LITERATURE

In addition to the product line handbooks listed below, the INTEL PRODUCT GUIDE (no charge, Order No. 210846-003) provides an overview of Intel’s complete product lines and customer services.

Consult the INTEL LITERATURE GUIDE (Order No. 210620) for a listing of Intel literature. TO ORDER literature in the U.S., write or call the INTEL LITERATURE DEPARTMENT, 3065 Bowers Avenue, Santa Clara, CA 95051, (800) 538-1876, or (800) 672-1833 (California only). TO ORDER literature from international locations, contact the nearest Intel sales office or distributor (see listings in the back of most any Intel literature).

Use the order blank on the facing page or call our TOLL FREE number listed above to order literature. Remember to add your local sales tax.

1985 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

*U.S. PRICE

QUALITY/RELIABILITY HANDBOOK (Order No. 210997-001) $15.00
Contains technical details of both quality and reliability programs and principles.

CHMOS HANDBOOK (Order No. 290005-001) $12.00
Contains data sheets only on all microprocessor, peripheral, microcontroller and memory CHMOS components.

MEMORY COMPONENTS HANDBOOK (Order No. 210830-004) $18.00

TELECOMMUNICATION PRODUCTS HANDBOOK (Order No. 230730-003) $12.00

MICROCONTROLLER HANDBOOK (Order No. 210918-003) $18.00

MICROSYSTEM COMPONENTS HANDBOOK (Order No. 230843-002) $25.00
Microprocessors and peripherals—2 Volume Set

DEVELOPMENT SYSTEMS HANDBOOK (Order No. 210940-003) $15.00

OEM SYSTEMS HANDBOOK (Order No. 210941-003) $18.00

SOFTWARE HANDBOOK (Order No. 230786-002) $12.00

MILITARY HANDBOOK (Order No. 210461-003) $15.00
Not available until June.

COMPLETE SET OF HANDBOOKS (Order No. 231003-002) $120.00
Get a 25% discount off the retail price of $160.

*U.S. Price Only
**U.S. LITERATURE ORDER FORM**

NAME: ___________________________ TITLE: ________________

COMPANY: ________________________________________________

ADDRESS: ________________________________________________

CITY: ___________________________ STATE: ________ ZIP: ________

COUNTRY: ________________________________________________

PHONE NO.: ___________________________

<table>
<thead>
<tr>
<th>ORDER NO.</th>
<th>TITLE</th>
<th>QTY.</th>
<th>PRICE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

POSTAGE AND HANDLING:
Add appropriate postage and handling to subtotal
10% U.S.
20% Canada

Subtotal ____________

Your Local Sales Tax ____________

Allow 4–6 weeks for delivery

Total ____________

Pay by Visa, MasterCard, Check or Money Order, payable to Intel Literature. Purchase Orders have a $50.00 minimum.

- [ ] Visa  Account No. ___________________________ Expiration ____________
- [ ] MasterCard  Date ____________

Signature: ____________________________

**Mail To:** Intel Literature Distribution
Mail Stop SC6-714
3065 Bowers Avenue
Santa Clara, CA 95051.

Customers outside the U.S. and Canada should contact the local Intel Sales Office or Distributor listed in the back of this book.

For information on quantity discounts, call the 800 number below:

**TOLL-FREE NUMBER:** (800) 548-4725

Prices good until 12/31/85.

Source HB
Mail To: Intel Literature Distribution
Mail Stop SC6-714
3065 Bowers Avenue
Santa Clara, CA 95051.
About Our Cover:
The design on our front cover is an abstract portrayal of microprocessors and associated peripherals as the building blocks which provide total systems development solutions. Intel superior technology and reliability provide easier solutions to specific development problems thereby cutting "time-to-market" and creating a greater market share.
Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, i, iCE, iCS, IDBP, iDIS, iICE, iLBX, iM, iMDDX, IMM, Insite, Intel, int, intIBOS, Intelevision, intelligent Identifier, intelligent Programming, Intellec, Intenlink, iOSP, iPDS, iRMX, iSBC, iSBX, iSDM, iSXN, KEPROM, Library Manager, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, OpeNET, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Ripplemode, RMX/80, RUPI, Seamless, SLD, SYSTEM 2000, and UPI, and the combination of ICE, iCS, iRMX, iSBC, iSBX, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Department
3065 Bowers Avenue
Santa Clara, CA 95051
Table of Contents

CHAPTER 1
OVERVIEW
Introduction ................................................................. 1-1

CHAPTER 2
MCS®-80/85 MICROPROCESSORS
8080A/8080A-1/8080A-2, 8-Bit N-Channel Microprocessor .................... 2-1
8085AH/8085AH-2/8085AH-1 8-Bit HMOS Microprocessors .................... 2-10
8155H/8156H/8155H-2/8156H-2 2048-Bit Static HMOS RAM with I/O Ports and Timer 2-26
8185/8185-2 1024 x 8-Bit Static RAM for MCS-85 ...................... 2-38
8205 High Speed 1 out of 8 Binary Decoder ................................... 2-43
8224 Clock Generator and Driver for 8080A CPU ................................... 2-48
8228/8238 System Controller and Bus Driver for 8080A CPU ........................ 2-53
8237A/8237A-4/8237A-5 High Performance Programmable DMA Controller .................... 2-57
8257/8257-5 Programmable DMA Controller ........................................ 2-72
8259A/8259A-2/8259A-8 Programmable Interrupt Controller ..................... 2-89
8755A/8755A-2 16, 384-Bit EPROM with I/O ........................................ 2-107
AP-59 Using the 8259A Programmable Interrupt Controller ................. 2-118

CHAPTER 3
IAPX 86, 88, 186, 188 MICROPROCESSORS
iAPX 86/10 16-Bit HMOS Microprocessor ....................................... 3-1
iAPX 88/10 8-Bit HMOS Microprocessor .......................................... 3-79
iAPX 188 High Integration 8-Bit Microprocessor ............................... 3-106
8089 8 & 16-Bit HMOS I/O Processor ............................................. 3-161
8087/8087-2/8087-1 Numeric Data Coprocessor .................................... 3-175
80130/80130-2 IAPX 86/30, 88/30, 186/30, 188/30 iRMX™ 86 Operating System Processors 3-198
80150/80150-2 IAPX 86/50, 88/50, 186/50, 188/50 CP/M®-86 Operating System Processors 3-220
8282/8283 Octal Latch ............................................................ 3-232
8284A/8284A-1 Clock Generator and Driver for IAPX 86, 88 Processors ............. 3-237
8286/8287 Octal Bus Transceiver .................................................. 3-245
8288 Bus Controller for IAPX 86, 88 Processors .................................. 3-250
82188 Integrated Bus Controller for IAPX 86, 88, 186, 188 Processors ....... 3-257
8289/8289-1 Bus Arbitrer .......................................................... 3-274
AP-67 8086 System Design ...................................................... 3-285
AP-123 Graphic CRT Design Using the Intel 8089 ..................................... 3-348
AP-113 Getting Started with the Numeric Data Processor ......................... 3-420
AP-143 Using the iAPX 86/20 Numeric Data Processor in a Small Business Computer 3-481
AP-144 Three Dimensional Graphics Application of the IAPX 86/20 Numeric Data Processor 3-504
AP-186 Introduction to the 80186 ................................................ 3-543

CHAPTER 4
IAPX 286 MICROPROCESSORS
iAPX 286/10 High Performance Microprocessor with Memory Management and Protection 4-1
80287 80-Bit HMOS Numeric Processor Extension .................................. 4-54
82258 Advanced DMA Controller Architectural Overview .......................... 4-79
82284 Clock Generator and Ready Interface for IAPX 286 Processors .............. 4-92
82288 Bus Controller for iAPX 286 Processors .................................... 4-100
82289 Bus Arbitr for iAPX 286 Processor Family ................................... 4-118

*CP/M is a Trademark of Digital Research, Inc.
CHAPTER 5
MEMORY CONTROLLERS

DATA SHEETS
8202A Dynamic RAM Controller ................................................. 5-1
8203 64K Dynamic RAM Controller ............................................. 5-15
8206/8206-2 Error Detection and Correction Unit ........................ 5-30
8207 Dual-Port Dynamic RAM Controller ..................................... 5-51
8208 Dynamic RAM Controller ................................................... 5-98

USERS MANUAL
Introduction .............................................................................. 5-117
Programming the 8207 ............................................................ 5-118
RAM Interface ........................................................................ 5-123
Microprocessor Interfaces ....................................................... 5-132
8207 with ECC (8206) ............................................................. 5-140
Appendix .................................................................................. 5-143

APPLICATION NOTES
AP-97A Interfacing Dynamic RAM to iAPX 86/88 Using the 8202A and 8203 5-147
AP-141 8203/8206/2164A Memory Design ................................ 5-183
AP-167 Interfacing the 8207 Dynamic RAM Controller to the iAPX 186 5-189
AP-168 Interfacing the 8207 Advanced Dynamic RAM Controller to the iAPX 286 5-194

ARTICLE REPRINTS
AR-364 FAE News 1/84 "8208 with 186" .................................... 5-201
AR-231 Dynamic RAM Controller Orchestrates Memory Systems .... 5-212

— VOLUME 2 —

SUPPORT PERIPHERALS
DATA SHEETS
8231A Arithmetic Processing Unit .............................................. 5-219
8253/8253-5 Programmable Interval Timer ................................ 5-229
8254 Programmable Interval Timer ........................................... 5-240
82C54 CHMOS Programmable Interval Timer ............................ 5-256
8255A/8255A-5 Programmable Peripheral Interface .................... 5-273
82C55A CHMOS Programmable Peripheral Interface .................. 5-294
8256AH Multifunction Microprocessor Support Controller ............ 5-317
8279/8279-5 Programmable Keyboard/Display Interface ............ 5-340

APPLICATION NOTES
AP-153 Designing with the 8256 ............................................... 5-352
AP-183 8256AH Application Note ............................................ 5-427

FLOPPY DISK CONTROLLERS
DATA SHEETS
8272A Single/Double Density Floppy Disk Controller ............... 5-444

APPLICATION NOTES
AP-116 An Intelligent Data Base System Using the 8272 ............ 5-463
AP-121 Software Design and Implementation of Floppy Disk Systems 5-504

HARD DISK CONTROLLERS
DATA SHEETS
82062 Winchester Disk Controller ............................................. 5-574
82064 Winchester Disk Controller with On-Chip Error Detection and Correction 5-601

UPI USERS MANUAL
Introduction ............................................................................. 5-635
Functional Description ............................................................ 5-639
Instruction Set ......................................................................... 5-656
Single-Step, Programming, and Power-Down Modes .................. 5-683
System Operation .................................................................. 5-688
Applications .......................................................................... 5-694
AP-161 Complex Peripheral Control with the UPI-42 .................. 5-750
AP-90 An 8741A/8041A Digital Cassette Controller ................. 5-806
CHAPTER 6
DATA COMMUNICATIONS
INTRODUCTION
Intel Data Communications Family Overview ........................................ 6-1

GLOBAL COMMUNICATIONS
DATA SHEETS
8251A Programmable Communication Interface .................................. 6-3
8273/8273-4 Programmable HDLC/SDLC Protocol Controller .............. 6-20
8274 Multi-Protocol Serial Controller (MPSC) ..................................... 6-48
82530/82530-6 Serial Communications Controller (SCC) .................... 6-85

APPLICATION NOTES
AP-16 Using the 8251 Universal Synchronous/Asynchronous Receiver/Transmitter .................................................. 6-113
AP-36 Using the 8273 SDLC/HDLC Protocol Controller ...................... 6-144
AP-134 Asynchronous Communications with the 8274 Multiple Protocol Serial Controller ................................................. 6-191
AP-145 Synchronous Communications with the 8274 Multiple Protocol Serial Controller .................................................. 6-228
AP-222 Asynchronous SDLC Communications with 82530 .................... 6-268

LOCAL AREA NETWORKS
DATA SHEETS
82501 Ethernet Serial Interface .......................................................... 6-288
82C502 Ethernet Transceiver Chip Data Sheet .................................... 6-299
82586 Local Area Network Coprocessor .............................................. 6-302
82588 Single Chip LAN Controller ...................................................... 6-336

ARTICLE REPRINTS
AR-345 Build a VLSI-Based Workstation for the Ethernet Environment ................................................................. 6-362
AR-346 VLSI Solutions for Tiered Office Networks .............................. 6-370
AR-342 Chips Support Two Local Area Networks ................................. 6-380

OTHER DATA COMMUNICATIONS
DATA SHEETS
8291A GPIB Talker/Listener ............................................................... 6-386
8292 GPIB Controller ........................................................................ 6-415
8294A Data Encryption Unit ............................................................... 6-430

APPLICATION NOTES
AP-66 Using the 8292 GPIB Controller .............................................. 6-442
AP-166 Using the 8291A GPIB Talker/Listener .................................... 6-496

ARTICLE REPRINTS
AR-208 LSI Transceiver Chips Complete GPIB Interface ...................... 6-528
AR-113 LSI Chips Ease Standard 488 Bus Interfacing ........................... 6-536

TUTORIAL
Data Encryption Tutorial .................................................................. 6-546
CHAPTER 7
ALPHANUMERIC TERMINAL CONTROLLERS
DATA SHEETS
8275H Programmable CRT Controller ........................................ 7-1
8276H Small System CRT Controller ........................................... 7-25
APPLICATION NOTES
AP-62 A Low Cost CRT Terminal Using the 8275 ......................... 7-42
ARTICLE REPRINTS
AR-178 A Low Cost CRT Terminal Does More with Less ............... 7-84
GRAPHICS DISPLAY PRODUCTS
DATA SHEETS
82720 Graphics Display Controller .............................................. 7-91
ARTICLE REPRINTS
AR-255 Dedicated VLSI Chip Lightens Graphic Display
Design Load ................................................................. 7-128
AR-298 Graphics Chip Makes Low Cost High Resolution, Color
Displays Possible .......................................................... 7-136
TEXT PROCESSING PRODUCTS
DATA SHEETS
82730 Text Coprocessor .......................................................... 7-143
82731 Video Interface Controller .............................................. 7-187
ARTICLE REPRINTS
AR-305 Text Coprocessor Brings Quality to CRT Displays ............ 7-206
AR-297 VLSI Coprocessor Delivers High Quality Displays ............ 7-214
AR-296 Mighty Chips .......................................................... 7-217
Numeric Index

80130/81030-2 iAPX 86/30, 88/30, 186/30, 188/30 iRMX* 86 Operating System Processors .... 3-198
80150/80150-2 iAPX 86/50, 88/50, 186/50, 188/50 C/PM*-86 Operating System Processors .... 3-220
80186 (iAPX 186) High Integration 16-Bit Microprocessor .............................................. 3-25, 3-543
80188 (iAPX 188) High Integration 8-Bit Microprocessor .................................................. 3-106
80286 (iAPX 286/10) High Performance Microprocessor with Memory Management and Protection ................................................................. 4-1
80287 80-Bit HMOS Numeric Processor Extension ................................................................. 4-54
8041A/8641A/8741A Universal Peripheral Interface 8-Bit Microcomputer .......................... 5-814, 5-635, 5-639
8042/8742 Universal Peripheral Interface 8-Bit Microcomputer ......................................... 5-826, 5-635, 5-639, 5-910
8080A/8080A-1/8080A-2, 8-Bit N-Channel Microprocessor .................................................. 2-1
8085AH/8085AH-2/8085AH-1 8-Bit HMOS Microprocessors ................................................. 2-10
8086 (iAPX 86/10) 16-Bit HMOS Microprocessor ................................................................. 3-1, 3-285
8087/8087-2/8087-1 Numeric Data Coprocessor ................................................................... 3-175, 3-420, 3-481, 3-504, 6-362
8088 (iAPX 88/10) 8-Bit HMOS Microprocessor ..................................................................... 3-79
8089 8 & 16-Bit HMOS I/O Processor .................................................................................... 3-161, 3-348
8155H/8156H/8155H-2/8156H-2 2048-Bit Static HMOS RAM with I/O Ports and Timer ....... 2-26
8185/8185-2 1024 x 8-Bit Static RAM for MCS®-85 .............................................................. 2-38
8202A Dynamic RAM Controller ............................................................................................ 5-1, 5-147
8203 64K Dynamic RAM Controller ....................................................................................... 5-15, 5-147, 5-183
8205 High Speed 1 out of 8 Binary Decoder ......................................................................... 2-43
8206/8206-2 Error Detection and Correction Unit ................................................................. 5-30, 5-183, 5-212
82062 Winchester Disk Controller .......................................................................................... 5-574, 5-846
82064 Winchester Disk Controller with On-Chip Error Detection and Correction .......... 5-601
8207 Dual-Port Dynamic RAM Controller ............................................................................. 5-51, 5-118, 5-123, 5-132, 5-140, 5-143, 5-183, 5-189, 5-194, 5-212
8208 Dynamic RAM Controller ............................................................................................. 5-98, 5-201
82188 Integrated Bus Controller for iAPX 86, 88, 186, 188 Processors .............................. 3-257
8224 Clock Generator And Driver for 8080A CPU ................................................................ 2-48
82258 Advanced DMA Controller Architectural Overview ..................................................... 4-79
8228/8238 System Controller and Bus Driver for 8080A CPU .............................................. 2-53
82284 Clock Generator and Ready Interface for iAPX 286 Processors ............................... 4-92
82288 Bus Controller for iAPX 286 Processors .................................................................... 4-100
82289 Bus Arbiter for iAPX 286 Processor Family ................................................................. 4-118
8231A Arithmetic Processing Unit ......................................................................................... 5-219
8237A/8237A-4/8237A-5 High Performance Programmable DMA Controller ..................... 2-57
8243 MCS-48 Input/Output Expander ...................................................................................... 5-635, 5-840
82501 Ethernet Serial Interface ............................................................................................. 6-288, 6-362, 6-380
82C502 Ethernet Transceiver Chip ......................................................................................... 6-299
8251A Programmable Communication Interface ..................................................................... 6-3, 6-113
8253/8253-5 Programmable Interval Timer .......................................................................... 5-229
82530/82530-6 Serial Communications Controller (SCC) .................................................... 6-85, 6-268
<table>
<thead>
<tr>
<th>Product Code</th>
<th>Description</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>8254</td>
<td>Programmable Interval Timer</td>
<td>5-240</td>
</tr>
<tr>
<td>8254A</td>
<td>CHMOS Programmable Interval Timer</td>
<td>5-256</td>
</tr>
<tr>
<td>8255A</td>
<td>Programmable Peripheral Interface</td>
<td>5-273, 7-84</td>
</tr>
<tr>
<td>8255A-5</td>
<td>CHMOS Programmable Peripheral Interface</td>
<td>5-294</td>
</tr>
<tr>
<td>8256AH</td>
<td>Multifunction Microprocessor Support Controller</td>
<td>5-317, 5-352, 5-427</td>
</tr>
<tr>
<td>8257/8257-5</td>
<td>Programmable DMA Controller</td>
<td>2-72</td>
</tr>
<tr>
<td>82586</td>
<td>Local Area Network Coprocessor</td>
<td>6-302, 6-362, 6-370, 6-380</td>
</tr>
<tr>
<td>82588</td>
<td>Personal Workstation Lan Control</td>
<td>6-336</td>
</tr>
<tr>
<td>8259A</td>
<td>Programmable Interrupt Controller</td>
<td>2-89, 2-118</td>
</tr>
<tr>
<td>8259A-2</td>
<td>CHMOS Programmable Interrupt Controller</td>
<td></td>
</tr>
<tr>
<td>8259A-8</td>
<td>Programmable Interrupt Controller</td>
<td></td>
</tr>
<tr>
<td>8272A</td>
<td>Single/Double Density Floppy Disk Controller</td>
<td>5-444, 5-463, 5-504, 7-128</td>
</tr>
<tr>
<td>82720</td>
<td>Graphics Display Controller</td>
<td>7-91, 7-128, 7-136, 7-206, 7-214, 7-217</td>
</tr>
<tr>
<td>8273/8273-4</td>
<td>Programmable HDLC/SDLC Protocol Controller</td>
<td>6-20, 6-144, 6-380</td>
</tr>
<tr>
<td>82730</td>
<td>Text Coprocessor</td>
<td>6-262, 7-136, 7-143, 7-206, 7-214, 7-217</td>
</tr>
<tr>
<td>82731</td>
<td>Video Interface Controller</td>
<td>7-187, 7-206</td>
</tr>
<tr>
<td>8274</td>
<td>Multi-Protocol Serial Controller (MPSC)</td>
<td>6-48, 6-191, 6-228, 6-380</td>
</tr>
<tr>
<td>8275H</td>
<td>Programmable CRT Controller</td>
<td>7-1, 7-42</td>
</tr>
<tr>
<td>8276H</td>
<td>Small System CRT Controller</td>
<td>7-25, 7-84</td>
</tr>
<tr>
<td>8279/8279-5</td>
<td>Programmable Keyboard/Display Interface</td>
<td>5-340</td>
</tr>
<tr>
<td>8282/8283</td>
<td>Octal Latch</td>
<td>3-232</td>
</tr>
<tr>
<td>8284A</td>
<td>Clock Generator and Driver for iAPX 86, 88 Processors</td>
<td>3-327</td>
</tr>
<tr>
<td>8286/8287</td>
<td>Octal Bus Transceiver</td>
<td>3-245</td>
</tr>
<tr>
<td>8288</td>
<td>Bus Controller for iAPX 86, 88 Processors</td>
<td>3-250, 6-362</td>
</tr>
<tr>
<td>8289/8989-1</td>
<td>Bus Arbiter</td>
<td>3-274</td>
</tr>
<tr>
<td>8291A</td>
<td>GPIB Talker/Listener</td>
<td>6-386, 6-496, 6-528, 6-536</td>
</tr>
<tr>
<td>8292</td>
<td>GPIB Controller</td>
<td>6-415, 6-442, 6-528, 6-536</td>
</tr>
<tr>
<td>8294A</td>
<td>Data Encryption Unit</td>
<td>6-430</td>
</tr>
<tr>
<td>8755A/8755A-2</td>
<td>16,384-Bit EPROM with I/O</td>
<td>2-107</td>
</tr>
</tbody>
</table>

viii
8202A
DYNAMIC RAM CONTROLLER

- Provides All Signals Necessary to Control 2117, or 2118 Dynamic Memories
- Directly Addresses and Drives Up to 64K Bytes Without External Drivers
- Provides Address Multiplexing and Strobes
- Provides a Refresh Timer and a Refresh Counter
- Refresh Cycles May be Internally or Externally Requested
- Provides Transparent Refresh Capability
- Fully Compatible with Intel® 8080A, 8085A, iAPX 88, and iAPX 86 Family Microprocessors
- Decodes CPU Status for Advanced Read Capability with the 8202A-1 or 8202A-3
- Provides System Acknowledge and Transfer Acknowledge Signals
- Internal Clock Capability with the 8202A-1 or 8202A-3

The Intel® 8202A is a Dynamic Ram System Controller designed to provide all signals necessary to use 2117 or 2118 Dynamic RAMs in microcomputer systems. The 8202A provides multiplexed addresses and address strobes, as well as refresh/access arbitration. The 8202A-1 or 8202A-3 support an internal crystal oscillator.

Figure 1. 8202A Block Diagram
Figure 2. Pin Configuration
Table 1. Pin Descriptions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL0</td>
<td>6</td>
<td>I</td>
<td>Address Low: CPU address inputs used to generate memory row address.</td>
</tr>
<tr>
<td>AL1</td>
<td>8</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL2</td>
<td>10</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL3</td>
<td>12</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL4</td>
<td>14</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL5</td>
<td>16</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL6</td>
<td>18</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH0</td>
<td>5</td>
<td>I</td>
<td>Address High: CPU address inputs used to generate memory column address.</td>
</tr>
<tr>
<td>AH1</td>
<td>4</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH2</td>
<td>3</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH3</td>
<td>2</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH4</td>
<td>1</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH5</td>
<td>39</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH6</td>
<td>38</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>BO</td>
<td>24</td>
<td>I</td>
<td>Bank Select Inputs: Used to gate the appropriate RAS0-RAS3 output for a memory cycle. B1/OP1 option used to select the Advanced Read Mode.</td>
</tr>
<tr>
<td>B1/OP1</td>
<td>25</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>PCS</td>
<td>33</td>
<td>I</td>
<td>Protected Chip Select: Used to enable the memory read and write inputs. Once a cycle is started, it will not abort even if PCS goes inactive before cycle completion.</td>
</tr>
<tr>
<td>WR</td>
<td>31</td>
<td>I</td>
<td>Memory Write Request.</td>
</tr>
<tr>
<td>RD/S1</td>
<td>32</td>
<td>I</td>
<td>Memory Read Request: S1 function used in Advanced Read mode selected by OP1 (pin 25).</td>
</tr>
<tr>
<td>REFREQ/ALE</td>
<td>34</td>
<td>I</td>
<td>External Refresh Request: ALE function used in Advanced Read mode, selected by OP1 (pin 25).</td>
</tr>
<tr>
<td>OUT0</td>
<td>7</td>
<td>O</td>
<td>Output of the Multiplexer: These outputs are designed to drive the addresses of the Dynamic RAM array. (Note that the OUT0-6 pins do not require inverters or drivers for proper operation.)</td>
</tr>
<tr>
<td>OUT1</td>
<td>9</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUT2</td>
<td>11</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUT3</td>
<td>13</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUT4</td>
<td>15</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUT5</td>
<td>17</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUT6</td>
<td>19</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>WE</td>
<td>28</td>
<td>O</td>
<td>Write Enable: Drives the Write Enable inputs of the Dynamic RAM array.</td>
</tr>
<tr>
<td>CAS</td>
<td>27</td>
<td>O</td>
<td>Column Address Strobe: This output is used to latch the Column Address into the Dynamic RAM array.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS0</td>
<td>21</td>
<td>O</td>
<td>Row Address Strobe: Used to latch the Row Address into the bank of dynamic RAMs, selected by the 8202A Bank Select pins (B0, B1/OP1).</td>
</tr>
<tr>
<td>RAS1</td>
<td>22</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>RAS2</td>
<td>23</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>RAS3</td>
<td>26</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>XACK</td>
<td>29</td>
<td>O</td>
<td>Transfer Acknowledge: This output is a strobe indicating valid data during a read cycle or data written during a write cycle. XACK can be used to latch valid data from the RAM array.</td>
</tr>
<tr>
<td>SACK</td>
<td>30</td>
<td>O</td>
<td>System Acknowledge: This output indicates the beginning of a memory access cycle. It can be used as an advanced transfer acknowledge to eliminate wait states. (Note: If a memory access request is made during a refresh cycle, SACK is delayed until XACK in the memory access cycle).</td>
</tr>
<tr>
<td>(X0) OP2</td>
<td>36</td>
<td>I/O</td>
<td>Oscillator Inputs: These inputs are designed for a quartz crystal to control the frequency of the oscillator. If X0/OP2 is connected to a 1kΩ resistor pulled to +12V then X1/CLK becomes a TTL input for an external clock.</td>
</tr>
<tr>
<td>(X1) CLK</td>
<td>37</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>N.C.</td>
<td>35</td>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td></td>
<td>Power Supply: +5V.</td>
</tr>
<tr>
<td>GND</td>
<td>20</td>
<td></td>
<td>Ground.</td>
</tr>
</tbody>
</table>

NOTE: Crystal mode for the 8202A-1 or 8202A-3 only.

Figure 3. Crystal Operation for the 8202A-1 and the 8202A-3
Functional Description
The 8202A provides a complete dynamic RAM controller for microprocessor systems as well as expansion memory boards. All of the necessary control signals are provided for 2117 and 2118 dynamic RAMs.

All 8202A timing is generated from a single reference clock. This clock is provided via an external oscillator or an on-chip crystal oscillator. All output signal transitions are synchronous with respect to this clock reference, except for the CPU handshake signals SACK and XACK (trailing edge).

CPU memory requests normally use the RD and WR inputs. The advanced READ mode allows ALE and S1 to be used in place of the RD input.

Failsafe refresh is provided via an internal refresh timer which generates internal refresh requests. Refresh requests can also be generated via the REFREQ input.

An on-chip synchronizer/arbitrator prevents memory and refresh requests from affecting a cycle in progress. The READ, WRITE, and external REFRESH requests may be asynchronous to the 8202A clock; on-chip logic will synchronize the requests, and the arbiter will decide if the requests should be delayed, pending completion of a cycle in progress.

Option Selection
The 8202A has two strapping options. When OP1 is selected (16K mode only), pin 32 changes from a RD input to an S1 input, and pin 34 changes from a REFREQ input to an ALE input. See "Refresh Cycles" and "Read Cycles" for more detail. OP1 is selected by tying pin 25 to +12V through a 5.1K ohm resistor on the 8202A-1 or 8202A-3 only.

When OP2 is selected, by connecting pin 36 to +12V through a 1K ohm resistor, pin 37 changes from a crystal input (X1) to the CLK input for an external TTL clock.

Refresh Timer
The refresh timer is used to monitor the time since the last refresh cycle occurred. When the appropriate amount of time has elapsed, the refresh timer will request a refresh cycle. External refresh requests will reset the refresh timer.

Refresh Counter
The refresh counter is used to sequentially refresh all of the memory’s rows. The 8-bit counter is incremented after every refresh cycle.

Address Multiplexer
The address multiplexer takes the address inputs and the refresh counter outputs, and gates them onto the address outputs at the appropriate time. The address outputs, in conjunction with the RAS and CAS outputs, determine the address used by the dynamic RAMs for read, write, and refresh cycles. During the first part of a read or write cycle, AL0-AL8 are gated to OUT0-OUT6, then AH0-AH8 are gated to the address outputs.

During a refresh cycle, the refresh counter is gated onto the address outputs. All refresh cycles are RAS-only refresh (CAS inactive, RAS active).

To minimize buffer delay, the information on the address outputs is inverted from that on the address inputs.

OUT0-OUT6 do not need inverters or buffers unless additional drive is required.

Synchronizer/Arbiter
The 8202A has three inputs, REFREQ/ALE (pin 34), RD (pin 32) and WR (pin 31). The RD and WR inputs allow an external CPU to request a memory read or write cycle, respectively. The REFREQ/ALE allows refresh requests to be requested external to the 8202A.

All three of these inputs may be asynchronous with respect to the 8202A’s clock. The arbiter will resolve conflicts between refresh and memory requests, for both pending cycles and cycles in progress. Read and write requests will be given priority over refresh requests.

System Operation
The 8202A is always in one of the following states:

a) IDLE
b) TEST Cycle
c) REFRESH Cycle
d) READ Cycle
e) WRITE Cycle

The 8202A is normally in the IDLE state. Whenever one of the other cycles is requested, the 8202A will leave the IDLE state to perform the desired cycle. If no other cycles are pending, the 8202A will return to the IDLE state.

<table>
<thead>
<tr>
<th>Description</th>
<th>Pin #</th>
<th>Normal Function</th>
<th>Option Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1/OP1</td>
<td>25</td>
<td>Bank (RAS) Select</td>
<td>Advanced-Read Mode (see text)</td>
</tr>
<tr>
<td>X0/OP2</td>
<td>36</td>
<td>Crystal Oscillator (8202A-1 or 8202A-3)</td>
<td>External Oscillator</td>
</tr>
</tbody>
</table>

Figure 4. 8202A Option Selection
Test Cycle

The TEST Cycle is used to check operation of several 8202A internal functions. TEST cycles are requested by activating the RD and WR inputs, independent of PCS. The TEST Cycle will reset the refresh address counter. It will perform a WRITE Cycle if PCS is low. The TEST Cycle should not be used in normal system operation, since it would affect the dynamic RAM refresh.

Refresh Cycles

The 8202A has two ways of providing dynamic RAM refresh:

1) Internal (failsafe) refresh
2) External (hidden) refresh

Both types of 8202A refresh cycles activate all of the RAS outputs, while CAS, WE, SACK, and XACK remain inactive.

Internal refresh is generated by the on-chip refresh timer. The timer uses the 8202A clock to ensure that refresh of all rows of the dynamic RAM occurs every 2 milliseconds. If REFRQ is inactive, the refresh timer will request a refresh cycle every 10-16 microseconds.

External refresh is requested via the REFRQ input (pin 34). External refresh control is not available when the Advanced-Read mode is selected. External refresh requests are latched, then synchronized to the 8202A clock.

The arbiter will allow the refresh request to start a refresh cycle only if the 8202A is not in the middle of a cycle.

Simultaneous memory request and external refresh request will result in the memory request being honored first. This 8202A characteristic can be used to "hide" refresh cycles during system operation. A circuit similar to Figure 5 can be used to decode the CPU's instruction fetch status to generate an external refresh request. The refresh request is latched while the 8202A performs the instruction fetch; the refresh cycle will start immediately after the memory cycle is completed, even if the RD input has not gone inactive. If the CPU's instruction decode time is long enough, the 8202A can complete the refresh cycle before the next memory request is generated.

Certain system configurations require complete external refresh requests. If external refresh is requested faster than the minimum internal refresh timer (tREF), then, in effect, all refresh cycles will be caused by the external refresh request, and the internal refresh timer will never generate a refresh request.

Read Cycles

The 8202A can accept two different types of memory Read requests:

1) Normal Read, via the RD input
2) Advanced Read, using the S1 and ALE inputs

The user can select the desired Read request configuration via the B1/OP1 hardware strapping option on pin 25.

<table>
<thead>
<tr>
<th>Pin 25</th>
<th>Normal Read</th>
<th>Advanced Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin 25</td>
<td>B1 input</td>
<td>+12 Volt Option</td>
</tr>
<tr>
<td>Pin 32</td>
<td>RD input</td>
<td>S1 input</td>
</tr>
<tr>
<td>Pin 34</td>
<td>REFRQ input</td>
<td>ALE input</td>
</tr>
<tr>
<td># RAM banks</td>
<td>4 (RAS 0-3)</td>
<td>2 (RAS 2-3)</td>
</tr>
<tr>
<td>Ext. Refresh Req.</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Figure 6. 8202A Read Options

Normal Reads are requested by activating the RD input, and keeping it active until the 8202A responds with an XACK pulse. The RD input can go inactive as soon as the command hold time (tCHS) is met.

Advanced Read cycles are requested by pulsing ALE while S1 is active; if S1 is inactive (low) ALE is ignored. Advanced Read timing is similar to Normal Read timing, except the falling edge of ALE is used as the cycle start reference.

If a Read cycle is requested while a refresh cycle is in progress, then the 8202A will set the internal delayed-SACK latch. When the Read cycle is eventually started, the 8202A will delay the active SACK transition until XACK goes active, as shown in the AC timing diagrams. This delay was designed to compensate for the CPU's READY setup and hold times. The delayed-SACK latch is cleared after every READ cycle.

Based on system requirements, either SACK or XACK can be used to generate the CPU READY signal. XACK will...
normally be used; if the CPU can tolerate an advanced READY, then SACK can be used, but only if the CPU can tolerate the amount of advance provided by SACK. If SACK arrives too early to provide the appropriate number of WAIT states, then either XACK or a delayed form of SACK should be used.

Write Cycles
Write cycles are similar to Normal Read cycles, except for the WE output. WE is held inactive for Read cycles, but goes active for Write cycles. All 8202A Write cycles are “early-write” cycles; WE goes active before CAS goes active by an amount of time sufficient to keep the dynamic RAM output buffers turned off.

General System Considerations
All memory requests (Normal Reads, Advanced Reads, Writes) are qualified by the PCS input. PCS should be stable, either active or inactive, prior to the leading edge of RD, WR, or ALE. Systems which use battery backup should pullup PCS to prevent erroneous memory requests, and should also pullup WR to keep the 8202A out of its test mode.

In order to minimize propagation delay, the 8202A uses an inverting address multiplexer without latches. The system must provide adequate address setup and hold times to guarantee RAS and CAS setup and hold times for the RAM. The 8202A tAD AC parameter should be used for this system calculation.

The B0-B1 inputs are similar to the address inputs in that they are not latched. B0 and B1 should not be changed during a memory cycle, since they directly control which RAS output is activated.

The 8202A uses a two-stage synchronizer for the memory request inputs (RD, WR, ALE), and a separate two stage synchronizer for the external refresh input (REFRQ). As with any synchronizer, there is always a finite probability of metastable states inducing system errors. The 8202A synchronizer was designed to have a system error rate less than 1 memory cycle every three years based on the full operating range of the 8202A.

A microprocessor system is concerned with the time data is valid after RD goes low. See Figure 7. In order to calculate memory read access times, the dynamic RAM’s A.C. specifications must be examined, especially the RAS-access time (tRAC) and the CAS-access time (tCAC). Most configurations will be CAS-access limited; i.e., the data from the RAM will be stable $t_{CC,\text{max}}(8202A) + t_{CAC}(\text{RAM})$ after a memory read cycle is started. Be sure to add any delays (due to buffers, data latches, etc.) to calculate the overall read access time.

Since the 8202A normally performs “early-write” cycles, the data must be stable at the RAM data inputs by the time CAS goes active, including the RAM’s data setup time. If the system does not normally guarantee sufficient write data setup, you must either delay the WR input signal or delay the 8202A WE output.

Delaying the WR input will delay all 8202A timing, including the READY handshake signals, SACK and XACK, which may increase the number of WAIT states generated by the CPU.

If the WE output is externally delayed beyond the CAS active transition, then the RAM will use the falling edge of WE to strobe the write data into the RAM. This WE transition should not occur too late during the CAS active transition, or else the WE to CAS requirements of the RAM will not be met.

![Figure 7. Read Access Time](image-url)
Figure 8. Typical 8088 System
ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias ............ 0°C to 70°C
Storage Temperature ................ -65°C to +150°C
Voltage On any Pin With Respect to Ground .... -0.5V to +7V
Power Dissipation .................. 1.5 Watts

*NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

\( T_A = 0°C \) to \( 70°C \); \( V_{CC} = 5.0V \pm 10% \), \( V_{CC} = 5.0V \pm 5% \) for 8202A-3, GND = 0V

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_C )</td>
<td>Input Clamp Voltage</td>
<td>(-1.0)</td>
<td>( V )</td>
<td>( I_C = -5 \text{ mA} )</td>
<td></td>
</tr>
<tr>
<td>( I_{CC} )</td>
<td>Power Supply Current</td>
<td>270</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( I_F )</td>
<td>Forward Input Current</td>
<td>-2.0</td>
<td>mA</td>
<td>( V_F = 0.45V )</td>
<td></td>
</tr>
<tr>
<td>All Other Inputs(^3)</td>
<td>(-320)</td>
<td>( \mu A )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( I_R )</td>
<td>Reverse Input Current(^3)</td>
<td>40</td>
<td>( \mu A )</td>
<td>( V_R = V_{CC} ) (Note 1)</td>
<td></td>
</tr>
<tr>
<td>( V_{OL} )</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td>V</td>
<td>( I_{OL} = 5 \text{ mA} )</td>
<td></td>
</tr>
<tr>
<td>All Other Outputs</td>
<td>0.45</td>
<td>V</td>
<td>( I_{OL} = 3 \text{ mA} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( V_{OH} )</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td>V</td>
<td>( V_{IL} = 0.65V )</td>
<td></td>
</tr>
<tr>
<td>All Other Outputs</td>
<td>2.6</td>
<td>V</td>
<td>( I_{OH} = 1 \text{ mA} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( V_{IL} )</td>
<td>Input Low Voltage</td>
<td>0.8</td>
<td>V</td>
<td>( V_{CC} = 5.0V ) (Note 2)</td>
<td></td>
</tr>
<tr>
<td>( V_{IH} )</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>V</td>
<td>( V_{CC} = 5.0V )</td>
<td></td>
</tr>
<tr>
<td>( V_{IH2} )</td>
<td>Option Voltage</td>
<td>V</td>
<td>(Note 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{IN} )</td>
<td>Input Capacitance</td>
<td>30</td>
<td>pF</td>
<td>( F = 1 \text{ MHz} )</td>
<td></td>
</tr>
<tr>
<td>( V_{BIAS} = 2.5V, V_{CC} = 5V )</td>
<td>( T_A = 25°C )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTES:
1. \( I_R = 200\mu A \) for pin 37 (CLK) for external clock mode.
2. For test mode RD & WR must be held at GND
3. Except for pin 36
4. 8202A-1 and 8202A-3 supports both OP\(_1\) and OP\(_2\). 8202A only supports OP\(_2\).
A.C. CHARACTERISTICS

\( T_a = 0^\circ C \) to \( 70^\circ C \), \( V_{CC} = 5V \pm 10\% \), \( V_{CC} = 5V \pm 5\% \) for 8202A-3

Measurements made with respect to \( RAS_0-RAS_3, CAS, WE, OUT_0-OUT_6 \) are at 2.4V and 0.8V. All other pins are measured at 1.5V. All times are in nsec.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_p )</td>
<td>Clock Period</td>
<td>40</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>( t_{PH} )</td>
<td>External Clock High Time</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PL} )</td>
<td>External Clock Low Time—above (&gt; 20 mHz)</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PL} )</td>
<td>External Clock Low Time—below (&lt;) 20 mHz</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{RC} )</td>
<td>Memory Cycle Time</td>
<td>10( tp ) – 30</td>
<td>12( tp )</td>
<td>4, 5</td>
</tr>
<tr>
<td>( t_{REF} )</td>
<td>Refresh Time (128 cycles—16K mode)</td>
<td>264( tp )</td>
<td>288( tp )</td>
<td></td>
</tr>
<tr>
<td>( t_{RP} )</td>
<td>RAS Precharge Time</td>
<td>4( tp ) – 30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{RSH} )</td>
<td>RAS Hold After CAS</td>
<td>5( tp ) – 30</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( t_{ASR} )</td>
<td>Address Setup to RAS</td>
<td>( tp ) – 30</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( t_{RAH} )</td>
<td>Address Hold From RAS</td>
<td>( tp ) – 10</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( t_{ASC} )</td>
<td>Address Setup to CAS</td>
<td>( tp ) – 30</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( t_{CAH} )</td>
<td>Address Hold from CAS</td>
<td>5( tp ) – 20</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( t_{CAS} )</td>
<td>CAS Pulse Width</td>
<td>5( tp ) – 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{WCS} )</td>
<td>WE Setup to CAS</td>
<td>( tp ) – 40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{WCH} )</td>
<td>WE Hold After CAS</td>
<td>5( tp ) – 35</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>( t_{RS} )</td>
<td>RD, WR, ALE, REFQ delay from RAS</td>
<td>5( tp )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{MRP} )</td>
<td>RD, WR setup to RAS</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>( t_{RMS} )</td>
<td>REFQ setup to RD, WR</td>
<td>2( tp )</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>( t_{RMP} )</td>
<td>REFQ setup to RAS</td>
<td>2( tp )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PCS} )</td>
<td>PCS Setup to RD, WR, ALE</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{AL} )</td>
<td>S1 Setup to ALE</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{LA} )</td>
<td>S1 Hold from ALE</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{CR} )</td>
<td>RD, WR, ALE to RAS Delay</td>
<td>( tp ) + 30</td>
<td>2( tp ) + 70</td>
<td>2</td>
</tr>
<tr>
<td>( t_{CC} )</td>
<td>RD, WR, ALE to CAS Delay</td>
<td>3( tp ) + 25</td>
<td>4( tp ) + 85</td>
<td>2</td>
</tr>
<tr>
<td>( t_{SC} )</td>
<td>CMD Setup to Clock</td>
<td>15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( t_{MRS} )</td>
<td>RD, WR setup to REFQ</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{CA} )</td>
<td>RD, WR, ALE to SACK Delay</td>
<td>2( tp ) + 47</td>
<td>2, 9</td>
<td></td>
</tr>
<tr>
<td>( t_{CX} )</td>
<td>CAS to XACK Delay</td>
<td>5( tp ) – 25</td>
<td>5( tp ) + 20</td>
<td></td>
</tr>
<tr>
<td>( t_{CS} )</td>
<td>CAS to SACK Delay</td>
<td>5( tp ) – 25</td>
<td>5( tp ) + 40</td>
<td>2, 10</td>
</tr>
<tr>
<td>( t_{ACK} )</td>
<td>XACK to CAS Setup</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{XW} )</td>
<td>XACK Pulse Width</td>
<td>( tp ) – 25</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>( t_{CK} )</td>
<td>SACK, XACK turn-off Delay</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{KCH} )</td>
<td>CMD Inactive Hold after SACK, XACK</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{LL} )</td>
<td>REFQ Pulse Width</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{CHS} )</td>
<td>CMD Hold Time</td>
<td>30</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>( t_{RFR} )</td>
<td>REFQ to RAS Delay</td>
<td>4( tp ) + 100</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>( t_{WW} )</td>
<td>WR to WE Delay</td>
<td>0</td>
<td>50</td>
<td>8</td>
</tr>
<tr>
<td>( t_{AD} )</td>
<td>CPU Address Delay</td>
<td>0</td>
<td>40</td>
<td>3</td>
</tr>
</tbody>
</table>
WAVEFORMS
Normal Read or Write Cycle

Advanced Read Mode
'WAVEFORMS (cont’d)
Memory Compatibility Timing

Write Cycle Timing
WAVEFORMS (cont'd)
Read or Write Followed By External Refresh

External Refresh Followed By Read or Write
WAVEFORMS (cont’d)

Clock And System Timing

Table 2 8202A Output Test Loading.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Test Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACK, XACK</td>
<td>$C_L = 30 \text{ pF}$</td>
</tr>
<tr>
<td>OUT0-OUT6</td>
<td>$C_L = 160 \text{ pF}$</td>
</tr>
<tr>
<td>RAS0-RAS3</td>
<td>$C_L = 60 \text{ pF}$</td>
</tr>
<tr>
<td>WE</td>
<td>$C_L = 224 \text{ pF}$</td>
</tr>
<tr>
<td>CAS</td>
<td>$C_L = 320 \text{ pF}$</td>
</tr>
</tbody>
</table>

NOTES:
1. $t_{SC}$ is a reference point only. ALE, RD, WR, and REFREQ inputs do not have to be externally synchronized to 8202A clock.
2. If $t_{RS}$ min and $t_{MRS}$ min are met then, $t_{CA}$, $t_{CR}$, and $t_{CC}$ are valid, otherwise $t_{CS}$ is valid.
3. $t_{ASR}$, $t_{RAH}$, $t_{ASC}$, $t_{CAH}$, and $t_{RSH}$ depend upon B0-B1 and CPU address remaining stable throughout the memory cycle. The address inputs are not latched by the 8202A.
4. For back-to-back refresh cycles, $t_{RC}$ max = 13tp
5. $t_{RC}$ max is valid only if $t_{MP}$ min is met (READ, WRITE followed by REFRESH) or $t_{MP}$ min is met (REFRESH followed by READ, WRITE).
6. $t_{RFR}$ is valid only if $t_{RS}$ min and $t_{MRS}$ min are met.
7. $t_{XW}$ min applies when RD, WR has already gone high. Otherwise XACK follows RD, WR.
8. WE goes high according to $t_{WCH}$ or $t_{WW}$, whichever occurs first.

A.C. TESTING LOAD CIRCUIT

9. $t_{CA}$ applies only when in normal SACK mode.
10. $t_{CS}$ applies only when in delayed SACK mode.
11. $t_{CHS}$ must be met only to ensure a SACK active pulse when in delayed SACK mode. XACK will always be activated for at least $t_{XW}$ (tp – 25 nS). Violating $t_{CHS}$ min does not otherwise affect device operation.
The typical rising and falling characteristic curves for the OUT, RAS, CAS and WE output buffers can be used to determine the effects of capacitive loading on the A.C. Timing Parameters. Using this design tool in conjunction with the timing waveforms, the designer can determine typical timing shifts based on system capacitive load.

A.C. CHARACTERISTICS FOR DIFFERENT CAPACITIVE LOADS

NOTE:
Use the Test Load as the base capacitance for estimating timing shifts for system critical timing parameters.

MEASUREMENT CONDITIONS:

\begin{align*}
T_A &= 25^\circ C \\
V_{CC} &= \pm 5V \\
t_p &= 50 \text{ ns}
\end{align*}

Pins not measured are loaded with the Test Load capacitance.
Example: Find the effect on $t_{CR}$ and $t_{CC}$ using 64 2118 Dynamic RAMs configured in 4 banks.

1. Determine the typical RAS and CAS capacitance:
   From the data sheet $RAS = 4 \text{ pF}$ and $CAS = 4 \text{ pF}$.
   \[
   \text{RAS load} = 64 \text{ pF} + \text{board capacitance}.
   \]
   \[
   \text{CAS load} = 256 \text{ pF} + \text{board capacitance}.
   \]
   Assume 2 pF/in (trace length) for board capacitance.

2. From the waveform diagrams, we determine that the falling edge timing is needed for $t_{CR}$ and $t_{CC}$. Next find the curve that best approximates the test load; i.e., 68 pF for RAS and 330 pF for CAS.

3. If we use 72 pF for RAS loading, then the $t_{CR}$ (max.) spec should be increased by about 1 ns. Similarly if we use 288 pF for CAS, then $t_{CC}$ (min.) and (max.) should decrease about 1 ns.
**8203**

64K DYNAMIC RAM CONTROLLER

- Provides All Signals Necessary to Control 64K (2164) and 16K (2117, 2118) Dynamic Memories
- Directly Addresses and Drives Up to 64 Devices Without External Drivers
- Provides Address Multiplexing and Strobes
- Provides a Refresh Timer and a Refresh Counter
- Provides Refresh/Access Arbitration
- Internal Clock Capability with the 8203-1 and the 8203-3
- Fully Compatible with Intel® 8080A, 8085A, iAPX 88, and iAPX 86 Family Microprocessors
- Decodes CPU Status for Advanced Read Capability in 16K mode with the 8203-1 and the 8203-3.
- Provides System Acknowledge and Transfer Acknowledge Signals
- Refresh Cycles May be Internally or Externally Requested (For Transparent Refresh)
- Internal Series Damping Resistors on All RAM Outputs

The Intel® 8203 is a Dynamic Ram System Controller designed to provide all signals necessary to use 2164, 2118 or 2117 Dynamic RAMs in microcomputer systems. The 8203 provides multiplexed addresses and address strobes, refresh logic, refresh/access arbitration. Refresh cycles can be started internally or externally. The 8203-1 and the 8203-3 support an internal crystal oscillator and Advanced Read Capability. The 8203-3 is a ±5% VCC part.

---

**Figure 1. 8203 Block Diagram**

**Figure 2. Pin Configuration**
### Table 1. Pin Descriptions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALO</td>
<td>6</td>
<td>I</td>
<td>Address Low: CPU address inputs used to generate memory row address.</td>
</tr>
<tr>
<td>AL1</td>
<td>8</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL2</td>
<td>10</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL3</td>
<td>12</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL4</td>
<td>14</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL5</td>
<td>16</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL6</td>
<td>18</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH0</td>
<td>5</td>
<td>I</td>
<td>Address High: CPU address inputs used to generate memory column address.</td>
</tr>
<tr>
<td>AH1</td>
<td>4</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH2</td>
<td>3</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH3</td>
<td>2</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH4</td>
<td>1</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH5</td>
<td>39</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH6</td>
<td>38</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>BO/AL7</td>
<td>24</td>
<td>I</td>
<td>Bank Select Inputs: Used to gate the appropriate RAS output for a memory cycle.</td>
</tr>
<tr>
<td>B1/OP1/AH7</td>
<td>25</td>
<td>I</td>
<td>B1/OP1 option used to select the Advanced Read Mode. (Not available in 64K mode.) See Figure 5. When in 64K RAM Mode, pins 24 and 25 operate as the AL7 and AH7 address inputs.</td>
</tr>
<tr>
<td>PCS</td>
<td>33</td>
<td>I</td>
<td>Protected Chip Select: Used to enable the memory read and write inputs. Once a cycle is started, it will not abort even if PCS goes inactive before cycle completion.</td>
</tr>
<tr>
<td>WR</td>
<td>31</td>
<td>I</td>
<td>Memory Write Request.</td>
</tr>
<tr>
<td>RD/S1</td>
<td>32</td>
<td>I</td>
<td>Memory Read Request: S1 function used in Advanced Read mode selected by OP1 (pin 25).</td>
</tr>
<tr>
<td>REF/RO/ALE</td>
<td>34</td>
<td>I</td>
<td>External Refresh Request: ALE function used in Advanced Read mode, selected by OP1 (pin 25).</td>
</tr>
<tr>
<td>OUT0</td>
<td>7</td>
<td>O</td>
<td>Output of the Multiplexer: These outputs are designed to drive the addresses of the Dynamic RAM array. (Note that the OUT0-7 pins do not require inverters or drivers for proper operation.)</td>
</tr>
<tr>
<td>OUT1</td>
<td>9</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUT2</td>
<td>11</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUT3</td>
<td>13</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUT4</td>
<td>15</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUT5</td>
<td>17</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUT6</td>
<td>19</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>WE</td>
<td>28</td>
<td>O</td>
<td>Write Enable: Drives the Write Enable inputs of the Dynamic RAM array.</td>
</tr>
<tr>
<td>CAS</td>
<td>27</td>
<td>O</td>
<td>Column Address Strobe: This output is used to latch the Column Address into the Dynamic RAM array.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS0</td>
<td>21</td>
<td>O</td>
<td>Row Address Strobe: Used to latch the Row Address into the bank of dynamic RAMs, selected by the 8203 Bank Select pins (B0, B1/OP1). In 64K mode, only RAS0 and RAS1 are available; pin 23 operates as OUT7 and pin 26 operates as the B0 bank select input.</td>
</tr>
<tr>
<td>RAS1</td>
<td>22</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>RAS2/OUT7</td>
<td>23</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>RAS3/B0</td>
<td>26</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>XACK</td>
<td>29</td>
<td>O</td>
<td>Transfer Acknowledge: This output is a strobe indicating valid data during a read cycle or data written during a write cycle. XACK can be used to latch valid data from the RAM array.</td>
</tr>
<tr>
<td>SACK</td>
<td>30</td>
<td>O</td>
<td>System Acknowledge: This output indicates the beginning of a memory access cycle. It can be used as an advanced transfer acknowledge to eliminate wait states. (Note: If a memory access request is made during a refresh cycle, SACK is delayed until XACK in the memory access cycle).</td>
</tr>
<tr>
<td>X0/OP2</td>
<td>36</td>
<td>I/O</td>
<td>Oscillator Inputs: These inputs are designed for a quartz crystal to control the frequency of the oscillator. If X0/OP2 is shorted to pin 40 (VCC) or if X0/OP2 is connected to +12V through a 1kΩ resistor then X1/CLK becomes a TTL input for an external clock. (Note: Crystal mode for the 8203-1 and the 8203-3 only).</td>
</tr>
<tr>
<td>X1/CLK</td>
<td>37</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>16K/64K</td>
<td>35</td>
<td>I</td>
<td>Mode Select: This input selects 16K mode (2117, 2118) or 64K mode (2164). Pins 23-28 change function based on the mode of operation.</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td></td>
<td>Power Supply: +5V.</td>
</tr>
<tr>
<td>GND</td>
<td>20</td>
<td></td>
<td>Ground.</td>
</tr>
</tbody>
</table>

### Functional Description

The 8203 provides a complete dynamic RAM controller for microprocessor systems as well as expansion memory boards. All of the necessary control signals are provided for 2164, 2118 and 2117 dynamic RAMs.

The 8203 has two modes, one for 16K dynamic RAMs and one for 64Ks, controlled by pin 35.
All 8203 timing is generated from a single reference clock. This clock is provided via an external oscillator or an on-chip crystal oscillator. All output signal transitions are synchronous with respect to this clock reference, except for the trailing edges of the CPU handshake signals SACK and XACK.

CPU memory requests normally use the RD and WR inputs. The Advanced-Read mode allows ALE and S1 to be used in place of the RD input.

Failsafe refresh is provided via an internal timer which generates refresh requests. Refresh requests can also be generated via the REFREQ input.

An on-chip synchronizer/arbitrator prevents memory and refresh requests from affecting a cycle in progress. The READ, WRITE, and external REFRESH requests may be asynchronous to the 8203 clock; on-chip logic will synchronize the requests, and the arbiter will decide if the requests should be delayed, pending completion of a cycle in progress.

16K/64K Option Selection

Pin 35 is a strap input that controls the two 8203 modes. Figure 4 shows the four pins that are multiplexed. In 16K mode (pin 35 tied to VCC or left open), the 8203 has two Bank Select inputs to select one of four RAS outputs. In this mode, the 8203 is exactly compatible with the Intel 8202A Dynamic RAM Controller. In 64K mode (pin 35 tied to GND), there is only one Bank Select input (pin 26) to select the two RAS outputs. More than two banks of 64K dynamic RAM's can be used with external logic.

Other Option Selections

The 8203 has two strapping options. When OP1 is selected (16K mode only), pin 32 changes from a RD input to an S1 input, and pin 34 changes from a REFREQ input to an ALE input. See “Refresh Cycles” and “Read Cycles” for more detail. OP1 is selected by tying pin 25 to +12V through a 5.1K ohm resistor on the 8203-1 or 8203-3 only.

When OP2 is selected, the internal oscillator is disabled and pin 37 changes from a crystal input (X1) to a CK input for an external TTL clock. OP2 is selected by shorting pin 36 (X0/OP2) directly to pin 40 (VCC). No current limiting resistor should be used. OP2 may also be selected by tying pin 36 to +12V through a 1KΩ resistor.

Refresh Timer

The refresh timer is used to monitor the time since the last refresh cycle occurred. When the appropriate amount of time has elapsed, the refresh timer will request a refresh cycle. External refresh requests will reset the refresh timer.

Refresh Counter

The refresh counter is used to sequentially refresh all of the memory’s rows. The 8-bit counter is incremented after every refresh cycle.

16K Function | 64K Function
---|---
23 | RAS2 | Address Output (OUT7)
24 | Bank Select (B0) | Address Input (AL7)
25 | Bank Select (B1) | Address Input (AH7)
26 | RAS3 | Bank Select (B0)

Inputs | Outputs
---|---
B1 | B0 | RAS0 | RAS1 | RAS2 | RAS3
16K Mode
0 | 0 | 0 | 1 | 1 | 1
0 | 1 | 1 | 0 | 1 | 1
1 | 0 | 1 | 1 | 0 | 1
1 | 1 | 1 | 1 | 1 | 0
64K Mode
— | 0 | 0 | 1 | — | —
— | 1 | 1 | 0 | — | —

Description | Pin # | Normal Function | Option Function
---|---|---|---
B1/OP1 (16K only) / AH7 | 25 | Bank (RAS) Select | Advanced-Read Mode (8203-1, -3)
X0/OP2 | 36 | Crystal Oscillator (8203-1 and 8203-3) | External Oscillator
Address Multiplexer
The address multiplexer takes the address inputs and the refresh counter outputs, and gates them onto the address outputs at the appropriate time. The address outputs, in conjunction with the RAS and CAS outputs, determine the address used by the dynamic RAMs for read, write, and refresh cycles. During the first part of a read or write cycle, AL0-AL7 are gated to OUT0-OUT7, then AH0-AH7 are gated to the address outputs.

During a refresh cycle, the refresh counter is gated onto the address outputs. All refresh cycles are RAS-only refresh (CAS inactive, RAS active).

To minimize buffer delay, the information on the address outputs is inverted from that on the address inputs. OUT0-OUT7 do not need inverters or buffers unless additional drive is required.

Synchronizer/Arbiter
The 8203 has three inputs, REFRQ/ALE (pin 34), RD (pin 32) and WR (pin 31). The RD and WR inputs allow an external CPU to request a memory read or write cycle, respectively. The REFRQ/ALE input allows refresh requests to be requested external to the 8203.

All three of these inputs may be asynchronous with respect to the 8203's clock. The arbiter will resolve conflicts between refresh and memory requests, for both pending cycles and cycles in progress. Read and write requests will be given priority over refresh requests.

System Operation
The 8203 is always in one of the following states:

a) IDLE
b) TEST Cycle
c) REFRESH Cycle
d) READ Cycle
e) WRITE Cycle

The 8203 is normally in the IDLE state. Whenever one of the other cycles is requested, the 8203 will leave the IDLE state to perform the desired cycle. If no other cycles are pending, the 8203 will return to the IDLE state.

Test Cycle
The TEST Cycle is used to check operation of several 8203 internal functions. TEST cycles are requested by activating the PCS, RD and WR inputs. The TEST Cycle will reset the refresh address counter and perform a WRITE Cycle. The TEST Cycle should not be used in normal system operation, since it would affect the dynamic RAM refresh.

Refresh Cycles
The 8203 has two ways of providing dynamic RAM refresh:

1) Internal (failsafe) refresh
2) External (hidden) refresh

Both types of 8203 refresh cycles activate all of the RAS outputs, while CAS, WE, SACK, and XACK remain inactive.

Internal refresh is generated by the on-chip refresh timer. The timer uses the 8203 clock to ensure that refresh of all rows of the dynamic RAM occurs every 2 milliseconds (128 cycles) or every 4 milliseconds (256 cycles). If REFRQ is inactive, the refresh timer will request a refresh cycle every 10-16 microseconds.

External refresh is requested via the REFRQ input (pin 34). External refresh control is not available when the Advanced-Read mode is selected. External refresh requests are latched, then synchronized to the 8203 clock.

The arbiter will allow the refresh request to start a refresh cycle only if the 8203 is not in the middle of a cycle.

When the 8203 is in the idle state a simultaneous memory request and external refresh request will result in the memory request being honored first. This 8203 characteristic can be used to "hide" refresh cycles during system operation. A circuit similar to Figure 7 can be used to decode the CPU's instruction fetch status to generate an external refresh request. The refresh request is latched while the 8203 performs the instruction fetch; the refresh cycle will start immediately after the memory cycle is completed, even if the RD input has not gone inactive. If the CPU's instruction decode time is long enough, the 8203 can complete the refresh cycle before the next memory request is generated.

If the 8203 is not in the idle state then a simultaneous memory request and an external refresh request may result in the refresh request being honored first.

![Figure 7. Hidden Refresh](image_url)
Certain system configurations require complete external refresh requests. If external refresh is requested faster than the minimum internal refresh timer (tRGR), then, in effect, all refresh cycles will be caused by the external refresh request, and the internal refresh timer will never generate a refresh request.

**Read Cycles**
The 8203 can accept two different types of memory Read requests:

1) Normal Read, via the RD input
2) Advanced Read, using the S1 and ALE inputs (16K mode only)

The user can select the desired Read request configuration via the B1/OP1 hardware strapping option on pin 25.

<table>
<thead>
<tr>
<th>Normal Read</th>
<th>Advanced Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin 25</td>
<td>B1 input</td>
</tr>
<tr>
<td>Pin 32</td>
<td>RD input</td>
</tr>
<tr>
<td>Pin 34</td>
<td>REFQ input</td>
</tr>
<tr>
<td># RAM banks</td>
<td></td>
</tr>
<tr>
<td>Ext. Refresh</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>OP1 (+12V)</td>
</tr>
<tr>
<td></td>
<td>S1 input</td>
</tr>
<tr>
<td></td>
<td>ALE input</td>
</tr>
<tr>
<td></td>
<td>4 (RAS 0-3)</td>
</tr>
<tr>
<td></td>
<td>2 (RAS 2-3)</td>
</tr>
</tbody>
</table>

**Figure 8. 8203 Read Options**

Normal Reads are requested by activating the RD input, and keeping it active until the 8203 responds with an XACK pulse. The RD input can go inactive as soon as the command hold time (tCHS) is met.

Advanced Read cycles are requested by pulsing ALE while S1 is active; if S1 is inactive (low) ALE is ignored. Advanced Read timing is similar to Normal Read timing, except the falling edge of ALE is used as the cycle start reference.

If a Read cycle is requested while a refresh cycle is in progress, then the 8203 will set the internal delayed-SACK latch. When the Read cycle is eventually started, the 8203 will delay the active SACK transition until XACK goes active, as shown in the AC timing diagrams. This delay was designed to compensate for the CPU's READY setup and hold times. The delayed-SACK latch is cleared after every READ cycle.

Based on system requirements, either SACK or XACK can be used to generate the CPU READY signal. XACK will normally be used; if the CPU can tolerate an advanced READY, then SACK can be used, but only if the CPU can tolerate the amount of advance provided by SACK. If SACK arrives too early to provide the appropriate number of WAIT states, then either XACK or a delayed form of SACK should be used.

**Write Cycles**
Write cycles are similar to Normal Read cycles, except for the WE output. WE is held inactive for Read cycles, but goes active for Write cycles. All 8203 Write cycles are "early-write" cycles; WE goes active before CAS goes active by an amount of time sufficient to keep the dynamic RAM output buffers turned off.

**General System Considerations**
All memory requests (Normal Reads, Advanced Reads, Writes) are qualified by the PCS input. PCS should be stable, either active or inactive, prior to the leading edge of RD, WR, or ALE. Systems which use battery backup should pullup PCS to prevent erroneous memory requests.

In order to minimize propagation delay, the 8203 uses an inverting address multiplexer without latches. The system must provide adequate address setup and hold times to guarantee RAS and CAS setup and hold times for the RAM. The tAD AC parameter should be used for this system calculation.

The B0-B1 inputs are similar to the address inputs in that they are not latched. B0 and B1 should not be changed during a memory cycle, since they directly control which RAS output is activated.

The 8203 uses a two-stage synchronizer for the memory request inputs (RD, WR, ALE), and a separate two stage synchronizer for the external refresh input (REFQ). As with any synchronizer, there is always a finite probability of metastable states inducing system errors. The 8203 synchronizer was designed to have a system error rate less than 1 memory cycle every three years based on the full operating range of the 8203.

A microprocessor system is concerned when the data is valid after RD goes low. See Figure 9. In order to calculate memory read access times, the dynamic RAM's A.C. specifications must be examined, especially the RAS-access time (tRAC) and the CAS-access time (tCAC). Most configurations will be CAS-access limited; i.e., the data from the RAM will be stable tCC,max (8203) + tCAC (RAM) after a memory read cycle is started. Be sure to add any delays (due to buffers, data latches, etc.) to calculate the overall read access time.

Since the 8203 normally performs "early-write" cycles, the data must be stable at the RAM data inputs by the time CAS goes active, including the RAM's data setup time. If the system does not normally guarantee sufficient write data setup, you must either delay the WR input signal or delay the 8203 WE output.

Delaying the WR input will delay all 8203 timing, including the READY handshake signals, SACK and XACK, which
may increase the number of WAIT states generated by the CPU.

If the WE output is externally delayed beyond the CAS active transition, then the RAM will use the falling edge of WE to strobe the write data into the RAM. This WE transition should not occur too late during the CAS active transition, or else the WE to CAS requirements of the RAM will not be met.

The RAS0-3, CAS, OUT0-7, and WE outputs contain on-chip series damping resistors (typically 20Ω) to minimize overshoot.

Some dynamic RAMs require more than 2.4V VIH. Noise immunity may be improved for these RAMs by adding pull-up resistors to the 8203's outputs. Intel RAMs do not require pull-up resistors.

---

Figure 9. Read Access Time

---

Figure 10. Typical 8088 System
Figure 11. 8086/256K Byte System
ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias: 0°C to 70°C
Storage Temperature: -65°C to +150°C
Voltage On any Pin: With Respect to Ground: -0.5V to +7V
Power Dissipation: 1.6 Watts

*NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vc</td>
<td>Input Clamp Voltage</td>
<td>-1.0</td>
<td>V</td>
<td>IC = -5 mA</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>Power Supply Current</td>
<td>290</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>Forward Input Current CLK, 64K/16K Mode select</td>
<td>-2.0</td>
<td>mA</td>
<td>VF = 0.45V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>All Other Inputs</td>
<td>-320</td>
<td>μA</td>
<td>VF = 0.45V</td>
<td></td>
</tr>
<tr>
<td>IR</td>
<td>Reverse Input Current</td>
<td>40</td>
<td>μA</td>
<td>VR = VCC; Note 1</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage SACK, XACK</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 5 mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>All Other Outputs</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 3 mA</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage SACK, XACK</td>
<td>2.4</td>
<td>V</td>
<td>VI = 0.65 V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>All Other Outputs</td>
<td>2.6</td>
<td>V</td>
<td>VOH = -1 mA</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Input Low Voltage SACK, XACK</td>
<td>0.8</td>
<td>V</td>
<td>VCC = 5.0V (Note 2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>All Other Outputs</td>
<td>0.8</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Option Voltage VCC</td>
<td>2.0</td>
<td>V</td>
<td>VCC = 5.0V</td>
<td></td>
</tr>
<tr>
<td>CIN</td>
<td>Input Capacitance</td>
<td>30</td>
<td>pF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTES:
1. IIR = 200 μA for pin 37 (CLK).
2. For test mode RD & WR must be held at GND.
3. Except for pin 36 in XTAL mode.
4. 8203-1 and 8203-3 supports both OP1 and OP2, 8203 only supports OP2.
A.C. CHARACTERISTICS

\[ T_j = 0^\circ C \text{ to } 70^\circ C; \ V_{CC} = 5V \pm 10\% \ (5.0V \pm 5\% \ for \ 8203-3); \ GND = 0V \]

Measurements made with respect to RAS0-RAS3, CAS, WE, OUT0-OUT6 are at 2.4V and 0.8V. All other pins are measured at 1.5V. All times are in nsec.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>tp</td>
<td>Clock Period</td>
<td>40</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>tpH</td>
<td>External Clock High Time</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tpL</td>
<td>External Clock Low Time—above (&gt; 20 mHz)</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tpL</td>
<td>External Clock Low Time—below (≤ 20 mHz)</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tRC</td>
<td>Memory Cycle Time</td>
<td>10tp - 30</td>
<td>12tp</td>
<td>4, 5</td>
</tr>
<tr>
<td>tREF</td>
<td>Refresh Time (128 cycles)</td>
<td>264tp</td>
<td>288tp</td>
<td></td>
</tr>
<tr>
<td>tRP</td>
<td>RAS Precharge Time</td>
<td>4tp - 30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tRSH</td>
<td>RAS Hold After CAS</td>
<td>5tp - 30</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>tASR</td>
<td>Address Setup to RAS</td>
<td>tp - 30</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>tRAH</td>
<td>Address Hold From RAS</td>
<td>tp - 10</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>tASC</td>
<td>Address Setup to CAS</td>
<td>tp - 30</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>tCAH</td>
<td>Address Hold from CAS</td>
<td>5tp - 20</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>tCAS</td>
<td>CAS Pulse Width</td>
<td>5tp - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWCS</td>
<td>WE Setup to CAS</td>
<td>tp - 40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IWCH</td>
<td>WE Hold After CAS</td>
<td>5tp - 35</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>tRS</td>
<td>RD, WR, ALE, REFREQ delay from RAS</td>
<td>5tp</td>
<td></td>
<td>2, 6</td>
</tr>
<tr>
<td>tMRP</td>
<td>RD, WR setup to RAS</td>
<td>0</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>tRMS</td>
<td>REFREQ setup to RD, WR</td>
<td>2tp</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>tRMP</td>
<td>REFREQ setup to RAS</td>
<td>2tp</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>tPCS</td>
<td>PCS Setup to RD, WR, ALE</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tAL</td>
<td>S1 Setup to ALE</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tLA</td>
<td>S1 Hold from ALE</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tCR</td>
<td>RD, WR, ALE to RAS Delay</td>
<td>tp + 30</td>
<td>21p + 70</td>
<td>2</td>
</tr>
<tr>
<td>tCC</td>
<td>RD, WR, ALE to CAS Delay</td>
<td>3tp + 25</td>
<td>4tp + 85</td>
<td>2</td>
</tr>
<tr>
<td>tSC</td>
<td>CMD Setup to Clock</td>
<td>15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>tMRS</td>
<td>RD, WR setup to REFREQ</td>
<td>5</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>tCA</td>
<td>RD, WR, ALE to SACK Delay</td>
<td>2tp + 47</td>
<td></td>
<td>2, 9</td>
</tr>
<tr>
<td>tCX</td>
<td>CAS to XACK Delay</td>
<td>5tp - 25</td>
<td>5tp + 20</td>
<td></td>
</tr>
<tr>
<td>tCS</td>
<td>CAS to SACK Delay</td>
<td>5tp - 25</td>
<td>5tp + 40</td>
<td>2, 10</td>
</tr>
<tr>
<td>tACK</td>
<td>XACK to CAS Setup</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tXW</td>
<td>XACK Pulse Width</td>
<td>tp - 25</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>tCK</td>
<td>SACK, XACK turn-off Delay</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tKCH</td>
<td>CMD Inactive Hold after SACK, XACK</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tLL</td>
<td>REFREQ Pulse Width</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tCHS</td>
<td>CMD Hold Time</td>
<td>30</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>tRF</td>
<td>REFREQ to RAS Delay</td>
<td>4tp + 100</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>tWW</td>
<td>WR to WE Delay</td>
<td>0</td>
<td>50</td>
<td>8</td>
</tr>
<tr>
<td>tAD</td>
<td>CPU Address Delay</td>
<td>0</td>
<td>40</td>
<td>3</td>
</tr>
</tbody>
</table>
WAVEFORMS
Normal Read or Write Cycle

Advanced Read Mode
WAVEFORMS (cont’d)
Memory Compatibility Timing

Write Cycle Timing
WAVEFORMS (cont'd)

Read or Write Followed By External Refresh

External Refresh Followed By Read or Write
WAVEFORMS (cont’d)
Clock And System Timing

Table 2. 8203 Output Loading. All specifications are for the Test Load unless otherwise noted.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Test Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACK, XACK</td>
<td>C_L = 50 pF</td>
</tr>
<tr>
<td>OUT0-OUT6</td>
<td>C_L = 160 pF</td>
</tr>
<tr>
<td>RAS0-RAS3</td>
<td>C_L = 60 pF</td>
</tr>
<tr>
<td>WE</td>
<td>C_L = 224 pF</td>
</tr>
<tr>
<td>CAS</td>
<td>C_L = 320 pF</td>
</tr>
</tbody>
</table>

NOTES:
1. tSC is a reference point only. ALE, RD, WR, and REFREQ inputs do not have to be externally synchronized to 8203 clock.
2. If tRS min and tRMS min are met then tCA, tCR, and tCC are valid, otherwise tCS is valid.
3. tASR, tRAN, tASC, tCAH and tRSH depend upon B0-B1 and CPU address remaining stable throughout the memory cycle. The address inputs are not latched by the 8203.
4. For back-to-back refresh cycles, tPC max = 13p
5. tRSH is valid only if tMP min is met (READ, WRITE followed by REFRESH) or tRMP min is met (REFRESH followed by READ, WRITE).
6. tRFR is valid only if tRS min and tRMS min are met.
7. tW min applies when RD, WR has already gone high. Otherwise XACK follows RD, WR.
8. WE goes high according to tWCH or tWW, whichever occurs first.

A.C. TESTING LOAD CIRCUIT

NOTE: C_L includes jig capacitance

9. t_CA applies only when in normal SACK mode de.
10. t_CS applies only when in delayed SACK mode.
11. tCHS must be met only to ensure a SACK active pulse when in delayed SACK mode. XACK will always be activated for at least tWW (10-25 nS). Violating tCHS min does not otherwise affect device operation.
The typical rising and falling characteristic curves for the OUT, RAS, CAS and WE output buffers can be used to determine the effects of capacitive loading on the A.C. Timing Parameters. Using this design tool in conjunction with the timing waveforms, the designer can determine typical timing shifts based on system capacitive load.

### A.C. CHARACTERISTICS FOR DIFFERENT CAPACITIVE LOADS

![Diagram](image_url)

**NOTE:**
Use the Test Load as the base capacitance for estimating timing shifts for system critical timing parameters.

**MEASUREMENT CONDITIONS:**
- $T_A = 25^\circ C$
- $V_{CC} = +5V$
- $t_P = 50 \text{ ns}$

- Pins not measured are loaded with the Test Load capacitance
Example: Find the effect on tCR and tCC using 32 2164 Dynamic RAMs configured in 2 banks.

1. Determine the typical RAS and CAS capacitance:
   From the data sheet RAS = 5 pF and CAS = 5 pF.
   \[ \text{RAS load} = 80 \text{ pF} + \text{board capacitance} \]
   \[ \text{CAS load} = 160 \text{ pF} + \text{board capacitance} \]
   Assume 2 pF/in (trace length) for board capacitance and for this example 4 inches for RAS and 8 inches for CAS.

2. From the waveform diagrams, we determine that the falling edge timing is needed for tCR and tCC. Next find the curve that best approximates the test load; i.e., 68 pF for RAS and 330 pF for CAS.

3. If we use 88 pF for RAS loading, then tCR (min.) spec should be increased by about 1 ns, and tCR (max.) spec should be increased by about 2 ns. Similarly if we use 176 pF for CAS, then tCC (min.) should decrease by 3 ns and tCC (max.) should decrease by about 7 ns.
8206/8206-2
ERROR DETECTION AND CORRECTION UNIT

- Detects and Corrects All Single Bit Errors
- Detects All Double Bit and Most Multiple Bit Errors
- 52 ns Maximum for Detection; 67 ns Maximum for Correction (16 Bit System)
- Syndrome Outputs for Error Logging
- 8206-2 Timing Optimized for single 8206 8MHz iAPX 186, 188, 86, 88 and 8207-2 Systems
- Separate Input and Output Busses—No Timing Strobes Required
- Expandable to Handle 80 Bit Memories
- Supports Reads With and Without Correction, Writes, Partial (Byte) Writes, and Read-Modify-Writes
- HMOS Technology for Low Power
- 68 Pin Leadless JEDEC Package
- Single +5V Supply

The HMOS 8206 Error Detection and Correction Unit is a high-speed device that provides error detection and correction for memory systems (static and dynamic) requiring high reliability and performance. Each 8206 handles 8 or 16 data bits and up to 8 check bits. 8206's can be cascaded to provide correction and detection for up to 80 bits of data. Other 8206 features include the ability to handle byte writes, memory initialization, and error logging.

Figure 1. 8206 Block Diagram
Figure 2. 8206–2 Block Diagram

Table 1. 8206 Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data In</td>
<td>1, 68-61, 59-53</td>
<td>I</td>
<td>Data In: These inputs accept a 16 bit data word from RAM for error detection and/or correction.</td>
</tr>
<tr>
<td>Check Bits In/Syndrome In</td>
<td>5</td>
<td>I</td>
<td>In a single 8206 system, or in the master in a multi-8206 system, these inputs accept the check bits (5 to 8) from the RAM. In a single 8206 16 bit system, CBl0-5 are used. In slave 8206's these inputs accept the syndrome from the master.</td>
</tr>
<tr>
<td>Data Out/Write Data In</td>
<td>51</td>
<td>I/O</td>
<td>Data Out/Write Data In: In a read cycle, data accepted by Dlo-15 appears at these outputs corrected if CRCT is low, or uncorrected if CRCT is high. The BM inputs must be high to enable the output buffers during the read cycle. In a write cycle, data to be written into the RAM is accepted by these inputs for computing the write check bits. In a partial-write cycle, the byte not to be modified appears at either DOO-7 if BM0 is high, or DOO-15 if BM1 is high, for writing to the RAM. When WZ is active, it causes the 8206 to output all zeros at D0-15, with the proper write check bits on CBO.</td>
</tr>
</tbody>
</table>
Table 1. 8206 Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYO/CBO/PPO0</td>
<td>23</td>
<td>O</td>
<td>Syndrome Out/Check Bits Out/Partial Parity Out: In a single 8206 system, or in the master in a multi-8206 system, the syndrome appears at these outputs during a read. During a write, the write check bits appear. In slave 8206's the partial parity bits used by the master appear at these outputs. The syndrome is latched (during read-modify-writes) by R/W going low.</td>
</tr>
<tr>
<td>SYO/CBO/PPO1</td>
<td>24</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO/PPO2</td>
<td>25</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO/PPO3</td>
<td>26</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO/PPO4</td>
<td>27</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO/PPO5</td>
<td>28</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO/PPO6</td>
<td>29</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO/PPO7</td>
<td>30</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO/PPO8</td>
<td>31</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>PPI0/POS0</td>
<td>13</td>
<td>I</td>
<td>Partial Parity In/Position: In the master in a multi-8206 system, these inputs accept partial parity bits 0 and 1 from the slaves. In a slave 8206 these inputs inform it of its position within the system (1 to 4). Not used in a single 8206 system.</td>
</tr>
<tr>
<td>PPI1/POS1</td>
<td>14</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>PPI2/NSL0</td>
<td>15</td>
<td>I</td>
<td>Partial Parity In/Number of Slaves: In the master in a multi-8206 system, these inputs accept partial parity bits 2 and 3 from the slaves. In a multi-8206 system these inputs are used in slave number 1 to tell it the total number of slaves in the system (1 to 4). Not used in other slaves or in a single 8206 system.</td>
</tr>
<tr>
<td>PPI3/NSL1</td>
<td>16</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>PPI4/CE</td>
<td>17</td>
<td>I/O</td>
<td>Partial Parity In/Correctable Error: In the master in a multi-8206 system this pin accepts partial parity bit 4. In slave number 1 only, or in a single 8206 system, this pin outputs the correctable error flag. CE is latched by R/W going low. Not used in other slaves.</td>
</tr>
<tr>
<td>PPI5</td>
<td>18</td>
<td>I</td>
<td>Partial Parity In: In the master in a multi-8206 system these pins accept partial parity bits 5 to 7. The number of partial parity bits equals the number of check bits. Not used in single 8206 systems or in slaves.</td>
</tr>
<tr>
<td>PPI6</td>
<td>19</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>PPI7</td>
<td>20</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>ERROR</td>
<td>22</td>
<td>O</td>
<td>Error: This pin outputs the error flag in a single 8206 system or in the master of a multi-8206 system. It is latched by R/W going low. Not used in slaves.</td>
</tr>
<tr>
<td>CRCT</td>
<td>52</td>
<td>I</td>
<td>Correct: When low this pin causes data correction during a read or read-modify-write cycle. When high, it causes error correction to be disabled, although error checking is still enabled.</td>
</tr>
<tr>
<td>STB</td>
<td>2</td>
<td>I</td>
<td>Strobe: STB is an input control used to strobe data at the DI inputs and check-bits at the CBI/SYI inputs. The signal is active high to admit the inputs. The signals are latched by the high-to-low transition of STB.</td>
</tr>
<tr>
<td>BM0</td>
<td>33</td>
<td>I</td>
<td>Byte Marks: When high, the Data Out pins are enabled for a read cycle. When low, the Data Out buffers are fristated for a write cycle. BM0 controls DO0-7. In partial (byte) writes, the byte mark input is low for the new byte to be written.</td>
</tr>
<tr>
<td>BM1</td>
<td>32</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>R/W</td>
<td>21</td>
<td>I</td>
<td>Read/Write: When high this pin causes the 8206 to perform detection and correction (if CRCT is low). When low, it causes the 8206 to generate check bits. On the high-to-low transition the syndrome is latched internally for read-modify-write cycles.</td>
</tr>
<tr>
<td>WZ</td>
<td>34</td>
<td>I</td>
<td>Write Zero: When low this input overrides the BM0-1 and R/W inputs to cause the 8206 to output all zeros at DO0-15 with the corresponding check bits at CBO0-7. Used for memory initialization.</td>
</tr>
<tr>
<td>M/S</td>
<td>4</td>
<td>I</td>
<td>Master/Slave: Input tells the 8206 whether it is a master (high) or a slave (low).</td>
</tr>
<tr>
<td>SEDCU</td>
<td>3</td>
<td>I</td>
<td>Single EDC Unit: Input tells the master whether it is operating as a single 8206 (low) or as the master in a multi-8206 system (high). Not used in slaves.</td>
</tr>
<tr>
<td>VCC</td>
<td>60</td>
<td>I</td>
<td>Power Supply: +5V</td>
</tr>
<tr>
<td>VSS</td>
<td>26</td>
<td>I</td>
<td>Logic Ground</td>
</tr>
<tr>
<td>VSS</td>
<td>43</td>
<td>I</td>
<td>Output Driver Ground</td>
</tr>
</tbody>
</table>
Table 2. 8206-2 Pin Description Differences over the 8206.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB10-5</td>
<td>5-10</td>
<td>I</td>
<td>Check Bits In: In an 8206-2 system, these inputs accept the check bits (5 to 6) from the RAM</td>
</tr>
<tr>
<td>SYO/CBO0</td>
<td>23</td>
<td>O</td>
<td>Syndrome Out/Check Bits Out: In an 8206-2 system, the syndrome appears at these outputs during a read. During a write, the write check bits appear. The syndrome is latched (during read-modify-writes) by R/W going low.</td>
</tr>
<tr>
<td>SYO/CBO1</td>
<td>24</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO2</td>
<td>25</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO3</td>
<td>27</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO4</td>
<td>28</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>SYO/CBO5</td>
<td>29</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td>17</td>
<td>O</td>
<td>Correctable Error: In an 8206-2 system, this pin outputs the correctable error flag. CE is latched by R/W going low.</td>
</tr>
<tr>
<td>WZ</td>
<td>34</td>
<td>I</td>
<td>Write Zero: When low this input overrides the BM0-1 and R/W inputs to cause the 8206-2 to output all zeros at DO0-16 with the corresponding check bits at CBO0-5. Used for memory initialization.</td>
</tr>
<tr>
<td>Strap High</td>
<td>4</td>
<td>I</td>
<td>Must be tied High.</td>
</tr>
<tr>
<td>Strap Low</td>
<td>3</td>
<td>I</td>
<td>Must be tied Low.</td>
</tr>
<tr>
<td>N.C.</td>
<td>11-16</td>
<td>I</td>
<td>Note: These pins have internal pull-up resistors but if possible should be tied high or low.</td>
</tr>
<tr>
<td></td>
<td>18-20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N.C.</td>
<td>30, 31</td>
<td>O</td>
<td>Note: These are no connect pins and should be left open.</td>
</tr>
</tbody>
</table>

FUNCTIONAL DESCRIPTION

The 8206 Error Detection and Correction Unit provides greater memory system reliability through its ability to detect and correct memory errors. It is a single chip device that can detect and correct all single bit errors and detect all double bit and some multiple bit errors. Some other odd multiple bit errors (e.g., 5 bits in error) are interpreted as single bit errors, and the CE flag is raised. While some even multiple bit errors (e.g., 4 bits in error) are interpreted as no error, most are detected as double bit errors. This error handling is a function of the number of check bits used by the 8206 (see Figure 2) and the specific Hamming code used. Errors in check bits are not distinguished from errors in a word.

For more information on error correction codes, see Intel Application Notes AP-46 and AP-73.

A single 8206 or 8206-2 handles 8 or 16 bits of data, and up to 5 8206's can be cascaded in order to handle data paths of 80 bits. For a single 8206 8 bit system, the D18-15, DO/WD18-15 and BM1 inputs are grounded. See the Multi-Chip systems section for information on 24-80 bit systems.

The 8206 has a “flow through” architecture. It supports two kinds of error correction architecture: 1) Flow-through, or correct-always; and 2) Parallel, or check-only. There are two separate 16-pin busses, one to accept data from the RAM (DI) and the other to deliver corrected data to the system bus (DO/WD). The logic is entirely combinatorial during a read cycle. This is in contrast to an architecture with only one bus, with bidirectional bus drivers that must first read the data and then be turned around to output the corrected data. The latter architecture typically requires additional hardware (latches and/or transceivers) and may be slower in a system due to timing skews of control signals.

<table>
<thead>
<tr>
<th>DATA WORD BITS</th>
<th>CHECK BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>32</td>
<td>7</td>
</tr>
<tr>
<td>40</td>
<td>7</td>
</tr>
<tr>
<td>48</td>
<td>8</td>
</tr>
<tr>
<td>56</td>
<td>8</td>
</tr>
<tr>
<td>64</td>
<td>8</td>
</tr>
<tr>
<td>72</td>
<td>8</td>
</tr>
<tr>
<td>80</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 3. Number of Check Bits Used by 8206
READ CYCLE

With the R/W pin high, data is received from the RAM outputs into the DI pins where it is optionally latched by the STB signal. Check bits are generated from the data bits and compared to the check bits read from the RAM into the CBI pins. If an error is detected the ERROR flag is activated and the correctable error flag (CE) is used to inform the system whether the error was correctable or not. With the BM inputs high, the word appears corrected at the DO pins if the error was correctable, or unmodified if the error was uncorrectable.

If more than one 8206 is being used, then the check bits are read by the master. The slaves generate a partial parity output (PPO) and pass it to the master. The master 8206 then generates and returns the syndrome to the slaves (SYO) for correction of the data.

The 8206 may alternatively be used in a “check-only” mode with the CRCT pin left high. With the correction facility turned off, the propagation delay from memory outputs to 8206 outputs is significantly shortened. In this mode the 8206 issues an ERROR flag to the CPU, which can then perform one of several options: lengthen the current cycle for correction, restart the instruction, perform a diagnostic routine, etc.

A syndrome word, five to eight bits in length and containing all necessary information about the existence and location of an error, is made available to the system at the SYO7-0 pins. Error logging may be accomplished by latching the syndrome and the memory address of the word in error.

WRITE CYCLE

For a full write, in which an entire word is written to memory, the data is written directly to the RAM, bypassing the 8206. The same data enters the 8206 through the WDI pins where check bits are generated. The Byte Mark inputs must be low to tristate the DO drivers. The check bits, 5 to 8 in number, are then written to the RAM through the CBO pins for storage along with the data word. In a multi-chip system, the master writes the check bits using partial parity information from the slaves.

In a partial write, part of the data word is overwritten, and part is retained in memory. This is accomplished by performing a read-modify-write cycle. The complete old word is read into the 8206 and corrected, with the syndrome internally latched by R/W going low. Only that part of the word not to be modified is output onto the DO pins, as controlled by the Byte Mark inputs. That portion of the word to be overwritten is supplied by the system bus. The 8206 then calculates check bits for the new word, using the byte from the previous read and the new byte from the system bus, and writes them to the memory.

READ-MODIFY-WRITE CYCLES

Upon detection of an error the 8206 may be used to correct the bit in error in memory. This reduces the probability of getting multiple-bit errors in subsequent read cycles. This correction is handled by executing read-modify-write cycles.

The read-modify-write cycle is controlled by the R/W input. After (during) the read cycle, the system dynamic RAM controller or CPU examines the 8206 ERROR and CE outputs to determine if a correctable error occurred. If it did, the dynamic RAM controller or CPU forces R/W low, telling the 8206 to latch the generated syndrome and drive the corrected check bits onto the CBO outputs. The corrected data is available on the DO pins. The DRAM controller then writes the corrected data and corresponding check bits into memory.

The 8206 may be used to perform read-modify-writes in one or two RAM cycles. If it is done in two cycles, the 8206 latches are used to hold the data and check bits from the read cycle to be used in the following write cycle. The Intel 8207 Advanced Dynamic RAM controller allows read-modify-write cycles in one memory cycle. See the System Environment section.

INITIALIZATION

A memory system operating with ECC requires some form of initialization at system power-up in order to set valid data and check bit information in memory. The 8206 supports memory initialization by the write zero function. By activating the WZ pin, the 8206 will write a data pattern of zeros and the associated check bits in the current write cycle. By thus writing to all memory at power-up, a controller can set memory to valid data and check bits. Massive memory failure, as signified by both data and check bits all ones or zeros, will be detected as an uncorrectable error.
MULTI-CHIP SYSTEMS

A single 8206 handles 8 or 16 bits of data and 5 or 6 check bits, respectively. Up to 5 8206's can be cascaded for 80 bit memories with 8 check bits.

When cascaded, one 8206 operates as a master, and all others as slaves. As an example, during a read cycle in a 32 bit system with one master and one slave, the slave calculates parity on its portion of the word—"partial parity"—and presents it to the master through the PPO pins. The master combines the partial parity from the slave with the parity it calculated from its own portion of the word to generate the syndrome. The syndrome is then returned by the master to the slave for error correction. In systems with more than one slave the above description continues to apply, except that the partial parity outputs of the slaves must be XOR'd externally. Figure 4 shows the necessary external logic for multi-chip systems. Write and read-modify-write cycles are carried out analogously. See the System Operation section for multi-chip wiring diagrams.

There are several pins used to define whether the 8206 will operate as a master or a slave. Tables 3 and 4 illustrate how these pins are tied.

Figure 4. External Logic For Multi-Chip Systems
Table 3. Master/Slave Pin Assignments

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Name</th>
<th>Master</th>
<th>Slave 1</th>
<th>Slave 2</th>
<th>Slave 3</th>
<th>Slave 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>M/S</td>
<td>+5V</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
</tr>
<tr>
<td>3</td>
<td>SEDCU</td>
<td>+5V</td>
<td>+5V</td>
<td>+5V</td>
<td>+5V</td>
<td>+5V</td>
</tr>
<tr>
<td>13</td>
<td>PPl0/POS0</td>
<td>PPI</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
</tr>
<tr>
<td>14</td>
<td>PPl1/POS1</td>
<td>PPI</td>
<td>Gnd</td>
<td>+5V</td>
<td>+5V</td>
<td>+5V</td>
</tr>
<tr>
<td>15</td>
<td>PPl2/NSL0</td>
<td>PPI</td>
<td>+5V</td>
<td>+5V</td>
<td>+5V</td>
<td>+5V</td>
</tr>
<tr>
<td>16</td>
<td>PPl3/NSL1</td>
<td>PPI</td>
<td>+5V</td>
<td>+5V</td>
<td>+5V</td>
<td>+5V</td>
</tr>
</tbody>
</table>

*See Table 3.

NOTE: Pins 13, 14, 15, 16 have internal pull-up resistors and may be left as N.C. where specified as connecting to +5V.

Table 4. NSL Pin Assignments for Slave 1

<table>
<thead>
<tr>
<th>Pin</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPl2/NSL0</td>
<td>Gnd</td>
<td>+5V</td>
<td>Gnd</td>
<td>+5V</td>
</tr>
<tr>
<td>PPl3/NSL1</td>
<td>Gnd</td>
<td>Gnd</td>
<td>+5V</td>
<td>+5V</td>
</tr>
</tbody>
</table>

The timing specifications for multi-chip systems must be calculated to take account of the external XOR gating in 3, 4, and 5-chip systems. Let tXOR be the delay for a single external TTL XOR gate. Then the following equations show how to calculate the relevant timing parameters for 2-chip (n=0), 3-chip (n=1), 4-chip (n=2), and 5-chip (n=2) systems:

Data-in to corrected data-out (read cycle) = TDVSV + TPVSV + TSVQV + ntXOR

Data-in to error flag (read cycle) = TDVSV + TPVEV + ntXOR

Data-in to correctable error flag (read cycle) = TDVSV + TPVSV + TSVCV + ntXOR

Write data to check-bits valid (full write cycle) = TVQVQ + TPVSV + ntXOR

Data-in to check-bits valid (read-mod-write cycle) = TDVSV + TPVSV + TVQVQ + TVQVQ + TPVSV + 2ntXOR

Data-in to check-bits valid (non-correcting read-modify-write cycle) = TDVQU + TVQVQ + TPVSV + ntXOR

HAMMING CODE

The 8206 uses a modified Hamming code which was optimized for multi-chip EDCU systems. The code is such that partial parity is computed by all 8206's in parallel. No 8206 requires more time for propagation through logic levels than any other one, and hence no one device becomes a bottleneck in the parity operation. However, one or two levels of external TTL XOR gates are required in systems with three to five chips. The code appears in Table 5. The check bits are derived from the table by XORing or XNOR-ing together the bits indicated by 'X's in each row corresponding to a check bit. For example, check bit 0 in the MASTER for data word 100110111010111 will be "0." It should be noted that the 8206 will detect the gross-error condition of all lows or all highs.

Error correction is accomplished by identifying the bad bit and inverting it. Table 5 can also be used as an error syndrome table by replacing the 'X's with '1's. Each column then represents a different syndrome word, and by locating the column corresponding to a particular syndrome the bit to be corrected may be identified. If the syndrome cannot be located then the error cannot be corrected. For example, if the syndrome word is 00110111, the bit to be corrected is bit 5 in the slave one data word (bit 21).

The syndrome decoding is also summarized in Tables 6 and 7 which can be used for error logging. By finding the appropriate syndrome word (starting with bit zero, the least significant bit), the result is either: 1) no error; 2) an identified (correctable) single bit error; 3) a double bit error; or 4) a multi-bit uncorrectable error.
Table 5. Modified Hamming Code Check Bit Generation

Check bits are generated by XOR'ing (except for the CB0 and CB1 data bits, which are XNOR'ed in the Master) the data bits in the rows corresponding to the check bits. Note there are 6 check bits in a 16-bit system, 7 in a 32-bit system, and 8 in 48-or-more-bit systems.

<table>
<thead>
<tr>
<th>BYTE NUMBER</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT NUMBER</td>
<td>0 1 2 3 4 5 6 7</td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>CHECK</td>
<td>CB0 = x x - x - x x - x - x - x - x - x - XNOR</td>
<td>CB1 = x - x - x - x - - x - x - x - x - x - XNOR</td>
</tr>
<tr>
<td></td>
<td>CB2 = - x - x - x x - - x - x - x - x - x - XOR</td>
<td>CB3 = x x x x x - - x - x - x - x - x - XOR</td>
</tr>
<tr>
<td></td>
<td>CB4 = - - x x x - - x - x - x - x - x - XNOR</td>
<td>CB5 = - - x x x x x - - x - x - x - x - XOR</td>
</tr>
<tr>
<td></td>
<td>CB6 = - - - x x x - - x - x - x - x - x - XOR</td>
<td>CB7 = - - - - x x x x x - - - - - x - x - XOR</td>
</tr>
<tr>
<td>DATA BITS</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7</td>
<td>8 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td></td>
<td>1 6 7 8 9 0 1 2 3</td>
<td>4 5 6 7 8 9 0 1</td>
</tr>
</tbody>
</table>

16 BIT OR MASTER

| SLAVE #1 |
|---|---|---|
| BYTE NUMBER | 2 | 3 |
| BIT NUMBER  | 0 1 2 3 4 5 6 7 | 0 1 2 3 4 5 6 7 |
| OPERATIONS  | - x x x x x - - x - x - x - x - x - XOR | x x x x x x x x x - - x - x - x - XOR |
| CHECK       | CB2 = - x x x x x - - x - x - x - x - x - XOR | CB3 = x x x x x x x x x - - x - x - x - XOR |
|             | CB4 = - - x x x - - x - x - x - x - x - XOR | CB5 = - - - x x x x x - - x - x - x - XOR |
|             | CB6 = - - - - x x x x x x - - - - - x - XOR | CB7 = - - - - - - - - - - x x x x x x x x XOR |
| DATA BITS   | 0 0 0 0 0 0 0 0 | 0 1 1 1 2 2 2 2 2 |
|             | 2 2 2 2 2 2 3 3 | 4 5 6 7 8 9 0 1 |

SLAVE #2

| SLAVE #3 |
|---|---|---|
| BYTE NUMBER | 4 | 5 |
| BIT NUMBER  | 0 1 2 3 4 5 6 7 | 0 1 2 3 4 5 6 7 |
| OPERATIONS  | x x - x - x x - x - x - x - x - XNOR | x x - x - x x - x - x - x - x - XOR |
| CHECK       | CB0 = x x - x - x x - x - x - x - x - XNOR | CB1 = x - x - x - x - - x - x - x - x - XOR |
|             | CB2 = - x - x - x x - - x - x - x - x - XOR | CB3 = x x x x x - - x - x - x - x - XOR |
|             | CB4 = - - x x x - - x - x - x - x - XOR | CB5 = - - x x x x x - - x - x - x - XOR |
|             | CB6 = - - - x x x - - x - x - x - x - XOR | CB7 = - - - - x x x x x - - - - - x - XOR |
| DATA BITS   | 3 3 3 3 3 3 3 3 | 3 3 3 3 3 3 3 3 |
|             | 4 4 4 4 4 4 4 4 4 | 4 4 4 4 5 5 5 5 5 |
|             | 5 5 5 5 6 6 6 6 6 | 6 6 6 6 6 6 6 7 7 |
|             | 7 7 7 7 7 7 7 7 7 | 7 7 7 7 7 7 7 7 7 |
|             | 8 9 0 1 2 3 4 5 6 7 | 8 9 0 1 2 3 4 5 6 7 |
|             | 4 5 6 7 8 9 0 1 2 3 | 4 5 6 7 8 9 0 1 2 3 |
|             | 6 7 8 9 0 1 2 3 4 5 | 6 7 8 9 0 1 2 3 4 5 |
|             | 7 8 9 0 1 2 3 4 5 6 | 7 8 9 0 1 2 3 4 5 6 |
|             | 8 9 0 1 2 3 4 5 6 7 | 8 9 0 1 2 3 4 5 6 7 |
|             | 9 0 1 2 3 4 5 6 7 8 | 9 0 1 2 3 4 5 6 7 8 |

SLAVE #4

SLAVE #2

SLAVE #3

SLAVE #4
Table 6. 8206 Syndrome Decoding

<table>
<thead>
<tr>
<th>Syndrome Bits</th>
<th>0 0 0 0 0</th>
<th>0 0 0 1 1</th>
<th>0 0 1 0 1</th>
<th>0 0 1 1 0</th>
<th>0 1 0 0 0</th>
<th>0 1 0 1 0</th>
<th>0 1 1 0 0</th>
<th>0 1 1 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>N CB0</td>
<td>CB1</td>
<td>CB2</td>
<td>CB3</td>
<td>D D</td>
<td>D D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>0 0 0 1 1</td>
<td>CB4</td>
<td>D D</td>
<td>5 D</td>
<td>6 D</td>
<td>7 D</td>
<td>8 D</td>
<td>9 D</td>
<td>10 D</td>
</tr>
<tr>
<td>0 0 1 0 1</td>
<td>CB5</td>
<td>D D</td>
<td>11 D</td>
<td>12 D</td>
<td>13 D</td>
<td>14 D</td>
<td>15 D</td>
<td>16 D</td>
</tr>
<tr>
<td>0 0 1 1 0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0 1 0 0 0</td>
<td>CB6</td>
<td>D D</td>
<td>25 D</td>
<td>26 D</td>
<td>27 D</td>
<td>28 D</td>
<td>29 D</td>
<td>30 D</td>
</tr>
<tr>
<td>0 1 0 1 0</td>
<td>D D</td>
<td>52 D</td>
<td>53 D</td>
<td>54 D</td>
<td>55 D</td>
<td>56 D</td>
<td>57 D</td>
<td>58 D</td>
</tr>
<tr>
<td>0 1 1 0 0</td>
<td>D D</td>
<td>29 D</td>
<td>31 D</td>
<td>32 D</td>
<td>33 D</td>
<td>34 D</td>
<td>35 D</td>
<td>36 D</td>
</tr>
<tr>
<td>0 1 1 1 0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>1 0 0 0 0</td>
<td>CB7</td>
<td>D D</td>
<td>43 D</td>
<td>44 D</td>
<td>45 D</td>
<td>46 D</td>
<td>47 D</td>
<td>48 D</td>
</tr>
<tr>
<td>1 0 0 1 1</td>
<td>D D</td>
<td>49 D</td>
<td>50 D</td>
<td>51 D</td>
<td>52 D</td>
<td>53 D</td>
<td>54 D</td>
<td>55 D</td>
</tr>
<tr>
<td>1 0 1 0 0</td>
<td>D D</td>
<td>56 D</td>
<td>57 D</td>
<td>58 D</td>
<td>59 D</td>
<td>60 D</td>
<td>61 D</td>
<td>62 D</td>
</tr>
<tr>
<td>1 1 0 0 0</td>
<td>D D</td>
<td>D D</td>
<td>D D</td>
<td>D D</td>
<td>D D</td>
<td>D D</td>
<td>D D</td>
<td>D D</td>
</tr>
<tr>
<td>1 1 0 1 0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>1 1 1 0 0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>1 1 1 1 0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

N = No Error
CBX = Error in Check Bit X
X = Error in Data Bit X
D = Double Bit Error
U = Uncorrectable Multi-Bit Error

SYSTEM ENVIRONMENT

The 8206 interface to a typical 32 bit memory system is illustrated in Figure 5. For larger systems, the partial parity bits from slaves two to four must be
XOR'ed externally, which calls for one level of XOR gating for three 8206's and two levels for four or five 8206's.

The 8206 is designed for direct connection to the Intel 8207 Advanced Dynamic RAM Controller. The 8207 has the ability to perform dual port memory control, and Figure 6 illustrates a highly integrated dual port RAM implementation using the 8206 and 8207. The 8206/8207 combination permits such features as automatic scrubbing (correcting errors in memory during refresh), extending RAS and CAS timings for Read-Modify-Writes in single memory cycles, and automatic memory initialization upon reset. Together these two chips provide a complete dual-port, error-corrected dynamic RAM subsystem.

Figure 6. Dual Port RAM Subsystem with 8206/8207 (32-bit bus)
The 8206-2 handles 8 or 16 bits of data. For 8 bit 8206-2 systems, the D1b-15, DO/WD1b-15 and BM1 inputs are grounded.

The 8206-2 is designed for direct connection to the Intel 8207-2 Advanced Dynamic RAM Controller. The 8207-2 has the ability to perform dual port memory control, and Figure 7 illustrates a highly integrated iAPX 186 RAM implementation using the 8206-2 and 8207-2. The 8206-2/8207-2 combination permits such features as automatic scrubbing (correcting errors in memory during refresh), extending RAS and CAS timings for Read-Modify-Writes in single memory cycles, and automatic memory initialization upon reset. Together these two chips provide a complete dual-port, error-corrected dynamic RAM subsystems.

**Table 7. 8206-2 Syndrome Decoding**

<table>
<thead>
<tr>
<th>Syndrome Bits</th>
<th>5 4 3 2 0 0 0 0 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>N CB0 CB1 D CB2 D D D</td>
</tr>
<tr>
<td>0 0 1</td>
<td>CB3 D D 0 0 1 2 D</td>
</tr>
<tr>
<td>0 1 0</td>
<td>CB4 D D 5 D 6 7 D</td>
</tr>
<tr>
<td>0 1 1</td>
<td>D 3 D D 4 D D D</td>
</tr>
<tr>
<td>1 0 0</td>
<td>CB5 D D 11 D D 12 D</td>
</tr>
<tr>
<td>1 0 1</td>
<td>D 8 9 D 10 D D D</td>
</tr>
<tr>
<td>1 1 0</td>
<td>D 13 14 D 15 D D D</td>
</tr>
<tr>
<td>1 1 1</td>
<td>D D D D D D D D D</td>
</tr>
</tbody>
</table>

N = No Error
CBX = Error in Check Bit X
X = Error in Data Bit X
D = Double Bit Error

**Figure 7. iAPX 186 RAM Correct Always Subsystem with the 8206-2 and the 8207-2**
MEMORY BOARD TESTING

The 8206 lends itself to straightforward memory board testing with a minimum of hardware overhead. The following is a description of four common test modes and their implementation.

Mode 0—Read and write with error correction.
Implementation: This mode is the normal 8206 operating mode.

Mode 1—Read and write data with error correction disabled to allow test of data memory.
Implementation: This mode is performed with CRCT deactivated.

Mode 2—Read and write check bits with error correction disabled to allow test of check bits memory.
Implementation: Any pattern may be written into the check bits memory by judiciously choosing the proper data word to generate the desired check bits, through the use of the 8206 Hamming code. To read out the check bits it is first necessary to fill the data memory with all zeros, which may be done by activating WZ and incrementing memory addresses with WE to the check bits memory held inactive, and then performing ordinary reads. The check bits will then appear directly at the SYO outputs, with bits CB0 and CB1 inverted.

Mode 3—Write data, without altering or writing check bits, to allow the storage of bit combinations to cause error correction and detection.
Implementation: This mode is implemented by writing the desired word to memory with WE to the check bits array held inactive.
NOTE:
The 8206 and 8206-2 is packaged in a 88 pin JEDEC TYPE A hermetic chip carrier

Figure 8. 8206 and 8206-2 Pinout Diagram
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .................. 0°C to 70°C
Storage Temperature .................... -65°C to +150°C
Voltage On Any Pin With Respect to Ground .......... -0.5V to +7V
Power Dissipation ......................... 1.5 Watts

*NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (Ta = 0°C to 70°C, VCC = 5.0V ± 10%, VSS = GND)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icc</td>
<td>Power Supply Current</td>
<td>-Single 8206, 8206-2 or Slave #1</td>
<td>270</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Master in Multi-Chip or Slaves #2, 3, 4</td>
<td>230</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>VIL1</td>
<td>Input Low Voltage</td>
<td></td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
</tr>
<tr>
<td>VIH1</td>
<td>Input High Voltage</td>
<td></td>
<td>2.0</td>
<td>VCC + 0.5V</td>
<td>V</td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>-DO</td>
<td>0.45</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-All Others</td>
<td>0.45</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>-DO, CBO</td>
<td>2.6</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-All Other Outputs</td>
<td>2.4</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>ILO</td>
<td>I/O Leakage Current</td>
<td>-PPI4/CE</td>
<td>± 20</td>
<td>μA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-DO/WDI-15</td>
<td>± 10</td>
<td>μA</td>
<td></td>
</tr>
<tr>
<td>ILI</td>
<td>Input Leakage Current</td>
<td>-PPI0-7, 5-7, CB16-7, SEDCU2</td>
<td>± 20</td>
<td>μA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-All Other Input Only Pins</td>
<td>± 10</td>
<td>μA</td>
<td></td>
</tr>
</tbody>
</table>

NOTES:
1. SECDU (pin 3) and MIS (pin 4) are device strapping options and should be tied to VCC or GND. Vih min = VCC - 0.5V and Vil max = 0.5V.
2. PPI0-7 (pins 13-20) and CB16-7 (pins 11, 12) have internal pull-up resistors and if left unconnected will be pulled to VCC.

A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING LOAD CIRCUIT

C0 INCLUDES JIG CAPACITANCE
### A.C. CHARACTERISTICS

(T<sub>A</sub> = 0°C to 70°C, V<sub>CC</sub> = +5V ± 10%, V<sub>SS</sub> = 0V, R<sub>L</sub> = 22Ω, C<sub>L</sub> = 50 pF; all times are in nsec.)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8206</th>
<th>8206-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
</tr>
<tr>
<td>TRHEV</td>
<td>ERROR Valid from R/W↑</td>
<td>25</td>
<td>40</td>
</tr>
<tr>
<td>TRHCVC</td>
<td>CE Valid from R/W↑ (Single 8206)</td>
<td>44</td>
<td>49</td>
</tr>
<tr>
<td>TRHQV</td>
<td>Corrected Data Valid from R/W↑</td>
<td>54</td>
<td>66</td>
</tr>
<tr>
<td>TRVSV</td>
<td>SYO/CBO/PPO Valid from R/W</td>
<td>42</td>
<td>46</td>
</tr>
<tr>
<td>TDVEV</td>
<td>ERROR Valid from Data/Check Bits In</td>
<td>52</td>
<td>57</td>
</tr>
<tr>
<td>TDVCV</td>
<td>CE Valid from Data/Check Bits In</td>
<td>70</td>
<td>76</td>
</tr>
<tr>
<td>TDVQV</td>
<td>Corrected Data Valid from Data/Check Bits In</td>
<td>67</td>
<td>74</td>
</tr>
<tr>
<td>TDVSV</td>
<td>SYO/PPO Valid from Data/Check Bits In</td>
<td>55</td>
<td>65</td>
</tr>
<tr>
<td>TBHQV.</td>
<td>Corrected Data Access Time</td>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>TDIXQX</td>
<td>Hold Time from Data/Check Bits In</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TBLQZ</td>
<td>Corrected Data Float Delay</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>TSHIV</td>
<td>STB High to Data/Check Bits In Valid</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>TIVSL</td>
<td>Data/Check Bits In to STB↓ Set-up</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>TSLIX</td>
<td>Data/Check Bits In from STB↓ Hold</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>TPVEV</td>
<td>ERROR Valid from Partial Parity In</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>TPQVSV</td>
<td>Corrected Data (Master) from Partial Parity In</td>
<td>61</td>
<td>1,3</td>
</tr>
<tr>
<td>TPVSV</td>
<td>Syndrome/Check Bits Out from Partial Parity In</td>
<td>43</td>
<td>1,3</td>
</tr>
<tr>
<td>TSVQV</td>
<td>Corrected Data (Slave) Valid from Syndrome</td>
<td>51</td>
<td>3</td>
</tr>
<tr>
<td>TSCVQ</td>
<td>CE Valid from Syndrome (Slave number 1)</td>
<td>48</td>
<td>3</td>
</tr>
<tr>
<td>TTVQV</td>
<td>Check Bits/Partial Parity Out from Write Data In</td>
<td>64</td>
<td>69</td>
</tr>
<tr>
<td>TRHSX</td>
<td>Check Bits/Partial Parity Out from R/W, WZ Hold</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TRLSX</td>
<td>Syndrome Out from R/W Hold</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TQXQX</td>
<td>Hold Time from Write Data In</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TSVRL</td>
<td>Syndrome Out to R/W↓, Set-up</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>TVQXL</td>
<td>Data/Check Bits In to R/W↓ Set-up</td>
<td>39</td>
<td>41</td>
</tr>
<tr>
<td>TVQQV</td>
<td>Uncorrected Data Out from Data In</td>
<td>32</td>
<td>38</td>
</tr>
<tr>
<td>TWQVL</td>
<td>Corrected Data Out from CRCT↓</td>
<td>30</td>
<td>33</td>
</tr>
<tr>
<td>LWLQL</td>
<td>WZ↑ to Zero Out</td>
<td>30</td>
<td>34</td>
</tr>
<tr>
<td>TWHQX</td>
<td>Zero Out from WZ↑ Hold</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**NOTES:**

1. A.C. Test Levels for CBO and DO are 2.4V and 0.8V.
2. T<sub>SHIV</sub> is required to guarantee output delay timings: TDVEV, TDVCV, TDQVQ, TDSVQ. T<sub>SHIV</sub> + TIVSL guarantees a min STB pulse width of 35 ns (45 ns for the 8206-8).
3. Not required for 8/16 bit systems.
WAVEFORMS

READ

STB

TSHDV

TIVSL

TSILX

R/W

SM

TBHQV

DI

CBI

VALID

TRHQV

DO

VALID

TRYSV

SYO

VALID

TDVSY

ERROR

TRHEV

VALID

TDVEV

CE

VALID

TDVCV

TRHCV
WAVEFORMS (Continued)

READ—MASTER/SLAVE

- STB
- R/W
- B/B
- DI/CB
- PPO (SLAVE)
- PPI (MASTER)
- DO (MASTER)
- SYO (MASTER)
- SYI (SLAVE)
- DO (SLAVE)
- ERROR
- CE

Valid

TSHIV
TIVSL
TSLIX
TBLOZ
TBHOV
TRSVV
TDVS
TPVQV
TPVSV
TPSVY
TRHEV
TRHCY
TSVGV
TSVCV

5-46
WAVEFORMS (Continued)

FULL WRITE

R/W

6M

DO/WDI

DATA OUT

WRITE DATA IN

SYO/CBO

SYN

FULL WRITE—MASTER/SLAVE

R/W

6M

DO/WDI

DATA OUT

WRITE DATA IN

PPO (SLAVE)

PPI (MASTER)

SYO/CBO

SYN

CB
WAVEFORMS (Continued)

READ MODIFY WRITE

- STB
- TSHIV
- TIVSL
- TSLIX
- R/W
- TDVRL
- TRSVY
- SM
- VALID
- TBHOV
- TRHDX
- TBLQZ
- DI
- CSI
- VALID
- TRHQV
- TDQOV
- TDXQX
- DO/WDI
- TRSVY
- TRLSX
- TQXQX
- SYO/CBO
- SYN
- CB
- TDVSY
- TQVGY
WAVEFORMS (Continued)

READ MODIFY WRITE—MASTER/SLAVE
WAVEFORMS (Continued)

NON-CORRECTING READ

WRITE ZERO
8207 DUAL-PORT DYNAMIC RAM CONTROLLER

- Provides All Signals Necessary to Control 16K (2118), 64K (2164A) and 256K Dynamic RAMs
- Directly Addresses and Drives up to 2 Megabytes without External Drivers
- Supports Single and Dual-Port Configurations
- Automatic RAM Initialization in All Modes
- Four Programmable Refresh Modes
- Transparent Memory Scrubbing in ECC Mode

The Intel 8207 Advanced Dynamic RAM Controller (ADRC) is a high-performance, systems-oriented, Dynamic RAM controller that is designed to easily interface 16K, 64K and 256K Dynamic RAMs to Intel and other microprocessor systems. A dual-port interface allows two different busses to independently access memory. When configured with an 8206 Error Detection and Correction Unit the 8207 supplies the necessary logic for designing large error-corrected memory arrays. This combination provides automatic memory initialization and transparent memory error scrubbing.

![Figure 1. 8207 Block Diagram](image-url)
Table 1. Pin description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEN</td>
<td>1</td>
<td>0</td>
<td>ADDRESS LATCH ENABLE: In two-port configurations, when Port A is running with iAPX 286 Status interface mode, this output replaces the ALE signal from the system bus controller of port A and generates an address latch enable signal which provides optimum setup and hold timing for the 8207. This signal is used in Fast Cycle operation only.</td>
</tr>
<tr>
<td>XACKA/</td>
<td>2</td>
<td>0</td>
<td>TRANSFER ACKNOWLEDGE PORTA/ACKNOWLEDGE PORTA: In non-ECC mode, this pin is XACKA and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle for Port A. XACKA is a Multibus-compatible signal. In ECC mode, this pin is ACKA which can be configured, depending on the programming of the X program bit, as an XACK or AACK strobe. The SA programming bit determines whether the AACK will be an early EAACKA or a late LAACKA interface signal.</td>
</tr>
<tr>
<td>ACKA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XACKB/</td>
<td>3</td>
<td>0</td>
<td>TRANSFER ACKNOWLEDGE PORT B/ACKNOWLEDGE PORT B: In non-ECC mode, this pin is XACKB and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle for Port B. XACKB is a Multibus-compatible signal. In ECC mode, this pin is ACKB which can be configured, depending on the programming of the X program bit, as an XACK or AACK strobe. The SB programming bit determines whether the AACK will be an early EAACKB or a late LAACKB interface signal.</td>
</tr>
<tr>
<td>ACKB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AACKA/WZ</td>
<td>4</td>
<td>0</td>
<td>ADVANCED ACKNOWLEDGE PORT A/WRITE ZERO: In non-ECC mode, this pin is AACKA and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the SA program bit for synchronous or asynchronous operation. In ECC mode, after a RESET, this signal will cause the 8206 to force the data to all zeros and generate the appropriate check bits.</td>
</tr>
<tr>
<td>XACKB/RW</td>
<td>5</td>
<td>0</td>
<td>ADVANCED ACKNOWLEDGE PORT B/READ/WRITE: In non-ECC mode, this pin is AACKB and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the SB program bit for synchronous or asynchronous operation. In ECC mode, this signal causes the 8206 EDCU to latch the syndrome and error flags and generate check bits.</td>
</tr>
<tr>
<td>DBM</td>
<td>6</td>
<td>0</td>
<td>DISABLE BYTE MARKS: This is an ECC control output signal indicating that a read or refresh cycle is occurring. This output forces the byte address decoding logic to enable all 8206 data output buffers. In ECC mode, this output is also asserted during memory initialization and the 8-cycle dynamic RAM wake-up exercise. In non-ECC systems this signal indicates that either a read, refresh or 8-cycle warm-up is in progress.</td>
</tr>
<tr>
<td>ESTB</td>
<td>7</td>
<td>0</td>
<td>ERROR STROBE: In ECC mode, this strobe is activated when an error is detected and allows a negative-edge triggered flip-flop to latch the status of the 8206 EDCU CE for systems with error logging capabilities. ESTB will not be issued during refresh cycles.</td>
</tr>
<tr>
<td>LOCK</td>
<td>8</td>
<td>1</td>
<td>LOCK: This input instructs the 8207 to lock out the port not being serviced at the time LOCK was issued.</td>
</tr>
<tr>
<td>Vcc</td>
<td>9</td>
<td>1</td>
<td>DRIVER POWER: +6 Volts. Supplies Vcc for the output drivers.</td>
</tr>
<tr>
<td></td>
<td>43</td>
<td></td>
<td>LOGIC POWER: +5 Volts. Supplies Vcc for the internal logic circuits.</td>
</tr>
<tr>
<td>CE</td>
<td>10</td>
<td>1</td>
<td>CORRECTABLE ERROR: This is an ECC input from the 8206 EDCU which instructs the 8207 whether a detected error is correctable or not. A high input indicates a correctable error. A low input inhibits the 8207 from activating WE to write the data back into RAM. This should be connected to the CE output of the 8206.</td>
</tr>
<tr>
<td>ERROR</td>
<td>11</td>
<td>1</td>
<td>ERROR: This is an ECC input from the 8206 EDCU and instructs the 8207 that an error was detected. This pin should be connected to the ERROR output of the 8206.</td>
</tr>
<tr>
<td>MUX/PCLK</td>
<td>12</td>
<td>O</td>
<td>MULTIPLEXER CONTROL/PROGRAMMING CLOCK: Immediately after a RESET this pin is used to clock serial programming data into the PDI pin. In normal two-port operation, this pin is used to select memory addresses from the appropriate port. When this signal is high, port A is selected and when it is low, port B is selected. This signal may change state before the completion of a RAM cycle, but the RAM address hold time is satisfied.</td>
</tr>
<tr>
<td>PSEL</td>
<td>13</td>
<td>O</td>
<td>PORT SELECT: This signal is used to select the appropriate port for data transfer. When this signal is high port A is selected and when it is low port B is selected.</td>
</tr>
<tr>
<td>PSEN</td>
<td>14</td>
<td>O</td>
<td>PORT SELECT ENABLE: This signal used in conjunction with PSEL provides contention-free port exchange on the data bus. When PSEN is low, port selection is allowed to change state.</td>
</tr>
<tr>
<td>WE</td>
<td>15</td>
<td>O</td>
<td>WRITE ENABLE: This signal provides the dynamic RAM array the write enable input for a write operation.</td>
</tr>
</tbody>
</table>
Table 1. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>FWR</td>
<td>16</td>
<td>I</td>
<td>FULL WRITE: This is an ECC input signal that instructs the 8207, in an ECC configuration, whether the present write cycle is normal RAM write (full write) or a RAM partial write (read-modify-write) cycle.</td>
</tr>
<tr>
<td>RESET</td>
<td>17</td>
<td>I</td>
<td>RESET: This signal causes all internal counters and state flip-flops to be reset and upon release of RESET, data appearing at the PDI pin is clocked in by the PCLK output. The states of the PDI, PCTLA, PCTLB and RFRQ pins are sampled by RESET going inactive and are used to program the 8207. An 8-cycle dynamic RAM warm-up is performed after clocking PDI bits into the 8207.</td>
</tr>
<tr>
<td>CAS0</td>
<td>18</td>
<td>O</td>
<td>COLUMN ADDRESS STROBE: These outputs are used by the dynamic RAM array to latch the column address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>CAS1</td>
<td>19</td>
<td>O</td>
<td>COLUMN ADDRESS STROBE: These outputs are used by the dynamic RAM array to latch the column address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>CAS2</td>
<td>20</td>
<td>O</td>
<td>COLUMN ADDRESS STROBE: These outputs are used by the dynamic RAM array to latch the column address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>CAS3</td>
<td>21</td>
<td>O</td>
<td>COLUMN ADDRESS STROBE: These outputs are used by the dynamic RAM array to latch the column address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>RAS0</td>
<td>22</td>
<td>O</td>
<td>ROW ADDRESS STROBE: These outputs are used by the dynamic RAM array to latch the row address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>RAS1</td>
<td>23</td>
<td>O</td>
<td>ROW ADDRESS STROBE: These outputs are used by the dynamic RAM array to latch the row address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>RAS2</td>
<td>24</td>
<td>O</td>
<td>ROW ADDRESS STROBE: These outputs are used by the dynamic RAM array to latch the row address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>RAS3</td>
<td>25</td>
<td>O</td>
<td>ROW ADDRESS STROBE: These outputs are used by the dynamic RAM array to latch the row address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>Vss</td>
<td>26</td>
<td>I</td>
<td>DRIVER GROUND: Provides a ground for the output drivers.</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>I</td>
<td>LOGIC GROUND: Provides a ground for the remainder of the device.</td>
</tr>
<tr>
<td>AO0</td>
<td>35</td>
<td>O</td>
<td>ADDRESS OUTPUTS: These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>AO1</td>
<td>34</td>
<td>O</td>
<td>ADDRESS OUTPUTS: These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>AO2</td>
<td>33</td>
<td>O</td>
<td>ADDRESS OUTPUTS: These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>AO3</td>
<td>32</td>
<td>O</td>
<td>ADDRESS OUTPUTS: These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>AO4</td>
<td>31</td>
<td>O</td>
<td>ADDRESS OUTPUTS: These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>AO5</td>
<td>30</td>
<td>O</td>
<td>ADDRESS OUTPUTS: These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>AO6</td>
<td>29</td>
<td>O</td>
<td>ADDRESS OUTPUTS: These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>AO7</td>
<td>28</td>
<td>O</td>
<td>ADDRESS OUTPUTS: These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>AO8</td>
<td>27</td>
<td>O</td>
<td>ADDRESS OUTPUTS: These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>BS0</td>
<td>36</td>
<td>I</td>
<td>BANK SELECT: These inputs are used to select one of four banks of the dynamic RAM array as defined by the program bits RB0 and RB1.</td>
</tr>
<tr>
<td>BS1</td>
<td>37</td>
<td>I</td>
<td>BANK SELECT: These inputs are used to select one of four banks of the dynamic RAM array as defined by the program bits RB0 and RB1.</td>
</tr>
<tr>
<td>AL0</td>
<td>38</td>
<td>I</td>
<td>ADDRESS LOW: These lower-order address inputs are used to generate the row address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AL1</td>
<td>39</td>
<td>I</td>
<td>ADDRESS LOW: These lower-order address inputs are used to generate the row address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AL2</td>
<td>40</td>
<td>I</td>
<td>ADDRESS LOW: These lower-order address inputs are used to generate the row address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AL3</td>
<td>41</td>
<td>I</td>
<td>ADDRESS LOW: These lower-order address inputs are used to generate the row address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AL4</td>
<td>42</td>
<td>I</td>
<td>ADDRESS LOW: These lower-order address inputs are used to generate the row address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AL5</td>
<td>44</td>
<td>I</td>
<td>ADDRESS LOW: These lower-order address inputs are used to generate the row address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AL6</td>
<td>45</td>
<td>I</td>
<td>ADDRESS LOW: These lower-order address inputs are used to generate the row address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AL7</td>
<td>46</td>
<td>I</td>
<td>ADDRESS LOW: These lower-order address inputs are used to generate the row address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AL8</td>
<td>47</td>
<td>I</td>
<td>ADDRESS LOW: These lower-order address inputs are used to generate the row address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AH0</td>
<td>48</td>
<td>I</td>
<td>ADDRESS HIGH: These higher-order address inputs are used to generate the column address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AH1</td>
<td>49</td>
<td>I</td>
<td>ADDRESS HIGH: These higher-order address inputs are used to generate the column address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AH2</td>
<td>50</td>
<td>I</td>
<td>ADDRESS HIGH: These higher-order address inputs are used to generate the column address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AH3</td>
<td>51</td>
<td>I</td>
<td>ADDRESS HIGH: These higher-order address inputs are used to generate the column address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AH4</td>
<td>52</td>
<td>I</td>
<td>ADDRESS HIGH: These higher-order address inputs are used to generate the column address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AH5</td>
<td>53</td>
<td>I</td>
<td>ADDRESS HIGH: These higher-order address inputs are used to generate the column address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AH6</td>
<td>54</td>
<td>I</td>
<td>ADDRESS HIGH: These higher-order address inputs are used to generate the column address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AH7</td>
<td>55</td>
<td>I</td>
<td>ADDRESS HIGH: These higher-order address inputs are used to generate the column address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AH8</td>
<td>56</td>
<td>I</td>
<td>ADDRESS HIGH: These higher-order address inputs are used to generate the column address for the internal address multiplexer.</td>
</tr>
<tr>
<td>PDI</td>
<td>57</td>
<td>I</td>
<td>PROGRAM DATA INPUT: This input programs the various user-selectable options in the 8207. The PCLK pin shifts programming data into the PDI input from optional external shift registers. This pin may be strapped high or low to a default ECC (PDI = Logic &quot;1&quot;) or non-ECC (PDI = Logic &quot;0&quot;) mode configuration.</td>
</tr>
<tr>
<td>RFRQ</td>
<td>58</td>
<td>I</td>
<td>REFRESH REQUEST: This input is sampled on the falling edge of RESET. If it is high at RESET, then the 8207 is programmed for internal refresh request or external refresh request with failsafe protection. If it is low at RESET, then the 8207 is programmed for external refresh without failsafe protection or burst refresh. Once programmed the RFRQ pin accepts signals to start an external refresh with failsafe protection or external refresh without failsafe protection or a burst refresh.</td>
</tr>
</tbody>
</table>
Table 1. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>59</td>
<td>I</td>
<td>CLOCK: This input provides the basic timing for sequencing the internal logic.</td>
</tr>
<tr>
<td>RDB</td>
<td>61</td>
<td>I</td>
<td>READ FOR PORT B: This pin is the read memory request command input for port B. This input also directly accepts the S1 status line from Intel processors.</td>
</tr>
<tr>
<td>WRB</td>
<td>62</td>
<td>I</td>
<td>WRITE FOR PORT B: This pin is the write memory request command input for port B. This input also directly accepts the S0 status line from Intel processors.</td>
</tr>
<tr>
<td>PEB</td>
<td>63</td>
<td>I</td>
<td>PORT ENABLE FOR PORT B: This pin serves to enable a RAM cycle request for port B. It is generally decoded from the port address.</td>
</tr>
<tr>
<td>PCTLB</td>
<td>64</td>
<td>I</td>
<td>PORT CONTROL FOR PORT B: This pin is sampled on the falling edge of RESET. It configures port B to accept command inputs or processor status inputs. If low after RESET, the 8207 is programmed to accept command or iAPX 286 status inputs or Multibus commands. If high after RESET, the 8207 is programmed to accept status inputs from iAPX 86 or iAPX 186 processors. The S2 status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept commands or iAPX 286 status, it should be tied low or it may be used as a Multibus-compatible inhibit signal.</td>
</tr>
<tr>
<td>RDA</td>
<td>65</td>
<td>I</td>
<td>READ FOR PORT A: This pin is the read memory request command input for port A. This input also directly accepts the S1 status line from Intel processors.</td>
</tr>
<tr>
<td>WRA</td>
<td>66</td>
<td>I</td>
<td>WRITE FOR PORT A: This pin is the write memory request command input for port A. This input also directly accepts the S0 status line from Intel processors.</td>
</tr>
<tr>
<td>PEA</td>
<td>67</td>
<td>I</td>
<td>PORT ENABLE FOR PORT A: This pin serves to enable a RAM cycle request for port A. It is generally decoded from the port address.</td>
</tr>
<tr>
<td>PCTLA</td>
<td>68</td>
<td>I</td>
<td>PORT CONTROL FOR PORT A: This pin is sampled on the falling edge of RESET. It configures port A to accept command inputs or processor status inputs. If low after RESET, the 8207 is programmed to accept command or iAPX 286 status inputs or Multibus commands. If high after RESET, the 8207 is programmed to accept status inputs from iAPX 86 or iAPX 186 processors. The S2 status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept commands or iAPX 286 status, it should be tied low or it may be connected to INHIBIT when operating with Multibus.</td>
</tr>
</tbody>
</table>

GENERAL DESCRIPTION

The Intel 8207 Advanced Dynamic RAM Controller (ADRC) is a microcomputer peripheral device which provides the necessary signals to address, refresh and directly drive 16K, 64K and 256K dynamic RAMs. This controller also provides the necessary arbitration circuitry to support dual-port access of the dynamic RAM array.

The ADRC supports several microprocessor interface options including synchronous and asynchronous connection to iAPX 86, iAPX 88, iAPX 186, iAPX 188, iAPX 286 and Multibus.

This device may be used with the 8206 Error Detection and Correction Unit (EDCU). When used with the 8206, the 8207 is programmed in the Error Checking and Correction (ECC) mode. In this mode, the 8207 provides all the necessary control signals for the 8206 to perform memory initialization and transparent error scrubbing during refresh.

FUNCTIONAL DESCRIPTION

Processor Interface

The 8207 has control circuitry for two ports each capable of supporting one of several possible bus structures. The ports are independently configurable allowing the dynamic RAM to serve as an interface between two different bus structures.

Each port of the 8207 may be programmed to run synchronous or asynchronous to the processor clock. (See Synchronous/Asynchronous Mode) The 8207 has been optimized to run synchronously with Intel's iAPX 86, iAPX 88, iAPX 186, iAPX 188 and iAPX 286. When the 8207 is programmed to run in asynchronous mode, the 8207 inserts the necessary synchronization circuitry for the RD, WR, PE, and PCTL inputs.
The 8207 achieves high performance (i.e., no wait states) by decoding the status lines directly from the iAPX 86, iAPX 88, iAPX 186, iAPX 188 and iAPX 286 processors. The 8207 can also be programmed to receive read or write Multibus commands or commands from a bus controller. (See Status/Command Mode)

The 8207 may be programmed to accept the clock of the iAPX 86, 88, 186, 188, or 286. The 8207 adjusts its internal timing to allow for the different clock frequencies of these microprocessors. (See Microprocessor Clock Frequency Option)

Figure 2 shows the different processor interfaces to the 8207 using the synchronous or asynchronous mode and status or command interface.
NOTE: ADDRESS LATCH NOT REQUIRED IN SINGLE-PORT MODE.

Fast-Cycle Synchronous-Status Interface

NOTE: ADDRESS LATCH NOT REQUIRED IN SINGLE-PORT MODE.

Fast-Cycle Asynchronous-Status Interface

Fast-Cycle Synchronous-Command Interface

Fast-Cycle Asynchronous-Command Interface

Figure 2B. Fast-cycle (CFS=1) Port Interfaces Supported by the 8207

Single-Port Operation

The use of an address latch with the iAPX 286 status interface is not needed since the 8207 can internally latch the addresses with an internal signal similar in behavior to the LEN output. This operation is active only in single-port applications when the processor is interfaced to port A.

Dual-Port Operation

The 8207 provides for two-port operation. Two independent processors may access memory controlled by the 8207. The 8207 arbitrates between each of the processor requests and directs data to or from the appropriate port. Selection is done on a priority concept that reassigns priorities based upon past history. Processor requests are internally queued.

Figure 3 shows a dual-port configuration with two iAPX 86 systems interfacing to dynamic RAM. One of the processor systems is interfaced synchronously using the status interface and the other is interfaced asynchronously also using the status interface.

Dynamic RAM Interface

The 8207 is capable of addressing 16K, 64K and 256K dynamic RAMs. Figure 4 shows the connection of the processor address bus to the 8207 using the different RAMs. The 8207 directly supports the 2118 RAM family or any RAM with similar timing requirements and responses including the Intel 2164A RAM.

The 8207 divides memory into as many as four banks, each bank having its own Row (RAS) and Column (CAS) Address Strobe pair. This organization permits RAM cycle interleaving and permits error scrubbing during ECC refresh cycles. RAM cycle interleaving overlaps the start of the next RAM cycle with the RAM Precharge period of the previous cycle. Hiding the precharge period of one RAM cycle behind the data access period of the next RAM cycle optimizes memory bandwidth and is effective as long as successive RAM cycles occur in alternate banks.

Successive data access to the same bank will cause the 8207 to wait for the precharge time of the previous RAM cycle.
NOTE:
*These components are not necessary when using the 80186 components. These functions are provided directly by the 80186.

Figure 3. 8086/80186 Dual Port System
If not all RAM banks are occupied, the 8207 reassigns the RAS and CAS strobes to allow using wider data words without increasing the loading on the RAS and CAS drivers. Table 2 shows the bank selection decoding and the word expansion, including RAS and CAS assignments. For example, if only two RAM banks are occupied, then two RAS and two CAS strobes are activated per bank. Program bits RB1 and RB0 are not used to check the bank select inputs BS1 and BS0. The system design must protect from accesses to "illegal", non-existent banks of memory, by deactivating the PEA, PEB inputs when addressing an illegal bank.

The 8207 can interface to fast (e.g., 2118-10) or slow (e.g., 2118-15) RAMs. The 8207 adjusts and optimizes internal timings for either the fast or slow RAMs as programmed. (See RAM Speed Option).

**Memory Initialization**

After programming, the 8207 performs eight RAM “warm-up” cycles to prepare the dynamic RAM for proper device operation. During “warm-up” some RAM parameters, such as tRAH, tASC, may not be met. This causes no harm to the dynamic RAM array. If configured for operation with error correction, the 8207 and 8206 EDCU will proceed to initialize all of memory (memory is written with zeros with corresponding check bits).

**Table 2. Bank Selection Decoding and Word Expansion**

<table>
<thead>
<tr>
<th>Program Bits</th>
<th>Bank Input</th>
<th>RAS/CAS Pair Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RB1</td>
<td>BS1</td>
<td>BS0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Because the time to initialize memory is fairly long, the 8207 may be programmed to skip initialization in ECC mode. The time required to initialize all of memory is dependent on the clock cycle time to the 8207 and can be calculated by the following equation:

\[ T_{\text{INIT}} = (2^{23}) \times T_{\text{CLCL}} \]

if \( T_{\text{CLCL}} = 125 \text{ ns} \) then \( T_{\text{INIT}} = 1 \text{ sec.} \)

**8206 ECC Interface**

For operation with Error Checking and Correction (ECC), the 8207 adjusts its internal timing and changes some pin functions to optimize performance and provide a clean dual-port memory interface between the 8206 EDCU and memory. The 8207 directly supports a master-only (16-bit word plus 6 check bits) system. Under extended operation and reduced clock frequency, the 8207 will support any ECC master-slave configuration up to 80 data bits, which is the maximum set by the 8206 EDCU. (See Extend Option)

Correctable errors detected during memory read cycles are corrected immediately and then written back into memory.

In a synchronous bus environment, ECC system performance has been optimized to enhance processor throughput, while in an asynchronous bus environment (the Multibus), ECC performance has been optimized to get valid data onto the bus as quickly as possible. Performance optimization, processor throughput or quick data access may be selected via the Transfer Acknowledge Option.

The main difference between the two ECC implementations is that, when optimized for processor throughput, RAM data is always corrected and an advanced transfer acknowledge is issued at a point when, by knowing the processor characteristics, data is guaranteed to be valid by the time the processor needs it.

When optimized for quick data access, (valid for Multibus) the 8206 is configured in the uncorrecting mode where the delay associated with error correction circuitry is transparent, and a transfer acknowledge is issued as soon as valid data is known to exist. If the ERROR flag is activated, then the transfer acknowledge is delayed until after the 8206 has instructed the 8206 to correct the data and the corrected data becomes available on the bus. Figure 5 illustrates a dual-port ECC system.

Figure 6 illustrates the interface required to drive the CRCT pin of the 8206, in the case that one port (PORT A) receives an advanced acknowledge (not Multibus-compatible), while the other port (PORT B) receives XACK (which is Multibus-compatible).

**Error Scrubbing**

The 8207/8206 performs error correction during refresh cycles (error scrubbing). Since the 8207 must refresh RAM, performing error scrubbing during refresh allows it to be accomplished without additional performance penalties.

Upon detection of a correctable error during refresh, the RAM refresh cycle is lengthened slightly to permit the 8206 to correct the error and for the corrected word to be rewritten into memory. Uncorrectable errors detected during scrubbing are ignored.

**Refresh**

The 8207 provides an internal refresh interval counter and a refresh address counter to allow the 8207 to refresh memory. The 8207 will refresh 128 rows every 2 milliseconds or 256 rows every 4 milliseconds, which allows all RAM refresh options to be supported. In addition, there exists the ability to refresh 256 row address locations every 2 milliseconds via the Refresh Period programming option.

The 8207 may be programmed for any of four different refresh options: Internal refresh only, External refresh with failsafe protection, External refresh without failsafe protection, Burst Refresh mode, or no refresh. (See Refresh Options)

It is possible to decrease the refresh time interval by 10%, 20% or 30%. This option allows the 8207 to compensate for reduced clock frequencies. Note that an additional 5% interval shortening is built-in in all refresh interval options to compensate for clock variations and non-immediate response to the internally generated refresh request. (See Refresh Period Options)

**External Refresh Requests after RESET**

External refresh requests are not recognized by the 8207 until after it is finished programming and preparing memory for access. Memory preparation includes 8 RAM cycles to prepare and ensure proper
Figure 5. Two-Port ECC Implementation Using the 8207 and the 8206
The differentiated reset pulse would be shorter than the system reset pulse by at least the programming period required by the 8207. The differentiated reset pulse first resets the 8207, and system reset would reset the rest of the system. While the rest of the system is still in reset, the 8207 completes its programming. Figure 7 illustrates a circuit to accomplish this task.

Within four clocks after RESET goes active, all the 8207 outputs will go high, except for PSEN, WE, and A00-2, which will go low.

**OPERATIONAL DESCRIPTION**

**Programming the 8207**

The 8207 is programmed after reset. On the falling edge of RESET, the logic states of several input pins are latched internally. The falling edge of RESET actually performs the latching, which means that the logic levels on these inputs must be stable prior to that time. The inputs whose logic levels are latched at the end of reset are the PCTLA, PCTLB, REFRQ, and PDI pins. Figure 8 shows the necessary timing for programming the 8207.

![Figure 7. 8207 Differentiated Reset Circuit](image-url)
Status/Command Mode

The two processor ports of the 8207 are configured by the states of the PCTL A and PCTL B pins. Which interface is selected depends on the state of the individual port's PCTL pin at the end of reset. If PCTL is high at the end of the reset, the 8086 Status interface is selected; if it is low, then the Command interface is selected.

The status lines of the 80286 are similar in code and timing to the Multibus command lines, while the status code and timing of the 8076 and 8088 are identical to those of the 80186 and 80188 (ignoring the differences in clock duty cycle). Thus there exists two interface configurations, one for the 80286 status or Multibus memory commands, which is called the Command interface, and one for 8086, 8088, 80186 or 80188 status, called the 8086 Status interface. The Command interface can also directly interface to the command lines of the bus controllers for the 8086, 8088, 80186 and the 80286.

The 8086 Status interface allows direct decoding of the status of the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. Table 3 shows how the status lines are decoded. While in the Command mode the iAPX 286 status can be directly decoded. Microprocessor bus controller read or write commands or Multibus commands can also be directed to the 8207 when in Command mode.

Refresh Options

Immediately after system reset, the state of the REFREQ input pin is examined. If REFREQ is high, the 8207 provides the user with the choice between self-refresh or user-generated refresh with failsafe protection. Failsafe protection guarantees that if the

### Table 3A. Status Coding of 8086, 80186 and 80286

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Function</th>
<th>8086/80186</th>
<th>80286</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2 S1 S0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>INTERRUPT</td>
<td>INTERRUPT</td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>I/O READ</td>
<td>I/O READ</td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>I/O WRITE</td>
<td>I/O WRITE</td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>HALT</td>
<td>IDLE</td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>INSTRUCTION FETCH</td>
<td>HALT</td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>MEMORY READ</td>
<td>MEMORY READ</td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>MEMORY WRITE</td>
<td>MEMORY WRITE</td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>IDLE</td>
<td>IDLE</td>
<td></td>
</tr>
</tbody>
</table>

### Table 3B. 8207 Response

<table>
<thead>
<tr>
<th>8207 Command</th>
<th>Function</th>
<th>8086/80186 Status Interface</th>
<th>80286 Status or Command Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCTL RD WR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>IGNORE</td>
<td>IGNORE</td>
<td>IGNORE*</td>
</tr>
<tr>
<td>0 0 1</td>
<td>IGNORE</td>
<td>READ</td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>IGNORE</td>
<td>WRITE</td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>IGNORE</td>
<td>IGNORE</td>
<td>IGNORE*</td>
</tr>
<tr>
<td>1 0 0</td>
<td>READ</td>
<td>IGNORE</td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>READ</td>
<td>INHIBIT</td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>WRITE</td>
<td>INHIBIT</td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>IGNORE</td>
<td>IGNORE</td>
<td></td>
</tr>
</tbody>
</table>

*Illegal with CFS=0
user does not come back with another refresh request before the internal refresh interval counter times out, a refresh request will be automatically generated. If the REFRQ pin is low immediately after a reset, then the user has the choice of a single external refresh cycle without failsafe, burst refresh or no refresh.

Internal Refresh Only

For the 8207 to generate internal refresh requests, it is necessary only to strap the REFRQ input pin high.

External Refresh with Failsafe

To allow user-generated refresh requests with failsafe protection, it is necessary to hold the REFRQ input high until after reset. Thereafter, a low-to-high transition on this input causes a refresh request to be generated and the internal refresh interval counter to be reset. A high-to-low transition has no effect on the 8207. A refresh request is not recognized until a previous request has been serviced.

External Refresh without Failsafe

To generate single external refresh requests without failsafe protection, it is necessary to hold REFRQ low until after reset. Thereafter, bringing REFRQ high for one clock period causes a refresh request to be generated. A refresh request is not recognized until a previous request has been serviced.

Burst Refresh

Burst refresh is implemented through the same procedure as a single external refresh without failsafe (i.e., REFRQ is kept low until after reset). Thereafter, bringing REFRQ high for at least two clock periods causes a burst of up to 128 row address locations to be refreshed.

In ECC-configured systems, 128 locations are scrubbed. Any refresh request is not recognized until a previous request has been serviced (i.e., burst completed).

No Refresh

It is necessary to hold REFRQ low until after reset. This is the same as programming External Refresh without Failsafe. No refresh is accomplished by keeping REFRQ low.

Option Program Data Word

The program data word consists of 16 program data bits, PD0—PD15. If the first program data bit PD0 is set to logic 1, the 8207 is configured to support ECC. If it is logic 0, the 8207 is configured to support a non-ECC system. The remaining bits, PD1—PD15, may then be programmed to optimize a selected configuration. Figures 9 and 10 show the Program words for non-ECC and ECC operation.

Using an External Shift Register

The 8207 may be configured to use an external shift register with asynchronous load capability such as a 74LS165. The reset pulse serves to parallel load the shift register and the 8207 supplies the clocking signal to shift the data in. Figure 11 shows a sample circuit diagram of an external shift register circuit.

Serial data is shifted into the 8207 via the PDI pin (57), and clock is provided by the MUX/PCLK pin (12), which generates a total of 16 clock pulses. After programming is complete, data appearing at the input of the PDI pin is ignored. MUX/PCLK is a dual-function pin. During programming, it serves to clock the external shift register, and after programming is completed, it reverts to a MUX control pin. As the pin changes state to select different port addresses, it continues to clock the shift register. This does not present a problem because data at the PDI pin is ignored after programming. Figure 8 illustrates the timing requirements of the shift register circuitry.

ECC Mode (ECC Program Bit)

The state of PDI (Program Data In) pin at reset determines whether the system is an ECC or non-ECC configuration. It is used internally by the 8207 to begin configuring timing circuits, even before programming is completely finished. The 8207 then begins programming the rest of the options.

Default Programming Options

After reset, the 8207 serially shifts in a program data word via the PDI pin. This pin may be strapped either high or low, or connected to an external shift register. Strapping PDI high causes the 8207 to default to a particular system configuration with error correction, and strapping it low causes the 8207 to default to a particular system configuration without error correction. Table 4 shows the default configurations.
### Figure 9. Non-ECC Mode Program Data Word

<table>
<thead>
<tr>
<th>PD15</th>
<th>PD8</th>
<th>PD7</th>
<th>PD0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>TM1</td>
<td>PPR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FFS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PL8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CI0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CI1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RB1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RBS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

#### PROGRAM DATA BIT NAME POLARITY/FUNCTION

<table>
<thead>
<tr>
<th>PD0</th>
<th>ECC</th>
<th>DI0</th>
<th>0</th>
<th>FOR NON-ECC MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD1</td>
<td>SA</td>
<td>SA</td>
<td>0</td>
<td>PORT A IS SYNCHRONOUS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SA</td>
<td>1</td>
<td>PORT A IS ASYNCHRONOUS</td>
</tr>
<tr>
<td>PD2</td>
<td>SB</td>
<td>SB</td>
<td>0</td>
<td>PORT B IS SYNCHRONOUS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SB</td>
<td>1</td>
<td>PORT B IS ASYNCHRONOUS</td>
</tr>
<tr>
<td>PD3</td>
<td>CFS</td>
<td>CFS</td>
<td>0</td>
<td>FAST-CYCLE iAPX 286 MODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CFS</td>
<td>1</td>
<td>SLOW-CYCLE iAPX 86 MODE</td>
</tr>
<tr>
<td>PD4</td>
<td>RFS</td>
<td>RFS</td>
<td>0</td>
<td>FAST RAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RFS</td>
<td>1</td>
<td>SLOW RAM</td>
</tr>
<tr>
<td>PD5</td>
<td>RBO</td>
<td>RBO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD6</td>
<td>RB1</td>
<td>RB1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD7</td>
<td>CI0</td>
<td>CI0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD8</td>
<td>CFI</td>
<td>CFI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD9</td>
<td>PL8</td>
<td>PL8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD10</td>
<td>EXT</td>
<td>EXT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD11</td>
<td>FFS</td>
<td>FFS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD12</td>
<td>PPR</td>
<td>PPR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD13</td>
<td>TM1</td>
<td>TM1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD14</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD15</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Figure 10. ECC Mode Program Data Word

<table>
<thead>
<tr>
<th>TM2</th>
<th>PD7</th>
<th>PD6</th>
<th>PD5</th>
<th>RBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBO</td>
<td>PPR</td>
<td>FFS</td>
<td>EXT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PL8</td>
<td>CI0</td>
<td>CI1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XB</td>
<td>XA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### PROGRAM DATA BIT NAME POLARITY/FUNCTION

<table>
<thead>
<tr>
<th>PD0</th>
<th>ECC</th>
<th>DI0</th>
<th>0</th>
<th>ECC MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD1</td>
<td>SA</td>
<td>SA</td>
<td>0</td>
<td>PORT A ASYNCHRONOUS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SA</td>
<td>1</td>
<td>PORT A SYNCHRONOUS</td>
</tr>
<tr>
<td>PD2</td>
<td>SB</td>
<td>SB</td>
<td>0</td>
<td>PORT B SYNCHRONOUS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SB</td>
<td>1</td>
<td>PORT B ASYNCHRONOUS</td>
</tr>
<tr>
<td>PD3</td>
<td>CFS</td>
<td>CFS</td>
<td>0</td>
<td>SLOW-CYCLE iAPX 86 MODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CFS</td>
<td>1</td>
<td>FAST-CYCLE iAPX 286 MODE</td>
</tr>
<tr>
<td>PD4</td>
<td>RFS</td>
<td>RFS</td>
<td>0</td>
<td>SLOW RAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RFS</td>
<td>1</td>
<td>FAST RAM</td>
</tr>
<tr>
<td>PD5</td>
<td>XA</td>
<td>XA</td>
<td>0</td>
<td>MULTIBUS-COMPATIBLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XA</td>
<td>1</td>
<td>ADVANCED ACKA NOT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MULTIBUS-COMPATIBLE</td>
</tr>
<tr>
<td>PD6</td>
<td>XB</td>
<td>XB</td>
<td>0</td>
<td>ADVANCED ACKB NOT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XB</td>
<td>1</td>
<td>MULTIBUS-COMPATIBLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MULTIBUS-COMPATIBLE</td>
</tr>
<tr>
<td>PD7</td>
<td>CI1</td>
<td>CI1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CI0</td>
<td>CI0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD8</td>
<td>PL8</td>
<td>PL8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD9</td>
<td>EXT</td>
<td>EXT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD10</td>
<td>FFS</td>
<td>FFS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD11</td>
<td>PPR</td>
<td>PPR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD12</td>
<td>TM2</td>
<td>TM2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD13</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD14</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD15</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Figures 9 and 10

- Figure 9. Non-ECC Mode Program Data Word
- Figure 10. ECC Mode Program Data Word
If further system flexibility is needed, one or two external shift registers can be used to tailor the 8207 to its operating environment.

**Synchronous/Asynchronous Mode (SA and SB Program Bits)**

Each port of the 8207 may be independently configured to accept synchronous or asynchronous port commands (RD, WR, PCTL) and Port Enable (PE) via the program bits SA and SB. The state of the SA and SB programming bits determine whether their associated ports are synchronous or asynchronous.

While a port may be configured with either the Status or Command interface in the synchronous mode, certain restrictions exist in the asynchronous mode. An asynchronous Command interface using the control lines of the Multibus is supported, and an asynchronous 8086 interface using the control lines of the 8086 is supported, with the use of TTL gates as illustrated in Figure 2. In the 8086 case, the TTL gates are needed to guarantee that status does not appear at the 8207's inputs too much before address, so that a cycle would start before address was valid.

**Microprocessor Clock Frequency Option (CFS and FFS Program Bits)**

The 8207 can be programmed to interface with slow-cycle microprocessors like the 8086, 8088, 80188 and 80186 or fast-cycle microprocessors like the 80286. The CFS bit configures the microprocessor interface to accept slow or fast cycle signals from either microprocessor group.

The FFS bit is used to select the speed of the microprocessor clock. Table 5 shows the various microprocessor clock frequency options that can be programmed.
The external clock frequency must be programmed so that the failsafe refresh repetition circuitry can adjust its internal timing accordingly to produce a refresh request as programmed.

**RAM Speed Option (RFS Program Bit)**

The RAM Speed programming option determines whether RAM timing will be optimized for a fast or slow RAM. Whether a RAM is fast or slow is measured relative to the 2118-10 (Fast) or the 2118-15 (Slow) RAM specifications.

**Refresh Period Options (C10, C11, and PLS Program Bits)**

The 8207 refreshes with either 128 rows every 2 milliseconds or 256 rows every 4 milliseconds. This translates to one refresh cycle being executed approximately once every 15.6 microseconds. This rate can be changed to 256 rows every 2 milliseconds or a refresh approximately once every 7.8 microseconds via the Period Long/Short, program bit PLS, programming option. The 7.8 microsecond refresh request rate is intended for those RAMs, 64K and above, which may require a faster refresh rate.

In addition to PLS program option, two other programming bits for refresh exist: Count Interval 0 (C10) and Count Interval 1, (C11). These two programming bits allow the rate at which refresh requests are generated to be increased in order to permit refresh requests to be generated close to the same 15.6 or 7.8 microsecond period when the 8207 is operating at reduced frequencies. The interval between refreshes is decreased by 0%, 10%, 20%, or 30% as a function of how the count interval bits are programmed. A 5% guardband is built-in to allow for any clock frequency variations. Table 6 shows the refresh period options available.

The numbers tabulated under Count Interval represent the number of clock periods between internal refresh requests. The percentages in parentheses represent the decrease in the interval between refresh requests. Note that all intervals have a built-in 5% (approximately) safety factor to compensate for minor clock frequency deviations and non-immediate response to internal refresh requests.

**Extend Option (EXT Program Bit)**

The Extend option lengthens the memory cycle to allow longer access time which may be required by the system. Extend alters the RAM timing to compensate for increased loading on the Row and Column Address Strobes; and in the multiplexed Address Out lines.

**Port Priority Option and Arbitration (PPR Program Bit)**

The 8207 has to internally arbitrate among three ports: Port A, Port B and Port C—the refresh port. Port C is an internal port dedicated to servicing refresh requests, whether they are generated internally by the refresh interval counter, or externally by the user. Two arbitration approaches are available via

---

**Table 6. Refresh Count Interval Table**

<table>
<thead>
<tr>
<th>Ref. Period (μS)</th>
<th>CFS</th>
<th>PLS</th>
<th>FFS</th>
<th>Count Interval C11, C10 (8207 Clock Periods)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00 (0%)</td>
</tr>
<tr>
<td>15.6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>236</td>
</tr>
<tr>
<td>7.8</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>118</td>
</tr>
<tr>
<td>15.6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>148</td>
</tr>
<tr>
<td>7.8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>74</td>
</tr>
<tr>
<td>15.6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>118</td>
</tr>
<tr>
<td>7.8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>59</td>
</tr>
<tr>
<td>15.6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>74</td>
</tr>
<tr>
<td>7.8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37</td>
</tr>
</tbody>
</table>

5-66
the Port Priority programming option, program bit PPR. PPR determines whether the most recently used port will remain selected (PPR = 1) or whether Port A will be favored or preferred over Port B (PPR = 0).

A port is selected if the arbiter has given the selected port direct access to the timing generators. The front-end logic, which includes the arbiter, is designed to operate in parallel with the selected port. Thus a request on the selected port is serviced immediately. In contrast, an unselected port only has access to the timing generators through the front-end logic. Before a RAM cycle can start for an unselected port, that port must first become selected (i.e., the MUX output now gates that port’s address into the 8207 in the case of Port A or B). Also, in order to allow its address to stabilize, a newly selected port’s first RAM cycle is started by the front-end logic. Therefore, the selected port has direct access to the timing generators. What all this means is that a request on a selected port is started immediately, while a request on an unselected port is started two to three clock periods after the request, assuming that the other two ports are idle. Under normal operating conditions, this arbitration time is hidden behind the RAM cycle of the selected port so that as soon as the present cycle is over a new cycle is started. Table 7 lists the arbitration rules for both options.

**Port LOCK Function**

The LOCK function provides each port with the ability to obtain uninterrupted access to a critical region of memory and, thereby, to guarantee that the opposite port cannot “sneak in” and read from or write to the critical region prematurely.

Only one LOCK pin is present and is multiplexed between the two ports as follows: when MUX is high, the 8207 treats the LOCK input as originating at PORT A, while when MUX is low, the 8207 treats LOCK as originating at PORT B. When the 8207 recognizes a LOCK, the MUX output will remain pointed to the locking port until LOCK is deactivated. Refresh is not affected by LOCK and can occur during a locked memory cycle.

**Table 7. The Arbitration Rules for the Most Recently Used Port Priority and for Port A Priority Options Are As follows:**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>If only one port requests service, then that port—if not already selected—becomes selected.</td>
</tr>
<tr>
<td>2a.</td>
<td>When no service requests are pending, the last selected processor port (Port A or B) will remain selected. (Most Recently Used Port Priority Option)</td>
</tr>
<tr>
<td>2b.</td>
<td>When no service requests are pending, Port A is selected whether it requests service or not. (Port A Priority Option)</td>
</tr>
<tr>
<td>3.</td>
<td>During reset initialization only Port C, the refresh port, is selected.</td>
</tr>
<tr>
<td>4.</td>
<td>If no processor requests are pending after reset initialization, Port A will be selected.</td>
</tr>
<tr>
<td>5a.</td>
<td>If Ports A and B simultaneously(*) request service while Port C is being serviced, then the next port to be selected is the one which was not selected prior to servicing Port C. (Most Recently Used Port Priority Option)</td>
</tr>
<tr>
<td>5b.</td>
<td>If Ports A and B simultaneously(*) request service while Port C is selected, then the next port to be selected is Port A. (Port A Priority Option)</td>
</tr>
<tr>
<td>6.</td>
<td>If a port simultaneously requests service with the currently selected port, service is granted to the selected port.</td>
</tr>
<tr>
<td>7.</td>
<td>The MUX output remains in its last state whenever Port C is selected.</td>
</tr>
<tr>
<td>8.</td>
<td>If Port C and either Port A or Port B (or both) simultaneously request service, then service is granted to the requester whose port is already selected. If the selected port is not requesting service, then service is granted to Port C.</td>
</tr>
<tr>
<td>9.</td>
<td>If during the servicing of one port, the other port requests service before or simultaneously with the refresh port, the refresh port is selected. A new port is not selected before the presently selected port is deactivated.</td>
</tr>
<tr>
<td>10.</td>
<td>Activating LOCK will mask off service requests from Port B if the MUX output is high, or from Port A if the MUX output is low.</td>
</tr>
</tbody>
</table>

* By “simultaneous” it is meant that two or more requests are valid at the clock edge at which the internal arbiter samples them.
Dual-Port Considerations

For both ports to be operated synchronously, several conditions must be met. The processors must be the same type (Fast or Slow Cycle) as defined by Table 8 and they must have synchronized clocks. Also when processor types are mixed, even though the clocks may be in phase, one frequency may be twice that of the other. So to run both ports synchronous using the status interface, the processors must have related timings (both phase and frequency). If these conditions cannot be met, then one port must run synchronous and the other asynchronous.

Figure 3 illustrates an example of dual-port operation using the processors in the slow cycle group. Note the use of cross-coupled NAND gates at the MUX output for minimizing contention between the two latches, and the use of flip flops on the status lines of the asynchronous processor for delaying the status and thereby guaranteeing RAS will not be issued, even in the worst case, until address is valid.

Processor Timing

In order to run without wait states, AACK must be used and connected to the SRDY input of the appropriate bus controller. AACK is issued relative to a point within the RAM cycle and has no fixed relationship to the processor's request. The timing is such, however, that the processor will run without wait states, barring refresh cycles, bank precharge, and RAM accesses from the other port. In non-ECC fast cycle, fast RAM, non-extended configurations (80286), AACK is issued on the next falling edge of the clock after the edge that issues RAS. In non-ECC, slow cycle, non-extended, or extended with fast RAM cycle configurations (8086, 80188, 80186), AACK is issued on the same clock cycle that issues RAS. Figure 14 illustrates the timing relationship between AACK, the RAM cycle, and the processor cycle for several different situations.

Port Enable (PE) setup time requirements depend on whether the associated port is configured for synchronous or asynchronous fast or slow cycle operation. In a synchronous fast cycle configuration, PE is required to be setup to the same clock edge as the status or commands. If PE is true (low), a RAM cycle is started; if not, the cycle is aborted. The memory cycle will only begin when both valid signals (PE and RD or WR) are recognized at a particular clock edge. In asynchronous operation, PE is required to be setup to the same clock edge as the internally synchronized status or commands. Externally, this allows the internal synchronization delay to be added to the status (or command)-to-PE delay time, thus allowing for more external decode time that is available in synchronous operation. The minimum synchronization delay is the additional amount that PE must be held valid. If PE is not held valid for the maximum synchronization delay time, it is possible that PE will go invalid prior to the status or command being synchronized. In such a case the 8207 aborts the cycle. If a memory cycle intended for the 8207 is aborted, then no acknowledge (AACK or XACK) is issued and the processor locks up in endless wait states. Figure 15 illustrates the status (command) timing requirements for synchronous and asynchronous systems. Figures 16 and 17 show a more detailed hook-up of the 8207 to the 8086 and the 80286, respectively.
NOTE:
1. The RAS and CAS shown in figure are different banks being accessed.

Figure 14. iAPX 286/8207 Synchronous-Status Timing Programmed in non-ECC Mode, C0 Configuration (Read Cycle)
Memory Acknowledge (AACK, XACK)

In system configurations without error correction, two memory acknowledge signals per port are supplied by the 8207. They are the Advanced Acknowledge strobe (AACK) and the Transfer Acknowledge strobe (XACK). The CFS programming bit determines for which processor AACKA and AACKB are optimized, either 80286 (CFS = 1) or 8086/186 (CFS = 0), while the SA and SB programming bits optimize AACK for synchronous operation ("early" AACK) or asynchronous operation ("late" AACK).

Both the early and late AACK strobos are three clocks long for CFS = 1 and two clocks long for CFS = 0. The XACK strobe is asserted when data is valid (for reads) or when data may be removed (for writes) and meets the Multibus requirements. XACK is removed asynchronously by the command going inactive. Since in asynchronous operation the 8207 removes read data before late AACK or XACK is recognized by the CPU, the user must provide for data latching in the system until the CPU reads the data. In synchronous operation, data latching is unnecessary since the 8207 will not remove data until the CPU has read it.

In ECC-based systems there is one memory acknowledge (XACK or AACK) per port and a programming bit associated with each acknowledge. If the X programming bit is active, the strobe is configured as XACK, while if the bit is inactive, the strobe is configured as AACK. As in non-ECC, the SA and SB programming bits determine whether the AACK strobe is early or late (EAACK or LAACK).

Data will always be valid a fixed time after the occurrence of the advanced acknowledge. Table 9 summarizes the various transfer acknowledge options.
NOTE:
*These components are not necessary when using the 80186. These functions are provided directly by the 80186.

Figure 16. 8086/80186, 8207 Single Port Non-ECC Synchronous Systems
Note: While the 8207 does not need the input addresses latched, A0, BHE must come from the latched address bus.

Figure 17. 80286 Hook-up to 8207 Non-ECC Synchronous System–Single Port.
Table 8. Processor Interface/Acknowledge Summary

<table>
<thead>
<tr>
<th>CYCLE</th>
<th>PROCESSOR</th>
<th>REQUEST TYPE</th>
<th>SYNC/ASYNC INTERFACE</th>
<th>ACKNOWLEDGE TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAST CYCLE CFS=1</td>
<td>80286</td>
<td>STATUS</td>
<td>SYNC</td>
<td>EAA CK</td>
</tr>
<tr>
<td></td>
<td>80286</td>
<td>STATUS</td>
<td>ASYNC</td>
<td>LAA CK</td>
</tr>
<tr>
<td></td>
<td>80286</td>
<td>COMMAND</td>
<td>SYNC</td>
<td>EAA CK</td>
</tr>
<tr>
<td></td>
<td>80286</td>
<td>COMMAND</td>
<td>ASYNC</td>
<td>LAA CK</td>
</tr>
<tr>
<td></td>
<td>8086/80186</td>
<td>STATUS</td>
<td>ASYNC</td>
<td>LAA CK</td>
</tr>
<tr>
<td></td>
<td>8086/80186</td>
<td>COMMAND</td>
<td>ASYNC</td>
<td>LAA CK</td>
</tr>
<tr>
<td></td>
<td>MULTIBUS</td>
<td>COMMAND</td>
<td>ASYNC</td>
<td>XACK</td>
</tr>
<tr>
<td>SLOW CYCLE CFS=0</td>
<td>8086/80186</td>
<td>STATUS</td>
<td>SYNC</td>
<td>EAA CK</td>
</tr>
<tr>
<td></td>
<td>8086/80186</td>
<td>STATUS</td>
<td>ASYNC</td>
<td>LAA CK</td>
</tr>
<tr>
<td></td>
<td>8086/80186</td>
<td>COMMAND</td>
<td>SYNC</td>
<td>EAA CK</td>
</tr>
<tr>
<td></td>
<td>8086/80186</td>
<td>COMMAND</td>
<td>ASYNC</td>
<td>LAA CK</td>
</tr>
<tr>
<td></td>
<td>MULTIBUS</td>
<td>COMMAND</td>
<td>ASYNC</td>
<td>XACK</td>
</tr>
</tbody>
</table>

Table 9. Memory Acknowledge Option Summary

<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Asynchronous</th>
<th>XACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Cycle</td>
<td>AACK Optimized for Local 80286</td>
<td>AACK Optimized for Remote 80286</td>
</tr>
<tr>
<td>Slow Cycle</td>
<td>AACK Optimized for Local 8086/186</td>
<td>AACK Optimized for Remote 8086/186</td>
</tr>
</tbody>
</table>

Test Modes

Two special test modes exist in the 8207 to facilitate testing. Test Mode 1 (non-ECC mode) splits the refresh address counter into two separate counters and Test Mode 2 (ECC mode) presets the refresh address counter to a value slightly less than rollover.

Test Mode 1 splits the address counter into two, and increments both counters simultaneously with each refresh address update. By generating external refresh requests, the tester is able to check for proper operation of both counters. Once proper individual counter operation has been established, the 8207 must be returned to normal mode and a second test performed to check that the carry from the first counter increments the second counter. The outputs of the counters are presented on the address output bus with the same timing as the row and column addresses of a normal scrubbing operation. During Test Mode 1, memory initialization is inhibited, since the 8207, by definition, is in non-ECC mode.

Test Mode 2 sets the internal refresh counter to a value slightly less than rollover. During functional testing other than that covered in Test Mode 1, the 8207 will normally be set in Test Mode 2. Test Mode 2 eliminates memory initialization in ECC mode. This allows quick examination of the circuitry which brings the 8207 out of memory initialization and into normal operation.

General System Considerations

The RAS₀₋₃, CAS₀₋₃, AO₀₋₈, output buffers were designed to directly drive the heavy capacitive loads associated with dynamic RAM arrays. To keep the RAM driver outputs from ringing excessively in the system environment and causing noise in other output pins it is necessary to match the output impedance of the RAM output buffers with the RAM array by using series resistors and to add series resistors to other control outputs for noise reduction if necessary. Each application may have different impedance characteristics and may require different series resistance values. The series resistance values should be determined for each application. In non-ECC systems unused ECC input pins should be tied high or low to improve noise immunity.

The 8207 is packaged in a 68-pin, leadless JEDEC type A hermetic chip carrier.
NOTE:

8207 is packaged in a 68 pin JEDEC Type A hermetic leadless chip carrier.

Figure 19. 8207 Pinout Diagram
A.C. CHARACTERISTICS

\(T_A = 0^\circ C \text{ to } 70^\circ C; \ V_{CC} = +5V \pm 5\%; \ V_{SS} \text{OV})\)

Measurements made with respect to \(RAS_0,3, \ CAS_0,3, \ AO_0,8,\) are a +2.4V and 0.8V. All other pins are measured at 2.0V and 0.8V. All times are nsec unless otherwise indicated. Testing done with specified test load.

### CLOCK AND PROGRAMMING

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>(8207-16, -8) (FFS=1)</th>
<th>(8207-12, -6) (FFS=0)</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>tF</td>
<td>Clock Fall Time</td>
<td>10</td>
<td>10</td>
<td>ns</td>
<td>3</td>
</tr>
<tr>
<td>-</td>
<td>tR</td>
<td>Clock Rise Time</td>
<td>10</td>
<td>10</td>
<td>ns</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>TCLCL</td>
<td>Clock Period</td>
<td>125 (8207-12, 8207-8, 8207-6)</td>
<td>150 (8207-16)</td>
<td>ns</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>TCL</td>
<td>Clock Low Time</td>
<td>150 (8207-16)</td>
<td>150 (8207-12, 8207-8, 8207-6)</td>
<td>ns</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>TCH</td>
<td>Clock High Time</td>
<td>180 (8207-16, 8207-12, 8207-8, 8207-6)</td>
<td>180 (TCLCL2:12)</td>
<td>ns</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>TRTVCL</td>
<td>Reset to CLKI Setup</td>
<td>40</td>
<td>55</td>
<td>ns</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>TRTH</td>
<td>Reset Pulse Width</td>
<td>4 TCLCL</td>
<td>4 TCLCL</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TPGVRTL</td>
<td>PCTL, PDI, RFRQ to RESET()</td>
<td>125</td>
<td>167</td>
<td>ns</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>TRTLPGX</td>
<td>PCTL, RFRQ to RESET() Hold</td>
<td>10</td>
<td>10</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TCLPC</td>
<td>PCLK from CLKI Delay</td>
<td>45</td>
<td>55</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>TPDVCL</td>
<td>PDin to CLKI Setup</td>
<td>60</td>
<td>85</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>TCLPDX</td>
<td>PDin to CLKI Hold</td>
<td>40</td>
<td>55</td>
<td>ns</td>
<td>6</td>
</tr>
</tbody>
</table>
ABSOLUTE MAXIMUM RATINGS

Ambient Temperature
Under Bias .............................. -0°C to +70°C
Storage Temperature ................. -65°C to +150°C
Voltage On Any Pin With Respect to Ground .............. -0.5V to +7V
Power Dissipation ..................... 2.5 Watts

NOTICE: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

$T_A=0^\circ \text{C to } 70^\circ \text{C}; V_{\text{CC}}=5.0V \pm 5\%; V_{\text{SS}}=\text{GND}$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{\text{IL}}$</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>+0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>$V_{\text{IH}}$</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>$V_{\text{CC}} + 0.5$</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>$V_{\text{OL}}$</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td></td>
<td>V</td>
<td>Note 1</td>
</tr>
<tr>
<td>$V_{\text{OH}}$</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td></td>
<td>V</td>
<td>Note 1</td>
</tr>
<tr>
<td>$V_{\text{RL}}$</td>
<td>RAM Output Low Voltage</td>
<td>0.45</td>
<td></td>
<td>V</td>
<td>Note 1</td>
</tr>
<tr>
<td>$V_{\text{RH}}$</td>
<td>RAM Output High Voltage</td>
<td>2.6</td>
<td></td>
<td>V</td>
<td>Note 1</td>
</tr>
<tr>
<td>$I_{\text{CC}}$</td>
<td>Supply Current</td>
<td>455</td>
<td></td>
<td>mA</td>
<td>$T_A=0^\circ \text{C}$</td>
</tr>
<tr>
<td>$I_{\text{IL}}$</td>
<td>Input Leakage Current</td>
<td>+10</td>
<td></td>
<td>$\mu A$</td>
<td>$0V \leq V_{\text{IN}} \leq V_{\text{CC}}$</td>
</tr>
<tr>
<td>$V_{\text{CL}}$</td>
<td>Clock Input Low Voltage</td>
<td>-0.5</td>
<td>+0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{\text{CH}}$</td>
<td>Clock Input High Voltage</td>
<td>3.8</td>
<td>$V_{\text{CC}} + 0.5$</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>$C_{\text{IN}}$</td>
<td>Input Capacitance</td>
<td>20</td>
<td></td>
<td>$pF$</td>
<td>$f_c = 1 \text{ MHz}$</td>
</tr>
</tbody>
</table>

NOTE 1:

$I_{\text{OL}} = 5 \text{ mA}$ and $I_{\text{OH}} = -0.2 \text{ mA}$ (Typically $I_{\text{OL}} = 10 \text{ mA}$ and $I_{\text{OH}} = -0.88 \text{ mA}$)

WE : $I_{\text{OL}} = 8 \text{ mA}$

A.C. Testing Load Circuit

A.C. Testing Input, Output Waveform

A.C. Testing inputs (except clock) are driven at 2.4V for a logic "1" and 0.45V for a logic "0" (clock is driven at 4.0V and 0.45V for logic "1" and "0" respectively). Timing measurements are made at 2.0V, 2.4V for logic "1" and 0.8 V for logic "0"
A.C. CHARACTERISTICS (Continued)

**RAM WARM-UP AND INITIALIZATION**

| 64 | TCLWZL | WZ from CLK\textsubscript{i} Delay | 40 | 55 | ns | 7 |

**SYNCHRONOUS \( \mu \)P PORT INTERFACE**

| 11 | TPEVCL | PE to CLK\textsubscript{i} Setup | 30 | 40 | ns | 2 |
| 12 | TKVCL  | RD, WR, PE, PCTL to CLK\textsubscript{i} Setup | 20 | 25 | ns | 1 |
| 13 | TCLKX  | RD, WR, PE, PCTL to CLK\textsubscript{i} Hold | 0  | 0  | ns | 1 |
| 14 | TKVCH  | RD, WR, PCTL to CLK\textsubscript{i} Setup | 20 | 30 | ns | 2 |

**ASYNCHRONOUS \( \mu \)P PORT INTERFACE**

| 15 | TRWVCL | RD, WR to CLK\textsubscript{i} Setup | 20 | 30 | ns | 8, 9 |
| 16 | TRWL   | RD, WR Pulse Width | 2TCLCL + 30 | 2TCLCL + 40 | ns |
| 17 | TRWLPEV| PE from RD, WR\textsubscript{i} Delay | CFS = 1 | TCLCL-20, TCLCL-30 | ns |
| 18 | TRWLPEX| PE to RD, WR\textsubscript{i} Hold | 2TCLCL + 30 | 2TCLCL + 40 | ns |
| 19 | TRWLPTV| PCTL from RD, WR\textsubscript{i} Delay | TCLCL-30 | TCLCL-40 | ns |
| 20 | TRWLPTX| PCTL to RD, WR\textsubscript{i} Hold | 2TCLCL + 30 | 2TCLCL + 40 | ns |
| 21 | TRWLPTV| PCTL from RD, WR\textsubscript{i} Delay | 2TCLCL-20 | 2TCLCL-30 | ns |
| 22 | TRWLPTX| PCTL to RD, WR\textsubscript{i} Hold | 3TCLCL + 30 | 3TCLCL + 40 | ns |
## A.C. CHARACTERISTICS (Continued)

### RAM INTERFACE

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>8207-16, -8 (FFS=1)</th>
<th>8207-12, -6 (FFS=0)</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>TAVCL</td>
<td>AL, AH, BS to CLK</td>
<td>35+1ASR</td>
<td>45+1ASR</td>
<td>ns</td>
<td>10</td>
</tr>
<tr>
<td>24</td>
<td>TCLAX</td>
<td>AL, AH, BS to CLK Hold</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>TCLLN</td>
<td>LEN from CLK Delay</td>
<td>35</td>
<td>45</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>TCLRSI</td>
<td>RASI from CLK Delay</td>
<td>35</td>
<td>45</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>TRCD</td>
<td>RAS to CAS Delay</td>
<td>CFS=1</td>
<td>TCLCL-25</td>
<td>ns</td>
<td>1, 14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CFS=0</td>
<td>TCLCL/2-25</td>
<td>75</td>
<td>11, 14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCLCL-2-30</td>
<td>70</td>
<td>12, 14</td>
</tr>
<tr>
<td>28</td>
<td>TCLRSH</td>
<td>RASI from CLK Delay</td>
<td>50</td>
<td>60</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>TRAH</td>
<td>CASI from CLK Delay</td>
<td>TLCCL2:10</td>
<td>TCLA/2:15</td>
<td>ns</td>
<td>1, 13, 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TLCCL/4:10</td>
<td>TCLA/4:15</td>
<td>35</td>
<td>11, 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCLA/L:70</td>
<td>12</td>
<td>12, 15</td>
</tr>
<tr>
<td>30</td>
<td>TASR</td>
<td>Row A0 to CAS Hold</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>TASC</td>
<td>Column A0 to CASI Setup</td>
<td>CFS=1</td>
<td>0</td>
<td>ns</td>
<td>13, 19, 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CFS=0</td>
<td>5</td>
<td>ns</td>
<td>13, 19, 20</td>
</tr>
<tr>
<td>32</td>
<td>TCAH</td>
<td>Column A0 to CAS Hold</td>
<td>TLCCL/4</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>TCLCSL</td>
<td>CASI from CLK Delay</td>
<td>TLCCL/4</td>
<td>TLCCL/1.8</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+30</td>
<td>+53</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>TCLCSH</td>
<td>CASI from CLK Delay</td>
<td>35</td>
<td>40</td>
<td>ns</td>
<td>1</td>
</tr>
<tr>
<td>35</td>
<td>TCLWS</td>
<td>WE from CLK Delay</td>
<td>50</td>
<td>60</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>TCLKL</td>
<td>XACKI from CLK Delay</td>
<td>35</td>
<td>45</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>TRWLTKH</td>
<td>XACKI from RDT, WRt Delay</td>
<td>50</td>
<td>55</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>TCLKKL</td>
<td>AA1I from CLK Delay</td>
<td>35</td>
<td>45</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>TCLAKH</td>
<td>AA1I from CLK Delay</td>
<td>50</td>
<td>60</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>TCLDL</td>
<td>DBM from CLK Delay</td>
<td>35</td>
<td>45</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### ECC INTERFACE

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>2TCLCL-40</th>
<th>2TCLCL-50</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>TWRLTIV</td>
<td>FWR from WRi Delay</td>
<td>CFS=1</td>
<td>2TCLCL-40</td>
<td>ns</td>
<td>1, 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CFS=0</td>
<td>TCLCL+ TCLA-40</td>
<td>50</td>
<td>2, 22</td>
</tr>
<tr>
<td>43</td>
<td>TFVCL</td>
<td>FWR to CLK Setup</td>
<td>40</td>
<td>50</td>
<td>ns</td>
<td>23</td>
</tr>
<tr>
<td>44</td>
<td>TCLFX</td>
<td>FWR to CLK Hold</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td>24</td>
</tr>
<tr>
<td>45</td>
<td>TEVCL</td>
<td>ERROR to CLK Setup</td>
<td>20</td>
<td>25</td>
<td>ns</td>
<td>25, 26</td>
</tr>
<tr>
<td>46</td>
<td>TLEX</td>
<td>ERROR to CLK Hold</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>TCLRL</td>
<td>R/W from CLK Delay</td>
<td>40</td>
<td>45</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>TCLRH</td>
<td>R/W from CLK Delay</td>
<td>50</td>
<td>60</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>TCEVCL</td>
<td>CE to CLK Setup</td>
<td>20</td>
<td>25</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>TCLCEX</td>
<td>CE to CLK Hold</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>TCLS</td>
<td>ESTB from CLK Delay</td>
<td>35</td>
<td>45</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

---

ECC INTERFACE

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>2TCLCL-40</th>
<th>2TCLCL-50</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>TWRLTIV</td>
<td>FWR from WRi Delay</td>
<td>CFS=1</td>
<td>2TCLCL-40</td>
<td>ns</td>
<td>1, 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CFS=0</td>
<td>TCLCL+ TCLA-40</td>
<td>50</td>
<td>2, 22</td>
</tr>
</tbody>
</table>

---

ECC INTERFACE

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>2TCLCL-40</th>
<th>2TCLCL-50</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>TWRLTIV</td>
<td>FWR from WRi Delay</td>
<td>CFS=1</td>
<td>2TCLCL-40</td>
<td>ns</td>
<td>1, 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CFS=0</td>
<td>TCLCL+ TCLA-40</td>
<td>50</td>
<td>2, 22</td>
</tr>
</tbody>
</table>

---

ECC INTERFACE

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>2TCLCL-40</th>
<th>2TCLCL-50</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>TWRLTIV</td>
<td>FWR from WRi Delay</td>
<td>CFS=1</td>
<td>2TCLCL-40</td>
<td>ns</td>
<td>1, 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CFS=0</td>
<td>TCLCL+ TCLA-40</td>
<td>50</td>
<td>2, 22</td>
</tr>
</tbody>
</table>
A.C. CHARACTERISTICS (Continued)

PORT SWITCHING AND LOCK

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>8207-16, -8 (FFS=1)</th>
<th>8207-12, -6 (FFS=0)</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>TCLMV</td>
<td>MUX from CLK1 Delay</td>
<td>Min. 45</td>
<td>Max. 55</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>TCLPNV</td>
<td>PSEN from CLK1 Delay</td>
<td>Min. TCL 60</td>
<td>Max. TCL 60</td>
<td>ns</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Min. TCL +35</td>
<td>Max. TCL +35</td>
<td>ns</td>
<td>29</td>
</tr>
<tr>
<td>54</td>
<td>TCLPSV</td>
<td>PSEL from CLK1</td>
<td>Min. 35</td>
<td>Max. 45</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>TLKVCL</td>
<td>LOCK to CLK1 Setup</td>
<td>Min. 30</td>
<td>Max. 40</td>
<td>ns</td>
<td>30, 31</td>
</tr>
<tr>
<td>56</td>
<td>TCLKK</td>
<td>LOCK to CLK1 Hold</td>
<td>Min. 10</td>
<td>Max. 10</td>
<td>ns</td>
<td>30, 31</td>
</tr>
<tr>
<td>57</td>
<td>TRWLLKV</td>
<td>LOCK from RD, WR Delay</td>
<td>Min. 2TCLCL-30</td>
<td>Max. 2TCLCL-40</td>
<td>ns</td>
<td>31, 32</td>
</tr>
<tr>
<td>58</td>
<td>TRWHLXK</td>
<td>LOCK to RDT, WR Hold</td>
<td>Min. 3TCLCL+30</td>
<td>Max. 3TCLCL+40</td>
<td>ns</td>
<td>31, 32</td>
</tr>
</tbody>
</table>

REFRESH REQUEST

| 59   | TRFVCL   | RFRQ to CLK1 Setup            | Min. 20            | Max. 25             | ns    |       |
| 60   | TCRFX    | RFRQ to CLK1 Hold             | Min. 10            | Max. 10             | ns    |       |
| 61   | TRFRH    | Failsafe RFRQ Pulse Width      | Min. TCLCL+30      | Max. TCLCL+40       | ns    | 33    |
| 62   | TRFXCL   | Single RFRQ Inactive to CLK1  | Min. 20            | Max. 30             | ns    | 34    |
| 63   | TBRFH    | Burst RFRQ Pulse Width         | Min. 2TCLCL+30     | Max. 2TCLCL+40      | ns    | 33    |

NOTES:
1. Specification when programmed in the Fast Cycle processor mode (APX 286 mode)
2. Specification when programmed in the Slow Cycle processor mode (APX 186 mode)
3. tR and tF are referenced from the 3.5V and 1.8V levels
4. RESET is internally synchronized to CLK Hence a set-up time is required only to guarantee its recognition at a particular clock edge
5. The first programming bit (POD) is also sampled by RESET going low
6. TLPDIX is guaranteed if programming data is shifted using PCLK
7. WZ is issued only in ECC mode
8. TRWVLCL is not required for an asynchronous command except to guarantee its recognition at a particular clock edge
9. Valid when programmed in either Fast or Slow Cycle mode
10. tASR is a user-specified parameter and its value should be added accordingly to tAVCL
11. When programmed in Slow Cycle mode and 125 ns < tCLCL < 200 ns
12. When programmed in Slow Cycle mode and 200 ns < tCLCL
13. Specification for Test Load conditions
14. tRCD (actual) = tRCD (specification) + 0.06 (AC_CL) - 0.06 (AC_CA) (These are first order approximations)
15. tRIAH (actual) = tRIAH (specification) + 0.06 (AC_CL) - 0.022 (AC_CA) (These are first order approximations)
16. tLCK
17. tASR (actual) = tASR (specification) + 0.06 (AC_CL) - 0.025 (AC_CA) (These are first order approximations)
18. tASC (actual) = tASC (specification) + 0.06 (AC_CL) - 0.025 (AC_CA) (These are first order approximations)
19. tASC is a function of clock frequency and thus varies with changes in frequency. A minimum value is specified
20. See 8207 DRAM Interface Tables 14-18
21. TRWVLCL is defined for both synchronous and asynchronous FWR. In systems in which FWR is decoded directly from the address inputs to the 8207 TCLFV is automatically guaranteed by TCLAV
22. TFRVCL is defined for synchronous FWR
23. TCLFV is defined for both synchronous and asynchronous FWR. In systems in which FWR is decoded directly from the address inputs to the 8207 TCLFV is automatically guaranteed by TCLAV
24. ERROR and CE are set-up to CLK1 in fast cycle mode and CLK1 in slow cycle mode
25. ERROR is set-up to the same edge as WE is referenced to, in RMW cycles
26. CE is set-up to the same edge as WE is referenced to, in RMW cycles
27. Synchronous mode only. Must arrive by the clock falling edge after the clock edge which recognizes the command in order to be effective
28. LOCK must be held active for the entire period the opposite port must be locked out. One clock after the release of LOCK the opposite port will be able to obtain access to memory
29. Asynchronous mode only. In this mode a synchronizer stage is used internally in the 8207 to synchronize up LOCK TRWVLXV and TRWVLXK are only required for guaranteeing that LOCK will be recognized for the requesting port, but these parameters are not required for correct 8207 operation
30. TFRFH and TBRFH pertain to asynchronous operation only
31. Single RFRQ cannot be supplied asynchronously
WAVEFORMS
Clock and Programming Timings

RAM Warm-up and Memory Initialization Cycles

NOTES:
1. When in non-ECC mode or in ECC mode with the TM2 programming bit on, there are no initialization cycles, when in ECC mode with TM2 off, the dummy cycles are followed by initialization cycles.
2. The present example assumes a RAS four clocks long.
WAVEFORMS (Continued)
Synchronous Port Interface

NOTE:
Actual transitions are programmable. Refer to Tables 12 and 13.
WAVEFORMS (Continued)
Asynchronous Port Interface

CLK

FAST/SLOW CYCLE
RD, WR

SLOW CYCLE
PCTL

FAST CYCLE
PCTL (INHIBIT)

FAST CYCLE
INTERNAL INHIBIT

INTERNAL CYCLE REQUEST
WAVEFORMS (Continued)
RAM Interface Timing
ECC and Non-ECC Mode

NOTE:
Actual transitions are programmable. Refer to Tables 12 and 13.
WAVEFORMS (Continued)
Port Switching and Lock Timing

NOTE:
Transients during MUX switching.

Refresh Request Timing

NOTE:
Transients during MUX switching.
WAVEFORMS (Continued)
ECC Interface Timing

NOTE:
1. This parameter is set-up to the falling edge of clock, as shown, for fast cycle configurations. It is set-up to the rising edge of clock if in slow cycle configurations. Table 13A shows which clock and clock edge these signals are set-up in the R/W L column.
2. CE is set-up to the same edge as WE is referenced to in RMW cycles.
## CONFIGURATION TIMING CHARTS

The timing charts that follow are based on 8 basic system configurations where the 8207 operates. Tables 10 and 11 give a description of non-ECC and ECC system configurations based on the 8207’s PD0, PD3, PD4, PD10 and PD11 programming bits.

### Table 10. Non-ECC System Configurations

<table>
<thead>
<tr>
<th>Timing Conf.</th>
<th>CFS(PD3)</th>
<th>RFS(PD4)</th>
<th>EXT(PD10)</th>
<th>FF5(PD11)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>iAPX286(0)</td>
<td>FAST RAM(0)</td>
<td>NOT EXT(0)</td>
<td>10 MHZ(1)</td>
</tr>
<tr>
<td>C0</td>
<td>iAPX286(0)</td>
<td>FAST RAM(0)</td>
<td>EXT(1)</td>
<td>10 MHZ(1)</td>
</tr>
<tr>
<td>C0</td>
<td>iAPX286(0)</td>
<td>SLOW RAM(1)</td>
<td>NOT EXT(0)</td>
<td>10 MHZ(1)</td>
</tr>
<tr>
<td>C0</td>
<td>iAPX286(0)</td>
<td>SLOW RAM(1)</td>
<td>EXT(1)</td>
<td>10 MHZ(1)</td>
</tr>
<tr>
<td>C0</td>
<td>iAPX286(0)</td>
<td>FAST RAM(0)</td>
<td>NOT EXT(0)</td>
<td>16 MHZ(0)</td>
</tr>
<tr>
<td>C1</td>
<td>iAPX286(0)</td>
<td>SLOW RAM(1)</td>
<td>NOT EXT(0)</td>
<td>16 MHZ(0)</td>
</tr>
<tr>
<td>C1</td>
<td>iAPX286(0)</td>
<td>FAST RAM(0)</td>
<td>EXT(1)</td>
<td>16 MHZ(0)</td>
</tr>
<tr>
<td>C2</td>
<td>iAPX286(0)</td>
<td>SLOW RAM(1)</td>
<td>EXT(1)</td>
<td>16 MHZ(0)</td>
</tr>
<tr>
<td>C3</td>
<td>iAPX186(1)</td>
<td>FAST RAM(0)</td>
<td>NOT EXT(0)</td>
<td>8 MHZ(0)</td>
</tr>
<tr>
<td>C3</td>
<td>iAPX186(1)</td>
<td>SLOW RAM(1)</td>
<td>NOT EXT(0)</td>
<td>8 MHZ(0)</td>
</tr>
<tr>
<td>C3</td>
<td>iAPX186(1)</td>
<td>FAST RAM(0)</td>
<td>EXT(1)</td>
<td>8 MHZ(0)</td>
</tr>
<tr>
<td>C3</td>
<td>iAPX186(1)</td>
<td>FAST RAM(0)</td>
<td>NOT EXT(0)</td>
<td>5 MHZ(1)</td>
</tr>
<tr>
<td>C3</td>
<td>iAPX186(1)</td>
<td>SLOW RAM(1)</td>
<td>NOT EXT(0)</td>
<td>5 MHZ(1)</td>
</tr>
<tr>
<td>C3</td>
<td>iAPX186(1)</td>
<td>SLOW RAM(1)</td>
<td>EXT(1)</td>
<td>5 MHZ(1)</td>
</tr>
<tr>
<td>C4</td>
<td>iAPX186(1)</td>
<td>SLOW RAM(1)</td>
<td>EXT(1)</td>
<td>8 MHZ(0)</td>
</tr>
</tbody>
</table>

### Table 11. ECC System Configurations

<table>
<thead>
<tr>
<th>Timing Conf.</th>
<th>CFS(PD3)</th>
<th>RFS(PD4)</th>
<th>EXT(PD10)</th>
<th>FF5(PD11)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>iAPX286(1)</td>
<td>SLOW RAM(0)</td>
<td>M/S EDCU(0)</td>
<td>10 MHZ(0)</td>
</tr>
<tr>
<td>C0</td>
<td>iAPX286(1)</td>
<td>SLOW RAM(0)</td>
<td>M EDCU(1)</td>
<td>10 MHZ(0)</td>
</tr>
<tr>
<td>C0</td>
<td>iAPX286(1)</td>
<td>FAST RAM(1)</td>
<td>M/S EDCU(0)</td>
<td>10 MHZ(0)</td>
</tr>
<tr>
<td>C0</td>
<td>iAPX286(1)</td>
<td>FAST RAM(1)</td>
<td>M EDCU(1)</td>
<td>10 MHZ(0)</td>
</tr>
<tr>
<td>C0</td>
<td>iAPX286(1)</td>
<td>FAST RAM(1)</td>
<td>M EDCU(1)</td>
<td>16 MHZ(1)</td>
</tr>
<tr>
<td>C1</td>
<td>iAPX286(1)</td>
<td>SLOW RAM(0)</td>
<td>M EDCU(1)</td>
<td>16 MHZ(1)</td>
</tr>
<tr>
<td>C2</td>
<td>iAPX286(1)</td>
<td>FAST RAM(1)</td>
<td>M/S EDCU(0)</td>
<td>16 MHZ(1)</td>
</tr>
<tr>
<td>C3</td>
<td>iAPX286(1)</td>
<td>SLOW RAM(0)</td>
<td>M/S EDCU(0)</td>
<td>16 MHZ(1)</td>
</tr>
<tr>
<td>C4</td>
<td>iAPX186(0)</td>
<td>SLOW RAM(0)</td>
<td>M/S EDCU(0)</td>
<td>5 MHZ(0)</td>
</tr>
<tr>
<td>C4</td>
<td>iAPX186(0)</td>
<td>FAST RAM(1)</td>
<td>M/S EDCU(0)</td>
<td>5 MHZ(0)</td>
</tr>
<tr>
<td>C4</td>
<td>iAPX186(0)</td>
<td>SLOW RAM(0)</td>
<td>M EDCU(1)</td>
<td>8 MHZ(1)</td>
</tr>
<tr>
<td>C4</td>
<td>iAPX186(0)</td>
<td>FAST RAM(1)</td>
<td>M EDCU(1)</td>
<td>8 MHZ(1)</td>
</tr>
<tr>
<td>C5</td>
<td>iAPX186(0)</td>
<td>SLOW RAM(0)</td>
<td>M/S EDCU(0)</td>
<td>8 MHZ(1)</td>
</tr>
<tr>
<td>C5</td>
<td>iAPX186(0)</td>
<td>FAST RAM(1)</td>
<td>M/S EDCU(0)</td>
<td>8 MHZ(1)</td>
</tr>
<tr>
<td>C6</td>
<td>iAPX186(0)</td>
<td>SLOW RAM(0)</td>
<td>M EDCU(1)</td>
<td>5 MHZ(0)</td>
</tr>
<tr>
<td>C6</td>
<td>iAPX186(0)</td>
<td>FAST RAM(1)</td>
<td>M EDCU(1)</td>
<td>5 MHZ(0)</td>
</tr>
</tbody>
</table>
Using the Timing Charts

The notation used to indicate which clock edge triggers an output transition is "n" or "nl", where "n" is the number of clock periods that have passed since clock 0, the reference clock, and "l" refers to rising edge and "r" to falling edge. A clock period is defined as the interval from a clock falling edge to the following falling edge. Clock edges are defined as shown below.

The clock edges which trigger transitions on each 8207 output are tabulated in Table 12 for non-ECC mode, and Table 13 for ECC mode. "H" refers to the high-going transition, and "L" to low-going transition; "V" refers to valid, and "V" to non-valid.

Clock 0 is defined as the clock in which the 8207 begins a memory cycle, either as a result of a port request which has just arrived, or of a port request which was stored previously but could not be serviced at the time of its arrival because the 8207 was performing another memory cycle. Clock 0 may be identified externally by the leading edge of RAS, which is always triggered on 01.

Notes for interpreting the timing charts.

1. **PSEL** - valid is given as the latest time it can occur. It is entirely possible for PSEL to become valid before the time given. In a refresh cycle, PSEL can switch as defined in the chart, but it has no bearing on the refresh cycle itself, but only on a subsequent cycle for one of the external ports.

2. **LEN** - low is given as the latest time it can occur. LEN is only activated by port A configured in Fast Cycle iAPX286 mode, and thus it is not activated by a refresh cycle, although it may be activated by port A during a refresh cycle.

3. **ADDRESS** - col is the time column address becomes valid.

4. In non-ECC mode the CAS, EAACK, LAACK and XACK outputs are not issued during refresh.

5. In ECC mode there are really seven types of cycles: Read without error, read with error, full write, partial write without error, partial write with error, refresh without error, and refresh with error. These cycles may be derived from the timing chart as follows:

   - A. Read without error: Use row marked 'RD, RF'.
   - B. Read with error: Use row marked 'RMW', except for EAACK and LAACK, which should be taken from 'RD, RF'. If the error is uncorrectable, WE will not be issued.
   - C. Full write: Use row marked 'WR'.
   - D. Partial write without error: Use row marked 'RMW', except that DBM and ESTB will not be issued.
   - E. Partial write with error: Use row marked 'RMW', except that DBM will not be issued. If the error is uncorrectable, WE will not be issued.
   - F. Refresh without error: Use row marked 'RD, RF', except that ESTB, EAACK, LAACK, and XACK will not be issued.
   - G. Refresh with error: Use row marked 'RMW' except that EAACK, LAACK, ESTB, and XACK will not be issued. If the error is uncorrectable WE will not be issued.

6. **XACK** - high is reset asynchronously by command going inactive and not by a clock edge.

7. **MUX** - valid is given as the latest time it can occur.
### Table 12 A. Timing Chart — Non-ECC Mode

<table>
<thead>
<tr>
<th>Cn</th>
<th>CYCLE</th>
<th>H</th>
<th>L</th>
<th>V</th>
<th>V</th>
<th>L</th>
<th>H</th>
<th>L</th>
<th>H</th>
<th>L</th>
<th>H</th>
<th>H</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>RD, RF</td>
<td>0†</td>
<td>3†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>3†</td>
<td>1†</td>
<td>4†</td>
</tr>
<tr>
<td>WR</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>5†</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>5†</td>
<td>1†</td>
<td>5†</td>
<td>2†</td>
<td>5†</td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>RD, RF</td>
<td>0†</td>
<td>5†</td>
<td>0†</td>
<td>6†</td>
<td>0†</td>
<td>6†</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>4†</td>
<td>1†</td>
<td>6†</td>
</tr>
<tr>
<td>WR</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>5†</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>5†</td>
<td>1†</td>
<td>5†</td>
<td>2†</td>
<td>5†</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>RD, RF</td>
<td>0†</td>
<td>5†</td>
<td>0†</td>
<td>6†</td>
<td>0†</td>
<td>6†</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>4†</td>
<td>1†</td>
<td>6†</td>
</tr>
<tr>
<td>WR</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>5†</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>5†</td>
<td>1†</td>
<td>5†</td>
<td>2†</td>
<td>5†</td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>RD, RF</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>3†</td>
<td>0†</td>
<td>3†</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>3†</td>
<td>0†</td>
<td>3†</td>
</tr>
<tr>
<td>WR</td>
<td>0†</td>
<td>3†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>4†</td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>RD, RF</td>
<td>0†</td>
<td>3†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>4†</td>
</tr>
<tr>
<td>WR</td>
<td>0†</td>
<td>3†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>4†</td>
<td>0†</td>
<td>4†</td>
<td></td>
</tr>
</tbody>
</table>

### Table 12 B. Timing Chart — Non-ECC Mode

<table>
<thead>
<tr>
<th>Cn</th>
<th>COL ADDR</th>
<th>COL ADDR</th>
<th>EAACK</th>
<th>LAACK</th>
<th>XACK</th>
<th>MUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>RD, RF</td>
<td>2†</td>
<td>1†</td>
<td>4†</td>
<td>2†</td>
<td>5†</td>
</tr>
<tr>
<td>WR</td>
<td>0†</td>
<td>2†</td>
<td>1†</td>
<td>4†</td>
<td>1†</td>
<td>4†</td>
</tr>
<tr>
<td>C1</td>
<td>RD, RF</td>
<td>3†</td>
<td>2†</td>
<td>5†</td>
<td>2†</td>
<td>5†</td>
</tr>
<tr>
<td>WR</td>
<td>0†</td>
<td>3†</td>
<td>1†</td>
<td>4†</td>
<td>1†</td>
<td>4†</td>
</tr>
<tr>
<td>C2</td>
<td>RD, RF</td>
<td>3†</td>
<td>2†</td>
<td>5†</td>
<td>3†</td>
<td>6†</td>
</tr>
<tr>
<td>WR</td>
<td>0†</td>
<td>3†</td>
<td>1†</td>
<td>4†</td>
<td>1†</td>
<td>4†</td>
</tr>
<tr>
<td>C3</td>
<td>RD, RF</td>
<td>2†</td>
<td>0†</td>
<td>2†</td>
<td>1†</td>
<td>3†</td>
</tr>
<tr>
<td>WR</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>2†</td>
<td>1†</td>
<td>3†</td>
</tr>
<tr>
<td>C4</td>
<td>RD, RF</td>
<td>2†</td>
<td>1†</td>
<td>3†</td>
<td>1†</td>
<td>3†</td>
</tr>
<tr>
<td>WR</td>
<td>0†</td>
<td>2†</td>
<td>0†</td>
<td>2†</td>
<td>1†</td>
<td>3†</td>
</tr>
<tr>
<td>Cn</td>
<td>CYCLE</td>
<td>H</td>
<td>L</td>
<td>V</td>
<td>V</td>
<td>L</td>
</tr>
<tr>
<td>----</td>
<td>-------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C0</td>
<td>RD, RF</td>
<td>01</td>
<td>51</td>
<td>01</td>
<td>61</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>WR</td>
<td>01</td>
<td>51</td>
<td>01</td>
<td>61</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>RMW</td>
<td>01</td>
<td>81</td>
<td>01</td>
<td>91</td>
<td>01</td>
</tr>
<tr>
<td>C1</td>
<td>WR</td>
<td>01</td>
<td>51</td>
<td>01</td>
<td>61</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>RMW</td>
<td>01</td>
<td>81</td>
<td>01</td>
<td>91</td>
<td>01</td>
</tr>
<tr>
<td>C2</td>
<td>RD, RF</td>
<td>01</td>
<td>61</td>
<td>01</td>
<td>71</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>WR</td>
<td>01</td>
<td>61</td>
<td>01</td>
<td>71</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>RMW</td>
<td>01</td>
<td>101</td>
<td>01</td>
<td>111</td>
<td>01</td>
</tr>
<tr>
<td>C3</td>
<td>RD, RF</td>
<td>01</td>
<td>61</td>
<td>01</td>
<td>71</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>WR</td>
<td>01</td>
<td>61</td>
<td>01</td>
<td>71</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>RMW</td>
<td>01</td>
<td>101</td>
<td>01</td>
<td>111</td>
<td>01</td>
</tr>
<tr>
<td>C4</td>
<td>RD, RF</td>
<td>01</td>
<td>31</td>
<td>01</td>
<td>41</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>WR</td>
<td>01</td>
<td>41</td>
<td>01</td>
<td>51</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>RMW</td>
<td>01</td>
<td>61</td>
<td>01</td>
<td>71</td>
<td>01</td>
</tr>
<tr>
<td>C5</td>
<td>RD, RF</td>
<td>01</td>
<td>31</td>
<td>01</td>
<td>41</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>WR</td>
<td>01</td>
<td>41</td>
<td>01</td>
<td>51</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>RMW</td>
<td>01</td>
<td>61</td>
<td>01</td>
<td>71</td>
<td>01</td>
</tr>
<tr>
<td>C6</td>
<td>RD, RF</td>
<td>01</td>
<td>31</td>
<td>01</td>
<td>41</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>WR</td>
<td>01</td>
<td>31</td>
<td>01</td>
<td>41</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>RMW</td>
<td>01</td>
<td>41</td>
<td>01</td>
<td>51</td>
<td>01</td>
</tr>
<tr>
<td>Cycle</td>
<td>WR</td>
<td>RD</td>
<td>RF</td>
<td>WR</td>
<td>RD</td>
<td>RF</td>
</tr>
<tr>
<td>-------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 13 B: Timing Chart - ECC Mode
Several DRAM parameters, but not all, are a direct function of 8207 timings, and the equations for these parameters are given in the following tables. The following is a list of those DRAM parameters which have NOT been included in the following tables, with an explanation for their exclusion.

READ, WRITE, READ-MODIFY-WRITE & REFRESH CYCLES
- tRAC: response parameter.
- tCAC: response parameter.
- tREF: See "Refresh Period Options".
- tCRP: must be met only if CAS-only cycles, which do not occur with 8207, exist.
- tRAH: See "A.C. Characteristics".
- tRCD: See "A.C. Characteristics".
- tASC: See "A.C. Characteristics".
- tASR: See "A.C. Characteristics".
- tOFF: response parameter.

READ & REFRESH CYCLES
- tRCH: WE always goes active after CAS goes active, hence tRCH is guaranteed by tCPN.

WRITE CYCLE
- tRC: guaranteed by tRWC.
- tRAS: guaranteed by tRRW.
- tCAS: guaranteed by tCRW.
- tWCS: WE always activated after CAS is activated, except in memory initialization, hence tWCS is always negative (this is important for RMW only) except in memory initialization; in memory initialization tWCS is positive and has several clocks of margin.
- tDS: system-dependent parameter.
- tDH: system-dependent parameter.
- tDHR: system-dependent parameter.

READ-MODIFY-WRITE CYCLE
- tRWD: don’t care in 8207 write cycles, but tabulated for 8207 RMW cycles.
- tCWD: don’t care in 8207 write cycles, but tabulated for 8207 RMW cycles.

Table 14. Non-ECC Mode – RD, RF Cycles

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Fast Cycle Configurations</th>
<th>Slow Cycle Configurations</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C0</td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>tRP</td>
<td>3TCLCL-T26</td>
<td>4TCLCL-T26</td>
<td>4TCLCL-T26</td>
</tr>
<tr>
<td>tCPN</td>
<td>3TCLCL-T35</td>
<td>3TCLCL-T35</td>
<td>3TCLCL-T35</td>
</tr>
<tr>
<td>tRSH</td>
<td>2TCLCL-T34</td>
<td>3TCLCL-T34</td>
<td>3TCLCL-T34</td>
</tr>
<tr>
<td>tCSH</td>
<td>4TCLCL-T26</td>
<td>6TCLCL-T26</td>
<td>6TCLCL-T26</td>
</tr>
<tr>
<td>tCAH</td>
<td>TCLCL-T34</td>
<td>2TCLCL-T34</td>
<td>2TCLCL-T34</td>
</tr>
<tr>
<td>tAR</td>
<td>2TCLCL-T26</td>
<td>3TCLCL-T26</td>
<td>3TCLCL-T26</td>
</tr>
<tr>
<td>tT</td>
<td>3/30</td>
<td>3/30</td>
<td>3/30</td>
</tr>
<tr>
<td>tRC</td>
<td>6TCLCL</td>
<td>8TCLCL</td>
<td>8TCLCL</td>
</tr>
<tr>
<td>tRAS</td>
<td>3TCLCL-T26</td>
<td>4TCLCL-T26</td>
<td>4TCLCL-T26</td>
</tr>
<tr>
<td>tCAS</td>
<td>3TCLCL-T34</td>
<td>5TCLCL-T34</td>
<td>5TCLCL-T34</td>
</tr>
<tr>
<td>tRCS</td>
<td>2TCLCL-TCL-T36-TBUF</td>
<td>2TCLCL-TCL-T36-TBUF</td>
<td>2TCLCL-TCL-T36-TBUF</td>
</tr>
</tbody>
</table>
### Table 15. Non-ECC Mode – WR Cycle

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Fast Cycle Configurations</th>
<th>Slow Cycle Configurations</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>tRP</td>
<td>3TCLCL-T26</td>
<td>2TCLCL-T26</td>
<td>1</td>
</tr>
<tr>
<td>tCPN</td>
<td>4TCLCL-T35</td>
<td>2.5TCLCL-T35</td>
<td>1</td>
</tr>
<tr>
<td>tRSH</td>
<td>4TCLCL-T34</td>
<td>4TCLCL-T34</td>
<td>1</td>
</tr>
<tr>
<td>tCSH</td>
<td>5TCLCL-T26</td>
<td>4TCLCL-T26</td>
<td>1</td>
</tr>
<tr>
<td>tCAH</td>
<td>TCLCL-T34</td>
<td>2TCLCL-T34</td>
<td>1</td>
</tr>
<tr>
<td>tAR</td>
<td>2TCLCL-T26</td>
<td>2TCLCL-T26</td>
<td>1</td>
</tr>
<tr>
<td>tT</td>
<td>3/30</td>
<td>3/30</td>
<td>1</td>
</tr>
<tr>
<td>tRWC</td>
<td>8TCLCL</td>
<td>6TCLCL</td>
<td>1</td>
</tr>
<tr>
<td>tRRW</td>
<td>5TCLCL-T26</td>
<td>4TCLCL-T26</td>
<td>1</td>
</tr>
<tr>
<td>tCRW</td>
<td>4TCLCL-T34</td>
<td>4TCLCL-T34</td>
<td>1</td>
</tr>
<tr>
<td>tWCH</td>
<td>3TCLCL+TCL+T34</td>
<td>3TCLCL+TCL+T34</td>
<td>1, 3</td>
</tr>
<tr>
<td>tWCR</td>
<td>4TCLCL+TCL+T26</td>
<td>3TCLCL+TCL+T26</td>
<td>1, 3</td>
</tr>
<tr>
<td>tWP</td>
<td>2TCLCL+TCL+T36-TBUF</td>
<td>2TCLCL+TCL+T36-TBUF</td>
<td>1</td>
</tr>
<tr>
<td>tRWL</td>
<td>3TCLCL-T36-TBUF-T36-TBUF</td>
<td>3TCLCL-T36-TBUF-T36-TBUF</td>
<td>1</td>
</tr>
<tr>
<td>tCWL</td>
<td>3TCLCL-T36-TBUF-T36-TBUF</td>
<td>3TCLCL-T36-TBUF-T36-TBUF</td>
<td>1</td>
</tr>
</tbody>
</table>
### Table 16 A. ECC Mode — RD, RF Cycles

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Fast Cycle Mode</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C₀</td>
<td>C₁</td>
</tr>
<tr>
<td>tRP</td>
<td>4TCLCL-T26</td>
<td>4TCLCL-T26</td>
</tr>
<tr>
<td>tCPN</td>
<td>3TCLCL-T35</td>
<td>3TCLCL-T35</td>
</tr>
<tr>
<td>tRSH</td>
<td>3TCLCL-T34</td>
<td>3TCLCL-T34</td>
</tr>
<tr>
<td>tCSH</td>
<td>6TCLCL-T26</td>
<td>6TCLCL-T26</td>
</tr>
<tr>
<td>tCAH</td>
<td>TCLCL-T34</td>
<td>2TCLCL-T34</td>
</tr>
<tr>
<td>tAR</td>
<td>2TCLCL-T26</td>
<td>3TCLCL-T26</td>
</tr>
<tr>
<td>tT</td>
<td>3/30</td>
<td>3/30</td>
</tr>
<tr>
<td>tRC</td>
<td>8TCLCL</td>
<td>8TCLCL</td>
</tr>
<tr>
<td>tRAS</td>
<td>4TCLCL-T26</td>
<td>4TCLCL-T26</td>
</tr>
<tr>
<td>tCAS</td>
<td>5TCLCL-T34</td>
<td>5TCLCL-T34</td>
</tr>
<tr>
<td>tRCS</td>
<td>TCLCL-T36</td>
<td>TCLCL-T36</td>
</tr>
</tbody>
</table>

### Table 16 B. ECC Mode — RD, RF Cycles

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Slow Cycle Mode</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C₄</td>
<td>C₅</td>
</tr>
<tr>
<td>tRP</td>
<td>2TCLCL-T26</td>
<td>2TCLCL-T26</td>
</tr>
<tr>
<td>tCPN</td>
<td>1.5TCLCL-T35</td>
<td>1.5TCLCL-T35</td>
</tr>
<tr>
<td>tRSH</td>
<td>3TCLCL-T34</td>
<td>3TCLCL-T34</td>
</tr>
<tr>
<td>tCSH</td>
<td>4TCLCL-T26</td>
<td>4TCLCL-T26</td>
</tr>
<tr>
<td>tCAH</td>
<td>2TCLCL-T34</td>
<td>2TCLCL-T34</td>
</tr>
<tr>
<td>tAR</td>
<td>2TCLCL-T26</td>
<td>2TCLCL-T26</td>
</tr>
<tr>
<td>tT</td>
<td>3/30</td>
<td>3/30</td>
</tr>
<tr>
<td>tRC</td>
<td>5TCLCL</td>
<td>5TCLCL</td>
</tr>
<tr>
<td>tRAS</td>
<td>3TCLCL-T26</td>
<td>3TCLCL-T26</td>
</tr>
<tr>
<td>tCAS</td>
<td>4TCLCL-T34</td>
<td>4TCLCL-T34</td>
</tr>
<tr>
<td>tRCS</td>
<td>0.5TCLCL-T36</td>
<td>0.5TCLCL-T36</td>
</tr>
</tbody>
</table>

-TBUF -TBUF -TBUF
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Fast Cycle Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C₀</td>
</tr>
<tr>
<td>tRP</td>
<td>3TCLCL-T26</td>
</tr>
<tr>
<td>tCPN</td>
<td>4TCLCL-T35</td>
</tr>
<tr>
<td>tRSH</td>
<td>5TCLCL-T34</td>
</tr>
<tr>
<td>tCSH</td>
<td>6TCLCL-T26</td>
</tr>
<tr>
<td>tCAH</td>
<td>TCLCL-T34</td>
</tr>
<tr>
<td>tAR</td>
<td>2TCLCL-T26</td>
</tr>
<tr>
<td>tT</td>
<td>3/30</td>
</tr>
<tr>
<td>tRWC</td>
<td>9TCLCL</td>
</tr>
<tr>
<td>tRRW</td>
<td>6TCLCL-T26</td>
</tr>
<tr>
<td>tCRW</td>
<td>5TCLCL-T34</td>
</tr>
<tr>
<td>tWCH</td>
<td>5TCLCL-T34</td>
</tr>
<tr>
<td>tWCR</td>
<td>6TCLCL-T26</td>
</tr>
<tr>
<td>tWP</td>
<td>3TCLCL-T36</td>
</tr>
<tr>
<td></td>
<td>-TBUF</td>
</tr>
<tr>
<td>tRWL</td>
<td>3TCLCL-T36</td>
</tr>
<tr>
<td></td>
<td>-TBUF</td>
</tr>
<tr>
<td>tCWL</td>
<td>3TCLCL-T36</td>
</tr>
<tr>
<td></td>
<td>-TBUF</td>
</tr>
</tbody>
</table>
Table 17 B. ECC Mode — WR Cycle

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Slow Cycle Mode</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$C_4$</td>
<td>$C_5$</td>
</tr>
<tr>
<td>tRP</td>
<td>2TCLCL–T26</td>
<td>2TCLCL–T26</td>
</tr>
<tr>
<td>tCPN</td>
<td>2.5TCLCL–T35</td>
<td>2.5TCLCL–T35</td>
</tr>
<tr>
<td>tRSH</td>
<td>5TCLCL–T34</td>
<td>5TCLCL–T34</td>
</tr>
<tr>
<td>tCSH</td>
<td>5TCLCL–T26</td>
<td>5TCLCL–T26</td>
</tr>
<tr>
<td>tCAH</td>
<td>2TCLCL–T34</td>
<td>2TCLCL–T34</td>
</tr>
<tr>
<td>tAR</td>
<td>2TCLCL–T26</td>
<td>2TCLCL–T26</td>
</tr>
<tr>
<td>tT</td>
<td>3/30</td>
<td>3/30</td>
</tr>
<tr>
<td>tRWC</td>
<td>7TCLCL</td>
<td>7TCLCL</td>
</tr>
<tr>
<td>tRRW</td>
<td>5TCLCL–T26</td>
<td>5TCLCL–T26</td>
</tr>
<tr>
<td>tCRW</td>
<td>5TCLCL–T34</td>
<td>5TCLCL–T34</td>
</tr>
<tr>
<td>tWCH</td>
<td>5TCLCL–T34</td>
<td>5TCLCL–T34</td>
</tr>
<tr>
<td>tWCR</td>
<td>5TCLCL–T26</td>
<td>5TCLCL–T26</td>
</tr>
<tr>
<td>tWP</td>
<td>3TCLCL–TCL</td>
<td>3TCLCL–TCL</td>
</tr>
<tr>
<td>tRWL</td>
<td>3TCLCL–TCL</td>
<td>3TCLCL–TCL</td>
</tr>
<tr>
<td>tCWL</td>
<td>3TCLCL–TCL</td>
<td>3TCLCL–TCL</td>
</tr>
</tbody>
</table>
### Table 18 A. ECC Mode — RMW

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Fast Cycle Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$C_0$</td>
</tr>
<tr>
<td>tRP</td>
<td>3TCLCL-T26</td>
</tr>
<tr>
<td>tCPN</td>
<td>4TCLCL-T35</td>
</tr>
<tr>
<td>tRSH</td>
<td>8TCLCL-T34</td>
</tr>
<tr>
<td>tCSH</td>
<td>9TCLCL-T26</td>
</tr>
<tr>
<td>tCAH</td>
<td>TCLCL-T34</td>
</tr>
<tr>
<td>tAR</td>
<td>2TCLCL-T26</td>
</tr>
<tr>
<td>tT</td>
<td>3/30</td>
</tr>
<tr>
<td>tRWC</td>
<td>12TCLCL</td>
</tr>
<tr>
<td>tRRW</td>
<td>9TCLCL-T26</td>
</tr>
<tr>
<td>tCRW</td>
<td>8TCLCL-T34</td>
</tr>
<tr>
<td>tRCS</td>
<td>TCLCL-T36</td>
</tr>
<tr>
<td></td>
<td>-TBUF</td>
</tr>
<tr>
<td>tRWD</td>
<td>6TCLCL-T26</td>
</tr>
<tr>
<td>tCWD</td>
<td>5TCLCL-T34</td>
</tr>
<tr>
<td>tWP</td>
<td>3TCLCL-T36</td>
</tr>
<tr>
<td></td>
<td>-TBUF</td>
</tr>
<tr>
<td>tRWL</td>
<td>3TCLCL-T36</td>
</tr>
<tr>
<td></td>
<td>-TBUF</td>
</tr>
<tr>
<td>tCWL</td>
<td>3TCLCL-T36</td>
</tr>
<tr>
<td></td>
<td>-TBUF</td>
</tr>
</tbody>
</table>
### Table 18 B. ECC Mode — RMW

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Slow Cycle Mode</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>tRP</td>
<td>2TCLCL-T26</td>
<td>2TCLCL-T26</td>
<td>2TCLCL-T26</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tCPN</td>
<td>2.5TCLCL-T35</td>
<td>2.5TCLCL-T35</td>
<td>2.5TCLCL-T35</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tRSH</td>
<td>7TCLCL-T34</td>
<td>7TCLCL-T34</td>
<td>5TCLCL-T34</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tCSH</td>
<td>7TCLCL-T26</td>
<td>7TCLCL-T26</td>
<td>5TCLCL-T26</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tCAH</td>
<td>2TCLCL-T34</td>
<td>2TCLCL-T34</td>
<td>2TCLCL-T34</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tAR</td>
<td>2TCLCL-T26</td>
<td>2TCLCL-T26</td>
<td>2TCLCL-T26</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tT</td>
<td>3/30</td>
<td>3/30</td>
<td>3/30</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>tRWC</td>
<td>9TCLCL</td>
<td>9TCLCL</td>
<td>7TCLCL</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tRRW</td>
<td>7TCLCL-T26</td>
<td>7TCLCL-T26</td>
<td>5TCLCL-T26</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tCRW</td>
<td>7TCLCL-T34</td>
<td>7TCLCL-T34</td>
<td>5TCLCL-T34</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tRCS</td>
<td>0.5TCLCL-T36</td>
<td>0.5TCLCL-T36</td>
<td>0.5TCLCL-T36</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-TBUF</td>
<td>-TBUF</td>
<td>-TBUF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tRWD</td>
<td>4TCLCL+TCL</td>
<td>4TCLCL+TCL</td>
<td>2TCLCL+TCL</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-T26</td>
<td>-T26</td>
<td>-T26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tCWD</td>
<td>4TCLCL+TCL</td>
<td>4TCLCL+TCL</td>
<td>2TCLCL+TCL</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-T34</td>
<td>-T34</td>
<td>-T34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWP</td>
<td>3TCLCL-TCL</td>
<td>3TCLCL-TCL</td>
<td>3TCLCL-TCL</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-T36-TBUF</td>
<td>-T36-TBUF</td>
<td>-T36-TBUF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tRWL</td>
<td>3TCLCL-TCL</td>
<td>3TCLCL-TCL</td>
<td>3TCLCL-TCL</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-T36-TBUF</td>
<td>-T36-TBUF</td>
<td>-T36-TBUF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tCWL</td>
<td>3TCLCL-TCL</td>
<td>3TCLCL-TCL</td>
<td>3TCLCL-TCL</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-T36-TBUF</td>
<td>-T36-TBUF</td>
<td>-T36-TBUF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**

1. Minimum
2. Value on right is maximum; value on left is minimum.
3. Applies to the eight warm-up cycles during initialization only.
4. Applies to the eight warm-up cycles and to the memory initialization cycles during initialization only.
5. TP = TCLCL
   T26 = TCLRSL
   T34 = TCLCSL
   T35 = TCLCSH
   T36 = TCLW
   TBUF = TTL Buffer delay
8208
DYNAMIC RAM CONTROLLER

- 0 Wait State, 8 MHz iAPX 186/188, iAPX 286 and iAPX 86/88 Interface
- Provides all Signals necessary to Control 64K (2164A) and 256K Dynamic RAMs
- Supports Synchronous or Asynchronous Microprocessor Interfaces
- Automatic RAM Warm-up

The Intel 8208 Dynamic RAM Controller is a high performance, systems oriented, Dynamic RAM controller that is designed to easily interface 64K and 256K Dynamic RAMs to Intel and other microcomputer systems. The 8208 is designed to easily interface to the iAPX 186, iAPX 188, iAPX 86, and the iAPX 88 by strapping the programming pin to logic 0.

Figure 1. Block Diagram and Pinout Diagram
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL0</td>
<td>5</td>
<td>I</td>
<td>ADDRESS LOW: These lower order address inputs are used to generate the row address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AL1</td>
<td>4</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL2</td>
<td>3</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL3</td>
<td>2</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL4</td>
<td>1</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL5</td>
<td>47</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL6</td>
<td>46</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL7</td>
<td>45</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AL8</td>
<td>44</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td>6</td>
<td>I</td>
<td>BANK SELECT: This input is used to select one of the two banks of the dynamic RAM array as defined by the program-bit RB.</td>
</tr>
<tr>
<td>A00</td>
<td>7</td>
<td>O</td>
<td>ADDRESS OUTPUTS: These outputs are designed to provide the row and column addresses, of either the CPU or the refresh counter, to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers. However, they typically need series resistors to match impedances.</td>
</tr>
<tr>
<td>A01</td>
<td>8</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A02</td>
<td>9</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A03</td>
<td>10</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A04</td>
<td>11</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A05</td>
<td>13</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A06</td>
<td>14</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A07</td>
<td>15</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A08</td>
<td>16</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>VSS</td>
<td>12</td>
<td>I</td>
<td>GROUND</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td></td>
<td>36</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>RAS0</td>
<td>19</td>
<td>O</td>
<td>ROW ADDRESS STROBE: These outputs are used by the dynamic RAM array to latch the row address, present on the A00-8 pins. These outputs are selected by the BS pin as programmed by program-bit RB. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>RAS1</td>
<td>18</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>CAS0</td>
<td>21</td>
<td>O</td>
<td>COLUMN ADDRESS STROBE: These outputs are used by the dynamic RAM array to latch the column address, present on the A00-8 pins. These outputs are selected by the BS pin as programmed by program-bit RB. These outputs drive the dynamic RAM array directly and need no external drivers.</td>
</tr>
<tr>
<td>CAS1</td>
<td>20</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>23</td>
<td>I</td>
<td>RESET: This active high signal causes all internal counters to be reset and upon release of RESET, data appearing at the PDI pin is clocked-in by the PCLK output. The states of the PDI, PCTL and RFREQ pins are sampled by RESET going inactive and are used to program the 8208. An 8 cycle dynamic RAM warm-up is performed after clocking PDI bits into the 8208.</td>
</tr>
<tr>
<td>WE/</td>
<td>25</td>
<td>O</td>
<td>WRITE ENABLE/PROGRAMMING CLOCK: Immediately after a RESET this pin becomes PCLK and is used to clock serial programming data into the PDI pin. After the 8208 is programmed this active high signal provides the dynamic RAM array the write enable input for a write operation.</td>
</tr>
<tr>
<td>PCLK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCC</td>
<td>24</td>
<td>I</td>
<td>POWER: +5 Volts.</td>
</tr>
<tr>
<td></td>
<td>48</td>
<td>I</td>
<td>POWER: +5 Volts.</td>
</tr>
<tr>
<td>AACK/</td>
<td>26</td>
<td>O</td>
<td>ADVANCE ACKNOWLEDGE/TRANSFER ACKNOWLEDGE: When the X programming bit is set to logic 0 this pin is AACK and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the S program-bit for synchronous or asynchronous operation. The S programming bit determines whether this strobe will be early or late. If another dynamic RAM cycle is in progress at the time of the new request, the XACK is delayed. When the X programming bit is set to logic 1 this pin is XACK and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle. XACK is a MULTIBUS compatible signal.</td>
</tr>
<tr>
<td>XACK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCTL</td>
<td>27</td>
<td>I</td>
<td>PORT CONTROL: This pin is sampled on the falling edge of RESET. It configures the 8208 to accept command inputs or processor status inputs. If PCTL is low after RESET the 8208 is programmed to accept bus command inputs. If PCTL is high after RESET the 8208 is programmed to accept status inputs from IAPX 86 or IAPX 186 type processors. The S2 status line should be connected to this input if programmed to accept IAPX 86 or IAPX 186 status inputs. When programmed to accept bus commands it should be tied low or it may be connected to INHIBIT when operating with MULTIBUS.</td>
</tr>
<tr>
<td>PE</td>
<td>28</td>
<td>I</td>
<td>PORT ENABLE: This pin serves to enable a RAM cycle request. It is generally decoded from the address bus.</td>
</tr>
</tbody>
</table>
### Table 1. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>WR</td>
<td>29</td>
<td>I</td>
<td>WRITE: This pin is the write memory request command input. This input also directly accepts the S0 status line from Intel processors.</td>
</tr>
<tr>
<td>RD</td>
<td>30</td>
<td>I</td>
<td>READ: This pin is the read memory request command input. This input also directly accepts the S1 status line from Intel processors.</td>
</tr>
<tr>
<td>CLK</td>
<td>31</td>
<td>I</td>
<td>CLOCK: This input provides the basic timing for sequencing the internal logic.</td>
</tr>
<tr>
<td>RFRQ</td>
<td>32</td>
<td>I</td>
<td>REFRESH REQUEST: This input is sampled on the falling edge of RESET. If RFRQ is high at RESET then the 8208 is programmed for internal-refresh request or external-refresh request with failsafe protection. If RFRQ is low at RESET then the 8208 is programmed for external-refresh without failsafe protection or burst-refresh. Once programmed the RFRQ pin accepts signals to start an external-refresh with failsafe protection or external-refresh without failsafe protection or a burst-refresh.</td>
</tr>
<tr>
<td>PDI</td>
<td>33</td>
<td>I</td>
<td>PROGRAM DATA INPUT: This input is sampled by RESET going low. It programs the various user selectable options in the 8208. The PCLK pin shifts programming data into the PDI input from an external shift register. This pin may be strapped low to a default IAPX 186 (PDI=Low) mode configuration.</td>
</tr>
<tr>
<td>AH0</td>
<td>43</td>
<td>I</td>
<td>ADDRESS HIGH: These higher order address inputs are used to generate the column address for the internal address multiplexer.</td>
</tr>
<tr>
<td>AH1</td>
<td>42</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH2</td>
<td>41</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH3</td>
<td>40</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH4</td>
<td>39</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH5</td>
<td>38</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH6</td>
<td>37</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH7</td>
<td>35</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>AH8</td>
<td>34</td>
<td>I</td>
<td></td>
</tr>
</tbody>
</table>
GENERAL DESCRIPTION

The Intel 8208 Dynamic RAM Controller is a microcomputer peripheral device which provides the necessary signals to address, refresh and directly drive 64K and 256K dynamic RAMs.

The 8208 supports several microprocessor interface options including synchronous and asynchronous operations for iAPX 86, iAPX 186, iAPX 188 and MULTIBUS.

FUNCTIONAL DESCRIPTION

Processor Interface

The 8208 has control circuitry capable of supporting one of several possible bus structures. The 8208 may be programmed to run synchronous or asynchronous to the processor clock. (See Synchronous/Asynchronous Mode) The 8208 has been optimized to run synchronously with Intel's iAPX 86, iAPX 88, iAPX 186 and iAPX 188. When the 8208 is programmed to run in asynchronous mode, the 8208 inserts the necessary synchronization circuitry for the RD, WR, PE, and PCTL inputs.

The 8208 achieves high performance (i.e. no wait states) by decoding the status lines directly from the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. The 8208 can also be programmed to receive read or write MULTIBUS commands or commands from a bus controller. (See Status/Command Mode)

The 8208 may be programmed to accept the clock of the iAPX 86, iAPX 88, iAPX 186 or 188. The 8208 adjusts its internal timing to allow for different clock frequencies of these microprocessors. (See Microprocessor Clock Frequency Option)

Figure 2 shows the different processor interfaces to the 8208 using the synchronous or asynchronous mode and status or command interface.
Dynamic RAM Interface

The 8208 is capable of addressing 64K and 256K dynamic RAMs. Figure 3 shows the connection of the processor address bus to the 8208 using the different RAMs. The 8208 directly supports the 2164A RAM family or any RAM with similar timing requirements and responses.

If not all RAM banks are occupied, the 8208 reassigns the RAS and CAS strobos to allow using wider data words without increasing the loading on the RAS and CAS drivers. Table 2 shows the bank selection decoding and the horizontal word expansion, including RAS and CAS assignments. For example, if only one RAM bank is occupied, then the two RAS and CAS strobos are activated with the same timing.

Table 2. Bank Selection Decoding and Word Expansion

<table>
<thead>
<tr>
<th>Program Bit RB</th>
<th>Bank Input BS</th>
<th>8208 RAS/CAS Pair Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>RAS&lt;sub&gt;0&lt;/sub&gt;, CAS&lt;sub&gt;0&lt;/sub&gt; to Bank 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Illegal</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>RAS&lt;sub&gt;0&lt;/sub&gt;, CAS&lt;sub&gt;0&lt;/sub&gt; to Bank 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>RAS&lt;sub&gt;1&lt;/sub&gt;, CAS&lt;sub&gt;1&lt;/sub&gt; to Bank 1</td>
</tr>
</tbody>
</table>

Program bit RB is not used to check the bank select input BS. The system design must protect from accesses to "illegal", non-existent banks of memory by deactivating the PE input when addressing an "illegal", non-existent bank of memory.

The 8208 adjusts and optimizes internal timings for either the fast or slow RAMs as programmed. (See RAM Speed Option)

Memory Initialization

After programming, the 8208 performs eight RAM "wake-up" cycles to prepare the dynamic RAM for proper device operation (during "warm-up" some RAM interface parameters may not be met, this should cause no harm to the dynamic RAM array).

Refresh

The 8208 provides an internal refresh interval counter, and a refresh address counter to allow the 8208 to refresh memory. The 8208 will refresh 128 rows every 2 milliseconds or 256 rows every 4 milliseconds, which allows all RAM refresh options to be supported. In addition, there exists the ability to refresh 256 row address locations every 2 milliseconds via the Refresh Period programming option.

The 8208 may be programmed for any of five different refresh options: Internal refresh only, External refresh with failsafe protection, External refresh without failsafe protection, Burst Refresh modes, or no refresh. (See Refresh Options)
It is possible to decrease the refresh time interval by 10% 20% or 30%. This option allows the 8208 to compensate for reduced clock frequencies. Note that an additional 5% interval shortening is built-in in all refresh interval options to compensate for clock variations and non-immediate response to the internally generated refresh request. (See Refresh Period Options)

External Refresh Requests after RESET

External refresh requests are not recognized by the 8208 until after it is finished programming and preparing memory for access. Memory preparation includes 8 RAM cycles to prepare and ensure proper dynamic RAM operation. The time it takes for the 8208 to recognize a request is shown below.

eq. 8208 System Response: TRESP = TPROG + TPREP
where: TPROG = (40) (TCLCL) which is programming time
TPREP = (8) (32) (TCLCL) which is the RAM warm-up time
if TCLCL = 125 ns then TRESP = 37 us

Reset

RESET is an asynchronous input, the falling edge of which is used by the 8208 to directly sample the logic levels of the PCTL, RFRQ, and PDI inputs. The internally synchronized falling edge of reset is used to begin programming operations (shifting in the contents of the external shift register, if needed, into the PDI input).

Differentiated reset is unnecessary when the default synchronization programming is used. (S=0)

Until programming is complete the 8208 registers but does not respond to command or status inputs. A simple means of preventing commands or status from occurring during this period is to differentiate the system reset pulse to obtain a smaller reset pulse for the 8208. The total time of the 8208 reset pulse and the 8208 programming time must be less than the time before the first command the CPU issues in systems that alter the default port synchronization programming bit (default is synchronous interface).

The differentiated reset pulse would be shorter than the system reset pulse by at least the programming period required by the 8208. The differentiated reset pulse first resets the 8208, and system reset would reset the rest of the system. While the rest of the system is still in reset, the 8208 completes its programming. Figure 4 illustrates a circuit to accomplish this task.

Within four clocks after RESET goes active, all the 8208 outputs will go high, except for A00-2, which will go low.

OPERATIONAL DESCRIPTION

Programming the 8208

The 8208 is programmed after reset. On the falling edge of RESET, the logic states of several input pins are latched internally. The falling edge of RESET actually performs the latching, which means that the logic levels on these inputs must be stable prior to that time. The inputs whose logic levels are latched at the end of reset are the PCTL, RFRQ, and PDI pins.

Status/Command Mode

The processor port of the 8208 is configured by the states of the PCTL pin. Which interface is selected depends on the state of the PCTL pin at the end of reset. If PCTL is high at the end of reset, the 80186 Status interface is selected; if it is low, then the MULTIBUS or Command interface is selected.

There exist two interface configurations, one for MULTIBUS memory commands, which is called the Command interface, and one for 8086, 8088, 80186 or 80188 status, called the 80186 Status interface. The Command interface also directly interfaces to the command lines of the bus controllers for the 8086, 8088.
The 80186 Status interface allows direct decoding of the status lines for the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. Table 3 shows how the status lines are decoded. Microprocessor bus controller read or write commands or MULTIBUS commands can also be directed to the 8208 when in Command mode.

Table 3. 8208 Response

<table>
<thead>
<tr>
<th>8208 Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCTL</td>
<td>RD</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Refresh Options

Immediately after system reset, the state of the REFREQ input pin is examined. If REFREQ is high, the 8208 provides the user with the choice between self-refresh and user-generated refresh with failsafe protection. Failsafe protection guarantees that if the user does not come back with another refresh request before the internal refresh interval counter times out, a refresh request will be automatically generated. If the REFREQ pin is low immediately after a reset, then the user has the choice of a single external refresh cycle without failsafe, burst refresh or no refresh.

Internal Refresh Only

For the 8208 to generate internal refresh requests, it is necessary only to strap the REFREQ input pin high.

External Refresh with Failsafe

To allow user-generated refresh requests with failsafe protection, it is necessary to hold the REFREQ input high until after reset. Thereafter, a low-to-high transition on this input causes a refresh request to be generated and the internal refresh interval counter to be reset. A high-to-low transition has no effect on the 8208. A refresh request is not recognized until a previous request has been serviced.

External Refresh without Failsafe

To generate single external refresh requests without failsafe protection, it is necessary to hold REFREQ low until after reset. Thereafter, bringing REFREQ high for one clock period will cause a refresh request to be generated. A refresh request is not recognized until a previous request has been serviced.

Burst Refresh

Burst refresh is implemented through the same procedure as a single external refresh without failsafe (i.e., REFREQ is kept low until after reset). Thereafter, bringing REFREQ high for at least two clock periods will cause a burst of up to 128 row address locations to be refreshed. Any refresh request is not recognized until a previous request has been serviced (i.e. burst is completed).

No Refresh

It is necessary to hold REFREQ low until after reset. This is the same as programming External Refresh without Failsafe. No refresh is accomplished by keeping REFREQ low.

Option Program Data Word

The program data word consists of 9 program data bits, P00-P08. If the first program data bit, P00 is set to logic 0, the 8208 is configured to support iAPX 186, 188, 86, or 88 systems. The remaining bits, P01-P08, may then be programmed to optimize a selected system configuration. A default of all zeros in the remaining program bits optimizes the 8208 timing for 8 MHz Intel CPUs using 150 nS (or faster) dynamic RAMs with no performance penalty. Figure 5 shows the various options that can be programmed into the 8208.

Using an External Shift Register

The 8208 may be programmed by using an external shift register with asynchronous load capability such as a 74LS165. The reset pulse serves to parallel load the shift register and the 8208 supplies the clocking signal (PCLK) to shift the data into the PDI programming pin. Figure 6 shows a sample circuit diagram of an external shift register circuit.

Serial data is shifted into the 8208 via the PDI pin (33), and clock is provided by the WE/PCLK pin (23), which generates a total of 9 clock pulses. After programming is complete, data appearing at the input of the PDI pin is ignored. WE/PCLK is a dual function pin.
During programming, it serves to clock the external shift register, and after programming is completed, it reverts to the write enable RAM control output pin. As the pin changes state to provide the write enable signal to the dynamic RAM array, it continues to clock the shift register. This does not present a problem because data at the PDI pin is ignored after programming. Figure 7 illustrates the timing requirements of the shift register.

<table>
<thead>
<tr>
<th>PD8</th>
<th>PD7</th>
<th>PD0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>FFS</td>
<td>PIL</td>
</tr>
<tr>
<td></td>
<td>C10</td>
<td>C11</td>
</tr>
<tr>
<td></td>
<td>RB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>S</td>
</tr>
</tbody>
</table>

Default Programming Options

After reset, the 8208 serially shifts in a program data word via the PDI pin. This pin may be strapped low, or connected to an external shift register. Strapping PDI low causes the 8208 to default to the iAPX 186 system configuration. Table 4 shows the characteristics of the default configuration. If further system flexibility is needed, one external shift register, like a 74LS165, can be used to tailor the 8208 to its operating environment. Figure 8 illustrates an iAPX 186 and 8208 system.

Table 4. Programming, PDI Pin Tied to Ground.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous 80186 interface</td>
<td></td>
</tr>
<tr>
<td>2 RAM banks occupied</td>
<td></td>
</tr>
<tr>
<td>Fast processor clock frequency (8 MHz)</td>
<td></td>
</tr>
<tr>
<td>Fast RAM (Note 1)</td>
<td></td>
</tr>
<tr>
<td>Refresh interval uses 118 clocks</td>
<td></td>
</tr>
<tr>
<td>128 row refresh in 2 ms; 256 row refresh in 4 ms</td>
<td></td>
</tr>
<tr>
<td>Advanced ACK strobe</td>
<td></td>
</tr>
</tbody>
</table>

NOTE:
1. For iAPX 186 systems either slow or fast (150 or 100 ns) RAMS are ok to use.

Figure 5. Program Data Word

Figure 6. Timing Illustrating External Shift Register Requirements for Programming the 8208.

NOTES:
- TRTVCL — Reset is an asynchronous input, if reset occurs before TRTVCL, then it is guaranteed to be recognized.
- TPGVCL — Minimum PDI valid time prior to reset going low.
- TCLPC — MUX/PCLK delay.
- TLOAD — Asynchronous load data propagation delay.
Figure 7. External Shift Register Interface

Figure 8. 8208 Interface to an 80186
Synchronous/Asynchronous Mode
(S program bit)

The 8208 may be independently configured to accept synchronous or asynchronous commands (RD, WR, PCTL) and Port Enable (PE) via the S program bit. The state of the S programming bit determines whether the interface is synchronous or asynchronous.

While the 8208 may be configured with either the 80186 Status or Command (MULTIBUS) interface in the Synchronous mode, certain restrictions exist in the Asynchronous mode. An Asynchronous-Command interface using the control lines of the MULTIBUS is supported, and an Asynchronous-80186 Status interface using the status lines of the 80186 is supported, with the use of TTL gates as illustrated in Figure 2. In the 80186 case, the TTL gates are needed to guarantee that status does not appear at the 8208's inputs too much before address, so that a cycle would start before address was valid.

Microprocessor Clock Cycle Option
(CFS and FFS program bits)

The 8208 is programmed to interface with microprocessors with "slow cycle" timing like the 8086, 8088, 80186, and 80188 cycle timing. The CFS bit configures the microprocessor interface to accept signals from this microprocessor group. The CFS programming bit must be programmed to logic 0.

The FFS option is used to select the speed of the microprocessor clock. Table 5 shows the various microprocessor clock frequency options that can be programmed. The external clock frequency must be programmed so that the failsafe refresh repetition circuitry can adjust its internal timing accordingly to produce a refresh request as programmed.

RAM Speed Option (RFS program bit)

Table 5. Microprocessor Clock Frequency Options.

<table>
<thead>
<tr>
<th>Program Bits</th>
<th>Processor</th>
<th>Clock Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFS</td>
<td>FFS</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>iAPX 86,88,186</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>iAPX 86,88,186</td>
</tr>
</tbody>
</table>

The 8208 refreshes with either 128 rows every 2 milliseconds or the 256 rows every 4 milliseconds. This translates to one refresh cycle being executed approximately once every 15.6 microseconds. This rate can be changed to 256 rows every 2 milliseconds or a refresh approximately once every 7.8 microseconds via the Period Long/Short, program bit PLS, programming option.

The Count Interval 0 (C10) and Count Interval 1 (C11) programming options allow the rate at which refresh requests are generated to be increased in order to permit refresh requests to be generated close to the 15.6 or 7.8 microsecond period when the 8208 is operating at reduced frequencies. The interval between refreshes is decreased by 0%, 10%, 20%, or 30% as a function of how the count interval bits are programmed. A 5% guardband is built-in to allow for any clock frequency variations. Table 6 shows the refresh period options available.

Table 6. Refresh Count Interval Table

<table>
<thead>
<tr>
<th>Ref. Period (μS)</th>
<th>CFS</th>
<th>PLS</th>
<th>FFS</th>
<th>Count Interval C11, C10 (8208 Clock Periods)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00 (0%)</td>
</tr>
<tr>
<td>15.6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>118</td>
</tr>
<tr>
<td>7.8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>59</td>
</tr>
<tr>
<td>15.6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>74</td>
</tr>
<tr>
<td>7.8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37</td>
</tr>
</tbody>
</table>
The numbers tabulated under Count Interval represent the number of clock periods between internal refresh requests. The percentages in parentheses represent the decrease in the interval between refresh requests. Note that all intervals have a built-in 5% (approximately) safety factor to compensate for minor clock frequency deviations and non-immediate response to internal refresh requests.

Processor Timing

In order to run without wait states, the AACK must be used and connected to the SRDY input of the appropriate bus controller. AACK is issued relative to a point within the RAM cycle and has no fixed relationship to the processor’s request. The timing is such, however, that the processor will run without wait states, barring refresh cycles, and bank precharge. In slow cycle, fast RAM configurations (8086, 80186), AACK is issued on the same clock cycle that issues RAS.

Port Enable (PE) set-up time requirements depend on whether the 8208 is configured for synchronous or asynchronous, fast or slow cycle operation. In a synchronous fast cycle configuration, PE is required to be set-up to the same clock edge as the commands. If PE is true (low), a RAM cycle is started; if not, the cycle is aborted.

In asynchronous operation, PE is required to be set-up to the same clock edge as the internally synchronized status or commands. Externally, this allows the internal synchronization delay to be added to the status (or command) -to-PE delay time, thus allowing for more external decode time than is available in synchronous operation.

The minimum synchronization delay is the additional amount that PE must be held valid. If PE is not held valid for the maximum synchronization delay time, it is possible that PE will go invalid prior to the status or command being synchronized. In such a case the 8208 aborts the cycle. If a memory cycle intended for the 8208 is aborted, then no acknowledge (AACK or XACK) is issued and the processor locks up in endless wait states.

Memory Acknowledge (AACK, XACK)

Two type of memory acknowledge signals are supplied by the 8208. They are the Advanced Acknowledge strobe (AACK) and the Transfer Acknowledge strobe (XACK). The S programming bit optimizes AACK for synchronous operation (“early” AACK) or asynchronous operation (“late” AACK). Both the early and late AACK strobes are two clocks long. The XACK strobe is asserted when data is valid (for reads) or when data may be removed (for writes) and meets the MULTIBUS requirements. XACK is removed asynchronously by the command going inactive.

Since in a asynchronous operation the 8208 removes read data before late AACK or XACK is recognized by the CPU, the user must provide for data latching in the system until the CPU reads the data. In synchronous operation data latching is unnecessary, since the 8208 will not remove data until the CPU has read it.

If the X programming bit is high, the strobe is configured as XACK, while if the bit is low, the strobe is configured as AACK.

Data will always be valid a fixed time after the occurrence of the advanced acknowledge. Thus, the advanced acknowledge may also serve as a RAM cycle timing indicator.

General System Considerations

The RASO, CASO, and A00-8 output buffers are designed to directly drive the heavy capacitive loads associated with dynamic RAM arrays. To keep the RAM driver outputs from ringing excessively in the system environment it is necessary to match the output impedance with the RAM array by using series resistors. Each application may have different impedance characteristics and may require different series resistance values. The series resistance values should be determined for each application.

Using the Timing Charts

The notation used to indicate which clock edge triggers an output transition is “n+1” or “n+1”, where “n” is the number of clock periods that have passed since clock 0, the reference clock, and “+1” refers to rising edge and “+1” to falling edge. A clock period is defined as the interval from a clock falling edge to the following falling edge. Clock edges are defined as shown below.

![Clock Edges Diagram](attachment://clock_edges_diagram.png)
The clock edges which trigger transitions on each 8208 output are tabulated in Table 7. "H" refers to the high-going transition, and "L" to low-going transition; "V" refers to valid, and "V" to non-valid. Clock 0 is defined as the clock in which the 8208 begins a memory cycle, either as a result of a port request which has just arrived, or of a port request which was stored previously but could not be serviced at the time of its arrival because the 8208 was performing another memory cycle. Clock 0 is identified externally by the leading edge of RAS, which is always triggered on 0.

**NOTES FOR INTERPRETING THE TIMING CHARTS:**
1. COLUMN ADDRESS is the time column address becomes valid.
2. The CAS, EAACK, LAACK and XACK outputs are not issued during refresh.
3. XACK - high is reset asynchronously by command going inactive and not be a clock edge.

### Table 7. Timing Chart.

<table>
<thead>
<tr>
<th>RAS</th>
<th>COLUMN ADDRESS</th>
<th>CAS</th>
<th>WE</th>
<th>EAACK</th>
<th>LAACK</th>
<th>XACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>CYCLE</td>
<td>L</td>
<td>H</td>
<td>V</td>
<td>L</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>RD, RF</td>
<td>0↑</td>
<td>2↓</td>
<td>0↑</td>
<td>2↓</td>
<td>0↑</td>
<td>3↓</td>
</tr>
<tr>
<td>WR</td>
<td>0↓</td>
<td>4↓</td>
<td>0↓</td>
<td>3↓</td>
<td>1↑</td>
<td>4↓</td>
</tr>
</tbody>
</table>

### 8208–DRAM Interface Parameter Equations

Several DRAM parameters, but not all, are a direct function of 8208 timings, and the equations for these parameters are given in the following tables. The following is a list of those DRAM parameters which have NOT been included in the following tables, with an explanation for their exclusion.

**READ, WRITE REFRESH CYCLES**

- tRAC: response parameter.
- tCAC: response parameter.
- tREF: See “Refresh Period Options”.
- tCRP: must be met only if CAS-only cycles, which do not occur with 8208, exist.
- tRAH: See “A.C. Characteristics”
- tRCD: See “A.C. Characteristics”
- tASC: See “A.C. Characteristics”
- tASR: See “A.C. Characteristics”
- tOFF: response parameter.

**WRITE CYCLE**

- tDS: system-dependent parameter.
- tDH: system-dependent parameter.
- tDHR: system-dependent parameter.

**NOTES:**
1. Minimum.
2. Value on right is maximum; value on left is minimum.
3. Applies to the eight warm-up cycles during initialization only.
4. T22=TCLRSL
   T29=TCLCSL
   T30=TLCWSH
   T31=TCLW
   TBUF=TTL buffer delay

### Table 8. RD, RF & WR Cycles

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Rd, RF Cycles</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>tRP</td>
<td>2TCLCL-T22</td>
<td>1</td>
</tr>
<tr>
<td>tCPN</td>
<td>2.5TCLCL-T30</td>
<td>1</td>
</tr>
<tr>
<td>tRSH</td>
<td>3TCLCL-T29</td>
<td>1</td>
</tr>
<tr>
<td>tCSH</td>
<td>3TCLCL-T22</td>
<td>1</td>
</tr>
<tr>
<td>tCAH</td>
<td>2TCLCL-T29</td>
<td>1</td>
</tr>
<tr>
<td>tAR</td>
<td>2TCLCL-T22</td>
<td>1</td>
</tr>
<tr>
<td>tT</td>
<td>3/30</td>
<td>2</td>
</tr>
<tr>
<td>tRC</td>
<td>4TCLCL</td>
<td>1</td>
</tr>
<tr>
<td>tRAS</td>
<td>2TCLCL-T22</td>
<td>1</td>
</tr>
<tr>
<td>tCAS</td>
<td>3TCLCL-T29</td>
<td>1</td>
</tr>
<tr>
<td>tRCS</td>
<td>1.5TCLCL-TCL-T31-TBUF</td>
<td>1</td>
</tr>
<tr>
<td>tRCH</td>
<td>0.5TCLCL-T29</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>WR Cycles</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>tRP</td>
<td>2TCLCL-T22</td>
<td>1</td>
</tr>
<tr>
<td>tCPN</td>
<td>2.5TCLCL-T30</td>
<td>1</td>
</tr>
<tr>
<td>tRSH</td>
<td>3TCLCL-T29</td>
<td>1</td>
</tr>
<tr>
<td>tCSH</td>
<td>4TCLCL-T22</td>
<td>1</td>
</tr>
<tr>
<td>tCAH</td>
<td>2TCLCL-T29</td>
<td>1</td>
</tr>
<tr>
<td>tAR</td>
<td>3TCLCL-T22</td>
<td>1</td>
</tr>
<tr>
<td>tT</td>
<td>3/30</td>
<td>2</td>
</tr>
<tr>
<td>tRC</td>
<td>6TCLCL</td>
<td>1</td>
</tr>
<tr>
<td>tRAS</td>
<td>4TCLCL-T22</td>
<td>1</td>
</tr>
<tr>
<td>tCAS</td>
<td>3TCLCL-T29</td>
<td>1</td>
</tr>
<tr>
<td>tWCH</td>
<td>3TCLCL-T29</td>
<td>1, 3</td>
</tr>
<tr>
<td>tWCR</td>
<td>4TCLCL-T22</td>
<td>1, 3</td>
</tr>
<tr>
<td>tWP</td>
<td>4TCLCL-T31-TBUF</td>
<td>1</td>
</tr>
<tr>
<td>tWL</td>
<td>4TCLCL-T31-TBUF</td>
<td>1</td>
</tr>
<tr>
<td>tCW</td>
<td>4TCLCL-T31-TBUF</td>
<td>1</td>
</tr>
<tr>
<td>tWCS</td>
<td>TCLCL-T31-TBUF</td>
<td>1</td>
</tr>
</tbody>
</table>
ABSOLUTE MAXIMUM RATINGS

Ambient Temperature
- Under Bias .................... 0°C to +70°C
- Storage Temperature .......... -65°C to +150°C
- Voltage On Any Pin With Respect to Ground ............ -5V to +7V
- Power Dissipation ............. 1.55 Watts

NOTICE: Stress above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A=0°C to +70°C, V_CC = 5.0V ±10%, V_SS=GND)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>+0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VH</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>V CC + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td></td>
<td></td>
<td>0.45</td>
<td>V</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td></td>
<td></td>
<td>2.4</td>
<td>V</td>
</tr>
<tr>
<td>VROL</td>
<td>RAM Output Low Voltage</td>
<td></td>
<td></td>
<td>0.45</td>
<td>V</td>
</tr>
<tr>
<td>VRDH</td>
<td>RAM Output High Voltage</td>
<td></td>
<td></td>
<td>2.6</td>
<td>V</td>
</tr>
<tr>
<td>ICC</td>
<td>Supply Current</td>
<td></td>
<td></td>
<td>280</td>
<td>mA</td>
</tr>
<tr>
<td>IIL</td>
<td>Input Leakage Current</td>
<td></td>
<td></td>
<td>+10</td>
<td>µA</td>
</tr>
<tr>
<td>VCL</td>
<td>Clock Input Low Voltage</td>
<td></td>
<td></td>
<td>-0.5</td>
<td>+0.6</td>
</tr>
<tr>
<td>VCH</td>
<td>Clock Input High Voltage</td>
<td></td>
<td></td>
<td>3.8</td>
<td>V CC + 0.5</td>
</tr>
<tr>
<td>CIN</td>
<td>Input Capacitance</td>
<td></td>
<td></td>
<td>20</td>
<td>pF</td>
</tr>
</tbody>
</table>

NOTES:
1. IOL = 8 mA and IOH = -0.2 mA (typically IOL = 10 mA and IOH = -0.25 mA)

A.C. Testing Load Circuit

A.C. Testing Input, Output Waveform

A.C. Testing inputs (except clock) are driven at 2.4V for a logic "1" and 0.45V for a logic "0" (clock is driven at 4.0V and 0.45V for logic "1" and "0" respectively). Timing measurements are made at 2.0V, 2.4V for logic "1" and 0.8 V for logic "0".

5-110 230734-002
A.C. CHARACTERISTICS (TA=0°C to +70°C, VCC = +5V ±10%, VSS = GND)

Measurements made with respect to RAS0, 1, CAS0, 1, A00-8 are at 2.4V and 0.8V. All other pins are measured at 2.0V and 0.8V. All times in nsec unless otherwise indicated. AC testing done with specified test load.

**CLOCK AND PROGRAMMING**

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>8208 (FFS=1)</th>
<th>8208-6 (FFS=0)</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>—</td>
<td>tF</td>
<td>Clock Rise Time</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>—</td>
<td>tF</td>
<td>Clock Fall Time</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>TCLCL</td>
<td>Clock Period</td>
<td>125</td>
<td>500</td>
<td>167</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>TCL</td>
<td>Clock Low Time</td>
<td>TCLCL/2-12</td>
<td>TCLCL/2-12</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TCH</td>
<td>Clock High Time</td>
<td>TCLCL/3+2</td>
<td>TCLCL/3+2</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>TRTVC</td>
<td>Reset to CLKi Setup</td>
<td>40</td>
<td>65</td>
<td>40</td>
<td>65</td>
</tr>
<tr>
<td>5</td>
<td>TRTH</td>
<td>Reset Pulse Width</td>
<td>4 TCLCL</td>
<td>4 TCLCL</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TPGVRTL</td>
<td>PCTL, PDI, RFF to RESETi Setup</td>
<td>TCLCL</td>
<td>TCLCL</td>
<td>ns</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>TRTLPGX</td>
<td>PCTL, RFF to RESETi Hold</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>TCLPC</td>
<td>PCLK from CLKi Delay</td>
<td>45</td>
<td>65</td>
<td>45</td>
<td>65</td>
</tr>
<tr>
<td>9</td>
<td>TPDVC</td>
<td>PDI to CLKi Setup</td>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>TCLPDX</td>
<td>PDI to CLKi Hold</td>
<td>40</td>
<td>65</td>
<td>40</td>
<td>65</td>
</tr>
</tbody>
</table>

**SYNCHRONOUS /P INTERFACE**

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>8208 (FFS=1)</th>
<th>8208-6 (FFS=0)</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>11</td>
<td>TKVCH</td>
<td>RD, WR, PCTL to CLKi Setup</td>
<td>20</td>
<td>30</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>TCLXX</td>
<td>RD, WR, PE, PCTL to CLKi Hold</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>TPEVCL</td>
<td>PE to CLKi Setup</td>
<td>30</td>
<td>50</td>
<td>30</td>
<td>50</td>
</tr>
</tbody>
</table>

**ASYNCHRONOUS /P INTERFACE**

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>8208 (FFS=1)</th>
<th>8208-6 (FFS=0)</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>14</td>
<td>TRWVCL</td>
<td>RD, WR to CLKi Setup</td>
<td>20</td>
<td>30</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>15</td>
<td>TRWL</td>
<td>RD, WR Pulse Width</td>
<td>2TCLCL +30</td>
<td>2TCLCL +50</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>TRWLPEV</td>
<td>PE from RD, WRi Delay</td>
<td>TCLCL-30</td>
<td>TCLCL-50</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>TRWLPEX</td>
<td>PE to RD, WRi Hold</td>
<td>2TCLCL +30</td>
<td>2TCLCL +50</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>TRWLPT</td>
<td>PCTL from RD, WRi Delay</td>
<td>TCLCL-30</td>
<td>TCLCL-50</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>TRWLPTX</td>
<td>PCTL to RD, WRi Hold</td>
<td>2TCLCL +30</td>
<td>2TCLCL +50</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

**RAM INTERFACE**

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>8208 (FFS=1)</th>
<th>8208-6 (FFS=0)</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>20</td>
<td>TAVCL</td>
<td>AL, AH, BS to CLKi Setup</td>
<td>45 + tASR</td>
<td>55 + tASR</td>
<td>ns</td>
<td>4</td>
</tr>
<tr>
<td>21</td>
<td>TCLAX</td>
<td>AL, AH, BS to CLKi Hold</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>TCLRSL</td>
<td>RASi from CLKi Delay</td>
<td>35</td>
<td>55</td>
<td>35</td>
<td>55</td>
</tr>
<tr>
<td>23</td>
<td>iRCD</td>
<td>RAS to CAS Delay</td>
<td>TCLCL/2-25</td>
<td>TCLCL/2-40</td>
<td>ns</td>
<td>5,7,8</td>
</tr>
<tr>
<td>24</td>
<td>TCLRSH</td>
<td>RASi from CLKi Delay</td>
<td>50</td>
<td>70</td>
<td>50</td>
<td>70</td>
</tr>
<tr>
<td>25</td>
<td>iASR</td>
<td>Row AO to RASi Setup</td>
<td>4,10</td>
<td>4,10</td>
<td>4,10</td>
<td>4,10</td>
</tr>
<tr>
<td>26</td>
<td>IRAH</td>
<td>Row AO to RASi Hold</td>
<td>TCLKL/4-10</td>
<td>TCLKL/4-15</td>
<td>ns</td>
<td>5,7,9</td>
</tr>
<tr>
<td>27</td>
<td>iASC</td>
<td>Column AO to CASi Setup</td>
<td>TCLKL/4-26</td>
<td>TCLKL/4-26</td>
<td>ns</td>
<td>5,7,11,12</td>
</tr>
<tr>
<td>28</td>
<td>iCAH</td>
<td>Column AO to CAS Hold</td>
<td>(See DRAM Interface Tables)</td>
<td>13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## A.C. CHARACTERISTICS (Continued)

### RAM Interface (Continued)

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>8208</th>
<th>8208-6</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>29</td>
<td>TCLCSL</td>
<td>CASI from CLKi Delay</td>
<td>TCLCL/4 + 30 + 30</td>
<td>TCLCL/1.8 + 53</td>
<td>TCLCL/4 + 30</td>
<td>70 ns</td>
</tr>
<tr>
<td>30</td>
<td>TCLCSH</td>
<td>CASI from CLKi Delay</td>
<td>50</td>
<td>50</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>31</td>
<td>TCLWH</td>
<td>WEI from CLKi Delay</td>
<td>TCLCL/4 + 30 + 30</td>
<td>TCLCL/1.8 + 53</td>
<td>TCLCL/4 + 30</td>
<td>70 ns</td>
</tr>
<tr>
<td>32</td>
<td>TCLWL</td>
<td>WEI from CLKi Delay</td>
<td>35</td>
<td>55</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>33</td>
<td>TCLTKL</td>
<td>XACKi from CLKi Delay</td>
<td>35</td>
<td>55</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>34</td>
<td>TRWLTKH</td>
<td>XACKi from RDi, WRi Delay</td>
<td>50</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>35</td>
<td>TCLAKL</td>
<td>XACKi from CLKi Delay</td>
<td>35</td>
<td>55</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>36</td>
<td>TCLAKH</td>
<td>XACKi from CLKi Delay</td>
<td>50</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
</tbody>
</table>

### REFRESH REQUEST

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Symbol</th>
<th>Parameter</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>TRFVCL</td>
<td>RFRQ to CLKi Setup</td>
<td>20 ns</td>
<td>30</td>
</tr>
<tr>
<td>38</td>
<td>TCLRFX</td>
<td>RFRQ to CLKi Hold</td>
<td>10 ns</td>
<td>10</td>
</tr>
<tr>
<td>39</td>
<td>TFRFH</td>
<td>Failsafe RFRQ Pulse Width</td>
<td>14 ns</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>TRFXCL</td>
<td>Single RFRQ inactive to CLKi Setup</td>
<td>15 ns</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>TBFH</td>
<td>Burst RFRQ Pulse Width</td>
<td>14 ns</td>
<td></td>
</tr>
</tbody>
</table>

### NOTES:

1. RESET is internally synchronized to CLK. Hence a set-up time is required only to guarantee its recognition at a particular clock edge.
2. The first programming bit (PD0) is also sampled by RESET going low. 
3. TCLPDX is guaranteed if programming data is shifted using PCLK.
4. tASR is a user specified parameter and its value should be added accordingly to TAVCL.
5. 125 ns < TCLCL < 200 ns.
6. 200 ns < TCLCL.
7. Specification for Test Load Conditions.
8. $t_{RCD} (actual) = t_{RCD} (specification) + 0.06 (\Delta C_{RAS}) - 0.06 (C_{CAS})$
   where $\Delta C = C_{load} - C_{actual}$ in pF.
9. $t_{RAH} (actual) > t_{RAH} (specification) + 0.022 (\Delta C_{AO})$
   where $\Delta C = C_{load} - C_{actual}$ in pF.
10. $t_{ASR} (actual) = t_{ASR} (specification) + 0.06 (\Delta C_{AO}) - 0.025 (\Delta C_{RAS})$
    where $\Delta C = C_{load} - C_{actual}$ in pF.
11. $t_{ASC} (actual) = t_{ASC} (specification) + 0.06 (\Delta C_{AO}) - 0.025 (\Delta C_{CAS})$
    where $\Delta C = C_{load} - C_{actual}$ in pF.
12. $t_{ASC}$ is a function of clock frequency and thus varies with changes in frequency. A minimum value is specified.
13. See 8208 DRAM Interface Tables.
14. TFRFH and TBFH pertain to asynchronous operation only.
15. Single RFRQ cannot be supplied asynchronously.
WAVEFORMS
Clock and Programming Timings

RAM Warm-up Cycles

Synchronous Port Interface

NOTE:
The present example assumes a RAS four clocks long.
WAVEFORMS (Continued)

Asynchronous Port Interface

Refresh Request Timing
WAVEFORMS (Continued)

RAM Interface Timing

Note: Actual transitions are programmable. See Table 7.
8207 User's Manual

AUGUST 1983
CHAPTER 1
INTRODUCTION

This guide is a supplement to the 8207 Data Sheet and is intended as a design aid and not a stand-alone description of the 8207. The reader should already have read and have a copy of the 8207 Data Sheet, 8206 Error Detection and Correction Unit Data Sheet (EDCU), a microprocessor Data Sheet, or a Multibus bus specification for interfacing to the 8207, and a dynamic RAM Data Sheet.

The Intel 8207 Advanced Dynamic RAM Controller is a high performance, highly integrated device designed to interface 16k, 64k, and 256k dynamic RAMS to Intel microprocessors. The 8207, with the 8206, provides complete control for memory initialization, error correction, and automatic error scrubbing.

The 8207 has several speed selected versions. The -16 and -12 parts are for clock speeds up to 16MHz and 12 MHz in “fast cycle” configurations, and up to 8 MHz and 6 MHz in “slow cycle” configurations. The -8 and -6 parts can only be used in slow cycle configurations and as a result have some relaxed A.C. timings.

NOTE:

(1) The most current Data Sheet is dated July, 1984
(2) All RAM cycle timings and references are based on Intel’s 2164A Dynamic RAMs, APR ’82
CHAPTER 2  
PROGRAMMING THE 8207

The many configurations of bus structures, RAM speeds, and system requirements that the 8207 supports require the 8207 to be programmable. The 8207 will modify its outputs to provide the best performance possible. The 8207 must be told what type of interface the memory commands will arrive on, what type of RAM (speed, refresh rate) is being used, the clock rate, and others.

The 8207 uses two means to be informed of the user’s requirements. It reads in a 16 bit serial program word and examines the logic states on several input pins. The pins that are sampled for a logic level give the user options on the types of refresh and memory command input timing.

Input Pin Options

The three input pins that configure part of the 8207 are: PCTL A, PCTL B, and REFREQ. Let’s examine the options in refresh types the REFREQ pin provides.

Refresh types:

The 8207 gives the user a choice of the following refresh types.

1) Internal Refresh: All refresh cycles are generated internally — based on an internal programmable time.

2) External Refresh with Failsafe: If the external logic does not generate a refresh cycle within the programmed period, the 8207 will.

3) External Refresh - No Failsafe or No Refresh; All refresh cycles are generated at times by the user. This is for systems that cannot tolerate the random delay imposed by refresh (i.e. graphics memory).

4) Burst Refresh: The 8207 generates up to 128 consecutive refresh cycles and must be requested by external logic. Memory requests will be performed when the burst is completed.

The 8207 examines the state of the REFREQ pin when RESET goes inactive. This timing is shown in the “Clock and Programming Timings” waveforms in the Data Sheet.

If REFREQ is sampled active by the falling edge of RESET, the 8207’s internal timer is enabled. The timer’s period is determined by the CIO, CI1, and PLS bits in the program word. External refresh cycles are generated by a low to high transition on the REFREQ input. This transition, besides generating a refresh cycle, also resets the internal timer to zero. Simply tie REFREQ to Vcc if internal refresh is required.

If REFREQ is seen low at the falling edge of RESET, the internal timer is deactivated. All refresh cycles must either be done by external logic or by accessing all RAM (internal) rows within a 2 ms period.

Once the no failsafe option is programmed, the 8207 will generate a burst of up to 128 refresh cycles when the REFREQ input goes from low to high and sampled high for two consecutive clock edges. These cycles are internally counted and the 8207 stops when the refresh address counter reaches the value XX1111111₂ (X = don’t care; see Refresh Counter section). If prior to the burst request the counter is at XX111111₀₂ then only 2 refresh cycles would be generated.
For a single refresh cycle to be generated via external logic, the REFRQ input will have to go from low to high and then sample high by a falling 8207 clock edge. Since external refresh requests typically arrive asynchronously with respect to the 8207’s clock, this requires the REFRQ to be synchronized to the 8207 clock when programmed in the failsafe mode. This is to ensure that the request is seen for one clock - no more, no less. If no external synchronization is performed, then the 8207 could do random burst cycles.

**Processor Interface Options:**

The PCTLA, PCTLB input pins will program the 8207 to accept either the standard demultiplexed RD and WR inputs, or to directly decode the status outputs of Intel’s iAPX86, 88 family of microprocessors. The state definitions of the status lines and their timings, relative to the processor clock, differ for the 8086 family and the iAPX286 processor. Table 1 illustrates how the 8207 interprets these inputs after the PCTL pins are programmed.

If PCTL is seen high, as RESET goes inactive, and 8086 status interface is enabled. The commands arriving at the 8207 are sampled by a rising clock edge. When PCTL is low, the 80286 status and Multibus command interface is selected. These commands are sampled by the 8207 by a falling clock edge.

More information on interfacing to processors is contained in the *Microprocessor Interface section*.

**Table 1. Status Coding of 8086, 80186 and 80286**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Function</th>
<th>80286</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>Interrupt</td>
<td>Interrupt</td>
</tr>
<tr>
<td>0 0 1</td>
<td>I/O Read</td>
<td>I/O Read</td>
</tr>
<tr>
<td>0 1 0</td>
<td>I/O Write</td>
<td>I/O Write</td>
</tr>
<tr>
<td>0 1 1</td>
<td>Halt</td>
<td>Idle</td>
</tr>
<tr>
<td>1 0 0</td>
<td>Instruction</td>
<td>Halt</td>
</tr>
<tr>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>Memory Read</td>
<td>Memory Read</td>
</tr>
<tr>
<td>1 1 0</td>
<td>Memory Write</td>
<td>Memory Write</td>
</tr>
<tr>
<td>1 1 1</td>
<td>Idle</td>
<td>Idle</td>
</tr>
</tbody>
</table>

**8207 Response**

<table>
<thead>
<tr>
<th>8207 Command</th>
<th>Function</th>
<th>80286 Status Interface</th>
<th>Command Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCTL RD WR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>Ignore</td>
<td>Ignore</td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>Ignore</td>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>Ignore</td>
<td>Write</td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>Ignore</td>
<td>Ignore</td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>Read</td>
<td>Ignore</td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>Read</td>
<td>Inhibit</td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>Write</td>
<td>Inhibit</td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>Ignore</td>
<td>Ignore</td>
<td></td>
</tr>
</tbody>
</table>

**Programming Word**

The 8207 requires more information to operate in a wide variety of systems: The 8207 alters its timings and pin functions to operate with the 8206 ECC chip. The programming options allow the designer to use asynchronous or synchronous buses, various clock rates, various speeds and types of RAM, and others. This is detailed in Table 2.

This data is supplied to the 8207 over the PDI input pin. There are two methods of supplying this data. One is to strap the PDI pin high or low with the subsequent restrictions on your system.
3 shows the required system configuration. Note that your only option when strapping this pin high or low is error correction or not.

If any other configurations are required, then the programming data will have to be supplied by one or two 74LS165 type shift registers. Note that the sense of the bits in the program word change between ECC and non-ECC configurations.

Table 2a.
Non-ECC Mode Program Data Word

<table>
<thead>
<tr>
<th>PD15</th>
<th>PD8</th>
<th>PD7</th>
<th>PD0</th>
<th>Name</th>
<th>Polarity/Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>TM1</td>
<td>PPR</td>
<td>FFS</td>
<td>EXT</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>CTS</td>
<td>PLS</td>
<td>CI0</td>
<td>CI1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>RB1</td>
<td>RB0</td>
<td>RFS</td>
<td>CFS</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>SB</td>
<td>SA</td>
<td>COMP</td>
<td>SMI</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>PBO</td>
<td>PBI</td>
<td>PBO</td>
<td>PBO</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>EXT</td>
<td>FFS</td>
<td>FFS</td>
<td>FFS</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>PPR</td>
<td>PPR</td>
<td>PPR</td>
<td>PPR</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>TM1</td>
<td>TM1</td>
<td>TM1</td>
<td>TM1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Program Data Bit | Name | Polarity/Function
---|---|---
PD0 | ECC | ECC = 0 For non-ECC mode
PD1 | SA | SA = 0 Port A is synchronous
PD2 | SB | SB = 0 Port B is asynchronous
PD3 | CFS | CFS = 0 Fast-cycle iAPX 286 mode
PD4 | RFS | RFS = 0 Fast RAM
PD5 | RBO | RAM bank occupancy
PD6 | RB1 | See Table 4
PD7 | CI1 | Count interval bit 1: see Table 6 in 8207 data sheet
PD8 | CI0 | Count interval bit 0: see Table 6 in 8207 data sheet
PD9 | PLS | PLS = 0 Long refresh period
PD10 | EXT | EXT = 0 Not extended
PD11 | FFS | FFS = 0 Fast CPU frequency
PD12 | PPR | PPR = 0 Most recently used port priority
PD13 | TM1 | TM1 = 0 Test mode 1 off
PD14 | 0 | Reserved must be zero
PD15 | 0 | Reserved must be zero
Table 2b.  
ECC Mode Program Data Word

<table>
<thead>
<tr>
<th>Program Data Bit</th>
<th>Name</th>
<th>Polarity/Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD0</td>
<td>ECC</td>
<td>ECC = 1 ECC mode</td>
</tr>
<tr>
<td>PD1</td>
<td>SA</td>
<td>SA = 0 Port A is asynchronous (late AACK)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SA = 1 Port A is synchronous (early AACK)</td>
</tr>
<tr>
<td>PD2</td>
<td>SB</td>
<td>SB = 0 Port B is synchronous (early AACK)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SB = 1 Port B is asynchronous (late AACK)</td>
</tr>
<tr>
<td>PD3</td>
<td>CFS</td>
<td>CFS = 0 Slow-cycle iAPX 86 mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CFS = 1 Fast-cycle iAPX 286 mode</td>
</tr>
<tr>
<td>PD4</td>
<td>RFS</td>
<td>RFS = 0 Slow RAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RFS = 1 Fast RAM</td>
</tr>
<tr>
<td>PD5</td>
<td>XA</td>
<td>XA = 0 Multibus-compatible XACKA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XA = 1 AACKA not multibus-compatible</td>
</tr>
<tr>
<td>PD6</td>
<td>XB</td>
<td>XB = 0 AACKB not multibus-compatible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XB = 1 Multibus-compatible XACKB</td>
</tr>
<tr>
<td>PD7</td>
<td>CI1</td>
<td>Count interval bit 1: see Table 6 in 8207 data sheet</td>
</tr>
<tr>
<td>PD8</td>
<td>CI0</td>
<td>Count interval bit 0: see Table 6 in 8207 data sheet</td>
</tr>
<tr>
<td>PD9</td>
<td>PLS</td>
<td>PLS = 0 Short refresh period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PLS = 1 Long refresh period</td>
</tr>
<tr>
<td>PD10</td>
<td>EXT</td>
<td>EXT = 0 Master and slave EDCU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXT = 1 Master EDCU only</td>
</tr>
<tr>
<td>PD11</td>
<td>FFS</td>
<td>FFS = 0 Slow CPU frequency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FFS = 1 Fast CPU frequency</td>
</tr>
<tr>
<td>PD12</td>
<td>PPR</td>
<td>PPR = 0 Port A preferred priority</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PPR = 1 Most recently used port priority</td>
</tr>
<tr>
<td>PD13</td>
<td>RB0</td>
<td>RAM bank occupancy</td>
</tr>
<tr>
<td>PD14</td>
<td>RB1</td>
<td>See Table 4</td>
</tr>
<tr>
<td>PD15</td>
<td>TM2</td>
<td>TM2 = 0 Test mode 2 enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TM2 = 1 Test mode 2 off</td>
</tr>
</tbody>
</table>

Table 3. 8207 Default Programming

- Port A is Synchronous—has early AACK
- Port B is Asynchronous—has late AACK
- Fast RAM
- Refresh Interval uses 236 clocks
- 128 Row refresh in 2 ms; 256 Row refresh in 4 ms
- Fast Processor Clock Frequency (16 MHz)
- "Most Recently Used" Priority Scheme
- 4 RAM banks occupied
Reset

If Port A is changed to an asynchronous interface (via the SA bit), then one of two precautions must be taken. Either a differentiated reset must be provided, or else software must not access the 8207 controller RAM for a short period. The 8207 is either adding or deleting internal synchronizing circuits. If a command is received during this changing, the 8207 may not perform properly. This is required only if Port A is changed to asynchronous, or if Port B is changed to synchronous.

Several of the bits in the program word determine a particular configuration of the 8207 (reference Tables 10, 11 and the 8207 Data Sheet). The bits are: CFS, CLOCK fast or slow; RFS, RAM access time fast or slow (fast refers to 100 ns - slow is everything greater); and EXT, for memory data word widths greater than 16 (22) bits. Generally speaking, CO is the fastest configuration at clock frequencies up to 16 MHz, both in the ECC or non-ECC charts. ‘C3’ is the fastest for 8 MHz clocks in non-ECC mode, and ‘C4’ is the fastest configuration when using ECC.

Take, for example, a 16 MHz 8207 clock with no error correction, a 16 bit word, and 150 ns (slow) dynamic RAMs. Table 10, in the 8207 data sheet, is used to arrive at the configuration “C1.” The Timing chart Table 12 in the 8207 Data Sheet is then used to determine which clock edge to reference all timings from. The Waveforms diagrams then are used to determine the delay from the clock edge.
CHAPTER 3
RAM INTERFACE

The 8207 takes the memory addresses from the microprocessor bus and multiplexes them into row and column addresses as required by dynamic RAMs. The only hardware requirement when interfacing the 8207 to dynamic RAM are series resistors on all the RAM outputs of the 8207, and proper layout of the traces (see Intel's RAM Data Sheets or the Memory Design Handbook). This section mainly details the effects and requirements of input signals to the 8207 on the RAM array.

The 8207 contains an internal address counter used for refresh and error scrubbing (when using the 8206 EDCU) cycles. The 8207 has 18 address inputs (AIL0-AIL8 and AIH0-AIH8) which are multiplexed to form 9 address outputs (A00-A08). There are also 2 bank select (BS0, BS1) inputs for up to 4 banks of RAM. The Bank Select inputs are decoded internally to generate RAS and CAS outputs.

Refresh Interval

The 8207 supports four different refresh techniques as described in the Refresh Options section. In addition, the rate at which refresh cycles are performed is programmable. This is necessary because the refresh period is generated from the CLK input, which may vary over a wide range of frequencies. Programming the Cycle Fast/Slow (CFS) and Frequency Fast/Slow (FFS) bits automatically reprograms the refresh timer to generate the correct refresh interval for a clock frequency of 16, 10, 8, or 5 MHz (CFS, FFS = 11, 10, 01, and 00, respectively). For clock frequencies between those, Count Interval (C11, C10) programming bits allow "fine tuning" of the refresh interval. Refresh will always be done often enough to satisfy the RAM's requirements without doing refresh more often than needed and wasting memory bandwidth for all clock frequencies.

Refresh Counter

The internal refresh address counter of the 8207 contains 20 bits as organized in Figure 1.

<table>
<thead>
<tr>
<th>Bank</th>
<th>19 18</th>
<th>17 16 15 14 13 12 11 10 9</th>
<th>8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col addr</td>
<td>19 18 Bank</td>
<td>Row addr</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1. 8207 Refresh Address Counter

In non-ECC mode, the refresh address counter does not count beyond bit 8. For standard RAMs, this will refresh 128 rows every 2 ms or 256 rows every 4 ms.

In ECC mode, the 8207 automatically checks the RAM for errors during refresh. This requires it to access each of the possible 220 words of memory. The 8207 does not delete any of these bits when used with 16k and 64k dynamic RAMs. Each column would be scrubbed 4 times with 16k RAMs, and twice with 64 RAMs. This will have no detrimental effect on reliability. Banks of RAM that are not occupied, as indicated to the 8207 by the RB0, RB1 programming bits, will not be scrubbed.

Bank Selects BS0, BS1; RB0, RB1

The 8207 is designed to drive up to 88 RAMs in various configurations. The 8207 takes 2 inputs, BS0, BS1, and decodes them based on 2 programming bits, RB0, RB1, to generate the required RAS/CAS strobes. Additionally, the 8207 will always recognize (not programmable) whether an access is made to the same RAM bank or to a different bank. The 8207 will interleave the accesses resulting in improved performance.
RAS and CAS Reallocation

The 8207's address lines are designed to drive up to 88 RAMs directly (through impedance matching resistors). The 4 RAS and CAS outputs drive up to 22 RAMs per bank (16 data plus 6 check bits with the 8206). Under these conditions, the 8207 will meet all RAM timing requirements. See Figure 2 for an example.

The 8207 can accommodate other configurations like a 32 bit error corrected memory system. Each bank would have 39 RAMs (32 + 7 check bits) with the total number of RAMs equal to 78. This is within the address drivers capability, but the 39 RAMs per bank exceeds the RAS and CAS drivers limits. The loading of the RAS/CAS drivers should not exceed 22 RAMs per bank, otherwise critical row, column address setup, and hold times would be violated.

In order to prevent these critical timings being violated, the 8207 will re-allocate the RAS and CAS drivers based on the RBO, RBI programming bits (see Table 4). If the RB0, RB1 bits are programmed for 2 banks, the 8207 will operate RAS0 and RAS1 as a pair along with RAS2 and RAS3, CAS0 and CAS1, and CAS2 and CAS3. Now the address drivers would be loaded by 78 RAMs and the RAS/CAS drivers by 20 RAMs. This relative loading is almost identical to the first case of four banks of 22 RAMs each. Drive reallocation allows a wide range of memory configurations to be used and still maintain optimal memory timings. Figure 3 shows a 32 bit non-error corrected configuration.

These programming bits do not help to qualify RAM cycles. Their purpose is to reallocate RAS/CAS drivers. For example, if there is one bank of RAM and the bank select inputs (BS0, BS1) select any other bank and no provision is made to deselect the 8207 (via PE), the 8207 will do a RAM cycle and issue an acknowledge. This happens irregardless of the RB0, RB1 programmed value. See the Optional RAM Bank's section to provide for this.
<table>
<thead>
<tr>
<th>Program Bits</th>
<th>Bank Input</th>
<th>RAS/CAS Pair Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RB1 RB0</td>
<td>B1 B0</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>RAS0-3, CAS0-3 to Bank 0</td>
</tr>
<tr>
<td>0 0</td>
<td>0 1</td>
<td>Illegal Bank Input</td>
</tr>
<tr>
<td>0 0</td>
<td>1 0</td>
<td>Illegal Bank Input</td>
</tr>
<tr>
<td>0 0</td>
<td>1 1</td>
<td>Illegal Bank Input</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>RAS0-1, CAS0-1 to Bank 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>RAS2-3, CAS2-3 to Bank 1</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
<td>Illegal Bank Input</td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td>Illegal Bank Input</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>RAS0, CAS0 to Bank 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
<td>RAS1, CAS1 to Bank 1</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>RAS2, CAS2 to Bank 2</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>Illegal Bank Input</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td>RAS0, CAS0 to Bank 0</td>
</tr>
<tr>
<td>1 1</td>
<td>0 1</td>
<td>RAS1, CAS1 to Bank 1</td>
</tr>
<tr>
<td>1 1</td>
<td>1 0</td>
<td>RAS2, CAS2 to Bank 2</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>RAS3, CAS3 to Bank 3</td>
</tr>
</tbody>
</table>

**Figure 3. 8207 2 RAM Bank Configuration**
Scrubbing

An additional function of the RB0, RB1 bits, besides RAS/CAS allocation, is to inform the 8207 of how many banks are physically present. The 8207 will, during the refresh cycle, read data from a location and check to see that data and check bits are correct. If there is an error, the 8207 lengthens the refresh cycle and writes the corrected data back into RAM. Scrubbing the entire memory greatly reduces the chance of an uncorrectable error occurring. See the Refresh section for more detail on scrubbing.

Refresh Cycles

The 8207 performs RAS only refresh cycles in non-ECC systems. It outputs all 8207 control signals except for CAS and acknowledges. The real delay in a system due to refresh would be a fraction of that value. The length of the refresh cycle is always 2tRP + tRAS, and varies based upon the programmed 8207 configuration.

In error-corrected systems, the refresh cycle is actually a read cycle. The 8207 outputs a row address, then all RAS outputs go active. Next, a column address is output and then CAS. The CAS output is based upon the RB0, RB1 allocation bits. Figure 4a shows the general timing for a four bank system, and Figure 4b shows a two bank system.

![Figure 4. Refresh Cycles for Error Corrected Systems](image-url)

(1) Measurements have shown a delay of 2-4% on program execution time compared to programs running without refresh.
The 8207 sends the read out word through the 8206 EDCU to check for any errors. If no errors, the refresh cycle ends. If an error is discovered, the 8207 lengthens the cycle. An error is determined if the ERROR output of the 8206 is seen active at the same edge that the 8207 issues the R/W output. The cycle is then lengthened to a RMW cycle. If the error was correctable, the corrected data is written back to the location it was read from. But, if the data is uncorrectable, the cycle is still lengthened to a RMW, but no write pulse is issued. To aid in stabilizing the RAM output data and the Error flag, pullup resistors of 10k ohms on the data out lines are recommended.

Scrubbing removes soft errors that may accumulate until a double-bit error occurs, which would halt the system. Hard single-bit failures will not stop the system, but could slow it down. This is because read and refresh cycles lengthen to correct the data.

For large RAM arrays some form of error logging or diagnostics should be considered.

**Interleaving**

The term “interleaving” is often used to refer to overlapping the cycle times of multiple banks (or boards or systems) of RAMs. This has the advantage of using relatively slow cycle time banks to achieve a faster perceived cycle time at the processing unit. The drawbacks of interleaving are more logic to handle the necessary control and, for maximum performance, the program should execute sequentially through the addresses.

Dynamic RAM cycles consist of 2 parts — the RAS active time (tRAS in Dynamic RAM Data Sheets) and precharge time (tRP). The sum of these two times are roughly equal to the cycle time of the RAM. The 8207 determines how long these two periods are, based on the configuration the user picked (via the programming bits). Bank interleaving, as used by the 8207, is slightly different than the previous definition. The 8207 will overlap the precharge time of one bank with the access time of another bank. In either case, the advantage is the effective cycle time is reduced without having to use faster RAMs.

For interleaving to take place there must be more than 1 bank of RAM connected to the 8207. Interleaving is not practical with 3 banks of RAM because 3 is not a power of 2 (the 2 bank inputs BS0, BS1). So, interleaving works only for 2 or 4 banks of RAM. Note that it is easy enough to use three banks of RAM where the bank select inputs are connected to the highest-order address line. For instance, if three banks of 2164s are used in an 8086 system, and located at address OH, bank selects BS0 and BS1 would be connected to microprocessor addresses A17 and A18, respectively. Banks 0-2 would be accessed in the address ranges OH - FFFFH, 10000H - 1FFFFH, and 20000H - 2FFFH, respectively. In this case, consecutive addresses are almost always in the same bank and very little interleaving can take place.

Figure 5 shows the effects on the performance of the processor with and without interleaving. In both examples, consecutive accesses to the same bank will add 1 wait state to the second access, but no wait states to consecutive accesses to different banks. Irregardless of the 8207 configuration, there will always be a minimum 1 wait state added without interleaving. Therefore, interleaving is very highly recommended!

Interleaving is accomplished by connecting the 8207’s BS0, BS1 inputs to the microprocessor’s low order word address lines. Each consecutive address is then located in a different bank of RAM. About 90% of memory accesses are sequential, so interleaving will occur about 90% of the time in a single port system.

In a dual port system, the advantages of interleaving are a function of the number of banks of memory. Since the memory accesses of the two ports are presumably independent, and both ports are continuously accessing memory, the 8207 arbiter will tend to interleave accesses from each port (i.e., Port A, Port
B, Port A, Port B, ...). If there are two banks of RAM interleaving will occur 50% of the time and, if there are four banks of RAM, interleaving will take place 75% of the time\(^1\). To the extent that a single port generates a majority of memory cycles, interleaving efficiency will approach 90% as described in the previous paragraph.

(1) Don't get confused here. The paragraph is talking about interleaving memory requests from both ports, and their probability of accessing one of the other banks of RAM where tRP has been satisfied. The 8207 will leave the RAM precharge time out if consecutive accesses go to different banks. The 8207 RAM timing logic does not care which port requests a RAM cycle.

**Optional RAM Banks**

Many users allow various RAM array sizes for customer options and future growth. Some care must be taken during the design to allow for this. Three items should be considered to permit optional RAM banks.

The first item is the total RAM size. The 8207 starts a memory cycle based only upon a valid status or command and PE active. So some logic will be required to deselect the 8207 (via PE) when the addressed location does not exist within the current memory size. A 7485 type magnitude comparator works well.

The second item to consider is the BS0, BS1 inputs. With one bank of RAM these inputs are tied to ground. Four banks of RAM require two address inputs. So, if the design ever needs four banks
of RAM, then the BS0, BS1 inputs must be connected to address lines. Selecting a non-existant RAM bank is illegal. Figure 6 shows a non-interleaved method.

![Figure 6. Non-Interleaved 8207 Selection Circuit](image)

With designs using interleaving, the least significant word address lines are connected to the BS0, BS1 inputs. With two banks of RAM, A1 from the Intel processor is connected to BS0. A2 is connected to BS1, but not allowed to function until four banks are present. However, A2 must still be used since addresses increase sequentially. Two possible ways of implementing this are shown in Figure 7 below.

![Figure 7. Interleaved 8207 Selection Circuits](image)
The final consideration is for the RAS/CAS outputs. Remember that when the RB0, RB1 bits are programmed for two banks, then RAS0, 1 operates in tandem (non-ECC mode/ECC mode - the CAS outputs also work in tandem). Figure 8 shows the proper layout.

![Figure 8. RAM Bank Layout](image)

**Write Enables - Byte Marks**

The write enable supplied by the 8207 cannot drive the RAM array directly. It is intended to be NAND with the processor supplied byte marks in a non-ECC system. In error-corrected systems, the write enable output should be inverted before being used by RAMs. Only full word read/writes are allowed in ECC systems. The changing of byte data occurs in the 8206 EDCU.

For single and dual port systems, the byte mark data (A0, BHE) must be latched. The 8207 can (and will) change the input addresses midway through a RAM cycle.

**Memory Warm-up and Initialization**

After programming, the 8207 performs 8 RAM warm-up cycles. The warm-up cycles are to prepare the RAMs for proper operation. If the 8207 is configured for ECC, it will then prewrite zeros into the entire array.

All RAS outputs are driven active for these cycles, once every 32 clock periods. The prewrite cycles are equivalent to write cycles, except all RAS and CAS will go active, data is generated by the 8206, and the address is generated by the 8207.

**RAM Cycles/Timings**

Tables 12 and 13 of the 8207 Data Sheet show on what clock edge each of the 8207 outputs are generated. This, together with the timing waveforms and A.C. parameters, allows the user to calculate the timings of the 8207 for each of its configurations. To make the job easier, Tables 14-18 of the 8207 Data Sheet precalculate dynamic RAM timings for each 8207 configuration and type of cycle. All that is required is to plug in numerical values for the 8207 parameters.
Write Cycles

The 8207 always issues WE after CAS has gone valid. These types of cycles are known as "late writes." The 8207 does this primarily to interface to the iAPX286 processor bus timings. Late writes require separate data in and data out traces to the RAM array, plus the additional drivers.

Data Latches

The 8207 is designed to meet data setup and hold times for the iAPX86 family processors when using a synchronous status interface (see Microprocessor Interface section). Other types of interfaces will require external data latches. This is because the CAS pulse is a fixed length - the user has no control (besides programming options) over lengthening CAS. When CAS goes inactive, data out of the RAMs will disappear. Asynchronous interfaces should use XACK or LAACK to latch the data.
CHAPTER 4
MICROPROCESSOR INTERFACES

The 8207 is designed to be directly compatible with all Intel iAPX86, 186, 188, and 286 processors. For maximum performance, the 8207 will directly decode the status lines and operate off of the processor's clock. Additionally, the 8207 interfaces easily to other bus types that support demultiplexed address and data with separate read and write strobes.

Bus Interfaces

The 8207 easily supports either an asynchronous or synchronous command timing. The command timing can also be adjusted for various processors via the PCTL pin.

MEMORY COMMANDS

There are four inputs for each port of the 8207 that initiate a memory cycle. The input pins are WR, RD, PCTL, and PE. The first three inputs connect directly to the iAPX 86, 88, 186, 188 S0-S2 outputs, respectively. For the 80286, the same connections are used except that PCTL is tied to ground. In all configurations PE is decoded from the address bus. Multibus type commands use the same input setup as the 80286.

COMMAND/STATUS INTERFACE

The status interface for the 80186 and the 80286 differ both in timing and meaning. The 8207 can be optimized for either processor by programming the PCTL input pin at RESET time. S2 in 80186 systems, connects directly to PCTL. When the processor is reset it drives S2 high for one clock, then tristates it. A pullup resistor to +5 will program the PCTL input for the 80186 status interface when RESET goes inactive. A pullup is required only if no component has this pullup internally.

To optimize the 8207 for the 80286 interface, PCTL is tied to ground and not used in 80286 systems. Multibus commands are similar in meaning to the 80286 status interface, and are programmed the same way. In Multibus type systems, PCTL can be used as an inhibit to allow shadow memory. PCTL would be driven high, when required, to prevent the 8207 from performing a memory cycle. It would be connected to the Multibus INH pin through an inverter.

SYNCHRONOUS/ASYNCHRONOUS COMMANDS

Each port of the 8207 can be configured to accept either a synchronous or asynchronous (via programming bits) memory request. Minimum memory request decode time (and maximum performance) is achieved using a synchronous status interface. This type of interface to the processor requires no logic for the user to implement.

An asynchronous interface is used with Multibus bus interfaces when the setup and hold times of the memory commands cannot be guaranteed. Synchronizers are added to the inputs and will require up to two clocks for the 8207 to recognize the command. It should be obvious that better performance will result if the 8207’s clock is run as fast as possible.

Figure 2 of the 8207 Data Sheet shows various combinations of interfaces. The additional logic for the asynchronous interfaces is used to either lengthen the command width, to meet the minimum 8207 spec, or to make sure the command does not arrive too soon before the address has stabilized.

PORT ENABLE

The PE inputs serve to qualify a memory request. A RAM cycle, once started, cannot be stopped. A RAM cycle starts if PE is seen active at the proper clock edge and a valid command is recognized. If PE is activated after a command has gone active and inactive, no cycle will start.
Types of logic that work well are 74138 and 7485. $\overline{PE}$ should be valid as much as possible before the command arrives because, as the address bus switches and settles, glitches on $\overline{PE}$ could either: disqualify a memory cycle; delay a memory cycle; or start a memory cycle when none should have. Refer to the Port Interface Waveforms in the Data Sheet. If Port Enable is not seen active by the next or same clock edge, no memory cycle will occur unless the command is removed and brought active again.

Back to Back Commands

Holding the RD, WR inputs active will not generate continuous memory cycles. Memory commands must go inactive for at least one clock period before another memory request at that port will be considered valid. Holding the inputs active will not keep the other port from gaining access to the RAM. The only signal that can prevent the other port’s gaining access to the RAM is LOCK.

Address Inputs (And LOCK)

Two pins control the address inputs on the 8207, MUX and LEN. Neither are used for single port 8086 based systems. MUX is used for dual port configurations, and LEN is used for single and dual port 80286 based systems. MUX is used to gate the proper ports addresses to the 8207. If the output is high, Port A is selected. If it is low, Port B is selected.

The cross coupled NAND gates, shown in the 8207 Data Sheet (Figure 3), are used to minimize contention when switching address buses. Use of a single inverter would have both outputs enabled simultaneously for a short period. The cross coupled hand gates allow only one output enabled.

MUX also allows the single LOCK input to be multiplexed between ports. Figure 9 shows how to multiplex the LOCK input for dual port systems. See the LOCK section for more information.

![Figure 9. Dual Port LOCK Input Circuit](image)

MUX TIMING

The MUX output is optimized by the Port Arbitration scheme, which is selected in the program word. Figure 10 shows the effects on memory selected in the program word. Figure 10 shows the effects on memory bandwidth with the different schemes. Port A Preferred optimizes consecutive cycles for Port A. Consecutive Port B cycles have at least 1 clock added to their cycle time. There would be no MUX delays for any Port A request.
The Most Recently Used scheme allows either port to generate consecutive cycles without any MUX delays. The first memory cycle for each port would have the 1 clock delay. But all others would not.

With either scheme, if both ports request the memory at their top speed, the 8207 will interleave the requests; Port A, Port B, Port A, Refresh, Port B.

![Diagram](image)

**Figure 10. Port Arbitration Effects**

**LEN**

LEN is used to hold the 80286 addresses when the 8207 cannot respond immediately. The 8207 will require a separate address latch, with the ALE input replaced with LEN. LEN optimizes the address setup and hold times for the 8207.

LEN goes from high to low when a valid 8207 command is recognized, which latches the 80286 address. This transition of LEN is independent of a memory cycle starting. The low to high transition will occur in the middle of a memory cycle so that the next address will be admitted and subsequently latched.

If Port B is to interface to an 80286 with the synchronous status interface, then LEN must be created using external logic. Figure 11 shows the equivalent 8207 circuit for Port B.

**LOCK**

The LOCK input allows each port uninterrupted access to memory. It does this by not permitting MUX to switch. It is not intended as a means to improve throughput of one of the ports. To do so is at the designer's risk. Obviously, LOCK is only used in dual port systems. The 8207 interprets LOCK as originating from the port that MUX is indicating.

(1) The 8207 will not malfunction if this is done. This is a system level concern. For example, a time dependent process may fail if the other port holds LOCK active, preventing its access of memory and relinquishing the bus.
LOCK from the 8086 may be connected directly to the 8207 or to the multiplexing logic. The 8207 requires additional logic when interfaced to an 80286. Figure 12 shows both the synchronous and asynchronous circuitry.

For 16 MHz operation, the 8207 ignores the LOCK input during the clock period that MUX switched. During 8 MHz operation, the 8207 will see LOCK as being active during the clock period when MUX switches.

The LOCK issued in Multibus bus systems may not be compatible with the 8207. The 8207 references LOCK from the beginning of a cycle, while Multibus references LOCK from the end of a cycle.
Multibus LOCK can be used if it meets the 8207 requirements. If the LOCK timing cannot be guaranteed, then additional logic is necessary. The logic would issue LOCK whenever a Multibus command is recognized. The drawback to this is that MUX cannot switch during the RAM cycle. This would delay the other port’s memory access by one or two clocks.

DEADLOCK

The designer should ensure that a deadlock hazard has not been created in the design. The simple interfaces shown previously will not create a deadlock condition when the 8207 controls all system memory. If LOCK is issued by both ports, then the above logic would need to be modified to remove LOCK.

Figure 13 shows an illustration of the problem with a single LOCK input.
Suppose the 8207 starts a locked string transfer for the processor. The Multibus bus port requests a memory cycle but must wait for the processor to remove LOCK. But the processor must access Multibus as part of the locked string transfer. We now have a deadlock. The solution is to force LOCK inactive whenever an access is made to non-8207 memory by the processor. By doing this we have now violated the purpose of LOCK, since the Multibus port could change data. Another solution is to ensure that locked data does not exist in physically separate memory.

**8207 Acknowledge’s**

The 8207 in non-ECC mode has two active acknowledge’s per port, AACK and XACK. The AACK output is configured into either an “early” or “late” AACK based on the SA, SB bits in the program data word. In ECC systems there is one Acknowledge per port, and it is configured to any one of the three (EACK, LAACK or XACK) by the programming bits.

The AACK pin is optimized for either the 80286 or the 8086, based upon the CFS programming bit (fast = 80286; slow = 8086). XACK conforms to the Multibus bus specification. XACK requires a tri-state buffer and must not drive the bus directly.

In synchronous systems, XACK will not go active if the memory command is removed prior to the clock period that issues XACK. In asynchronous systems, the AACK pin can also serve as an advanced RAM cycle timing indicator.

Data out, in synchronous systems, should not have to be latched. The 8207 was designed to meet the data setup and hold times of Intel processors, the 8086 family, and the 80286. In asynchronous systems, the 8207 will remove data before the processor recognizes the Acknowledge (LAACK or XACK). In these systems, the data should be latched with transparent type latches (Intel 8282/8283).

**Output Data Control**

**Non-ECC**

In single port systems, Intel processors supply the necessary timing signals to control the input or output of data to the RAMs. These control signals are DEN and DT/R. Refer to the microprocessor handbook for their explanation. If these signals are not available, then PSEN and DBM provide the same function. They can be used directly to control the 8286/8287 bus drivers of the 8207.

Because of the single set of data in/out pins of the RAMs, data must be multiplexed between the two ports in dual port systems. The 8207 provides two outputs for contention-free switching. PSEL operates the same as the MUX output, in that a high selects Port A and a low selects Port B. PSEN acts to enable the selected port. The timing is shown in the 8207 Data Sheet, Port Switching Timing section.

The easiest means of using PSEL and PSEN is shown in Figure 14. At no time will both ports be enabled simultaneously.

![Figure 14. PSEL and PSEN Interface Circuit](image)
Data Bus - Single Port

Recall that the 8207 always performs a late write cycle and that this requires separate data in and out buses. One option for the data bus is shown in Figure 3 of the 8207 Data Sheet. It requires separate data in and out traces on the processor board.

The second option is to keep the processor’s combined data, bus but separate the data at the 8207 RAM. This is shown in Figure 15.

![Data Bus Circuit Diagram](image)

Figure 15. Data Bus Circuit

Data Bus - Dual Port

*Non-ECC*

The multiplexed data of the 8207 RAM must be kept isolated so that an access by one port does not affect another port. Figure 16 illustrates the control logic.
Figure 16. Dual Port Data Bus Control Circuitry
CHAPTER 5
8207 WITH ECC (8206)

This section points out the proper control of the 8206 EDCU by the 8207.

The 8207 performs error correction during read and refresh cycles (scrubbing), and initializes memory after power up to prevent false errors from causing interrupts to the processor. Since the 8207 must refresh RAM, performing scrubbing during refresh allows it to be accomplished without any additional performance penalty. Upon detection of a correctable error during scrubbing, the RAM refresh cycle is lengthened slightly to permit the 8206 to correct the error and for the corrected word to be rewritten into memory. Uncorrectable errors detected during scrubbing are ignored, since the processor may never access that memory location.

Correctable errors detected during a memory read cycle are corrected immediately and written back into memory.

Synchronous/Asynchronous Buses

The many types of configurations that are supported by the 8207/8206 combination can be broken down into two classes: ECC for synchronous or for asynchronous buses.

In synchronous bus systems, performance is optimized for processor throughput. In asynchronous buses, performance is optimized to get valid data onto the bus as quickly as possible (Multibus). While possible to optimize the 8207/8206 for processor throughput in Multibus systems, it is not Multibus compatible. The performance optimization is selected via the XA/XB and SA/SB programming bits.

When optimized for processor throughput, an advanced acknowledge (AACK - early or late) is issued at some point (based on the type of processor) so that data will be valid when the processor needs it.

When optimized for quick data access, an XACK is issued as soon as valid data is known to exist. If the data was invalid (based on the ERROR flag), then the XACK is delayed until the 8206 corrects the data and the data is on the bus.

The correct on error mode is of no real benefit to non-Multibus users. The earliest acknowledge (EAACK) is delayed by one clock to allow for the delays through the 8206. This imposes a 1 wait state delay.

Byte Marks

The only real difference to the 8207 system when adding the 8206 is the treatment of byte writes. Because the encoded check bits apply only to a whole word (including check bits), byte writes must not be permitted at the RAM. Instead, the altering of byte data is done at the 8206. The byte marks previously sent to RAM are now sent to the 8206. These byte marks must also qualify the output enables of the data drivers.

The DBM output of the 8207 is meant to be nanded with the processors byte marks. This output is activated only on reads or refreshes. On write cycles, this output stays high which would force the 8206 byte mark input low. When low, the internal 8206 data out buffers are tristated so that new data may be gated into the device.
Read Modify Writes - ECC

A RMW cycle occurs whenever a processor wants to do byte writes or when the 8207 has detected an error during read or refresh (scrubbing) cycles. A byte write is detected by the FWR input to the 8207 and is based on the processor supplied byte marks.

At the start of a RMW cycle, DBM stays high, which, when qualified with the byte marks, will enable the data out buffer of the 8206 for the unmodified byte, and tristates the buffer for the new byte; R/W is high, which tells the 8206 to do error detection and correcting (if CRCT is low). The 8206 can latch data and check bits from the RAM via the STB input, but the 8207 does not use this feature. Instead, the 8207 keeps CAS active the entire length of the RMW cycle to hold data at the 8206. The new byte data from the processor goes to the 8206 and to the RAM. The 8207 would have corrected any errors just read, so the old and new bytes of data, plus their check bits, are available at the RAM, and the 8207 generates a write pulse. The data driver for the unmodified byte must not have been enabled, otherwise erroneous data would be written to RAM and possibly made valid (if it was stable) by the 8206.

Data Buffer Control - ECC

The control of the data buffers is essentially the same as in non-ECC systems, with a few exceptions.
The processor's byte marks must now qualify the output enable logic. The reason was described earlier in the RMW section. This applies to both single and dual port configurations. A refresh cycle outputs all the control signals that a read cycle will, except for an acknowledge. If complete buffer control is left to the 8207, then it would occasionally (during refreshes) put data on the processor bus. The DEN and DT/R signals must be qualified by the PE input. PE would have to be latched for the entire cycle by PSEN.

Test Modes

Neither of the two test modes of the 8207 are to be used in a design. Both test modes reset the refresh address counter to a specific value, which interrupts the refresh sequence and causes loss of data.

In error corrected systems, a reset pulse causes the 8207/8206 to write over the entire RAM array. Test Mode 2 appears to bypass the prewrite sequence. But, the refresh counter is reset to a value of 1F7 (H). So, besides interrupting the refresh sequence, the 8207 still prewrites the 8 locations specified by the counter.

To not overwrite the RAM data, the 8207 RESET will have to be isolated from the system reset logic in ECC systems.
APPENDIX I

8207/8208 Performance

The following performance charts were based upon Figure 3 in the 8207 Data Sheet, and apply to the 8208 as well. All RAM access delays are based upon Intel dynamic RAMs. The charts show the performance of a single cycle with no precharge, refresh, port switching, or arbitration delays.

The read access calculations are: the margin between the 8207 starting a memory cycle to data valid at the processor - 8207 RAS or CAS from clock delay - DRAM RAS or CAS access - 8286 propagation delay - processor setup.

Assume the RAS/CAS drivers are loaded with 150 pf, and the 8286 is driving a 300 pf data bus.

80286 (example)

\[
\text{RAS Access: } 3 \text{TCLCL} - 8207 \text{TCLRSL} - 2118 \text{tRAC} - 8286 \text{TIVO} - 80286 \text{t8} \\
= (3)62.5 - 35 \text{ max} - 100 \text{ max} - 22 - 10 \\
= 20 \text{ ns}
\]

80186 (example)

\[
\text{CAS Access: } 2 \text{TCLCL} - 8207 \text{TCLCSL} - 2164A \text{tCAC} - 8286 \text{TIVO} - 80186 \text{TDVCL} \\
= (2)125 - 115 \text{ max} - 85 \text{ max} - 22 - 20 \\
= 8 \text{ ns}
\]
### 8207 Performance (EDC synchronous status interface)

#### Table 5a. Wait States for Different μP and RAM Combinations

<table>
<thead>
<tr>
<th>Wait states at full CPU speed</th>
<th>RAM speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100 ns</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>Freq</td>
</tr>
<tr>
<td>80286</td>
<td>8 MHz</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>80186, 8086/88-2</td>
<td>8 MHz</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>8086/88</td>
<td>5 MHz</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not (1) compatible with RAM parameters

#### Table 5b. μP Clock Frequency for Different μP and RAM Combinations

<table>
<thead>
<tr>
<th>Maximum frequency for one wait-state (4)</th>
<th>RAM speed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>Freq</td>
</tr>
<tr>
<td>80286</td>
<td>8 MHz</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>80186, 8086/88-2</td>
<td>8 MHz</td>
</tr>
<tr>
<td>8086/88</td>
<td>5 MHz</td>
</tr>
</tbody>
</table>

8207 Performance (EDC synchronous status interface)

8207 Performance (EDC synchronous status interface)
### Table 6a. Wait States for Different µP and RAM Combinations

<table>
<thead>
<tr>
<th>Wait states at full CPU speed</th>
<th>RAM speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100 ns</td>
</tr>
<tr>
<td>CPU</td>
<td>Freq</td>
</tr>
<tr>
<td>80286</td>
<td>8 MHz</td>
</tr>
<tr>
<td>80186, 8086/88-2</td>
<td>8 MHz</td>
</tr>
<tr>
<td>8086/88</td>
<td>5 MHz</td>
</tr>
</tbody>
</table>

(1) Numbers in lower right corners are the programmed configurations of the 8207.

### Table 6b. µP Clock Frequency for Different µP and RAM Combinations

<table>
<thead>
<tr>
<th>Maximum frequency for no wait-state (4)</th>
<th>RAM speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Freq</td>
</tr>
<tr>
<td>80286</td>
<td>8 MHz</td>
</tr>
<tr>
<td>80186, 8086/88-2</td>
<td>8 MHz</td>
</tr>
<tr>
<td>8086/88</td>
<td>5 MHz</td>
</tr>
</tbody>
</table>

(1) The 2164A tRAH parameter is not satisfied.
(2) 150 ns 64K DRAMs with tCAC = 100 ns won’t run with 0 wait-states, because they have a longer CAS access time than the 2164A-15 (tCAC = 85 ns).
(3) Numbers in lower right corners are the programmed configurations of the 8207.
(4) To meet read access time.
8207 Performance (multibus interface)

This is an asynchronous, command interface. Worst case data and transfer acknowledge (XACK#) delays. Including synchronization and data buffer delays, are:

Table 7a. Non-EDC system

<table>
<thead>
<tr>
<th>RAM speed</th>
<th>100 ns</th>
<th>120 ns</th>
<th>150 ns</th>
<th>200 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data access time</td>
<td>289ns</td>
<td>299ns</td>
<td>322ns</td>
<td>380ns</td>
</tr>
<tr>
<td>XACK# access time</td>
<td>333ns</td>
<td></td>
<td></td>
<td>450ns</td>
</tr>
</tbody>
</table>

Table 7b. EDC system

<table>
<thead>
<tr>
<th>RAM speed</th>
<th>100 ns</th>
<th>120 ns</th>
<th>150 ns</th>
<th>200 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data access time (read)</td>
<td>359ns (324 ns)[1]</td>
<td>369ns (334 ns)</td>
<td>392ns (357 ns)</td>
<td>450ns (415 ns)</td>
</tr>
<tr>
<td>XACK# access time</td>
<td>400 ns-RD, WR</td>
<td>588 ns-Byte Write</td>
<td>520 ns-RD, WR</td>
<td>806 ns-Byte WR</td>
</tr>
</tbody>
</table>

(1) Numbers in parentheses are for when 8206 is in check-only mode (8206 doesn’t do error correction until after an error is detected.)
Interfacing Dynamic RAM to iAPX 86, 88 Systems Using the Intel 8202A and 8203

Brad May
Peripheral Component Applications Engineering
INTRODUCTION

The designer of a microprocessor-based system has two basic types of devices available to implement a random access read/write memory — static or dynamic RAM. Dynamic RAMs offer many advantages. First, dynamic RAMs have four times the density (number of bits per device) of static RAMs, and are packaged in a 16-pin DIP package, as opposed to the 20-pin or larger DIPs used by static RAMs; this allows four times as many bytes of memory to be put on a board, or alternatively, a given amount of memory takes much less board space. Second, the cost per bit of dynamic RAMs is roughly one-fourth that of statics. Third, static RAMs use about one-sixth the power of static RAMs, so power supplies may be smaller and less expensive. These advantages are summarized in Table 1.

On the other hand, dynamic RAMs require complex support functions which static RAMs don’t, including

- address multiplexing
- timing of addresses and control strobes
- refreshing, to prevent loss of data
- arbitration, to decide when refresh cycles will be performed.

In addition, dynamic RAMs may not always be able to transfer data as fast as high-performance microprocessors require; wait states must be generated in this case. The circuitry required to perform these functions takes up board space, costs money, and consumes power, and so detracts from the advantages that make dynamic RAMs so appealing. Obviously, the amount of support circuitry should be minimized.

The Intel 8202A and 8203 are LSI dynamic RAM controller components. Either of these 40-pin devices alone does all of the support functions required by dynamic RAMs. This results in a minimum of board space, cost, and power consumption, allowing maximum advantage from the use of dynamic RAMs.

<table>
<thead>
<tr>
<th>Density (No. of bits)</th>
<th>2164-15 (Dynamic)</th>
<th>2167-70 (Static)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of pins</td>
<td>64K</td>
<td>16K</td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>Cycle time (ns)</td>
<td>150</td>
<td>70</td>
</tr>
<tr>
<td>Active power (ma)</td>
<td>300</td>
<td>70</td>
</tr>
<tr>
<td>Standby power (ma)</td>
<td>60</td>
<td>125</td>
</tr>
<tr>
<td>Approx. cost per bit (millicents/bit)</td>
<td>45</td>
<td>40</td>
</tr>
</tbody>
</table>

In addition, dynamic RAMs may not always be able to transfer data as fast as high-performance microprocessors require; wait states must be generated in this case. The circuitry required to perform these functions takes up board space, costs money, and consumes power, and so detracts from the advantages that make dynamic RAMs so appealing. Obviously, the amount of support circuitry should be minimized.

The Intel 8202A and 8203 are LSI dynamic RAM controller components. Either of these 40-pin devices alone does all of the support functions required by dynamic RAMs. This results in a minimum of board space, cost, and power consumption, allowing maximum advantage from the use of dynamic RAMs.

Figure 1. Implemented Cost of Static vs. Dynamic RAM
Figure 1 shows the relative cost of static and dynamic RAM, including support circuitry, as a function of memory size, using the Intel 8202A or 8203. For any memory larger than 16KBytes, the dynamic RAM is less expensive. Since the cost of the dynamic RAM controller is relatively independent of memory size, the cost advantage for dynamic RAM increases with increasing memory size.

This Application Note will describe the techniques of interfacing a dynamic RAM memory to an iAPX-86 or iAPX-88 system using either the 8202A or 8203 dynamic RAM controller. Various configurations of the 8086 and 8088 microprocessors, and those timings which they satisfy, are described. The Note concludes with examples of particular system implementations.

**DYNAMIC RAMS**

This section gives a brief introduction to the interfacing requirements for Dynamic RAMs. Later sections will describe the operation of the Intel 8202A and 8203 Dynamic RAM Controllers.

**Device Description**

The pinout of two popular families of dynamic RAMs, the Intel 2118 and 2164A, are shown in Figure 2. The 2118 is a 16,384 word by 1-bit dynamic MOS RAM. The 2164 is a 65,536 word by 1-bit dynamic MOS RAM. Both parts operate from a single +5v supply with a ± 10% tolerance, and both use the industry standard 16-lead pinout.

The two parts are pinout-compatible with the exception of the 2164 having one extra address input (A7, pin 9); this pin is a no-connect in the 2118. Both parts are also compatible with the next generation of 256K dynamic RAMs (262,144 word by 1-bit), which will use pin 1 (presently a no-connect on both the 2118 and 2164A) for the required one extra address input (A8). This makes it possible to use a single printed circuit board layout with any of these three types of RAM.

**Addressing**

Each bit of a dynamic RAM is individually addressable. Thus, a 2164A, which contains 216 (or 65,536) bits of information, requires 16-bit addresses; similarly, the 2118, which contains 214 (or 16,384) bits, requires 14-bit addresses.

In order to reduce the number of address pins required (and thus reduce device cost), dynamic RAMs time-multiplex addresses in two halves over the same pins. Thus a 2164A needs only 8 address pins to receive 16-bit addresses, and the 2118 needs only 7 for its 14-bit addresses. The first address is called the row address, and the second is called the column address. The row address is latched internal to the RAM by the falling edge of the RAS (Row Address Strobe) control input; the column address is latched by the falling edge of the CAS (Column Address Strobe) control input. This operation is illustrated in Figure 3.

Dynamic RAMS may be visualized as a two-dimensional array of single-bit storage cells arranged across the surface of the RAM’s die. In the case of the 2164A, this array would consist of 28 (or 256) rows and 28 (or 256) columns, for a total of 216 (or 65,526) total bit cells (Figure 4). This is the source of the “row address” and “column address” terminology. Bear in mind that any given RAM may not be physically implemented as described here; for instance, the 2164A actually contains four arrays, each one 27 rows by 27 columns.
Memory Cycles

In this Application Note, we will discuss three types of memory cycles — read, write, and RAS-only refresh. Dynamic RAMs may perform other types of cycles as well; these are described in the dynamic RAM's data sheet.

Whether data is read or written during a memory cycle is determined by the RAM's WE control input. Data is written only when WE is active.

During a read cycle, the CAS input has a second function, other than latching the column address. CAS also enables the RAM data output (pin 14) when active, assuming RAS is also active. Otherwise, the data output is 3-stated. This allows multiple dynamic RAMs to have their data outputs tied in common.

During write cycles, data on the RAM data input pin is latched internally to the RAM by the falling edge of CAS or WE, whichever occurs last. If WE goes active before CAS (the usual case, called an "early write"), write data is latched by the falling edge of CAS. If WE goes active after CAS (called a "late write"), data is latched by the falling edge of WE (see Figure 5).

Late writes are useful in some systems where it is desired to start the memory cycle as quickly as possible, to maximize performance, but the CPU cannot get the write data to the dynamic RAMs quickly enough to be latched by CAS. By delaying WE, more time is allowed for write data to arrive at the dynamic RAMs.

Note that when "late write" is performed, CAS goes active while WE is still inactive; this indicates a read cycle, so the RAM enables its data output. So, if "late write" cycles are performed by a system, the RAM data inputs and data outputs must be electrically isolated from each other to prevent contention. If no "late writes" are performed, the RAM data inputs and data outputs may be tied together at the RAM to reduce the number of board traces.
Access Times

Each dynamic RAM has two different access times quoted for it — access time from RAS active (\(t_{\text{RAC}}\)) and access time from CAS active (\(t_{\text{CAC}}\)); these are illustrated in Figure 6. How do you know which to use? This depends on the timings of your RAM controller. First, the worst case delay from the memory read command active to RAS active (\(t_{\text{CR}}\)) and CAS active (\(t_{\text{CC}}\)) must be determined. Then the read data access time is the larger of the \(t_{\text{CR}}\) (Controller) + \(t_{\text{RAC}}\) (RAM) or \(t_{\text{CC}}\) (Controller) + \(t_{\text{CAC}}\) (RAM). An alternative way to determine whether to use \(t_{\text{RAC}}\) or \(t_{\text{CAC}}\) is to look at the dynamic RAM parameter for RAS active to CAS active delay, \(t_{\text{RCD}}\). \(t_{\text{RCD}}\) is a calculated value, and is shown on dynamic RAM data sheets as a reference point only. If the delay from RAS to CAS is less than or equal to \(t_{\text{RCDmax}}\), then \(t_{\text{RAC}}\) is the limiting access time parameter; if, on the other hand, the delay from RAS to CAS is greater than \(t_{\text{RCDmax}}\), then \(t_{\text{CAC}}\) is the limiting parameter. \(t_{\text{RCDmax}}\) is not an operating limit, and this spec may be exceeded without affecting operation of the RAM. \(t_{\text{RCDmin}}\), on the other hand, is an operating limit, and the RAM will not operate properly if this spec is violated.
Refresh

One unique requirement of dynamic RAMs is that they be refreshed in order to retain data. To see why this is so, we must look briefly at how a dynamic RAM is implemented.

Dynamic RAMs achieve their high density and low cost mostly because of the very simple bit-storage cell they use, which consists only of one transistor and a capacitor. The capacitor stores one bit as the presence (or absence) of charge. This capacitor is selectively accessed for reading and writing by enabling its associated transistor (see Figure 7).

Unfortunately, if left for very long, the charge will leak out of the capacitor, and the data will be lost. To prevent this, each bit-cell must be periodically read, the charge on the capacitor amplified, and the capacitor recharged to its initial state. The circuitry which does this amplification of charge is called a "sense amp". This must be done for every bit-cell every 2 ms or less to prevent loss of data.

Each column in a dynamic RAM has its own sense amp, so refresh can be performed on an entire row at a time. Thus, for the 2118, it is only necessary to refresh each of its 128 rows every 2 ms. Each row must be addressed via the RAM's address inputs to be refreshed. To simplify
refresh, the 2164A is implemented in such a way that its refresh requirements are identical to the 2118; 128 rows every 2 ms. Some other 64K RAMs require 256 row refresh every 4 ms.

Refresh can be performed by a special cycle called a RAS-only refresh, shown in Figure 8. Only a row address is sent; that row is refreshed. No column address is sent, and no data is read or written during this cycle. Intel dynamic RAM controllers use this technique.

Any read, write, or read-modify-write cycle also refreshes the row addressed. This fact may be used to refresh the dynamic RAM without doing any special refresh cycles. Unfortunately, in general you cannot be sure that every row of every dynamic RAM in a system will be read from or written to every 2 ms, so refresh cannot be guaranteed by this method alone, except in special applications.

A third technique for refresh is called hidden refresh. This method is not popular in microprocessor systems, so it is not described here, but more information is available in the dynamic RAM’s data sheet.

Three techniques for timing when refresh cycles are performed are in common use: burst refresh, distributed refresh, and transparent refresh.

Burst refresh means waiting almost 2 ms from the last time refresh was performed, then refreshing the entire memory with a “burst” of 128 refresh cycles. This method has the inherent disadvantage that during the time refresh is being performed (more than 40 microseconds for 128 rows) no read or write cycles can be performed. This severely limits the worst case response time to interrupts and makes this approach unsuitable for many systems.

As long as every row of the RAM is refreshed every 2 ms, the distribution of individual refresh cycles is unimportant. Distributed refresh takes advantage of this fact by performing a single refresh cycle every 2 ms/128, or about every 15 microseconds. In this way, the refresh requirements of the RAM are satisfied, but the longest time that read and write cycles are delayed because of refresh is minimized. Those few dynamic RAMs which use 256 row refresh allow 4 ms for the refresh to be completed, so the distributed refresh period is still 15 microseconds.

The third technique is called transparent (or “hidden” or “synchronous”) refresh. This takes advantage of the fact that many microprocessors wait a fixed length of time after fetching the first opcode of an instruction to decode it. This time is necessary to determine what to do next (i.e., fetch more opcode bytes, fetch operands, operate on internal registers, etc.); this time may be longer than the time required for a RAM refresh cycle. If the status outputs of the CPU can be examined to determine which memory cycles are opcode fetches, a refresh cycle may be performed immediately afterward (Figure 9). In this way, refresh cycles will never interfere with read or write cycles, and so appear “transparent” to the microprocessor.

Transparent refresh has the disadvantage that if the microprocessor ever stops fetching opcodes for very
long, due to a HOLD, extended DMA transfers, or when under hardware emulation, no refresh cycles will occur and RAM data will be lost. This puts restrictions on the system design. Also, high speed microprocessors do not allow sufficient time between opcode fetches and subsequent bus cycles for a complete RAM refresh cycle to be performed, so they must wait for the refresh cycle to complete before they can do a subsequent bus cycle. These microprocessors cannot use transparent refresh to any advantage. Transparent refresh is useful for microprocessors like the Intel 8085 operating at low clock frequencies.

The 8086 and 8088, however, prefetch opcodes into a queue which is several bytes long. This prefetching is independent of the actual decoding and execution of the opcodes, and there is no time at which it can be guaranteed that the 8086 or 8088 will not request a memory cycle. So transparent refresh is not applicable to these microprocessors.

The 8202A and 8203 perform distributed and/or transparent refresh. Each device has an internal timer which automatically generates a distributed refresh cycle every 15.6 microseconds or less. In addition, an external refresh request input (REFRQ) allows the microprocessor’s status to be decoded to generate a refresh cycle for transparent refresh. If, for whatever reason, no external REFREQ is generated for 15 microseconds, the internally generated refresh will take over, so memory integrity will be guaranteed.

**Arbitration**

Because RAMs cannot do a read or write cycle and a refresh cycle at the same time, some form of arbitration must be provided to determine when refresh cycles will be performed.

Arbitration may be done by the microprocessor or by the dynamic RAM controller. Microprocessor arbitration may be implemented as follows:

A counter, running from the microprocessor’s clock, is used to time the period between refresh cycles. At terminal count, the arbitration logic asserts the bus request signal to prevent the microprocessor from performing any more memory cycles. When the microprocessor responds with a bus grant, the arbitration logic generates a refresh cycle (or cycles, if burst refresh is
used). After refresh is complete, the arbitration logic releases the bus. This method has several disadvantages: First, time is wasted in exchanging bus control, which would not be required if the RAM controller did arbitration. Second, while refresh is being performed, all bus activity is stopped; for instance, even if the microprocessor is executing out of ROM at the time, it must stop until refresh is over. Third, bursts of DMA transfers must be kept very short, as refresh cannot be performed while DMA is in progress.

Some microprocessors, such as the Zilog Z-80, generate refresh cycles themselves after instruction fetches. This removes the need for external arbitration logic, but still has several disadvantages: First, DMA bursts still must be kept short to allow the CPU to do refresh. Second, this method adds to the complexity of the microprocessor, without removing the need for the RAM controller which is still required to do address multiplexing and RAS, CAS and WE timing. Microprocessor refresh can cause problems of RAM compatibility; for instance, the Z-80 only outputs a 7-bit refresh address, which means some 64K RAMs which use 256 row refresh cannot be used with the Z-80. Also, since the Z-80 refresh cycle is a fixed length (no wait states), faster speed selections of the Z-80 are not compatible with slower dynamic RAMs. Third, systems employing multiprocessing or DMA are harder to implement, because of the difficulty in insuring the microprocessor will be able to perform refresh.

It is preferable to have arbitration performed by the dynamic RAM controller itself. This method avoids all the problems described above, but introduces a complication. If the microprocessor issues a read or write command while the dynamic RAM is in the middle of a refresh cycle, the RAM controller must make the microprocessor wait until it is done with the refresh before it can complete the read or write cycle. This means that from when the microprocessor activates the read or write signal, the time until the cycle can be completed can vary over a range of roughly 200 to 700 ns. Because of this, an acknowledge signal from the dynamic RAM controller is required to tell the microprocessor the memory cycle it requested is complete. This signal goes to the microprocessor's READY logic.

**Memory Organization**

As each dynamic RAM operates on only one bit at a time, multiple RAMs must be operated in parallel to operate on a word at a time. RAMs operated in this way are called a bank of RAM. A bank consists of as many RAMs as there are bits in the memory word. When used in this way, all address and control lines are tied to all RAMs in the bank.

A single bank of RAM will provide 64K words of memory in the case of the 2164A, or 16K words in the case of the 2118. To provide more memory words, multiple banks of RAM are used. In this case, all address, CAS, and WE lines are tied to all RAMs, but each bank of RAM has its own RAS. Each bank knows whether it is being addressed during a read or write operation by whether or not its RAS input was activated — if not, then all other inputs are ignored during that cycle.

Data outputs for RAMs in corresponding bit positions in each of the banks may be tied in common, since they are 3-state outputs; even though CAS is connected to all banks of RAM, only that bank whose RAS is active will enable its data outputs in response to CAS going active. Data inputs for RAMs in corresponding bit positions in each of the banks are also tied in common.

**INTEL DYNAMIC RAM CONTROLLERS**

The Intel 8202A and 8203 Dynamic RAM Controllers each provide all the interface logic needed to use dynamic RAMs in microprocessor systems, in a single chip. Either the 8202A or 8203 allow a dynamic RAM memory to be implemented using a minium of components, board space, and power, and in less design time than any other approach.

The following sections will describe each of these controllers in detail.

**8202A**

**FUNCTIONAL DESCRIPTION**

The 8202A provides total dynamic RAM control for 4K and 16K dynamic RAMs, including the Intel 2104A, 2117, and 2118. The pinout and simplified logic diagram of the 8202A are shown in Figures 10 and 11.

The 8202A is always in one of the following states:

a) IDLE
b) TEST cycle
c) REFRESH cycle
d) READ cycle
e) WRITE cycle

The 8202A is normally in the idle state. Whenever a cycle is requested, the 8202A will leave the idle state to perform the desired cycle; if no cycle requests are pending, the 8202A will return to the idle state. A refresh cycle request may originate internally or externally to
A test cycle is requested by activating the RD and WR inputs simultaneously, independent of PCS (Protected Chip Select). The test cycle will reset the refresh address counter to zero and perform a write cycle. A test cycle should not be allowed to occur in normal system operation, as it interferes with normal RAM refresh.

A refresh cycle performs a RAS-only refresh cycle of the next lower consecutive row address after the one previously refreshed. A refresh cycle may be requested by activating the REFRQ input to the 8202A; this input is latched on the next 8202A clock. If no refresh cycles are requested for a period of about 13 microseconds, the 8202A will generate one internally. By refreshing one row every 15.6 microseconds or sooner, all 128 rows will be refreshed every 2 ms. Because refresh requests are generated by the 8202A itself, memory integrity is insured, even if the rest of the system should halt operation for an extended period of time.

The arbiter logic will allow the refresh cycle to take place only if there is not another cycle in progress at the time.

A read cycle may be requested by activating the RD input, with PCS (Protected Chip Select) active. In the Advanced Read mode, a read cycle is requested if the microprocessor's S1 status line is high at the falling edge of ALE (Address Latch Enable) and PCS is active. If a dynamic RAM cycle is terminated prematurely, data loss may result. The 8202A chip select is "protected" in that once a memory cycle is started, it will go to completion, even if the 8202A becomes de-selected.

A write cycle may be requested by activating the WR input, with PCS active; this is the same for the normal and Advanced Read modes.

**BLOCK DIAGRAM**

Let's look at the detailed block diagram in Figure 12 to see how the 8202A satisfies the interface requirements of the dynamic RAM.

**Address Multiplexing**

Address multiplexing is achieved by a 3-to-1 multiplexer
internal to the 8202A; the three inputs are the row address (AL₀₆), column address (AH₀₆), and refresh row address (generated internally). When the 8202A is in the Idle state, the multiplexer selects the row address, so it is prepared to start a memory cycle. If a refresh cycle is requested either internally or externally, the address multiplexer will select the refresh row address long enough before RAS goes active to satisfy the RAM's t_{ASR} parameter.

To minimize propagation delays, the 8202A address outputs (OUT₀₆) are inverted from the address inputs. This has no effect on RAM operation; inverters are not needed on the address outputs.

Doing this multiplexing internally minimizes timing skews between the address, RAS, and CAS, and allows higher performance than would otherwise be possible.

Refresh Counter

The next row to be refreshed is determined by the refresh counter, which is implemented as a 7-bit ripple-carry counter. During each refresh cycle, the counter is

Figure 12. 8202A Detailed Block Diagram

Figure 13. Detailed 8202A Refresh Cycle
incremented by one in preparation for the next refresh cycle (a refresh cycle is shown in detail in Figure 13).

When the 8202A enters TEST mode, the refresh counter is cleared. This feature is useful for automatic testing of the refresh counter function. Because the address outputs are inverted, the first refresh address after clearing the counter in test mode is 7FH, and the addresses decrease for subsequent refresh cycles.

**RAS Decoding**

Which bank of RAM is selected for a memory cycle is determined by the RAS decoder from the B₀₋₁ inputs, which normally come from the microprocessor address bus. The 8202A Timing Generator produces an internal RAS pulse which strobes the RAS decoder, generating the appropriate external RAS pulse. The B₀₋₁ inputs are not latched, so they must be held valid for the length of the memory cycle. During a refresh cycle, all the RAS outputs are activated, refreshing all banks at once.

**Oscillator**

The 8202A operates from a single reference clock with a frequency between 18.432 MHz and 25 MHz; this clock is used by the synchronization, arbitration, and timing generation logic. This clock may be generated by an onboard crystal oscillator, or by an external TTL-compatible clock source. When using the internal oscillator (available only on part number D8202A-1 or D8202A-3), a fundamental-mode crystal is attached to pins 36 and 37 (X₀ and X₁), as shown in Figure 14. The external TTL clock option is selected by pulling pin 36 (OP₂) to +12v through 1K ohm resistor, and attaching the clock input to pin 37 (CLK).

**Command Decoder**

The command decoder takes the commands from the bus and generates internal memory request (MEMR), and TEST signals.

The 8202A has two bus interface modes: the "normal" mode, and the "Advanced Read" mode. In the normal mode, the 8202A interfaces to the usual bus RD and WR signals.

In the Advanced Read mode, the 8202A interfaces to the Intel microprocessor bus signals ALE, S₁, and WR. S₁ must be high on the falling edge of ALE for read cycles, and WR must be low for write cycles (write cycles are the same as for normal read mode). The 8085A S₁ may be used directly by the 8202A; the 8086 and 8088 S₁ must be inverted. ALE and WR must be qualified by PCS.

The Advanced Read mode is useful for reducing read data access time, and thus wait states. This mode is used mainly with 8085A systems.

If both RD and WR are active at once (regardless of the state of PCS), the internal TEST signal is generated and the 8202A performs a test cycle as described above. One or both of RD and WR should have pull-up resistors to prevent the 8202A from inadvertently being put into test mode, as the RD and WR signals are 3-stated by the microprocessor when RESET or HOLD are active. Since the test mode resets the refresh address counter, the refresh sequence will be interrupted, and data loss may result.

**Refresh Timer and REFREQ**

The 8202A contains a counter, operated from the internal clock to time the period from the last refresh cycle. When the counter times out, an internal refresh request is generated. This refresh period is proportional to the 8202A's clock period, and varies from 10.56 to 15.625 microseconds. Even at the lowest refresh rate, all the rows of the dynamic RAM will be refreshed every 2 ms.

The 8202A has an option of reducing the refresh rate by a factor of two, for use with 4K RAMS. These RAMs have only 64 rows to refresh every 2 ms, so need refresh cycles only half as often. This option is selected by pull-
ing pin 18 (AL6/OP3) to +12v through a 5.1K ohm resistor. This pin normally serves as the high-order row address input for the address multiplexer, but it is no longer needed for this function, as 4K RAMs have one less address input.

A refresh cycle may also be requested externally by activating the REFRQ input. This input is latched, so it only needs to be held active a maximum of 20 ns. If the 8202A is currently executing a memory cycle, it will complete that cycle, and then perform the refresh cycle. The internal and external refresh requests are ORed together before going to the arbiter.

The REFRQ input cannot be used in the Advanced Read mode, as the REFRQ pin is used for ALE in this mode.

REFRQ is most often used to implement transparent refresh, as explained in the section Dynamic RAMS — Refresh. This technique is not useful in iAPX 86 and iAPX 88 systems, so REFRQ is normally tied to ground.

The refresh timer is reset as soon as a refresh cycle is started (whether it was requested internally or externally). The time between refresh cycles (tREF) is measured from when the first cycle is started, not when it was requested, which occurs sometime earlier. Of course, tREFmin does not apply if REFRQ is used — you may externally request refresh cycles as often as you wish.

**Arbiter**

This is the hardest section of a dynamic RAM controller to implement. If a read or write arrives at the same time as a refresh request, the arbiter must decide which one to service first. Also, if a read, write, or refresh request arrives when another cycle is already in progress, the arbiter must delay starting the new cycle until the current cycle is complete.

Both of the internal signals REFR (refresh request) and MEMR (memory cycle request) are synchronized by D-type master-slave flip-flops before reaching the arbiter. These circuits have been optimized to resolve a valid logic state in as short a time as possible. Of course, with any synchronizer, there is a probability that it will fail — not be able to settle in one logic state or the other in the allowed amount of time, resulting in a memory failure — but the 8202A has been designed to have less than one system memory failure every three years, based on operation in the worst case system timing environments.

Both synchronizers and the arbiter are operated from the 8202A's internal clock. Assuming the 8202A is initially in an idle state, one full clock period after the synchronizers sample the state of the MEMREQ and REFRQ signals, the arbiter examines the REFR and MEMR outputs of the synchronizers. If MEMR is active, the arbiter will activate START to begin the memory cycle (either read or write) on that clock. If REFR is active (regardless of the state of MEMR), the arbiter will activate START and REF to begin a refresh cycle on that clock. Once the cycle is complete, the Cycle Timing Generator will generate an end-of-cycle (EOC) signal to clear the arbiter and allow it to respond to any new or pending requests on the next clock.

Once a memory cycle is started, it cannot be stopped, regardless of the state of the RD/S1, WR, ALE, or PCS inputs. This is necessary, as ending a dynamic RAM cycle prematurely may cause loss of data. Note, however, that the RAM WE output is directly gated by the WR input, so if WR is removed prematurely, the RAM WE pulse-width spec (tWP) may be violated, causing a memory failure.

What happens if a memory request and refresh request occur simultaneously?

If the 8202A is in the idle state, the memory request will be honored first.

If the 8202A is not in the idle state (a memory or refresh cycle is in progress) then the memory cycle will lose priority and the refresh cycle will be honored first.

Remember, if the 8202A is performing a cycle, the arbiter doesn’t arbitrate again until the end of that cycle. So the memory and refresh cycles are “simultaneous” if they both happen early enough to reach the arbiter before it finishes the current cycle. This arbitration arrangement gives memory cycles priority over refresh cycles, but insures that a refresh cycle will be delayed at most one RAM cycle.

**Refresh Lock-Out**

As a result of the 8202A operation, transparent refresh circuits like the one shown in Figure 15 should not be used. This circuit uses the RD input, with some qualifying logic, to activate REFRQ whenever the microprocessor does an opcode fetch. This circuit will work fine, as long as the 8202A never has to generate an internal refresh request, which is unlikely (if nothing else, the system RESET pulse is probably long enough that the 8202A will throw in a couple of refreshes while the microprocessor is reset). If the 8202A ever does generate its own refresh, there is a probability that the microprocessor will try to fetch an opcode while the
refresh is still in progress. If that happens, the 8202A will finish the refresh, see both the RD and REFRQ inputs active, honor the REFRQ first, and start a second refresh. In the meantime, the microprocessor is sitting in wait states, waiting for the 8202A to complete the opcode fetch. When the 8202A finishes the second refresh, it will see both RD and REFRQ active again, and will start a third refresh, etc. The system "locks up" with the microprocessor sitting in wait states ad infinitum, and the 8202A doing one refresh cycle after another.

When the cycle is complete, the Cycle Timing Generator sends an end-of-cycle (EOC) pulse to the arbiter to enable it to respond to new or pending cycle requests.

Minimum and maximum values for the 8202A parameters tCR (Command to RAS active delay) and tCC (Command to CAS active delay) differ by one 8202A clock period. This is because the commands (RD, WR, ALE) must be synchronized to the 8202A's clock; this introduces a ± one clock period (tp) uncertainty due to the fact that the command may or may not be sampled on the first clock after it goes active, depending on the set-up time. If RD or ALE and WR are synchronous to the 8202A's clock, and the set-up time (tsd) is met, the smaller number of clock periods will apply.

All 8202A output timings are specified for the capacitive loading in the data sheet. Typical output characteristics are shown in the data sheet for capacitive loads ranging from 0 to 660 pF, these can be used to calculate the effect of different loads than those specified in the data sheet on output timings. All address, RAS, CAS, and WE drivers are identical, so these characteristic curves apply to all outputs.

**SACK AND XACK**

Because refresh cycles are performed asynchronously to the microprocessor's operation (except during transparent refresh), the microprocessor cannot know when it activates RD or WR if a refresh cycle is in progress, and therefore, it can't know how long it will take to complete the memory cycle.

This added consideration requires an acknowledge or "handshake" signal from the 8202A to tell the microprocessor when it may complete the memory cycle. This acknowledge would be used to generate the microprocessor's READY input — the microprocessor will sit in wait states until the 8202A acknowledges the memory cycle. Two signals are generated for this purpose by the 8202A; they are called system acknowledge (SACK) and transfer acknowledge (XACK). They serve the same purpose but differ in timing.

XACK is a Multibus-compatible signal, and is not activated until the read or write cycle has been completed by the RAMs. In a microprocessor system, however, there is a considerable delay from when the 8202A acknowledges the memory cycle until the microprocessor actually terminates the cycle. This delay is due to the time required to combine this acknowledge with other sources of READY in the system, synchronize READY to the microprocessor's clock, sample the state of READY, and respond to an active READY signal. As a result, more wait states than necessary may actual-
Figure 17. 8202A Timing Relative To CLK
ly be generated by using XACK. SACK is activated earlier in the cycle to improve performance of microprocessors by compensating for the delays in the microprocessor responding to XACK, and thus eliminating unneeded wait states which might be generated as a result of XACK timing. The system designer may use one or the other acknowledge signal, or use both in different parts of the system, at his option.

SACK and XACK are activated by the Cycle Timing Generator, but they can be de-activated only by the microprocessor removing its RD or WR request, or by activating ALE when in the advanced read mode. As the SACK and XACK signals are used to generate READY for the microprocessor, this is necessary to give the microprocessor as much time as it needs to respond to its READY input.

Delayed SACK Mode

SACK may be activated at one of two different times in the memory cycle; the earlier case is called "normal SACK" and the later is called "delayed SACK" (Figure 18). Delayed SACK occurs if the memory request was received by the 8202A while it was doing a refresh cycle. In this case, the memory cycle will be delayed some length of time while the refresh cycle completes; SACK is delayed to ensure the microprocessor will generate enough wait states. This is a concern mostly for read cycles.

Because of the way the delayed SACK mode is implemented in the 8202A, if the RD or WR input is activated while a refresh cycle is in progress, regardless of whether or not the 8202A is chip-selected, the internal delayed SACK mode flip-flop will be set. The next 8202A memory cycle will have SACK delayed, even if that cycle was not actually delayed due to a refresh cycle in progress. The delayed SACK flip-flop will be reset at the end of that cycle, and the 8202A will return to normal SACK operation. The same thing happens in Advanced Read mode if S1 is high at the falling edge of ALE during a refresh cycle, once again regardless of the state of PCS.

8203

The 8203 is an extension of the 8202A architecture which allows the use of 64K dynamic RAMs. It is pinout compatible with the 8202A and shares identical A.C. and D.C. parameters with that part. The description of the 8202A applies to this part also, with the modifications below.

ENHANCEMENTS

1. Supports 16K or 64K dynamic RAMs. 4K RAM mode, selected by pulling AL6/OP3 (pin 18) to +12v, is not supported.

2. Allows a single board design to use either 16K or 64K RAMs, without changing the controller, and only making between two and four jumper changes to reconfigure the board.

3. May operate from external TTL clock without the +12v pull-up which the 8202A requires (a +5v or +12v pull-up may be used).

The pinout of the 8203 is shown in Figure 19. This pinout is identical to the 8202A, with the exception of the five highlighted pins. The function of these is described below. The simplified block diagram is similar to the 8202A's, in Figure 11.

![Figure 18. Delayed SACK Mode](5-162)
The goal of the 8203 is to provide a pin- and timing-compatible upgrade of the 8202A for use with 64K RAMs. The difficulty in doing this is that 64K RAMs require an additional address input compared to 16K RAMs, and thus the 8203 needs three more pins (one more RAM address output, and two more inputs to its internal address multiplexer). Since all but one of the 8202A's pins are already used, this is clearly a challenge — some functionality must be sacrificed to gain 64K RAM support. The 8203 reduces the maximum number of banks supported from four to two for 64K RAMs.

Pin 35 (16K/64K) is used to tell the 8203 whether it is being used to control 16K RAMs or 64K RAMs. When tied to Vcc or left unconnected, the 8203 operates in the 16K RAM mode; in this mode all the remaining pins function identically to the 8202A. When tied to ground, it operates in the 64K RAM mode, and pins 23 through 26 change function to enable the 8203 to support 64K RAMs. Pin 35 (16K/64K) contains an internal pull-up — when unconnected, this input is high, and the 8203 operates identically to the 8202A. This maintains pinout compatibility with the 8202A, in which pin 35 is a no-connect, so the 8203 may be used in 8202A sockets with no board modifications.

When the 8203 is in the 64K RAM mode, four pins change function, as shown in Table 2. The pins change function in this particular way to allow laying out a board to use either 16K or 64K RAMs with a minimum of jumpers, as shown in Figure 20. This figure shows the 8203 with two banks of RAM. Banks 0 and 1 may be either 16K RAMs or 64K RAMs; banks 2 and 3 may only be 16K RAMs, as the 8203 supports two banks of 64K RAM. For clarity, only those connections which are important in illustrating the 8203 jumper options are shown.

![Fig. 19 8203 Pinout](image-url)

**16K Mode and 64K Mode**

The 8203 Pinout Table:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vcc</td>
</tr>
<tr>
<td>2</td>
<td>AH6</td>
</tr>
<tr>
<td>3</td>
<td>AH5</td>
</tr>
<tr>
<td>4</td>
<td>AH4</td>
</tr>
<tr>
<td>5</td>
<td>X1/CLK</td>
</tr>
<tr>
<td>6</td>
<td>X2/OP2</td>
</tr>
<tr>
<td>7</td>
<td>OUT0</td>
</tr>
<tr>
<td>8</td>
<td>OUT1</td>
</tr>
<tr>
<td>9</td>
<td>OUT2</td>
</tr>
<tr>
<td>10</td>
<td>OUT3</td>
</tr>
<tr>
<td>11</td>
<td>OUT4</td>
</tr>
<tr>
<td>12</td>
<td>OUT5</td>
</tr>
<tr>
<td>13</td>
<td>OUT6</td>
</tr>
<tr>
<td>14</td>
<td>OUT7</td>
</tr>
<tr>
<td>15</td>
<td>OUT8</td>
</tr>
<tr>
<td>16</td>
<td>OUT9</td>
</tr>
<tr>
<td>17</td>
<td>OUT10</td>
</tr>
<tr>
<td>18</td>
<td>OUT11</td>
</tr>
<tr>
<td>19</td>
<td>OUT12</td>
</tr>
<tr>
<td>20</td>
<td>OUT13</td>
</tr>
<tr>
<td>21</td>
<td>OUT14</td>
</tr>
<tr>
<td>22</td>
<td>OUT15</td>
</tr>
<tr>
<td>23</td>
<td>OUT16</td>
</tr>
<tr>
<td>24</td>
<td>OUT17</td>
</tr>
<tr>
<td>25</td>
<td>OUT18</td>
</tr>
<tr>
<td>26</td>
<td>OUT19</td>
</tr>
<tr>
<td>27</td>
<td>OUT20</td>
</tr>
<tr>
<td>28</td>
<td>AL0</td>
</tr>
<tr>
<td>29</td>
<td>AL1</td>
</tr>
<tr>
<td>30</td>
<td>AL2</td>
</tr>
<tr>
<td>31</td>
<td>AL3</td>
</tr>
<tr>
<td>32</td>
<td>AL4</td>
</tr>
<tr>
<td>33</td>
<td>AL5</td>
</tr>
<tr>
<td>34</td>
<td>AL6</td>
</tr>
<tr>
<td>35</td>
<td>AL7</td>
</tr>
<tr>
<td>36</td>
<td>AL8</td>
</tr>
<tr>
<td>37</td>
<td>AL9</td>
</tr>
<tr>
<td>38</td>
<td>AL10</td>
</tr>
<tr>
<td>39</td>
<td>AL11</td>
</tr>
<tr>
<td>40</td>
<td>AL12</td>
</tr>
<tr>
<td>41</td>
<td>AL13</td>
</tr>
<tr>
<td>42</td>
<td>AL14</td>
</tr>
<tr>
<td>43</td>
<td>AL15</td>
</tr>
<tr>
<td>44</td>
<td>AL16</td>
</tr>
<tr>
<td>45</td>
<td>AL17</td>
</tr>
<tr>
<td>46</td>
<td>AL18</td>
</tr>
<tr>
<td>47</td>
<td>AL19</td>
</tr>
<tr>
<td>48</td>
<td>AL20</td>
</tr>
</tbody>
</table>

**Figure 20. 8203 Jumper Options**
Table 2. 16K/64K Mode Selection

<table>
<thead>
<tr>
<th>Pin #</th>
<th>16K Function</th>
<th>64K Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>RAS₂</td>
<td>Address Output (OUT₇)</td>
</tr>
<tr>
<td>24</td>
<td>Bank Select (B₀)</td>
<td>Address Input (AL₇)</td>
</tr>
<tr>
<td>25</td>
<td>Bank Select (B₁)</td>
<td>Address Input (AH₇)</td>
</tr>
<tr>
<td>26</td>
<td>RAS₃</td>
<td>Bank Select (B₀)</td>
</tr>
</tbody>
</table>

Jumpers J1-J4 may be used to chip select the 8203 over various address ranges. For example, if two banks of 16K RAMs are replaced with two banks of 64K RAMs, the address space controlled by the 8203 increases from 32K words to 128K words. If four banks of 16K RAMs are replaced with one bank of 64K RAMs, no chip select jumers are needed.

In the 64K RAM mode, pins 24 and 25 (B₀(AL₇) and B₁(AH₇)) change function from bank select inputs to address inputs for the 64K RAM. Since the bank select inputs normally come from the address bus anyway, no jumper changes are required here. The bank select function moves to pin 26 (RAS₃(B₀)); since only two bank of 64K RAM is supported, only one bank select input is needed in this mode, not two. Jumpers J6 and J7 are shorted in the 64K RAM mode to connect pin 26 (B₀) to the address bus. In the 16K RAM mode, these jumpers must be disconnected, as pin 26 junctions as the RAS₃ output; in the 64K RAM mode, this bank is not populated, so RAS₃ is not needed.

Pin 23 serves two functions: in the 16K RAM mode it is the RAS output for bank 2 (RAS₂), in the 64K RAM mode is the high order RAM address output (OUT₇), which goes to pin 9 of the 64K RAMs. This requires no jumpers as when using 16K RAMs, pin 9 is a no-connect, and when using 64K RAMs, bank 2 is depopulated, so RAS₂ is not used.

This arrangement allows converting a board from 16K RAMs to 64K RAMs with no change to the controller and changing a maximum of three jumpers.

+5v External Clock Option

Just as with the 8202A, the user has the option of an external TTL clock instead of the internal crystal oscillator as the timing reference for the 8203; unlike the 8202A, he does not need to tie pin 36 (X₀/OP₂) to +12v to select this option—this pin may be tied to either +5v or +12v. If pin 36 is tied to +12v, a 1K ohm (±5%) series resistor must be used, just as for the 8202A. If pin 36 is tied to +5v, it must be tied directly to pin 40 (Vᵲᵅ) with no series resistor. This is because pin 36 must be within one Schottky diode voltage drop (roughly 0.5v) of pin 40 to select the external TTL clock option; a series resistor may cause too great a voltage drop for the external clock option to be selected. For the same reason, the trace from pin 36 to 40 should be kept as short as practical.

Test Cycle

An 8203 test cycle is requested by activating the RD, WR, and PCS inputs simultaneously. By comparison, an 8202A test cycle requires activating only the RD and WR inputs simultaneously, independent of PCS. Like the 8202A, and 8203 test cycle resets the address counter to zero and performs a write cycle.
BLOCK DIAGRAM

A simplified block diagram of the 8203 is shown in Figure 21. It is identical to the 8202A except for the following differences:

1. The 3:1 address multiplexer is 8 bits wide, instead of 7 bits wide, to support the addressing requirements of the 64K RAM.
2. The refresh address counter is 8 bits. This allows it to support RAMs which use either the 128-row or 256-row refresh schemes. Regardless of which type of RAM is used, the refresh counter cycles through 256 rows every 4 ms. RAMs which use 128-row refresh treat the eighth address bit as a “don’t care” during refresh, so they see the equivalent of 128-row refresh every 2 ms. In either case the rate of internally-generated refresh cycles is the same—at least one every 15.6 microseconds.

INTEL iAPX-86 AND iAPX-88

Device Descriptions

The iAPX-86 and iAPX-88 are advanced 16-bit microprocessor families, based on the 8086 and 8088 microprocessors, respectively. While both have a similar architecture and are software compatible, the 8086 transfers data over a 16-bit bus, while the 8088 uses an 8-bit data bus (but has a 16-bit internal bus).

Min and Max Modes

In order to support the widest possible range of applications, the 8086 and 8088 can operate in one of two modes, called minimum and maximum modes. This allows the user to define certain processor pins to “tailor” the 8086 or 8088 to the intended system. These modes are selected by strapping the MN/MX (minimum/maximum) input pin to Vcc or ground.

In the minimum mode, the microprocessor supports small, single-processor systems using a minimum of components. In this mode, the 8086 or 8088 itself generates all the required bus control signals (Figure 22).

In the maximum mode, the microprocessor supports larger, higher performance, or multiprocessing systems. In this mode, the 8086 or 8088 generates status outputs which are decoded by the Intel 8288 Bus Controller to provide an extensive set of bus control signals, and Multibus compatibility (Figure 23). This allows higher performance RAM operation because the memory read and write commands are generated more quickly than is possible in the minimum mode. The maximum mode is the one most often used in iAPX-86 and iAPX-88 systems.

Figure 22. 8086 Minimum Mode
The Alternate Configuration is not an operating mode of the 8086 or 8088 per se, but uses TTL logic along with the status outputs of the microprocessor to generate the RAM read and/or write control signals (Figure 24). The alternate configuration may be used with the microprocessor in either minimum or maximum mode. This configuration is advantageous because it activates the memory read and write signals even earlier than the maximum mode, leading to higher performance. It is possible to generate either the RAM read or write signal using this configuration, and generate the other RAM control signal using the min or max mode in the normal configuration.

Each of the three system configurations may be used with buffers on the address, data, or control bus for increased electrical drive capability.

**Performance vs. Wait States**

Before starting a discussion of timing analyses, it’s worthwhile to look at the effect of wait states on the iAPX-86 and iAPX-88.
For most microprocessors, the effect of, say, one wait state on execution times is straightforward. If a bus cycle normally is three clocks long, adding a wait state to every bus cycle will make all bus cycles four clocks, decreasing performance by 33%. This is multiplied by the percentage of time that the microprocessor is doing bus cycles (some instructions take a long time to execute, so the microprocessor skips a few bus cycles).

The effect of wait states on the iAPX-86 and iAPX-88 is not so straightforward, however.

The 8086 and 8088 microprocessors consist of two processing units: the execution unit (EU) executes instructions, and the bus interface unit (BIU) fetches instructions, reads operands, and writes results. During periods when the EU is busy executing instructions, the BIU "looks ahead" and fetches more instructions from the next consecutive addresses in memory; these are stored in an internal queue. This queue is four bytes long for the 8088 and six bytes long for the 8086; under most conditions, the BIU can supply the next instructions without having to perform a memory cycle. Only when the program doesn't proceed serially (e.g. a Jump or Call instruction) does the EU have to wait for the next instruction to be fetched from memory. Otherwise, the instruction fetch time "disappears" as it is proceeding in parallel with execution of previously fetched instructions. The EU then has to wait for the BIU only when it needs to read operands from memory or write results to memory. As a result, the 8086 and 8088 are less sensitive to wait states than other microprocessors which don't use an instruction queue. The effect of wait states on 8086 execution time compared to the Motorola 68000 and Zilog Z8000 for a typical mix of software is summarized in Table 3.[1]

### Table 3. Effects of Wait States on Execution Time

<table>
<thead>
<tr>
<th>Processor</th>
<th>Execution Time Increase Over 0 Wait State Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 Wait State</td>
</tr>
<tr>
<td>iAPX 86/10 (measured)</td>
<td>8.3%</td>
</tr>
<tr>
<td>Z8000 (computed)</td>
<td>19.1%</td>
</tr>
<tr>
<td>68000 (computed)</td>
<td>15.9%</td>
</tr>
</tbody>
</table>

The BIU can fetch instructions faster than the EU can execute them, so wait states only affect performance to the extent that they make the EU wait for the transfer of operands and results. How much this affects program execution time is a function of the software; programs that contain many complex instructions like multiplies and divides and register operations are slowed down less than programs that contain primarily simple instructions. The effect of wait states on the 8086 and 8088 is always less than on other microprocessors which don't use an instruction queue.


---

**Figure 25. 8086 Max Mode System**
Timing Analysis

This section will look at two specific system configurations to show how the 8203 timing requirements are satisfied by the 8086. Methods of determining the worst case number of wait states for the various configurations are also given.

The timings of the 8202A and 8203 are identical; only the 8203 is referred to for the remainder of this note, but all comments apply equally to the 8202A. All timings are worst case over the range of $T_A = 0 - 70^\circ C$ and $V_{CC} = +5v \pm 10\%$ for the test conditions given in the devices’ data sheets.

Example 1. 8086 Max Mode System (5 MHz)

This example (Figure 25) is representative of a typical medium-size microprocessor system. Example 1 requires one wait state (worst case) for memory cycles. Example 2 also uses an 8086 in Max mode at 5 MHz, but uses external logic to reduce the number of wait states to zero for both read and write cycles.

DYNAMIC RAM INTERFACE

First, look at the timing requirements of the dynamic RAM to ensure they are satisfied by the 8203. Memory compatibility timings are shown in the 8203 data sheet (Figure 26). Seven 8203 timings are given, not counting $t_{AD}$, which will be discussed in the next section. These timings are summarized in Table 4.

![Figure 26. Memory Compatibility Timing](image-url)

Table 4. Memory Compatibility Timings
(all parameters are minimums)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{ASC}$</td>
<td>Column Address Set-Up Time</td>
<td>$t_p$ - 30</td>
</tr>
<tr>
<td>$t_{ASR}$</td>
<td>Row Address Set-Up Time</td>
<td>$t_p$ - 30</td>
</tr>
<tr>
<td>$t_{CAH}$</td>
<td>Column Address Hold Time</td>
<td>$5t_p$ - 30</td>
</tr>
<tr>
<td>$t_{CAS}$</td>
<td>CAS Pulse Width</td>
<td>$5t_p$ - 10</td>
</tr>
<tr>
<td>$t_{RAH}$</td>
<td>Row Address Hold Time</td>
<td>$t_p$ - 10</td>
</tr>
<tr>
<td>$t_{RCD}[1]$</td>
<td>RAS to CAS Delay Time</td>
<td>$2t_p$ - 40</td>
</tr>
<tr>
<td>$t_{RSH}$</td>
<td>RAS Hold Time from CAS</td>
<td>$5t_p$ - 30</td>
</tr>
</tbody>
</table>

$[1]t_{RCD\text{min}} = t_{RAH\text{min}} + t_{ASC\text{min}} = 2t_p - 40$

This parameter is the minimum RAS active to CAS active delay.

These timings are all a function of the 8203’s clock period ($t_p$); they may be adjusted to be compatible with slower dynamic RAMs by slowing the 8203’s clock (increasing $t_p$). The frequency of the 8203’s clock may be varied from 18.432 MHz to 25 MHz; for best performance, the 8203 should be operated at the highest possible frequency compatible with the chosen dynamic RAM. In most cases, $t_{RAH}$ or $t_{CAS}$ will be the frequency limiting parameter, but the 8203 can operate at its maximum frequency with most dynamic RAMs available.

$t_{ASR}$ applies only to refresh cycles. When the 8203 is in the Idle state (not performing any memory or refresh cycles) the address multiplexer allows the AL0:7 inputs (the RAM row address) to propagate through to the 8203 OUT0:7 pins, which are connected to the RAM address pins. So in read or write cycles, the row address will propagate directly from the address bus to the
RAM; the row address set-up time in this case is determined by the microprocessor's timing (see the next section). At the beginning of a refresh cycle, the 8203 has to switch its internal multiplexer to direct the refresh row address to the RAMs before activating RAS; the $t_{\text{ASR}}$ parameter in Table 4 refers to this case only.

Assume the Intel 2164A-20 RAM (200 ns access time) is used. Equations 1(a)-(h) show that this RAM is compatible at the 8203's maximum operating frequency of 25 MHz ($t_p = 1/(25 \text{ MHz}) = 40 \text{ ns}$). This frequency will be used for now; once the rest of the system timings are calculated, the minimum 8203 frequency which will provide the same system performance can also be determined.

\[
\begin{align*}
(\text{a}) & \quad t_{\text{ASC}} = t_p - 30 = 10 \quad \text{(Equation 1.)} \\
(\text{b}) & \quad t_{\text{ASR}} = t_p - 30 = 10 \\
(\text{c}) & \quad t_{\text{CAH}} = 5t_p - 30 = 170 \\
(\text{d}) & \quad t_{\text{CAS}} = 5t_p - 10 = 190 \\
(\text{e}) & \quad t_{\text{RAH}} = t_p - 10 = 30 \\
(\text{f}) & \quad t_{\text{RCD}}^{[1]} = 2t_p - 40 = 40 \\
(\text{g}) & \quad t_{\text{RP}} = 4t_p - 30 = 130 \\
(\text{h}) & \quad t_{\text{RSH}} = 5t_p - 30 = 170 \\
\end{align*}
\]

\[t_{\text{RCDmin}} = t_{\text{RAHmin}} + t_{\text{ASCmin}} = 2t_p - 40\]

[1] May be calculated as 

\[
t_{\text{RCDmin}} = t_{\text{RAHmin}} + t_{\text{ASCmin}} = 2t_p - 40
\]

ADDRESS SET-UP AND HOLD TIME MARGINS

The microprocessor must put the memory address on the address bus early enough in the memory cycle for it to pass through the 8203 and meet the row address set-up time to RAS ($t_{\text{ASR}}$) requirement of the dynamic RAM (Figure 27). Since the address propagates directly through the 8203, this set-up time is a function of how long the microprocessor holds the address on the bus before activating the RD or WR command, the address delay through the 8203 ($t_{\text{ADmax}}$), and how long the 8203 waits before activating RAS ($t_{\text{CRmin}}$). This is shown in Figure 28, and calculated in Equation 2. This and all following equations show timing margins; a positive result indicates extra margin, a zero result says the parameter is just met, and a negative result indicates it is not met for worst-case conditions.

Row Address Set-Up Time Margin (Equation 2.)

\[
= \text{CPU Address to RD Delay + RAS Active Delay - Address Delays}
\]

\[
= T_{\text{CLC}}(5\text{MHz}) + T_{\text{CLML min}}(8288) + t_{\text{CRmin}}(8203) - [\text{Greater of } T_{\text{CLAVmax}}(8086) + T_{\text{IVOVmax}}(8282) \text{ or } T_{\text{CLHmax}}(8288) + T_{\text{SHVmax}}(8282)] - t_{\text{ADmax}}(8203) - t_{\text{ASR}}(2164A - 20)
\]

\[
= [200 + 10 + [40 + 30] - [\text{Greater of (110 + 30) or (15 + 45)] - 40 - 0
\]

\[
= 100
\]

![Figure 27. Address Set-Up and Hold Time Margins](image-url)
Figure 28. Address Set-up Time Margin
Figure 29. Address Hold Time Margin
Similarly, the microprocessor must maintain the memory address long enough to satisfy the column address hold time \( t_{\text{CAH}} \) of the RAM; the 8203 \( T_{\text{ADmin}} \) parameter should be used for this calculation.

More importantly, the 8203 bank select \( B_0 \) inputs are also not latched; these are used directly to decode which RAS output is activated during read or write cycles, so these inputs must be held valid until RAS goes inactive. Since \( B_0 \) are usually taken directly from the address bus, this determines the address hold time required of the system (Figure 29). These are easily satisfied by the 8086 as shown by Equation 3. \( N \) represents the number of wait states. This equation can be tried with various values for \( N \) (starting with 0 and increasing) until the equation is satisfied, or it can be set equal to zero (meaning no excess margin remains) and solved for \( N \) directly; the fractional value for \( N \) that results must be rounded up to get the worst-case number of wait states to satisfy this particular parameter. No wait states are required to meet address hold times.

Address Hold Time Margin \( (N = 0) \) (Equation 3.)

\[
\text{CPU Address Hold Time, from RD Active - RAS Inactive Delays} = (3 + N)T_{\text{CLL}}(5\text{MHz}) + T_{\text{CLLHmin}}(8288) + T_{\text{SHOVmin}}(8282) - T_{\text{Hmax}}(8203) - T_{\text{RSHmax}}(8203) = 3(200) + 2 + 10 - 35 - [4(40) + 85] - [5(40) + 30] = 102
\]

READ DATA ACCESS TIME MARGIN

Read data access times determine how many wait states are required for read cycles. Remember that dynamic RAMs have two access time parameters, RAS access time \( t_{\text{RAC}} \) and CAS access time \( t_{\text{CAC}} \). Either one may be the limiting factor in determining RAM access time, as explained in the section Dynamic RAM - Access Times, above. Here \( t_{\text{CAC}} \) is the limiting factor, as

\[
t_{\text{CAC}} + t_{\text{RAC}} \geq t_{\text{RSH}}.
\]

This timing is shown in Figures 30 and 31, and is calculated in Equation 4. In this system, one wait state is required to satisfy the read data access time requirements of the system; the margin is -50 ns, which is too large a difference to be made up by using a faster RAM.

\[\text{[1]} \quad \text{Not specified — use } 2 \text{ ns} \]
\[\text{[2]} \quad \text{Not specified in } 8203 \text{ data sheet; } t_{\text{RSHmax}}(8203) = 5t_p + 30\]
Figure 31. Read Data Access Time Margin
Read Data Access  
(Equation 4.)

**Time Margin (N = 0)**

\[

t = CPU \text{ RD Active to Data Valid Delay} - \\
CAS \text{ Active Delay} - \text{Data Delays}
\]

\[
\begin{align*}
  &= (2 + N)TCCL(5MHz) - TCLMLmax(8288) - \\
  &\quad t_{CCmax}(8023) - t_{CAcmax}(2164A-20) - \\
  &\quad t_{pmax}(74S373)^{[1]} - TIVOVmax(8286) - \\
  &\quad TCLCLmin(8086) - \\
  &= 2(200) - 35 - [4(40) + 85] - 110 - \\
  &\quad 30^{[1]} - 30 - 30 \\
  &= -80 \Rightarrow 1 \text{ wait state needed (N = 1)}
\end{align*}
\]

**WRITE DATA SET-UP AND HOLD TIME MARGINS**

In write cycles, the write data must

1. reach the dynamic RAMs long enough before \( \overline{CAS} \) to meet the RAM's data set-up time parameter, \( t_{DS} \) (Figures 32 and 33), and

2. be held long enough after \( \overline{CAS} \) to meet the RAM's data hold time parameter \( t_{DH} \) (Figures 32 and 34).

Data set-up time margin is calculated in Equation 5, and data hold time margin is given in Equation 6. Again, these are margins, so a positive number indicates that system timing requirements are met for worst-case timings. Data hold time is a function of the number of 8086 wait states, represented as \( N \), as is the read data access time margin. No wait states are required to meet this parameter.

---

**Write Data Set-Up Time Margin**  
(Equation 5.)

\[

t = CPU \text{ WR Active to Data Valid Delay} + \\
CAS \text{ Delay} - Data Delay
\]

\[
\begin{align*}
  &= TCLMLmin(8288) + t_{CCmin}(8203) - \\
  &\quad TCDVmax(8086) - TIVOVmin(8286) - \\
  &\quad t_{pmin}(2164A-20) - \\
  &= 10 + [3(40) + 25] - 110 - 30 - 0 \\
  &= 15
\end{align*}
\]

**Write Data Hold Time**  
(Equation 6.)

Margin (\( N = 0 \))

\[

t = CPU \text{ Data Hold Time, from AMWC} + \\
Data Delays - CAS \text{ Active Delay} + \\
(2 + N)TCCL(5MHz) + TCLCHmin(8284A) + \\
TCHDmin(8086) + TIVOVmin(8286) - \\
TCLMLmax(8288) - t_{CCmax}(8023) - \\
t_{DHmin}(2164A-20) - \\
\begin{align*}
  &= 2(200) + [\frac{3}{2}(200) - 15] + 10 \\
  &\quad + 5 - 35 - [4(40) + 85] - 45 \\
  &= 308
\end{align*}
\]

\(^{[1]}\) \( t_{p74S373} \) is the greater of \( t_{pHL} \) (from data) or \( t_{pLH} \) (from data) and is compensated for \( V_{CC} \) and temperature variations, and is derated for a 300pF load (T.I. spec is at 15pF).

\[
 t_{p74S373} = 13ns + 0.05ns/pF(300 - 15)pF \\
+ 2.75ns = 30ns
\]

Where 13ns is T.I. spec value

0.05ns/pF is derating factor for excess capacitive load

(300 - 15) is excess capacitive load

2.75ns is compensation for \( T_A \) and \( V_{CC} \) variation

---

**Figure 32. Write Data Set-Up and Hold Time Margins**
SACK SET-UP TIME MARGIN

As explained earlier, SACK (and XACK) are "hand-shaking" signals used to tell the microprocessor when it may terminate the bus cycle in progress. Thus, SACK timing determines how many wait states will be generated, as opposed to how many wait states are actually required for proper operation, which is determined by the read data access time for read cycles and by the write data hold time for write cycles. If SACK causes more wait states than are required, there is a performance penalty, but the system operates; if too few wait states are generated, the system will not function.

SACK and XACK serve the same function; they differ only in timing. XACK is Multibus compatible, and is activated only when the read data is actually on the bus (in a read cycle) or when the write data has been latched into the RAM (in a write cycle). SACK is activated earlier in the memory cycle than XACK to compensate for delays in the microprocessor responding to this signal to terminate the cycle. Use of SACK is normally preferable, as it results in the fewest possible wait states being generated. But in some systems, SACK will not generate a sufficient number of wait states, so XACK or a delayed form of SACK must be used. Note that the number of wait states generated by SACK and XACK will vary, depending on whether a refresh cycle is in progress when the memory cycle was requested, and if a refresh cycle is in progress, how near it is to completion. SACK is sampled by the 8284A Clock Generator Chip's RDY1 or RDY2 input. The 8284A can be programmed to treat these inputs as either synchronous or asynchronous inputs by tying its ASYNC input (pin 15) either high or low, respectively. SACK must be treated as asynchronous unless it has been synchronized to the microprocessor's clock with an external flip-flop.

SACK set-up time is shown in Figures 35 and 36, and is calculated in Equation 7. This equation indicates that, at worst case, one wait state will be generated (N = 1). This satisfies the requirements of the system, namely one wait state for reads and zero (or more) wait states for writes.

\[
\text{SACK Set-Up Time Margin (N = 0) \quad (Equation 7.)}
\]

\[
= \text{RD or WR Active to SACK Active Delay}
\]

\[
= (N)TCLCL(5MHz) + t_{PLHmin}(7404)[1] - TCLMLmax(8288) - t_{CAmax}(8203) - t_{Smin}(74S74)
\]

\[
= 0 + 1 - 35 - [2(40) + 47] - 3
\]

\[
= -164 \rightarrow 1 \text{ wait state will be generated (N = 1)}
\]

We have only looked at "worst case" SACK set-up time so far, to determine the maximum number of wait states that will be generated (assuming no delays due to a refresh cycle in progress). We should look at "best

Figure 34. Write Data Hold Time Margin
Figure 35. SACK Set-Up Time Margin

Figure 36. SACK Set-Up Time Margin
case" SACK timing also, to make sure enough wait states are always generated. Note that in Figure 35, SACK goes through an external 74S74 flip-flop; this samples SACK on-half clock cycle earlier than the 8284A does (on the same clock edge that activates MRDC or AMWC), effectively reducing SACK set-up time by one-half clock period. This guarantees the proper number of wait state will be generated for "best case" SACK timing. Adding this flip-flop does not increase the worst case number of wait states generated by SACK.

In the case where a memory cycle is requested while a refresh cycle is in progress, the memory cycle will be delayed by a variable amount of time, depending on how near the refresh cycle is to completion. This delay may be as long as one full memory cycle if the refresh was just starting; this time is about 650 ns, depending on the 8203's clock frequency. SACK set-up, read data set-up, and write data hold times to the microprocessor's clock are not the same as in the usual case where there is no refresh interference. In this case, SACK is delayed until the read or write cycle has been completed by the RAM, so that there is no possibility of terminating the cycle too soon.

**PCS SET-UP TIME MARGIN**

The 8203's RD, WR, and ALE inputs must be qualified by PCS in order to perform a memory cycle. If the PCS active set-up time parameter (tPCS) is violated, the memory cycle will be delayed. In this case all maximum delays normally measured from command (tCR, tCC, tCA) will be measured instead from PCS active and will be increased by tPCS (20 ns). Minimum tCR, tCC, tCA delays remain the same, but are measured from command or PCS whichever goes active later. If tPCS is violated, care must be taken that PCS does not glitch low while RD, WR, or ALE is active, erroneously triggering a memory cycle. tPCS is not violated in this system, however (Equation 8).

**PCS Set-Up Time Margin** (Equation 8.)

\[
\text{CPU Address Valid to Command Active Delay} - \text{PCS Decode Time} = \text{TCLCL}(5\text{MHz}) + \text{TCLMLmin}(8288) - \left[ \text{Greater of TCLAVmax}(8086) + \text{TIVOVmax}(8282) \text{ or TCLCHmax}(8288) + \text{TSHOVmax}(8282) \right] - t_{\text{pmax}}(8205) - t_{\text{pmin}}(8203) + 200 + 10 - \left[ \text{Greater of (110 + 30) or (15 + 45)} \right] - 18 - 20 = 32
\]

**RAM DATA OUT HOLD TIME MARGIN**

The 8203 CAS output is only held valid for a fixed length of time during a read cycle, after that the RAM data outputs are 3-stated. This time is not long enough to allow the 8086 to read the data from the bus, so the data must be latched externally. This latch should be a transparent type and should be strobed by XACK from the 8203. Because the minimum time from XACK active to CAS inactive is only 10 ns, a latch with a data hold time requirement of 10 ns or less (such as a 74S373) should be used (see Equation 9).

**RAM Data Out Hold Time Margin,** (Equation 9.)

*From XACK Active*

\[
= t_{\text{ACKmin}}(8203) + t_{\text{offmin}}(2164A - 20) - t_{\text{Hmin}}(74S373)[1] = 10 + 0 - 10 = 0
\]

**OTHER CALCULATIONS**

Equations 3, 4, 6 and 7 may be solved directly for N, where N is the number of wait states, to find how many wait states are required at a given frequency. Alternatively, a number may be substituted for N and these equations solved for the 8086's clock period, TCLCL, to find the maximum microprocessor frequency possible with N wait states. Note that the clock high and low times (TCHCL and TCLCH) are also a function of TCLCL. Be sure to use the proper speed selection of the 8086 in this calculation, as various A.C. parameters are different and the result may be different for different speed selections of the 8086, even at the same frequency. Be sure to check the other equations at this frequency to make sure they are OK, too.

Finally, for given values of TCLCL and N, Equations 3, 4, 6, and 7 may be checked to find the lowest 8203 clock frequency which will allow the same system performance, if it is desired to operate at some frequency other than the 25 MHz we assumed.

**CONCLUSION**

This design will operate with, at worst case, one wait state (except for refresh) at microprocessor frequencies up to 6 MHz, using slow (200 ns access time) dynamic RAMs. At 6 MHz, it is limited by a lack of SACK set-up

[1] A 74S373 must be used to meet this timing requirement. Even though worst case margin is 0 ns, this is not a critical timing, as valid data will hold on the latch inputs for a considerable time after the RAM outputs 3-state.
time. At 5 MHz, the 8203 can be operated at any clock frequency from 18.432 MHz to 25 MHz, still with only one wait state.

Example 2. 8086 Alternate Configuration System (5 MHz)

Figure 37 shows another 8086 Max mode system at 5 MHz, but this time using the Alternate Configuration, which allows it to operate with no wait states (except for refresh).

The system in the previous example was limited by SACK set-up time. SACK set-up time can be improved by sampling SACK later; this has been done by changing the clock edge used to sample SACK, allowing roughly ½ clock period longer. SACK set-up time (and read data access time and write data hold time) margin can also be improved by activating the RD or WR inputs of the 8203 earlier in the 8086’s earlier in the 8086's 8288 Bus Controller. Altogether, these changes allow about one 8086 clock period more set-up time for SACK.

Let's look at this logic in more detail. An Intel 8205 (A8) is used to decode the 8086's status outputs D0-2. An opcode fetch, memory read, or memory write decode to 8205 outputs 4, 5, and 6, respectively. These outputs go to the D inputs of two 74S74 flip-flops. The Q output of flip-flop A10.2 is an advanced memory read signal and the Q output of A11.2 is an advanced memory write signal. As shown in Figure 37, the 8203 is not activated for opcode fetches, but it can be if 8205 outputs 4 and 5 are ORed with the unused 74S00 gate (A9.4) and the Q output of A10.2 used instead of Q. Both flip-flops are clocked by the trailing edge of ALE to generate the advanced commands. Flip-flop A10.1 is clocked by the trailing edge of either AMWC (Advanced Memory Write Command) or MRDC (Memory Read Command) from the 8288 bus controller (A6), indicating that the 8086 has completed the memory cycle. A10.1, in turn, presets both the A10.2 and A11.2 flip-flops to terminate the advanced memory read and write signals to the 8202A. A10.1 is then preset to its initial state by ALE going active at the start of the next bus cycle.

Because RAM write cycles are started very early in the 8086’s bus cycle using this logic, the 8203 will activate CAS to the RAMs (latching write data) before the data is valid from the 8086. This requires delaying WE to the RAMs and performing a “late write” (explained earlier under Dynamic RAMs) in order to allow more time for the write data to arrive. But the WE signal must not be delayed so long that there is no longer enough data hold time, measured from when WE goes active; or that the WE active to CAS inactive spec or the RAM (tRWL) is violated. None of the control signals from the 8086 or 8288 bus controller satisfy both of these timing constraints, so such a signal is generated by flip-flop A11.1, which serves to delay AMWC from the bus controller by an amount of time equal to TCLCH (the low time of the 8086’s clock). A11.1 is also preset by A10.1 at the end of the memory cycle. The Q output of A11.1 is ANDed with WE from the 8203 by A14.1 to form a delayed RAM WE. As in the previous example, this signal is then ANDed with BHE and AO to form the WE for the high and low bytes of RAM, respectively.

A total of four packages (three 14-pin and one 16-pin) of TTL logic are required.

The dynamic RAM interface timings are identical to the last example (Equations 1 (a)-(h)); 2164A-20 RAMs will be used again.

ADDRESS SET-UP AND HOLD TIME MARGINS

Address set-up and hold time margins are given in Equations 10 and 11, respectively. An 8086-2 microprocessor has been used instead of the standard 8086, as this speed-selected part gives better address set-up to RD or WR times, which this design needs since it uses advanced RD and WR commands.

Row Address Set-Up Time Margin

\[
\text{Row Address Set-Up Time Margin}[1] = \text{CPU Address to Adv. RD Delay} \\
+ \text{RAS Delay - Address*Delays} \\
= \text{TCLCH,min(8284A)} + \text{TC helium(8288)}[2] \\
+ \text{tPLH,min(74S00)}[3] + \text{tPLH,min(74S74)}[2] \\
+ \text{tC,min(8203)} - \text{Greater of TCLAV,max(8086 - 2) + TIVOV,max(8282) or TCLLH,max(8288) + TSHOV,max(8282)} \\
- \text{tAD,max(803) - tASR.min(2164A-20)} \\
= \frac{1}{2}(200) + 15 + 2 + 1 + 2 + [40 + 30] \\
- \text{Greater of (60 + 30) or (15 + 45) - 40 - 0} \\
= 63
\]

[1] Read or write cycles only. Eq. 1b gives this timing for refresh cycles.
Address Hold Time Margin (N = 0) (Equation 11.)
\[ \text{Address Hold Time Margin} = \text{CPU Address Hold Time from Adv. RD} \]
\[ - \text{Active - RAS Inactive Delays} \]
\[ = (3+N)T_{CL}(5\text{MHz}) + T_{CH}(\text{min}(8284A)) + T_{SH}(8288) + T_{O}(8284A) - T_{CLR}(8280) \]
\[ = (3)200 + [\frac{1}{2}(200) + 2] + 2 + 5 - 35 \]
\[ = 175 \]

READ DATA ACCESS TIME MARGIN

Read data access time margin is shown in Equation 12; no wait states are required for read cycles, even with 200 ns access time RAMs.

Read Data Access Time (Equation 12.) Margin (N = 0)
\[ = \text{Adv. RD to Data Valid Delay - CAS Delay} \]
\[ - \text{Read Data Delays} \]
\[ = (2+N)T_{CL}(5\text{MHz}) + T_{CH}(\text{min}(8284A)) + T_{SH}(8288) + T_{O}(8284A) - T_{CLR}(8280) \]
\[ = (2)200 + [\frac{1}{2}(200) + 2] - 15 - 10 - 20 \]
\[ = 3 \]

WRITE DATA SET-UP AND HOLD TIME MARGINS

Write data set-up and hold times are shown in Equations 13 and 14, respectively. No wait states are required during write cycles. Note that write data set-up has been guaranteed by delaying WE from the 8203 with clocked AMWC from the bus controller and performing "late write" cycles; write data set-up time would not be satisfied otherwise. Equation 15 verifies that WE has not been delayed too long to meet the RAM's WE active to RAS inactive set-up time (t_{RW}). The RAM's WE active to CAS inactive set-up time (t_{CWL}) is also satisfied, since CAS does not go inactive until at least 20 ns after RAS.

Write Data Set-Up Time Margin (Equation 13.)
\[ = \text{CPU Data to Clocked AMWC Set-Up} \]
\[ + \text{WE Delays} - \text{Data Delays} \]
\[ = T_{CL}(8284A) + T_{PH}(74S74) \]
\[ + (2)T_{PH}(74S32) \]
\[ - T_{O}(8286) - T_{O}(8286) \]
\[ = [\frac{1}{2}(200) - 15] + 2 + (2)2 - 60 - 30 - 0 \]
\[ = 34 \]

Write Data Hold Time Margin (N = 0) (Equation 14.)
\[ = \text{CPU Data Hold Time from Clocked AMWC} \]
\[ + \text{Data Delays - WE Delays} \]
\[ = (2+N)T_{CL}(5\text{MHz}) + T_{CH}(\text{min}(8286)) - T_{PH}(74S74) - T_{PH}(74S32) - T_{RW}(8284A) - T_{CWL}(8284A) \]
\[ = (2)200 + 10 + 5 - 10 - (2)7 - 45 \]
\[ = 346 \]

SACK SET-UP TIME MARGIN

Equation 16 shows that SACK set-up time is satisfied; no wait states will be generated for read or write cycles (except for refresh).

SACK Set-Up Time Margin (N = 0) (Equation 16.)
\[ = (1+N)T_{CL}(5\text{MHz}) - T_{CH}(\text{max}(8288)) \]
\[ - T_{PH}(74S74) - T_{PH}(74S74) \]
\[ = 200 - 35 - 5 - 10 [2(40) + 47] - 3 \]
\[ = 20 \]

[3] t_{SK}(74S74) is max. skew between t_{PH}(Q output, from CLK) of two Q outputs in same package — use = 2 ns.
Figure 37. 8086 Alternate Configuration System

Notes: Symbol \( \uparrow \) indicates connection to V\textsubscript{cc} through 1K pull up
--- indicates additional circuitry to zero watt states
PCS Set-Up Time Margin (Equation 17.)

\[ t_{PCS} = \text{CPU Address Valid to Adv. RD or Adv. WR Delay - PCS Decode Time} \]
\[ = \text{TCLCH}_{\text{min}}(8284A) + \text{TCHLL}_{\text{min}}(8288)[1] \]
\[ + \text{t}_{\text{PLH}}(74S00) + \text{t}_{\text{PHL}}(74S74)[1] \]
\[ - \text{TCLAV}_{\text{max}}(8086-2) - \text{TIVOV}_{\text{max}}(8282) \]
\[ - \text{t}_{\text{max}}(74S138)[3] - \text{t}_{\text{PCS}}(8203) \]
\[ = \frac{3}{5}(200) - 15 + 2 + 1 + 2 - 60 - 30 - 12 - 20 \]
\[ = 1 \]

**PCS SET-UP TIME MARGIN**

PCS set-up time for the 8203 \( t_{PCS} \) is satisfied, but not with as much margin in the last example (Figure 17).


This is because the RD and WR commands are activated earlier in the microprocessor’s bus cycle, leaving less time to decode PCS from the address bus.

**CONCLUSION**

This design will operate with a guaranteed zero wait states up to 5 MHz using slow (200 ns access time) RAMs. At this frequency, it is limited by both read and write data set-up times, and to a lesser extent, by SACK set-up time. Using faster RAMs will not raise the maximum frequency, as write data and SACK set-up times are not affected by the RAM speed. The 8203 operating frequency must be 25 MHz.

This design can be used (with some modifications) to allow one wait state performance up to 8086 clock frequency of 8 MHz.
8203/8206/2164A Memory Design

Brad May
Peripheral Applications
ABSTRACT

This Application Note shows an error corrected dynamic RAM memory design using the 8203 64K Dynamic RAM Controller, 8206 Error Detection and Correction Unit and 150 ns 64K Dynamic RAMs with a minimum of additional logic.

The goals of this design are to:
1. Control 128K words x 16 bits (256 KB) of 64K dynamic RAM.
2. Support 150 ns dynamic RAMs.
3. Write corrected data back into dynamic RAM when errors are detected during read operations.
4. To use a minimum of additional logic.

It is not the goal of this design to:
1. Provide the maximum possible performance.
2. Provide features like error logging, automatic error scrubbing and dynamic RAM initialization on power-up, or diagnostics, although these features can be added.

DESIGN

Figure 1 shows a memory design using the 8206 with Intel's 8203 Dynamic RAM Controller and 150 ns 64K Dynamic RAMs. As few as three additional ICs complete the memory control function (Figure 2).

For simplicity, all memory cycles are implemented as single-cycle read-modify-writes, shown in Figure 3. This cycle differs from a normal read or write primarily when the dynamic RAM write enable (WE) is activated. In a normal write cycle, WE is activated early in the cycle; in a read cycle, WE is inactive. A read-modify-write cycle consists of two phases. In the first phase, WE is inactive, and data is read from the dynamic RAM; for the second phase, WE is activated and the (modified) data is written into the same word in the dynamic RAM.

Dynamic RAMs have separate data input and output pins so that modified data may be written, even as the original data is being read. Therefore data may be read and written in only one memory cycle.

Figure 1. 8203/8206 Memory System
In order to do read-modify-writes in one cycle, the dynamic RAM's CAS strobe must be active long enough for the 8206 to access data from the dynamic RAM, correct it, and write the corrected data back into the dynamic RAM. CAS active time is an 8203 spec (t_{CAS}), and is dependent on the 8203's clock frequency. The clock frequency and dynamic RAM must be chosen to satisfy Equation 1.

\[
t_{\text{CASmin}} \geq t_{\text{CAC}} + t_{\text{DQVQ}} + t_{\text{QVQ}} + t_{\text{DS}} + t_{\text{CWL}}
\]

The 8203 itself performs normal reads and writes. In order to perform read-modify-writes, all that is needed is to change the timing of the WE signal. In this design, WE is generated by the interface logic in Figure 2—the 8203 WE output is not used. All other dynamic RAM control signals come from the 8203. A 20-ohm damping resistor is used to reduce ringing of the WE signal. These resistors are included on-chip for all 8203 outputs.

The interface logic generates the R/W input to the 8206. This signal is high for read cycles and low for write cycles. During a read-modify-write cycle, R/W is first high, then low. The falling edge of R/W tells the 8206 to latch its syndrome bits internally and generate corrected check bits to be written to dynamic RAM. Corrected data is already available from the DO pins. No control signals at all are required to generate corrected data.

R/W is generated by delaying CAS from the 8203 with a TTL-buffered delay line. This allows the 8206 sufficient time to generate the syndrome; this delay, \( t_{\text{DELAY1}} \), must satisfy Equation 2.

\[
t_{\text{DELY1}} \geq t_{\text{CAC}} + t_{\text{DQVQ}} + t_{\text{QVQ}} + t_{\text{DS}} + t_{\text{CWL}}
\]

The 8206 uses multiplexed pins to output first the syndrome word and then check bits. This same R/W signal may be used to latch the syndrome word externally for error logging. The 8206 also supplies two useful error signals. ERROR signals the presence of an error in the data or check bits. CE tells if the error is correctable (single bit in error) or uncorrectable (multiple bits in error).

In the event that an uncorrectable error is detected, the 8206 will force the Correctable Error (CE) flag low; this may be used as an interrupt to the CPU to halt execution and/or perform an error service routine. In this case the 8206 outputs data and check bits just as they were read, so that the data in the dynamic RAM is left unaltered, and may be inspected later.

After R/W goes low, sufficient time is allowed for the 8206 to generate corrected check bits, then the interface logic activates WE to write both corrected data and check bits into dynamic RAM. WE is generated by delaying CAS from the 8203 with the same delay line.
used to generate R/W. This delay, $t_{\text{DELAY} 2}$, must be long enough to allow the 8206 to generate valid check bits, but not so long that the $t_{\text{CWl}}$ spec of the RAM is violated. This is expressed by Equation 3.

$$t_{\text{DELAY} 1} + \text{TRVSV} \leq t_{\text{DELAY} 2} \leq t_{\text{CASmin}} - t_{\text{CWl}}$$

Unlike other EDC chips, errors in both data and check bits are automatically corrected, without programming the chip to a special mode.

Since the 8203 terminates $\text{CAS}$ to the dynamic RAMs a fixed length of time after the start of a memory cycle, a latch is usually needed to maintain data on the bus until the 8086 completes the read cycle. This is conveniently done by connecting $XACK$ from the 8203 to the STB input of the 8206. This latches the read data and check bits using the 8206's internal latches.

The 8086, like all 16-bit microprocessors, is capable of reading and writing single byte data to memory. Since the Hamming code works only on entire words, if you want to write one byte of the word, you have to read the entire word to be modified, do error correction on it, merge the new byte into the old word inside the 8206, generate check bits for the new word, and write the whole word plus check bits into dynamic RAM. A byte write is implemented as a Read-Modify-Write.

Why bother with error correction on the old word? Suppose a bit error had occurred in the half of the old word not to be changed. This old byte would be combined with the new byte, and new check bits would be generated for the whole word, including the bit in error. So the bit error now becomes "legitimate"; no error will be detected when this word is read, and the system will crash. You can see why it is important to eliminate this bit error before new check bits are generated. Byte writes are difficult with most EDC chips, but easy with the 8206.

Referring again to Figure 2, the 8206 byte mark inputs ($BM_0$, $BM_1$), are generated from A0 and BHE, respectively, of the 8086’s address bus, to tell the 8206 which byte is being written. The 8206 performs error correction on the entire word to be modified, but tri-states its DO/WDI pins for the byte to be written; this byte is provided from the data bus by enabling the corresponding 8286 transceiver. The 8206 then generates check bits for the new word.

During a read cycle, $BM_0$ and $BM_1$ are forced inactive, i.e., the 8206 outputs both bytes even if 8086 is only reading one. This is done since all cycles are implemented as read-modify-writes, so both bytes of data (plus check bits) must be present at the dynamic RAM data input pins to be rewritten during the second phase of the read-modify-write. Only those bytes actually be-
ing read by the 8086 are driven on the data bus by enabling the corresponding 8286 transceiver.

The output enables of the 8286 transceivers (OEB0, OEB1) are qualified by the 8086 RD, WR commands and the 8203 CS. This serves two purposes:
1. It prevents data bus contention during read cycles.
2. It prevents contention between the transceivers and the 8206 DO pins at the beginning of a write cycle.

CONCLUSION
Thanks to the use of a 68-pin package, the 8206 Error Detection and Correction Unit is able to implement an architecture with separate 16 pin input and output busses. The resulting simplification of control requirements allows error correction to be easily added to an 8203 memory subsystem with a minimal amount of interface logic.
Interfacing the 8207 Dynamic RAM Controller to the iAPX 186
INTRODUCTION

Most microprocessor based workstation designs today use large amounts of DRAM for program storage. A drawback to DRAMs is the many critical timings that must be met. This control function could easily equal the area of the DRAM array if implemented with discrete logic.

The VLSI 8207 Advanced Dynamic RAM Controller (ADRC) performs complete DRAM timing and control. This includes the normal RAM 8 warm-up cycles, various refresh cycles and frequencies, address multiplexing, and address strobe timings. The 8207’s system interface and RAM timing and control are programmable to permit it to be used in most applications.

Integrating all of the above functions (plus a dual port and error correcting interfaces) allows the user to realize significant cost savings over discrete logic. For example, comparing the 8207 to the iSBC012B 512K byte RAM board (where the DRAM control is done entirely with TTL), an 8207 design saved board space (3 in² vs 10 in²); required less power (420 ma vs 1220 ma); and generated less heat. Moreover, design time was reduced, and increased margins were achieved due to less skewing of critical timings. This comparison is based on a single port design and did not include the 8207’s RAM warm-up, dual-port and error correcting features. If these features were fully implemented, there would be no change to the 8207 figures, listed above, while the TTL figures would easily double.

This Application Note will illustrate an iAPX design with the 8207 controlling the dynamic RAM array. The reader should be familiar with the 8207 data sheet, the 80186 data sheet, and a RAM data sheet*.

DESIGN GOALS

The main objective of this design is for the 80186 to run with no wait states with a Dynamic RAM array. The design uses one port of the 8207. The dual port and error correcting interfaces of the 8207 are covered in separate Application Notes.

The size of the RAM array is 4 banks of 64k RAMs or 512k bytes. The memory is to be interfaced locally to the 80186.

USING THE 8207

The three areas to be considered when designing in the 8207 are:

- 8207 programming logic
- Microprocessor interface
- RAM array

8207 Programming

The 8207 requires up to two 74LS165 shift registers for programming. This design needs one 8 bit shift register, as shown in Figure 1. The 16 bits in the Program Data Word are set as shown in Figure 2. Refresh is done internally, so the REFREQ input must be tied high. The memory commands are iAPX 86 status, so

---

*All RAM references in this Application Note are based on Intel’s 2164A 64k Dynamic RAM.
the timing of EAACK will always guarantee 2 clocks of address hold time from RAS.

Acknowledgment Setup Time
The margin between the 8207 issuing EAACK and the 80186 ready input for no wait states minus delays from clock edges, logic delays, and setup time is calculated as follows.

1 clock - 8207 TCLAKL max – 74S30 tPLH @ 15 pf – 80186 TSRYCL > 0

125 ns – 35 – 22 – 35 = 33 ns

Read Access Margin
The 8207 starts a memory cycle on the falling clock edge between the 80186’s T1 and T2. Data must be valid within 2 clocks. Valid data from the RAMs is based upon the CAS access period minus buffer, clock, setup requirements.

2 TCLCL – 8207 TCLCSL @ 150 pf (t34) – DRAM tCAC – 74S240 propagation delay @ 50 pf – additional bus loading delay (250 pf)(1) – 74S240 delay @ 50 pf – 80186 TDVCL > 0

250 ns – 122 – 85 – 7 – 7 – 7 – 20 = 2 ns

Write Data Setup and Hold Margin
Data from the processor must be valid when WE is issued by the 8207 to meet the RAM specification tDS (2164A = 0 ns), and then held for a minimum of 30 ns.

(1) 74STTL logic derated by .05 ns/pf. 74STTL buffers (240, 37) derated by .025 ns/pf.
the PCTLA input must be high when RESET goes inactive.

The differential reset circuit shown in the Data Sheet is necessary only to ensure that memory commands are not received by the 8207 when Port A is changed from synchronous to asynchronous (vice versa for Port B). This design keeps Port A synchronous so no differential reset circuit is needed.

**Microprocessor Interface**

To achieve no wait states, the 8207 must connect directly to the microprocessor's CLKOUT and status lines. The 8207 Acknowledge (BAACK) must connect to the SRDY input of the 80186. When the 80186 is reset, it tristates the status lines. The 8207 PCTLA input requires a high to decode the proper memory commands. This is accomplished by using a pull-up resistor or some component that incorporates a pull-up on S2.

The 8207 address inputs are connected directly to the latched/demultiplexed address bus.

**RAM Array**

The 8207 provides complete control of all RAM timings, warm up cycles, and refresh cycles. All write cycles are "late writes." During write cycles, the data out lines go active. This requires separate data in/out lines in the RAM array.

To operate the 80186 with no wait states, it is necessary to choose sufficiently fast DRAMs. The 50 ns version of the 2164A allows operating the 80186 at 8 MHz, and the 200 ns version up to 7 MHz.

**HARDWARE DESIGN**

Figure 3 shows a block diagram of the design, and Figure 4 is a timing diagram showing the relationship between the 8207 and the 80186.

**8207 Command Setup**

Two events must occur for a command to be recognized by the 8207. The 80186 status outputs are sampled by a rising clock edge and Port Enable (PE) is sampled by the next falling clock edge (refer to the Data Sheet wave forms).

The command timing is determined by the period between the status being issued and the first rising clock edge of the 8207, minus setup and delays.

\[
\begin{align*}
80186 \text{ status valid to 8207 rising clock} & \quad \text{status from clock delay} - 8207 \text{ setup to clock } \geq 0 \\
1 \text{ TCLCL} & - 80186 \text{ TCHSV max} - 8207 \text{ TKVCH min } \geq 0 \\
125 \text{ ns } - 55 - 20 = 50 \text{ ns}
\end{align*}
\]

PE is a chip select for a valid address range. It can be generated from the address bus or from the 80186's programmable memory selects. This design uses an inverted A19. The timing is determined by the interval between the address becoming valid and the falling clock edge, minus setup and delays.

\[
\begin{align*}
80186 \text{ address valid to 8207 falling clock edge} & \quad 80186 \text{ address from clock delay } - 8283 \text{ latch delays } - 8207 \text{ PE setup } \geq 0 \\
1 \text{ TCLCL} & - 80186 \text{ TCLAV max} - 8283 \text{ IVOV @ 300 pf} - 8207 \text{ TPEVCL } \geq 0 \\
125 \text{ ns } - 44 - 22 - 30 = 29 \text{ ns}
\end{align*}
\]

The hold times are 0 ns and are met.

**Address Setup**

For an 80186 design, the 8207 requires the address to be stable before RAS goes active, and to remain stable for 2 clocks. Unused 80186 address inputs should be tied to Vcc. tASR is a RAM specification. If it is greater than zero, this must be added to the address setup time of the 8207. Address setup is the interval between addresses being issued and RAS going active, minus appropriate delays.

\[
\begin{align*}
80186 \text{ address valid to 8207 RAS active} & \quad 80186 \text{ address from clock delay } - \text{ bus delays} - (8207 \text{ setup } + \text{ RAM } t_{\text{ASR}}) \geq 0 \\
\text{TCLCL} + 8207 \text{ TCLRSL min } & @ 150 \text{ pf}^{(1)} - 80186 \text{ TCLAV max } - 8283 \text{ IVOV max @ 300 pf} - (8207 \text{ TAVCL min } + \text{ DRAM } t_{\text{ASR}}) \geq 0 \\
125 \text{ ns } + 0 - 44 - 22 -(35 + 0) = 24 \text{ ns}
\end{align*}
\]

The address hold time of 2 clocks + 0 ns is always met, since the addresses are latched by the 8282/3. Even when the processor is in wait states (for refresh),

\[\text{(1) Not specified—use 0 ns.}\]
The hold time, tDH, is from WE going low to the 80186 DEN going high plus buffer delays minus WE from clock delays.

\[
TCLCL - 80186 TCVCTX \text{ min + 74S32 tPD(2) min + 74S240 tPHZ (min)(2) + 250 pf bus delays + 74S240 propagation delay min - 8207 TCLW max - 74S37 tPHL @ 50 pf - 142 pf loading delays - DRAM tDH} \geq 0
\]

\[
62.5 \text{ ns} + 10 + 2 + 3 + 7 + 3.5 - 35 - 3.5 - 30 = 19.5 \text{ ns}
\]

All margins are actually better by about 10-20 ns. No improvement in timing was allowed for lower capacitive loads when additional buffers are used (i.e. the 80186 address out delay is at 200 pf, but the 8283 latch only loads these lines with about 20 pf).

**SUMMARY**

The 8207 supports the 80186 microprocessor running with no wait states. The 8207 interfaces easily between the microprocessor and dynamic RAM. There are no difficult timings to be resolved by the designer using external logic.

![Figure 4. 8207/80186 timing relationship](image-url)
Interfacing the 8207 Advanced Dynamic RAM Controller to the iAPX 286
INTRODUCTION

The 80286 high speed microprocessor pushes microprocessor based systems to new performance levels. However, its high speed bus requires special design considerations to utilize that performance. Interfacing the 80286 to a dynamic RAM array require many timings to be analyzed, refresh cycle effects on bus timing examined, minimum and maximum signal widths noted, and the list continues.

The 8207 Advanced Dynamic RAM Controller was specifically designed to solve all interfacing issues for the 80286, provide complete control and timing for the DRAM array, plus achieve optimum system performance. This includes the normal RAM 8 warm-up cycles, various refresh cycles and frequencies, address multiplexing, and address strobe timings. The 8207 Dynamic RAM Controller’s system interface and RAM timing and control are programmable to permit it to be used in most applications.

Integrating these functions (plus dual port and error correcting interfaces) allows the user to realize significant savings in both engineering design time, PC board space and product cost. For example, in comparing the 8207 to the ISBCOl2B 512k byte RAM board (where the DRAM timing and control is done entirely with TTL), the 8207 design saved board space (3 in 2 vs 10 in2); used less power (420 ma vs 1220 ma); reduced the design time; and increased margins due to less skewing of timings. The comparison is based upon a single port 8207 design and does not include its RAM warm-up, dual port, error correcting, and error scrubbing or RAM interleaving features.

This Application Note will detail an 80286 and 8207 design. The reader should have read the 8207 and the 80286 data sheets, a DRAM data sheet*, and have them available for reference.

DESIGN GOALS

The main objective of this design is to run the RAM array without wait states, to maximize the 80286’s performance, and to use as little board space as possible. The 80286 will interface synchronously to Port A of the 8207 and the 8207 will control 512k bytes of RAM (4 banks using 64k DRAMs). The dual port and error correcting features of the 8207 are covered in separate Application Notes.

8207 INTERFACE

The 8207 Memory design can be subdivided into three sections:

- Programming the 8207.
- The 80286/8207 interface.
- The Dynamic RAM array.

Programming the 8207

The RAM timing is configured via the 16 bit program word that the 8207 shifts-in when reset. This can require two 74LS165 shift registers to provide complete DRAM configurability. The 8207 defaults to the configuration shown in Table 1 when PDI is connected to ground. This design does not need the flexibility the shift registers would allow since standard 8207/80286 clock frequencies, DRAM speeds and refresh rates are used. Table 1 details the 8207/80286 configuration and Table 10 in the Data Sheet identifies “CO” as the configuration of the 8207 all timings will be referenced to (80286 mode at 16 MHz using fast RAMs = CO).

Table 1. Default Non-ECC programming, PD1 pin (57) tied to ground.

| Port A is Synchronous (EAACKA and XACKA) |
| Port B is Asynchronous (LAACKB and XACKB) |
| Fast-cycle Processor Interface (10 or 16 MHz) |
| Fast RAM 100/120 ns RAM |
| Refresh Interval uses 236 clocks |
| 128 Row refresh in 2 ms; 256 Row refresh in 4 ms |
| Fast Processor Clock Frequency (16 MHz) |
| “Most Recently Used” Priority Scheme |
| 4 RAM banks occupied |

The 8207 will accept 80286 status inputs when the PCTLA pin is sampled low at reset. This pin is not necessary for an 80286 design (besides programming) and is tied to ground.

Refresh is the final option to be programmed. If the Refresh pin is sampled high at reset, an internal timer

*All RAM references in this Application Note are based upon Intel’s CMOS 51C64-12 64k Dynamic RAM. Any DRAM with similar timings will function. Refer to section 4.4.
is enabled, and if low at reset, this timer is disabled. The first method is the easiest to implement, so the RFRQ pin is tied to Vcc.

The differential reset circuit shown in the Data Sheet is necessary only to ensure that memory commands are not received by the 8207 when Port A is changed from synchronous to asynchronous (vice versa for Port B). This design keeps Port A synchronous so no differential reset circuit is needed.

RAM Array
The 8207 completely controls all RAM timings, warm-up cycles, and refresh cycles. To determine if a particular RAM will work with the 8207, calculate the margins provided by the 8207 (Table 15, 16—8207 Data Sheet) and ensure they are greater than the RAM requirement. An additional consideration is the access times of the RAMs. The access time of the system is dependent upon the number of data buffers between the 80286 and the DRAMs. To operate the 80286 at zero wait states requires access times of 100-120 ns. Slower RAMs can be used (150 ns) by either adding a wait state (programming the 8207 for "Cl") or reducing the clock frequency (to 14.9 MHz approximately and maintaining the CO configuration.).

All write cycles are "late writes" and the data out lines of the RAM will go active. This will require separate data in and out lines in the RAM array. Another consideration for the RAM array is the proper layout of the RAM, and impedance matching resistors on the 8207 outputs. Proper layout is covered in Intel's RAM Data Sheets and Application Notes.

Microprocessor Array
To achieve no wait state operation, the 8207's clock input must be connected to the 80286's clock input. The EAACK (early acknowledge) output of the 8207 must connect to the SRDY input of the 82284. The 8207's address inputs connect directly to the address bus and must be set internally, when a valid command is received. LEN can go high in two clock cycles if the RAM cycle started (RAS going low) at the same time LEN went low. If the 8207 is doing a refresh cycle, the 80286 will be put into wait states until the memory cycle can start. LEN will then go high two clocks after RAS starts, since addresses are no longer needed for the current RAM cycle. Thus the low period of LEN could be much longer than listed in the Data Sheet.

DESIGNING THE HARDWARE
Figure 1 shows a detailed block diagram of the design and Figure 2 shows the timing relationship between the 8207 and the 80286.

The following analysis of six parameters will confirm that the design will work. These six system parameters are generally considered the most important in any microprocessor—Dynamic RAM design.

8207 Command Setup Margin
Two events must occur for the 8207 to start a memory cycle. Either RD or WR active (low) and PE must be low when the 8207 samples these pins on a falling clock edge. If PE is not valid at the same clock edge that samples RD or WR active, the memory cycle will be aborted and no acknowledge will be issued.

The command setup time is based upon the status being valid at the first falling clock edge.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>80286 status valid to 8207 falling clock</td>
<td>80286 status from clock delay - 8207 command setup to clock ≤ 0</td>
</tr>
<tr>
<td>TCLCL - 80286 t12 (max) - 8207 TKVCL (min) ≤ 0</td>
<td>62.5 - 40ns - 20ns = 2.5ns</td>
</tr>
</tbody>
</table>

PE is decoded from the address bus and must be set up to the same falling clock edge that recognizes the RD, WR inputs. This margin is determined from the clock edge that issues the address and the clock edge that will recognize RD or WR, minus decoding logic delays.

There are 2 clocks between addresses being issued by the 80286 and PE being sampled by the 8207. Then the 80286 address delay from the clock edge and decoding logic delays are subtracted from this interval. This margin must be greater than 0.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2TCLCL - 80286 t13 (max) - 8207 TPEVCL (min) ≤ 0</td>
<td>125 - 60 - 30 = 35ns</td>
</tr>
</tbody>
</table>

The address decode logic must use no more than 35 ns (and less is better). Figure 3 shows an easy implementation which uses a maximum of 12 ns.

The 8207 requires a zero ns hold time and is always met.
Figure 1. 80286 to 8207, non-ECC, Synchronous System Single Port

NOTE: THE 8207 REQUIRES SERIES RESISTORS ON ALL OUTPUTS.
Figure 2. 80286/8207 Timing—"CO"

Figure 3. Address Decode Logic
Address Setup Margin

The 8207 must have stable addresses up to two clocks after RAS goes active. This is of no concern to the user, since LEN latches the address internally and will not admit a new address until two clocks after RAS goes active.

Addresses must be stable at least 35 ns (tAVCL) before RAS goes active to allow for propagation delays through the 8207, if a RAM cycle is not delayed by the 8207.

tASR is a RAM specification. If it is greater than zero, tASR must be added to the address setup time of the 8207. Address setup is the interval between addresses being issued, by the 80286, and RAS going active, minus appropriate delays.

The margin is determined from the number of clocks between addresses being issued from the 80286 to RAS going active. Exactly when RAS goes active is unimportant, since here we are interested only in the clock edge.

2TCLCL - 80286 t13 (max) - 8207 TAVCL
(min) ≤ 0
125 - 60ns - 35ns = 35ns

Acknowledge Setup Margin

The 8207 acknowledge (EAACK) can be issued at any point in the 80286 bus cycle (end of $1$ or $2$ of Ts or Tc). If EAACK is issued at the end of $2$ (Ts or Tc), the 80286 will complete the current bus cycle. If EAACK is issued at the end of $1$ of Tc, the 82284 will not generate READY to the 80286 in time to end the current bus cycle. A new Tc would then be generated and EAACK would now be sampled in time to terminate the bus cycle. EAACK is 3 clocks long in order to meet setup and hold times for either condition.

We need the margin between the 8207 issuing EAACK and the 82284 needing it. Figure 4, shows a worst case example.

TCLCL - 8207 TCLAML max - 82284 t11 ≤ 0
62.5 - 35 - 15 = 12.5ns

Read Access Margin

The 8207 will typically start a memory cycle (i.e. RAS goes low) at the end of $1$ of Ts. But if the start of a memory cycle is delayed (by a refresh cycle for instance), then RAS will be delayed. In the first case,
this represents 3 clocks and the second case could require 4 clocks to meet the data setup requirements of the 80286. In either case, data must be valid at the end of $T_c$. The 8207 holds CAS active long enough to ensure valid data is received by the 80286 in either case.

DRAMs specify two access times, RAS access ($t_{RAC}$) and CAS access ($t_{CAC}$). Both access periods must be calculated and the one with the least margin used. Also the number of data buffers should be kept to a minimum. Too many buffers would require either faster (more expensive) DRAMs, or a reduction in the performance of the CPU (by adding wait states).

**RAS Access Margin**

$$3T_{CLCL} - 8207\ T_{CLRSL} \ max \ @ \ 150 \ \text{pf} - \ \text{DRAM} \ t_{RAC} - 74S240 \ \text{propagation delay max} \ @ \ 50 \ \text{pf} - 80286 \ t_8 \leq 0$$

$$187.5 - 35 - 120 - 7 - 10 = 15.5\text{ns}$$

**CAS Access Margin**

$$2T_{CLCL} - 8207\ T_{CLCSL} \ max \ @ \ 150 \ \text{pf} - \ \text{DRAM} \ t_{CAA} \ (or \ t_{CAC} - 74S240 \ \text{tplh max} \ @ \ 50 \ \text{pf} - 80286 \ t_8 \leq 0$$

$$125 - 35 - 60 - 7 - 10 = 13\text{ns}$$

By solving each equation for $t_{RAC}$ and $t_{CAC}$, the speed requirement of the RAM can be determined.

$$\text{DRAM } t_{RAC} = 3T_{CLCL} - 8207\ T_{CLRSL} - 74S240 \ \text{tplh} - 80286 \ t_8 = 135.5\text{ns}$$

$$\text{DRAM } t_{CAC} = 2T_{CLCL} - 8207\ T_{CLCSL} - 74S240 \ \text{tplh} - 80286 \ t_8 = 73\text{ns}$$

So any DRAM that has a RAS access period less than 135 ns, a CAS access period less than 73 ns, and meets all requirements in the DRAM Interface Timing (Table 15, 16—8207 Data Sheet), will work.

**Write Data Setup and Hold Margin**

Write data from the processor must be valid when the 8207 issues WE to meet the DRAM specification $t_{DS}$ and then held to meet the $t_{DH}$ requirement. Some write cycles will be byte writes and the information to determine which byte is decoded from AO and BHE/ since the 80286's address bus is pipelined, these two signals can change before the RAM cycle starts, hence they must be latched by LEN. PSEN is used in the WE term to shorten the WE pulse. Its use is not essential.

Data must be set up to the falling edge of WE, since WE occurs after CAS. The 2 clocks between valid write data and WE going active (at the RAM’s) minus propagation delays determines the margin.

$$2 T_{CLCL} - 80286 \ t_{14} \ (\max) \ @ \ 100 \ \text{pf} - 74S240 \ \text{tplh} + 8207 \ T_{CLW} \ (\min)^1 + 74S10 \ \text{tplh} @ 192 \ \text{pf}^2 - \ \text{DRAM } t_{DS} = 0$$

$$125 - 50 - 7 + 0 + 14 - 0 = 82\text{ns}$$

The timing of the 8207's acknowledge is such that data will be kept valid by the 80286, for more than two clocks after WE goes active. This easily meets all RAM $t_{DH}$ specifications.

**SUMMARY**

The 8207 complements the 80286's performance and high integration with its own performance, integration and ease of use. No critical timings or logic design has been left to the designer. The 80286/8207 combination allows users to realize maximum performance from their simpler design.

1. Not specified. Assume no delay for worst case analysis.
2. STTL derated by .05ns/pf.
APPLICATION BRIEF

INTERFACING THE DYNAMIC RAM CONTROLLER TO THE iAPX 186

Jim Sleezer

1.0 INTRODUCTION

The 80186 microprocessor has integrated about 20 typically used system components into the same package as the microprocessor. This integration saves board space and design-in time. The 8208 Dynamic RAM Controller continues this system level integration. It is designed to control up to 256 Kbytes of Dynamic RAM (DRAM) using 64 K x 1 DRAMs, and up to 1 Mbyte using 256 K x 1 DRAMs.

Besides generating all DRAM control and timings, the 8208 allows various refresh types, frequencies, and microprocessor interfaces. Additionally, the 8208 does the 8 DRAM warm-up cycles back-to-back to prepare for operation.

By integrating the entire RAM timing and programmable refresh types, refresh rates, and interfaces into a single package, the user realizes significant savings in development time and board space. For example, a quick comparison of the 8208 versus a TTL implementation (using just the DRAM timing logic from Intel's iSBC012B memory board) yielded the following results:

1) a reduction in board space (10 in\(^2\) to 3 in\(^2\)),
2) a reduction in power (1.2'A to 300 mA), and
3) much less design time (1 day).

The difference would be greater still if RAM warm-up, refresh, and interface programmability were added to the TTL implementation.

This Application Note will examine an 8208 to 80186 design. The reader should already have read the 8208 Data Sheet, the 80186 Data Sheet, and a DRAM Data Sheet*.

* While all DRAM references in this Application Note are based upon Intel's 2164A-15 64 K x 1 Dynamic RAM, any DRAM that meets the timing requirements in the Data Sheet, Table 8, and A.C. Characteristics, plus satisfies the Read Data Access Margin, will work.
2.0 HARDWARE DESIGN

An 8208 design can be divided into three areas: programming the 8208, DRAM compatibility, and system interface. While each topic will be covered in this Application Note, the 8208's programming logic defaults to an 8 MHz 80186 synchronous status interface with 150 ns access RAMs. All programming, RAM timings, and interface issues are satisfied for that configuration.

2.1.0 8208 PROGRAMMING

On the trailing edge of Reset, the 8208 samples the levels on two input pins and clocks in a 9 bit serial programming word. One input pin controls the type of refresh to be performed, while the other input pin alters the edge on which the 8208 samples memory commands. The program word further configures the 8208 for a refresh rate as a function of 8208 clock frequency, synchronous or asynchronous operation, and either an advanced acknowledge or Multibus compatible acknowledge.

2.1.1 REFRESH TYPES

If the REFRQ pin is sampled high at reset, an internal refresh timer is enabled; a low disables it. Both modes allow an external refresh cycle request by pulsing the REFRQ pin. An external request is generated by a low-to-high transition, and sampled by an 8208 (clock edge). Burst refresh occurs only when the timer is disabled and the REFRQ pin is sampled by two falling clock edges. The easiest method is to tie the REFRQ pin to Vcc (through a pull-up resistor); refresh cycles are transparent to the user.

2.1.2 8208 COMMANDS

The 8208 alters the point at which it samples a command and its response to the command inputs, based on the level sampled on PCTL when reset goes inactive. A high enables the status interface and a rising clock edge is used (this would be the middle of the T1 state; refer to the Timing Diagram). If low, the Multibus compatible interface is selected and a falling edge is used to allow for more propagation delay.
When the status interface is used, the status lines must be externally pulled up. The 80186 will tristate them when reset and the proper level (high) may not be seen by the 8208.

2.1.3 PROGRAM WORD

The program word defaults to a synchronous interface, fast acknowledge (for no wait states), and a refresh rate compatible with an 8 MHz clock (128 row/2 ms; 256 row/4 ms). When operating the 8208 at 8 MHz, most designs will not need to alter any programming bits and the PDI input pin can be tied to ground. If the 8208 is not run at 8 MHz a 74LS165-type shift register is needed to adjust for a proper refresh rate; otherwise, refresh cycles would not be performed often enough and data would be corrupted.

2.1.3.1 REFRESH RATE OPTIONS (C1O, C1T, PLS, FFS)

These four programming bits permit almost any DRAM to be used without wasting memory bandwidth. The combination of these four bits selects one of sixteen clock intervals as shown in Table 1.

<table>
<thead>
<tr>
<th>CFS</th>
<th>PLS</th>
<th>FFS</th>
<th>Count Interval C11, C10 (8208 Clock Periods)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>00 (0%)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>118</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>59</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>74</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37</td>
</tr>
</tbody>
</table>

Table 1. Refresh Count Intervals
The 8208 does not alter any other of its functions with these four bits. To determine which combination of bits to use, examine the following equation:

\[
\text{Equation 1. Refresh Rate} = \text{count interval} \times \text{8208 clock period}
\]

\[14.6 \text{ usec} = \text{count interval} \times 190 \text{ ns}\]

\[14.6 \text{ usec} / 0.190 = 76.8 \text{ count interval}\]

The next fastest Count Interval of 74 is chosen from Table 1. The bit configuration is: PLS = 1; FFS = 0; CI1 = 0; CIO = 0, and generates seventy-four 8208 clocks between refresh cycles. A refresh cycle can be delayed up to one 8208 RAM cycle from the time it was requested to the time it is serviced. Thus, the 14.6 usec refresh rate is chosen to allow for these delays. The 190 ns clock period was picked at random. The refresh timer is restarted when the cycle is requested and not when the cycle begins executing. Note the difference in the sense of the programming bits. **PLS = 0** is the same as **PLS = 1**. This notation is used throughout the Data Sheet.

2.1.3.3 INTERFACE OPTIONS (S, X)

The S programming bit adds synchronizers to the 8208's inputs when input signals cannot meet setup and hold times. The RD, WR inputs are still decoded as determined by PCTL, but these inputs will be sampled on a falling edge (status or command interface). The X bit allows either an 80186 (8086) no wait state acknowledge or an XACK (Multibus) type acknowledge. A synchronous interface should use the advanced acknowledge and an asynchronous interface the XACK acknowledge. XACK is removed by the inactive edge of RD or WR. If RD or WR goes inactive before the 8208 issues XACK, then no XACK is issued.

2.1.3.3 OTHER OPTIONS (CFS, RB, RFS)

The CFS bit must be set to zero. This bit is reserved for future speed enhancements of the 8208. RFS has no effect on 8208 timings and may be set to either state. It is to be used with faster 8208's. RB is to allow for 32 bit wide memory arrays. If an 8 or 16 bit wide system is used, set this bit to its active state (RB = 0). The Bank Select pin must not select a RAM bank that is not physically present.
2.2 MICROPROCESSOR INTERFACE

The 8208's timings are optimized for an 8086 and 80186 system. The synchronous status interface offers the best performance (i.e., no wait states) and is the easiest to implement.

2.3 DRAM COMPATIBILITY

Table 2 lists the equations to determine whether a particular DRAM will work with the 8208. Four other questions are listed in the A.C. Characteristics Section in the 8208 Data Sheet.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Rd. RF Cycles</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRP</td>
<td>2TCLCL—T26</td>
<td>1</td>
</tr>
<tr>
<td>ICPN</td>
<td>2.5TCLCL—T35</td>
<td>1</td>
</tr>
<tr>
<td>IRSH</td>
<td>3TCLCL—T34</td>
<td>1</td>
</tr>
<tr>
<td>ICAM</td>
<td>3TCLCL—T34</td>
<td>1</td>
</tr>
<tr>
<td>IAR</td>
<td>2TCLCL—T26</td>
<td>1</td>
</tr>
<tr>
<td>IT</td>
<td>3/30</td>
<td>2</td>
</tr>
<tr>
<td>IRC</td>
<td>4TCLCL</td>
<td>1</td>
</tr>
<tr>
<td>IRAS</td>
<td>2TCLCL—T26</td>
<td>1</td>
</tr>
<tr>
<td>ICAS</td>
<td>3TCLCL—T34</td>
<td>1</td>
</tr>
<tr>
<td>IRC5</td>
<td>1.5TCLCL—TCL—T36—TBUF</td>
<td>1</td>
</tr>
<tr>
<td>ICHIP</td>
<td>0.5TCLCL—T34</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>WR Cycles</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRP</td>
<td>2TCLCL—T26</td>
<td>1</td>
</tr>
<tr>
<td>ICPN</td>
<td>2.5TCLCL—T35</td>
<td>1</td>
</tr>
<tr>
<td>IRSH</td>
<td>3TCLCL—T34</td>
<td>1</td>
</tr>
<tr>
<td>ICAM</td>
<td>4TCLCL—T26</td>
<td>1</td>
</tr>
<tr>
<td>IAR</td>
<td>2TCLCL—T26</td>
<td>1</td>
</tr>
<tr>
<td>IT</td>
<td>3/30</td>
<td>2</td>
</tr>
<tr>
<td>IRC</td>
<td>6TCLCL</td>
<td>1</td>
</tr>
<tr>
<td>IRAS</td>
<td>4TCLCL—T26</td>
<td>1</td>
</tr>
<tr>
<td>ICAS</td>
<td>TCLCL—T26</td>
<td>1</td>
</tr>
<tr>
<td>IWHC</td>
<td>3TCLCL—T34</td>
<td>1, 3</td>
</tr>
<tr>
<td>IWC5</td>
<td>4TCLCL—T26</td>
<td>1, 3</td>
</tr>
<tr>
<td>IWP</td>
<td>4TCLCL—T36—TBUF</td>
<td>1</td>
</tr>
<tr>
<td>IRWL</td>
<td>4TCLCL—T36—TBUF</td>
<td>1</td>
</tr>
<tr>
<td>ICMWL</td>
<td>4TCLCL—T36—TBUF</td>
<td>1</td>
</tr>
<tr>
<td>TWCS</td>
<td>TCLCL—T36—TBUF</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2. DRAM Equations

These equations merely determine if the 8208 will provide proper margins for a DRAM. Whether a RAM works properly in a system is another issue. The Hardware Design Example section examines most of the important system timings.
3.0 HARDWARE DESIGN EXAMPLE

The objective is to have the 80186 run without wait states when accessing a DRAM array. The total amount of DRAM is 128K bytes and will be organized as 1 bank of 64K words.

Figure 1 is a block diagram of our design showing all relevant logic. The programming shift register is not needed since the 8208 will be operating at 8 MHz, and the other default values are required. A data buffer is required in a no wait state design, since during reads the 8208 CAS line drives data onto the bus up to 50 ns past the end of T4. If another bus cycle were starting, then the multiplexed address/data lines would conflict with the driven data bus. This would reduce the system's address to ALE setup margins. Figure 2 is a timing diagram of the design.

The timing parameters that are examined ensure that this portion of the system will operate properly. The parameters are:

1. Command setup and hold margin.
2. Address setup and hold margin.
3. Acknowledge setup and hold margin.
4. Write data setup and hold margin.
5. Read access setup and hold margin.

3.1 ACKNOWLEDGE SETUP AND HOLD MARGIN

The 8208 early acknowledge (AACK) is intended to be connected to the SRDY input on the 80186 after being inverted. The AACK is issued at the beginning of T2 and must be valid at the beginning of T3.

\[ 1TCLCL - 8208 \text{TCLAKL max} - 7410 \text{tPLH} @ 15 \text{ pf} - 80186 \text{TSRYCL min} _0 \]

\[ 125 \text{ ns} - 35 - 22 - 35 \]

\[ = 33 \text{ ns} \]
The 80186 hold requirements, TCLSRY, of 15 ns is always met. The 15 ns hold
time applies only when READY is being looked at by the 80186. Transitions
that occur anywhere else in the bus cycle have no effect. AACK is two clocks
long and is issued from a falling clock edge. AACK would always be sampled
one clock into its duration. There would be a hold time of about 1 clock.

3.2 COMMAND SETUP AND HOLD MARGIN

Two events must occur for the 8208 to recognize a valid memory command. The
80186 status outputs are sampled by a rising clock edge (middle of T1
typically) and PE is sampled on the very next falling clock edge. If PE is
not sampled at this point, no memory cycle will start. The status lines would
have to go inactive before requesting another memory cycle.

The status setup margin is referenced to the middle of T4 or T1, and is
required to be valid by the middle of T1.

\[
\text{TCHCH} - 80186 \quad \text{TCHSV max} - 8208 \quad \text{TKVCH min} = 0
\]

\[
125 \text{ ns} - 55 \text{ ns} - 20 \text{ ns}
\]

\[
= 50 \text{ ns}
\]

PE setup margin is referenced to the beginning of T1 and must be valid by the
end of T1. PE selects the 8208 for a valid address range. It can be
generated from either the address bus or using the 80186's programmable chip
selects.

\[
\text{TCLCL} - 80186 \quad \text{TCLCSV max} - 8208 \quad \text{TPEVCL min} = 0
\]

\[
125 - 66 - 30
\]

\[
= 29 \text{ ns}
\]

Both PE and the RD, WR, and PCTL inputs require a 0 ns hold time to their
respective clock edges.
The 8208 latches this information internally for cases when a refresh cycle delays a memory cycle from starting. Thus, a cycle will start when the refresh cycle finishes, even if the status signals have gone inactive. The hold margin is always met.

3.3 ADDRESS SETUP AND HOLD MARGINS

The 8208 requires the addresses to be stable before RAS goes active, and to remain stable for two clock periods thereafter. Unused address inputs should be pulled up to Vcc with a resistor.

The 8208 generates a margin of 0 ns minimum for the DRAM specification tASR when the 8208 specification TAVCL is met. If some DRAM is found that needs a more positive margin for tASR, then this requirement must be added to TAVCL.

The setup margin is between the clock edge that addresses are issued from to the 8208 issuing RAS, minus delays.

\[
1 \cdot T_{CL} + 8208 \cdot T_{LRLS} \cdot \text{min}\{1\} \text{ (@ 150 pf)} - 80186 \cdot T_{LAV} \cdot \text{max} - 8282 \cdot I_{OV} \cdot \text{max (@ 300 pf)} - [8208 \cdot T_{AVCL} \cdot \text{min} + \text{DRAM tASR}] = 0
\]

\[
125 \text{ ns} + 0 - 44 - 30 - (35 + 0) = 16 \text{ ns}
\]

The 8208's address bus is divided into two halves. ALO-8 becomes the DRAM row address outputs and AHO-8 becomes the column addresses (64K DRAMs would need ALO-7 and AHO-7 connected to the address bus, AL8, AH8 would be tied to Vcc). Internally, the 8208 latches AHO-8 with CAS to provide for tCAH - column address hold time. This latching occurs near the end of T2 for read cycles and near the end of T3 for write cycles. When the RAM cycle is delayed due to refresh, the timing of AACK will ensure the two clock hold requirement. No equation is provided since this happens internally.

[1] Since this is not specified, 0 will be used for analysis only. Based upon design information this value would be about 20 ns.
3.4 WRITE DATA SETUP AND HOLD MARGIN

During write cycles, data from the 80186 must be valid at the DRAM when CAS goes low, and satisfy the DRAM tDS specification. Data must then be held valid and referenced to CAS long enough to meet the DRAM specification tDH. In this design example DEN is the limiting factor in the data setup margin. DEN is active before data is issued by the microprocessor, but there is a significant delay before the buffer is active. The result is that write data will be valid at the buffer before it is fully capable of transmitting data. The margin is referenced to the clock edge that issues DEN and the clock edge that issues CAS, minus delays.

\[
TCHCL + 1 \text{TCLCL} = 8208 \text{TCLCSL} \text{min} (@150 \text{pf}) - \\
80186 \text{TCVCTV max} - 74LS245 \text{TPZH max} - \text{DRAM TDS} _0
\]

\[
55 + 125 + 62.5 - 70 - 40 - 0
\]

\[
= 132 \text{ ns}
\]

The hold margin is referenced to the edge that issues CAS and when valid data disappears. DEN is the controlling signal because it can go inactive before the data bus is floated by the microprocessor.

\[
1 \text{TCLCL} + 1 \text{TCLCH} + 80186 \text{TCVCTX min} + 74LS245 \text{TPLZ min}[1] - \\
8208 \text{TCLCSL max} (@150 \text{pf}) - \text{DRAM TDH}_0
\]

\[
125 \text{ ns} + 55 + 10 + 7.5 - 121 - 30
\]

\[
= 46.5 \text{ ns}
\]

The WE pulse length may cause problems with back-to-back bus cycles. Shortening the pulse width will not cause any other problems. The easiest solution is to factor in a shorter width signal, such as AACK, as is done in the design example.

[1] This parameter is not specified. For analysis, either assume 0 ns or use a more realistic value, such as one-half of typical.
3.5 READ DATA ACCESS MARGIN

The design example requires a buffer in the data path because the 8208 will not stop driving data onto the bus until after the end of T4. With back-to-back bus cycles this would cause bus contention and reduce address to ALE setup margins. The DRAM access parameter used is called "TCAC", and is referenced from the CAS active edge - not RAS. This parameter varies widely between manufacturers. When analyzing read access margins, some trade-off between buffer speed and TCAC delays must be considered.

The 8208 starts a memory cycle, typically, at the end of T1, and data must be valid at the end of T3. With [refresh cycle] delayed bus cycles, data would still have to be valid in two clocks. The timing of the AACK signal guarantees this. From this two clock margin, buffer delays, TCAC delays, and others must be subtracted.

\[
2 \text{TCLCL} - 8208 \text{TCLCSL max (@ 150 pf)} - \text{DRAM TCAC max (@100 pf)} - \text{buffer delays max} - 80186 \text{TDVCL min} = 0
\]

\[
250 \text{ ns} - 121 - 85 - 12 - 20 = 12 \text{ ns}
\]

4.0 SUMMARY

The 8208 solves most of the many design issues faced when adding a dynamic RAM array by giving the designer options. Options for various types of DRAMs, clock speeds, and system configurations. The margins that were just examined showed that the 8208 has plenty of margin in a system. Several margins were even higher. The READ DATA ACCESS MARGIN, for example, is considerably greater. The access time for DRAMS is specified with 100 pf loads, yet this was not added into the equation. Each designer should verify this analysis as specifications from manufacturer's change, without notice.
8208 Timing Diagram
Read then Write Cycle with one Bank of RAM

1. Status setup margin
2. Address setup margin
3. Acknowledge setup margin
4. Write data setup margin
5. Read access margin based upon TCAC
6. Shortened WE pulse

5-211
Dynamic-RAM Controller Orchestrates Memory Systems

Jim Nudif
Mel Basses
Intel Corporation
Santa Clara, California

October 1982

"Reprinted from ELECTRONICS, Sept. 8, 1982 Copyright (c) McGraw-Hill, Inc 1982 All rights reserved."
Dynamic-RAM controller orchestrates memory systems

Up to 88 chips take their cues from an n-channel MOS IC that both housekeeps and supports error-corrected dual-port memories

by Jim Nadir and Mel Bazes, Intel Corp., Santa Clara, Calif

Designing a dynamic-random-access-memory system means balancing the goals of high performance, reliability, and versatility against the often contrary aims of economy, simplicity, and compactness. In the last five or so years, the advent of dynamic-RAM controller chips relieved designers of some of the onus of tending to the needs of dynamic chips: standard supportive integrated circuits brought together the counters, timers, multiplexers, and other elements needed.

But controllers diverged into two types. One bought the high performance to ride with fast memory systems at the expense of functionality, while the other took on more and more functions to do a complete but slower job. The 8207—an advanced dynamic-RAM controller—blunts the horns of that dilemma and also solves a variety of less severe design problems.

A dynamic-RAM controller is charged with making a dynamic memory system appear static to the host processor. At a minimum, therefore, the controller takes over refreshing the memory chips, multiplexing the row and column addresses, generating control signals, timing the precharge period, and signaling the processor when data is available or no longer needed. But, beyond those local housekeeping chores, the controller can also go a long way to solving more global design problems, like sharing memory between two processors, not to mention detecting and correcting errors.

To realize this potential for a highly integrated solution, the 8207 has a dual-port interface and, when used with the 8206 error-checking and -correction unit, ensures data integrity in large dynamic-RAM systems. In addition to doing the jobs of refreshing, address multiplexing, and control timing, the unit supports memory-bank interleaving for pipelined accesses, overlaying RAM and read-only-memory locations, and initializing RAM.

The exact implementation of most of these functions is programmable, letting designers tailor their systems in detail. Systems containing up to 88 dynamic-RAM chips—whether 16-, 64-, or 256-k versions—in one, two, or four banks need only a single 8207 and no external buffering. Attesting to the high performance claimed, the 8207 mates dynamic RAMs having 100-nanosecond access times to the iAPX-286 processor operating at 8 megahertz without introducing any wait states.
To achieve that speed and include all those functions, the 8207 relies on a dense, high-speed n-channel MOS process (n-MOS II) and requires a chip some 230 by 200 mils in area. To meet the rigors of operation with even faster processors, novel logic and integrated-circuit designs are employed. Replacing the two-phase logic common in n-MOS ICs, single-phase edge-triggered logic simplifies logic and circuit design, precludes problems of clock-pulse overlap, and reduces the sensitivity to clock high and low times. Voltage-controlled capacitive loads form the delay elements that time critical output pulses, such as the address strobes, and compensate the output-switching delays for variations in power-supply voltage, temperature, and processing.

A low 20-ns setup time for input signals is achieved by cutting the RC delay of input-protection devices and moving the TTL-to-MOS signal buffering from the input pads to the pulse generators. A short 35-ns delay from input to output switching is achieved by triggering the output generators directly from the external clock, saving a buffer delay time. With the resulting high-speed performance and a high level of integration, the 8207 successfully attacks the stringent requirements of today's memory systems.

One system feature gaining popularity currently is the use of multiple processors operating on shared data to obtain higher performances and reliability. For example, a separate processor dedicated to input/output tasks frees the main processor for full-time data processing. Alternatively, multiple main processors can execute different tasks simultaneously. In all such cases, sharing a common memory space among the cooperating processors is the key to effective operation.

Unfortunately, when more than one processor accesses shared memory through a single bus, the limited bus bandwidth and the time spent in exchanging bus control slow down data transfers. Dual-port memory systems overcome this limitation by giving two processors access to a common memory through two independent buses. The 8207 includes a dual-port interface to simplify the design of shared memory systems.

Two-port memories can be used with multiprocessor or multitasking architectures. In the former, independent processors run independent programs, sharing only a common memory. Multitasking processors cooperate on different parts of the same task.

An example of a multiprocessing architecture is the dynamic video display (Fig. 1) that provides a window on a processor's memory. Centering the display over a data table, for example, immediately reveals how program execution affects the data, which aids in debugging programs. If a microcomputer is implemented with a dual-port memory—the second port for a dynamic video display—then the prototype itself can serve as a development and debugging system, reverting to single-port operation in the final version.

A dual-port architecture in a multitasking environment, on the other hand, adds a margin of safety to a shared-resource bus, such as Intel's Multibus. Although one of the biggest benefits of such a bus is the sharing of expensive peripherals among several users' programs, an intimidating problem is that a single program gone haywire can easily corrupt the entire system. A two-port memory, properly configured, circumvents this occurrence. Because each port has its own address, data, and control lines, problems on one side are confined by hardware to that side.

Port of call

As a general rule for multitasking architectures, one port of a two-port memory operates in a local environment, and the other port runs remotely, off the expandable shared-resource bus. The local processor is likely to require a synchronous port to reap the benefit of higher performance. Remote buses, in contrast, are usually configured asynchronously. Unless programmed other-

Dynamic-RAM controllers get in step

Synchronous and asynchronous signals have different requirements for interfacing with a controller. The terms synchronous and asynchronous are conventionally applied to dynamic random-access memory depending on whether it exists in a local or a remote environment, respectively. However, they more properly characterize the dynamic-RAM controllers, for the RAMs themselves need no clocks—the only restrictions as to the start of a memory access cycle involve ensuring that the refresh and precharge requirements are satisfied.

Because the controller decides both when to refresh and whether or not to precharge and other timing requirements have been met, it does need a clock. Incoming commands can either always arrive with a fixed relationship to the controller's clock or have no particular relationship to it. The former are, of course, synchronous operations, the latter asynchronous.

The major difference between an asynchronous and a synchronous controller (or port of a controller, in the case of the dual-port 8207) is that the asynchronous controller must first synchronize the incoming commands to its own internal clock. From that point on, the asynchronous controller looks just like a synchronous device.

Whereas various techniques for synchronization are available off chip, on-chip synchronization is restricted to the resolution and sampling of states of a flip-flop. The incoming command is clocked into a resolving flip-flop. After a predetermined time, a sampling flip-flop reads the state of the resolving flip-flop, thereby synchronizing the command. Assuming that both flip-flops are triggered on the same edge of the controller's internal clock, the fastest that an asynchronous signal can be synchronized is one clock period. The slowest synchronization takes two clock periods; on the average, getting the signals in step takes one and a half clock cycles.

Because the processor typically requires four or fewer clock periods to complete a cycle, adding a cycle and a half for synchronizing increases the access time by approximately 25%. Synchronous controllers are therefore always preferred when the environment permits them, and local environments, such as single-board computers, generally do so.
wise, the 8207 configures one port synchronously, and the other asynchronously. For specific applications, both ports may be programmed as either synchronous or asynchronous (see "Dynamic-RAM controllers get in step," p. 129).

Whether the ports are programmed for synchronous or asynchronous operation, some mechanism must decide which processor will gain access to memory when both request it almost simultaneously. That mechanism consists of arbitration logic that controls access and always leaves one port selected. When a port is selected, its associated control and interface signals are passed directly to the RAM timing logic by the command multiplexer (Fig. 2). Both ports' command and control lines, after being synchronized, go into both the command multiplexer and the arbitration logic.

However, the arbitration logic enables the command multiplexer to pass only commands that appear at the selected port. At the same time as a command appears at a selected port, arbitration logic initiates the cycle-control logic that completes the timing of the RAM cycle that ensues. If a command appears on the unselected port, it will not get through the multiplexer to initiate a RAM cycle but will instead wait in the status-command multiplexer and the arbitration logic.

The arbitration logic examines all port requests, including the internal refresh port. The refresh-request port is subject to arbitration like the other two ports, except that it is always assigned a higher priority than an unselected external access port. Thus, refreshing can be delayed, at most, one RAM cycle.

While the current RAM cycle is running, the arbiter determines the next cycle to be initiated. Thus, the arbitration time of two or more simultaneous port requests is hidden by the memory cycle time. In other words, in cases where both a selected and an unselected port request access simultaneously, the arbitration time for the unselected port does not extend that port's access time, which is delayed by one memory cycle anyway. Only when an unselected port requests a free memory does the arbitration time slow access, because then the command must pass through the arbitration logic before a RAM cycle can be initiated. To minimize such delays in most cases, there are two arbitration algorithms to be selected by the user.

The first algorithm, intended for multiprocessing environments, automatically returns the arbiter to a designated preferred port, generally the higher-performance, synchronous port. Thus any command on the selected port generally has immediate access, whereas any command arriving at the unselected port must wait.

The second, or last-accessed-port, algorithm, which is applicable in multitasking environments, leaves the most recently accessed port as the selected port. This algorithm optimizes port selection for task passing in a multitasking environment. In task passing, the host processor sends a task to an execution processor; until the task is received, the execution processor seldom accesses memory. Conversely, once the task is passed, the host processor seldom accesses memory until the task is completed. Thus, the ports are used in spurts.

Because timely refreshing is needed to preserve dynamic-RAM data, a refresh request is always serviced on the next available cycle. The refresh algorithm, however, may be selected by the user. The options available are: no refresh, user-generated single refresh, automatic refresh, or user-generated burst refresh.

No refresh would be selected for applications like bit-mapped-video displays, where continuous, sequential access of all RAM locations itself refreshes every cell periodically. User-generated refresh modes allow the designer greater control over power dissipation, for example, in large memory systems. Automatic refreshing, in which the controller itself times the refresh interval and initiates the operation, lets the designer ignore the refresh requirements entirely. As mentioned, the refresh requests are subject to arbitration just like other access requests. However, once a burst refresh is selected, it remains active until completed.

Cleaning up errors

Ensuring data integrity is a major concern in large dynamic-RAM systems, particularly because of their susceptibility to soft errors caused by alpha-particle radiation. Various parity encoding techniques have been developed to detect and correct memory-word errors [Electronics, June 2, 1982, p. 153]. The parity bits, called check bits when used for correction as well as detection, are stored in the memory array along with their associated data word. When the data is read, the check bits are regenerated and compared with the stored check bits. If an error exists, whether in the retrieved check bits or in the retrieved data word, the result of the comparison—called the syndrome—gives the location in the group of the bit in error.

Two drawbacks surface in the design of any memory system that is to be protected by error-correction circuitry. First, the memory-word width must be increased to store the check bits; second, extra time must be allotted for the error-correction circuitry to generate the check bits on write cycles, plus more time to regenerate and compare the check bits on read cycles. The 8207 provides several ways to minimize both problems.

Error-correction schemes require a smaller proportion of check bits to protect wider memory words. For example, an 8-bit word needs 5 check bits, for a 63% increase in memory. Put the other way around, 38% of the available memory would be dedicated to the check bits. Six check bits are required to protect a 16-bit data word—only a 27% overhead. Clearly, the wider the memory array, the more economical the error correction.

The 38% overhead necessary to protect such 8-bit–bus machines as the 8088 or 8085 makes error correction an unattractive proposition. However, if the memory width could be doubled, with the 8088 accessing only half a word at a time, the overhead would drop to 27%.

Reading a double-width word, checking for soft errors, and then sending the desired portion of the word to the processor presents no major problems, unlike writing to such an array. The check bits cannot be calculated from only a portion of the word—they must be calculated for
2. Arbitrer's labor. Two external ports plus the internal refresh port can request access to the memory system at once. Arbitration logic decides which to service, based on programmable algorithms. High-speed logic design cuts the delay from input to output switching to 55 ns.

the entire word at once. Whenever the processor writes a partial word to memory, it must first read the entire word, check it, substitute for that portion of the word to be rewritten, and recalculate the check bits. Only then can the entire word be written to memory. The 8207, working in conjunction with the 8206 error-checking and -correction unit, contains mechanisms to expedite this potentially arduous process.

Whenever the 8207 performs a partial-write cycle, it initiates a read-modify-write cycle wherein the entire memory word is first read and latched into the 8206 (Fig. 3). After the retrieved data has been verified as correct, new data is supplied to the RAM, half from the processor and half from the 8206, which also generates the check bits for the entire new word.

Control signals—called byte marks—specify which portion of the new data word is coming from the processor and which from the 8206. The byte marks determine whether the processor or the 8206 drives the RAM data bus—for example, if the 8206 is driving one portion of the data bus, the processor is prevented from driving the same portion. The byte-mark signals simply disable the appropriate transceivers. If, on the other hand, the processor is driving a portion of the RAM data bus, the byte marks change the 8206 data outputs to inputs, allowing the 8206 to read the data from the processor and calculate new check bits.

The ability of the 8207 to handle memories organized as one, two, or four banks allows tradeoffs between the cost and performance of an error-correction system. For maximum performance, memory would be organized in four banks, each 16 bits wide. In applications requiring error correction, but where maximum performance is not critical, concatenation of RAM banks into two banks of 32-bit words, or even one bank of 64-bit words, can make error correction very economical.

Holding to high performance

Even though the cost of error correction has thus been reduced to where it becomes an attractive solution, the problem remains of minimizing performance degradation. Tackling that challenge depends on the particulars of the configuration, such as whether the memory is to be used with a high-performance local processor, as system memory on a shared-resource bus, or is to be shared between a local high-performance processor and a shared-resource bus.

The method chosen to handle errors depends on the type of bus. Intel's Multibus is the kind that requires data to be valid prior to the issuance of a transfer-
acknowledge signal, in contrast to the local buses of the iAPX-86, -186, and -286 processors. A local bus will usually be synchronous, with a single processor or coprocessor group attached to it; the processor characteristics are known, as is the processor's response to a transfer-acknowledge signal.

With Multibus and other shared-resource buses, the processor types that will eventually be connected are not known in advance, and the buses themselves are generally asynchronous. Hence the time between the transfer-acknowledge signal and data becoming valid is not known. Therefore, the rule with such buses is to acknowledge a transfer only when data is valid. (On some asynchronous buses, the acknowledgment is issued earlier to compensate for synchronization delay at the receiving processor.)

Two basic configurations for checking and correcting errors derive from these system considerations and the fact that it takes longer to correct data than to detect an error. One is for buses that connect to processors and coprocessors receiving a transfer acknowledge prior to data becoming valid, and the other for buses that connect to processors receiving a transfer acknowledge after data is valid. Both configurations are supported by the 8206-8207 team.

Buses among the former type of processors always get corrected data from the 8206, whether an error exists or not, and will carry a transfer acknowledge from the 8207 before data becomes valid on the bus. Though this means data is delayed for error correction on every transaction, the extra delay is immaterial, since it is hidden behind the processor's response time to the transfer-acknowledge signal. By the time the processor requires data, it is already corrected and on the bus. As a result, system performance is not degraded at all because of single-bit errors.

For buses among processors that receive the transfer acknowledge after the data is valid, the 8206 always checks for errors but does not routinely correct data. In this mode, RAM data passes through faster, because the 8207 will issue an acknowledgment sooner. If, however, an error is found, the 8207 will lengthen the cycle, command the 8206 to correct the data, and delay the transfer-acknowledge signal until the corrected data can be placed on the bus. For those buses with an acknowledge-synchronization delay, the 8207 can be programmed to issue the acknowledgment earlier to compensate for the delay.

**Power-up problems**

Another problem with memories protected by ECC circuits crops up when the power is turned on. At power-up, the data stored in memory is completely random; any attempt to read or perform a partial write will be aborted because the check bits will indicate multiple, and therefore uncorrectable, errors. For processors whose word width is the same as that of the memory array, the processor could simply initialize the entire memory array, taking some additional time and software. For memories whose word width is greater than that of the processor, however, initialization of the memory is not possible unless the error-checking or -correction circuitry is disabled by hardware, for example, by gating off the error flags.

The 8207 is equipped to deal with the initialization problem by itself. At system reset, the 8207 performs
4. Interleaving. Overlapping accesses to different banks increases memory throughput. Once the column-address hold time is satisfied, the 8207 starts a second cycle, pulling the second row-address strobe low.

Eight cycles on all banks at once to warm up the dynamic RAMs, a typical RAM requirement for stable operation. The chip then individually initializes all memory locations to 0, adding the proper check bits. Though all memory banks could be initialized in parallel, that would require more power than any other memory operation, calling for a heftier and more expensive power supply needed only at system reset.

One final problem associated with memories protected by error-correction circuitry stems from the fact that only data that is accessed by the processor is corrected. If the processor continually accesses one particular segment of memory, the rest of the array may be accumulating soft errors. The possibility of two soft errors accumulating in a word of seldom accessed memory now becomes significant—and not all double-bit errors are correctable in simple ECC schemes. The 8207 scrubs memories to clean up this problem. During each refresh cycle, one word of memory is read, checked for errors, and if necessary, corrected before data is written back to memory. Because scrubbing occurs during refresh cycles with a read cycle replacing a row-address-strobe-only refresh cycle, no performance penalty is incurred. Scrubbing rids the entire memory of errors at least once every 16 seconds, reducing the probability of two soft errors accumulating in the same word almost to nil.

Bells and whistles

All dynamic RAMs require a recovery period for precharging internal lines after each access. If the processor were immediately to reaccess the RAM, the controller would have to delay it until the precharge time was over. By automatically organizing memory into banks so that sequential addresses are in different banks, the 8207 is usually able to hide the precharge time of one bank behind the access time of another. That organization follows from using the 2 least significant bits of the address to select the bank. Of course, a break in the program flow, such as would be caused by a jump or call instruction, raises the probability that the same bank may be immediately re-accessed. This probability is less in four-bank memories than in two-bank configurations.

Further performance advantages are gleaned by organizing memory into multiple banks. For example, the 8207 can speed throughput by pipelining cycles. Once the row and column addresses to one bank have been latched, the controller sends the row address for the next cycle to the next bank (Fig. 4).

The 8207's manifold features can be tailored to a given system with the use of a serial programming pin. This pin can either be strapped high or low to select one of two default modes or be programmed by means of a shift register. The external register is completely controlled by the 8207, eliminating any local processor support. Sixteen bits are shifted into the 8207 to configure up to nine different features. The bits are arranged in order of increasing importance; using a shift register with less than 16 bits permits just those features needed to be programmed.

Programmable features of the processor interface include the choice of arbitration algorithm, clock compensation, and preferred port. At the RAM interface, the user can specify fast or slow memory chips, indicate bank configuration, and select the optimal refreshing scheme. In anticipation of the next generation of 256-K dynamic RAMs, the 8207 can support a 256-row--1-millisecond refresh convention, in addition to the 128-row--2-ms one for current 16- and 64-K parts.

Helping facilitate system design is a self-programming processor interface. By decoding the command input pins at power-up, the 8207 automatically determines whether it is connected to the status lines of an 8086, iAPX-286 or to the command lines of the Multibus. Because the 8207 can directly decode the status lines of Intel microprocessors, it can anticipate the next memory cycle and start a new cycle before actually receiving a command. This extra pipelining enables the designer to specify slower RAMs then would otherwise be required.
The Intel® 8231A Arithmetic Processing Unit (APU) is a monolithic HMOS LSI device that provides high performance fixed and floating point arithmetic and floating point trigonometric operations. It may be used to enhance the mathematical capability of a wide variety of processor-oriented systems. Chebyshev polynomials are used in the implementation of the APU algorithms.

All transfers, including operand, result, status and command information, take place over an 8-bit bidirectional data bus. Operands are pushed onto an internal stack and commands are issued to perform operations on the data in the stack. Results are then available to be retrieved from the stack.

Transfers to and from the APU may be handled by the associated processor using conventional programmed I/O, or may be handled by a direct memory access controller for improved performance. Upon completion of each command, the APU issues an end of execution signal that may be used as an interrupt by the CPU to help coordinate program execution.
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>2</td>
<td></td>
<td>Power: +5 Volt power supply.</td>
</tr>
<tr>
<td>VDD</td>
<td>16</td>
<td></td>
<td>Power: +12 Volt power supply.</td>
</tr>
<tr>
<td>VSS</td>
<td>1</td>
<td></td>
<td>Ground.</td>
</tr>
<tr>
<td>CLK</td>
<td>23</td>
<td>I</td>
<td>Clock: An external, TTL compatible, timing source is applied to the CLK pin.</td>
</tr>
<tr>
<td>RESET</td>
<td>22</td>
<td>I</td>
<td>Reset: The active high reset signal provides initialization for the chip. RESET also terminates any operation in progress. RESET clears the status register and places the 8231A into the idle state. Stack contents and command registers are not affected (5 clock cycles).</td>
</tr>
<tr>
<td>CS</td>
<td>18</td>
<td>I</td>
<td>Chip Select: CS is an active low input signal which selects the 8231A and enables communication with the data bus.</td>
</tr>
<tr>
<td>A0</td>
<td>21</td>
<td>I</td>
<td>Address: In conjunction with the RD and WR signals, the A0 control line establishes the type of communication that is to be performed with the 8231A as shown below:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A0</td>
<td>RD</td>
<td>WR</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>RD</td>
<td>20</td>
<td>I</td>
<td>Read: This active low input indicates that data or status is to be read from the 8231A if CS is low.</td>
</tr>
<tr>
<td>WR</td>
<td>19</td>
<td>I</td>
<td>Write: This active low input indicates that data or a command is to be written into the 8231A if CS is low.</td>
</tr>
<tr>
<td>EACK</td>
<td>3</td>
<td>I</td>
<td>End of Execution: This active low input clears the end of execution output signal (END). If EACK is tied low, the END output will be a pulse that is one clock period wide.</td>
</tr>
<tr>
<td>SVACK</td>
<td>4</td>
<td>I</td>
<td>Service Request: This active low input clears the service request output (SVREQ).</td>
</tr>
<tr>
<td>END</td>
<td>24</td>
<td>O</td>
<td>End: This active low, open-drain output indicates that execution of the previously entered command is complete. It can be used as an interrupt request and is cleared by EACK, RESET or any read or write access to the 8231.</td>
</tr>
</tbody>
</table>

COMMAND STRUCTURE

Each command entered into the 8231A consists of a single 8-bit byte having the format illustrated below:

<table>
<thead>
<tr>
<th>Bits 7-2</th>
<th>SINGLE</th>
<th>.bits 5-0</th>
<th>FIXED</th>
<th>OPERATION CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Bits 0-4 select the operation to be performed as shown in the table. Bits 5-6 select the data format appropriate to the selected operation. If bit 5 is a 1, a fixed point data format is specified. If bit 5 is a 0, floating point format is specified. Bit 6 selects the precision of the data to be operated upon by fixed point commands only (if bit 5 = 0, bit 6 must be 0). If bit 6 is 1, single-precision (16-bit) operands are assumed. If bit 6 is 0, double-precision (32-bit) operands are indicated. Results are undefined for all illegal combinations of bits in the command byte. Bit 7 indicates whether a service request is to be issued after the command is executed. If bit 7 is a 1, the service request output (SVREQ) will go high at the conclusion of the command and will remain high until reset by a low level on the service acknowledge pin (SVACK) or until completion of execution of the succeeding command where service request (bit 7) is 0. Each command issued to the 8231A requests post execution service based upon the state of bit 7 in the command byte. When bit 7 is a 0, SVREQ remains low.
Table 2. 32-Bit Floating Point Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Hex(1) Code</th>
<th>Stack Contents(2) After Execution</th>
<th>Status Flags(3) Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS</td>
<td>Inverse Cosine of A</td>
<td>0 6</td>
<td>R U U U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>ASIN</td>
<td>Inverse Sine of A</td>
<td>0 5</td>
<td>R U U U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>ATAN</td>
<td>Inverse Tangent of A</td>
<td>0 7</td>
<td>R B U U</td>
<td>S, Z</td>
</tr>
<tr>
<td>CHSF</td>
<td>Sign Change of A</td>
<td>0 3</td>
<td>R B C D</td>
<td>S, Z</td>
</tr>
<tr>
<td>COS</td>
<td>Cosine of A (radians)</td>
<td>0 3</td>
<td>R B U U</td>
<td>S, Z</td>
</tr>
<tr>
<td>EXP</td>
<td>e^A Function</td>
<td>0 A</td>
<td>R B U U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>ADD</td>
<td>Add A and B</td>
<td>0 1</td>
<td>R C D U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>FDIV</td>
<td>Divide B by A</td>
<td>0 13</td>
<td>R C D U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiply A and B</td>
<td>0 1</td>
<td>R C D U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>FIXS</td>
<td>Subtract A from B</td>
<td>0 1</td>
<td>R C D U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>LOG</td>
<td>Common Logarithm (base 10) of A</td>
<td>0 0</td>
<td>R B U U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>POPF</td>
<td>Stack Pop</td>
<td>0 1</td>
<td>B C D A</td>
<td>S, Z</td>
</tr>
<tr>
<td>PTOF</td>
<td>Stack Push</td>
<td>0 1</td>
<td>A B C</td>
<td>S, Z</td>
</tr>
<tr>
<td>PUI</td>
<td>Push A onto Stack</td>
<td>0 1</td>
<td>R A B C</td>
<td>S, Z</td>
</tr>
<tr>
<td>PWR</td>
<td>B^A Power Function</td>
<td>0 1</td>
<td>R C D U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>SIN</td>
<td>Sine of A (radians)</td>
<td>0 0</td>
<td>R B U U</td>
<td>S, Z</td>
</tr>
<tr>
<td>SQRT</td>
<td>Square Root of A</td>
<td>0 0</td>
<td>R B C U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>TAN</td>
<td>Tangent of A (radians)</td>
<td>0 0</td>
<td>R B U U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>XCHF</td>
<td>Exchange A and B</td>
<td>0 0</td>
<td>B A C D</td>
<td>S, Z</td>
</tr>
</tbody>
</table>

Table 3. 32-Bit Integer Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Hex(1) Code</th>
<th>Stack Contents(2) After Execution</th>
<th>Status Flags(3) Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHSD</td>
<td>Sign Change of A</td>
<td>0 3</td>
<td>R B C D</td>
<td>S, Z, O</td>
</tr>
<tr>
<td>DADD</td>
<td>Add A and B</td>
<td>0 2</td>
<td>R C D A</td>
<td>S, Z, C, E</td>
</tr>
<tr>
<td>DDIV</td>
<td>Divide B by A</td>
<td>0 2</td>
<td>R C D U</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>DMUL</td>
<td>Multiply A and B (R = lower 32-bits)</td>
<td>0 2</td>
<td>R C D U</td>
<td>S, Z, O</td>
</tr>
<tr>
<td>DMUU</td>
<td>Multiply A and B (R = upper 32-bits)</td>
<td>0 2</td>
<td>R C D U</td>
<td>S, Z, O</td>
</tr>
<tr>
<td>DSUB</td>
<td>Subtract A from B</td>
<td>0 1</td>
<td>R C D A</td>
<td>S, Z, C, O</td>
</tr>
<tr>
<td>FIXD</td>
<td>Floating Point to Integer Conversion</td>
<td>0 1</td>
<td>R B C U</td>
<td>S, Z, O</td>
</tr>
<tr>
<td>POPD</td>
<td>Stack Pop</td>
<td>0 1</td>
<td>B C D A</td>
<td>S, Z</td>
</tr>
<tr>
<td>PTOD</td>
<td>Stack Push</td>
<td>0 1</td>
<td>A B C</td>
<td>S, Z</td>
</tr>
<tr>
<td>XCHD</td>
<td>Exchange A and B</td>
<td>0 1</td>
<td>B A C D</td>
<td>S, Z</td>
</tr>
</tbody>
</table>

Table 4. 16-Bit Integer Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Hex(1) Code</th>
<th>Stack Contents(2) After Execution</th>
<th>Status Flags(3) Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHSS</td>
<td>Change Sign of A</td>
<td>0 3</td>
<td>R L</td>
<td>S, Z, O</td>
</tr>
<tr>
<td>FIXS</td>
<td>Floating Point to Integer Conversion</td>
<td>0 1</td>
<td>R B U L</td>
<td>S, Z, O</td>
</tr>
<tr>
<td>POPS</td>
<td>Stack Pop</td>
<td>0 7</td>
<td>A U L</td>
<td>S, Z</td>
</tr>
<tr>
<td>PTOS</td>
<td>Stack Push</td>
<td>0 7</td>
<td>A U L</td>
<td>S, Z</td>
</tr>
<tr>
<td>SADD</td>
<td>Add A and A</td>
<td>0 6</td>
<td>R B U L</td>
<td>S, Z, C, E</td>
</tr>
<tr>
<td>SDIV</td>
<td>Divide A by A</td>
<td>0 6</td>
<td>R B U L</td>
<td>S, Z, C, E</td>
</tr>
<tr>
<td>SMUL</td>
<td>Multiply A by A (R = lower 16-bits)</td>
<td>0 6</td>
<td>R B U L</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>SMUO</td>
<td>Multiply A by A (R = upper 16-bits)</td>
<td>0 6</td>
<td>R B U L</td>
<td>S, Z, E</td>
</tr>
<tr>
<td>SSUB</td>
<td>Subtract A from A</td>
<td>0 6</td>
<td>R B U L</td>
<td>S, Z, C, E</td>
</tr>
<tr>
<td>XCHS</td>
<td>Exchange A and A</td>
<td>0 6</td>
<td>A U L</td>
<td>S, Z, C, E</td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td>0 6</td>
<td>A U L</td>
<td>S, Z, C, E</td>
</tr>
</tbody>
</table>

Notes: 1. In the hex code column, SVREQ is a 0.
2. The stack initially is composed of four 32-bit numbers (A, B, C, D). A is equivalent to Top Of Stack (TOS) and B is Next On Stack (NOS). Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A, B, C, or D).
3. The stack initially is composed of eight 16-bit numbers (A_U, A_L, B_U, B_L, C_U, C_L, D_U, D_L). A_U is the TOS and A_L is NOS. Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A_U, A_L, B_U, B_L, . . .).
4. Nomenclature: Sign (S); Zero (Z); Overflow (O); Carry (C); Error Code Field (E).
DATA FORMATS

The 8231A arithmetic processing unit handles operands in both fixed point and floating point formats. Fixed point operands may be represented in either single (16-bit operands) or double precision (32-bit operands), and are always represented as binary, two’s complement values.

SINGLE PRECISION FIXED POINT FORMAT

```
VALUE
```

The sign (positive or negative) of the operand is located in the most significant bit (MSB). Positive values are represented by a sign bit of zero (S = 0). Negative values are represented by the two’s complement of the corresponding positive value with a sign bit equal to 1 (S = 1). The range of values that may be accommodated by each of these formats is -32,768 to +32,767 for single precision and -2,147,483,648 to +2,147,483,647 for double precision.

Floating point binary values are represented in a format that permits arithmetic to be performed in a fashion analogous to operations with decimal values expressed in scientific notation.

\[(5.83 \times 10^2) (8.16 \times 10^1) = (4.75728 \times 10^4)\]

In the decimal system, data may be expressed as values between 0 and 10 times 10 raised to a power that effectively shifts the implied decimal point right or left the number of places necessary to express the result in conventional form (e.g., 47,572.8). The value portion of the data is called the mantissa. The exponent may be either negative or positive.

The concept of floating point notation has both a gain and a loss associated with it. The gain is the ability to represent the significant digits of data with values spanning a large dynamic range limited only by the capacity of the exponent field. For example, in decimal notation if the exponent field is two digits wide, and the mantissa is five digits, a range of values (positive or negative) from \(1.0000 \times 10^{-99}\) to \(9.9999 \times 10^{99}\) can be accommodated. The loss is that only the significant digits of the value can be represented. Thus there is no distinction in this representation between the values 123451 and 123452, for example, since each would be expressed as: \(1.2345 \times 10^5\). The sixth digit has been discarded. In most applications where the dynamic range of values to be represented is large, the loss of significance, and hence accuracy of results, is a minor consideration. For greater precision a fixed point format could be chosen, although with a loss of potential dynamic range.

The 8231A is a binary arithmetic processor and requires that floating point data be represented by a fractional mantissa value between .5 and 1 multiplied by 2 raised to an appropriate power. This is expressed as follows:

\[\text{value} = \text{mantissa} \times 2^{\text{exponent}}\]

For example, the value 100.5 expressed in this form is \(0.1100 \times 27\). The decimal equivalent of this value may be computed by summing the components (powers of two) of the mantissa and then multiplying by the exponent as shown below:

\[\text{value} = (2^{-1} + 2^{-2} + 2^{-5} + 2^{-6}) \times 2^7\]
\[= 0.5 + 0.25 + 0.03125 + 0.002963\] \(\times 128\)
\[= 0.78515625 \times 128\]
\[= 100.5\]

FLOATING POINT FORMAT

The format for floating point values in the 8231A is given below. The mantissa is expressed as a 24-bit (fractional) value; the exponent is expressed as a two’s complement 7-bit value having a range of \(-64\) to \(+63\). The most significant bit is the sign of the mantissa (0 = positive, 1 = negative), for a total of 32 bits. The binary point is assumed to be to the left of the most significant mantissa bit (bit 23). All floating point data values must be normalized. Bit 23 must be equal to 1, except for the value zero, which is represented by all zeros.

The range of values that can be represented in this format is \pm (2.7 \times 10^{-20} \text{ to } 9.2 \times 10^{18}) \text{ and zero.}

FUNCTIONAL DESCRIPTION

STACK CONTROL

The user interface to the 8231A includes access to an 8 level 16-bit wide data stack. Since single precision fixed point operands are 16-bits in length, eight such values may be maintained in the stack. When using double precision fixed point or floating point formats four values may be stored. The stack in these two configurations can be visualized as shown below:

```
<table>
<thead>
<tr>
<th>TOS</th>
<th>A2</th>
<th>A1</th>
<th>B2</th>
<th>B1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOS</td>
<td>A4</td>
<td>A3</td>
<td>A2</td>
<td>A1</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Data are written onto the stack, eight bits at a time, in the order shown (A1, A2, A3, ... .). Data are removed from the stack in reverse byte order (A4, A3, A2 ... .). Data should be entered onto the stack in multiples of the number of bytes appropriate to the chosen data format.
DATA ENTRY

Data entry is accomplished by bringing the chip select (CS), the command/data line (A0), and WR low, as shown in the timing diagram. The entry of each new data word "pushes down" the previously entered data and places the new byte on the top of stack (TOS). Data on the bottom of the stack prior to a stack entry are lost.

DATA REMOVAL

Data are removed from the stack in the 8231A by bringing chip select (CS), command/data (A0), and RD low as shown in the timing diagram. The removal of each data word redefines TOS so that the next successive byte to be removed becomes TOS. Data removed from the stack rotates to the bottom of the stack.

COMMAND ENTRY

After the appropriate number of bytes of data have been entered onto the stack, a command may be issued to perform an operation on that data. Commands which require two operands for execution (e.g., add) operate on the TOS and NOS values. Single operand commands operate only on the TOS.

Commands are issued to the 8231A by bringing the chip select (CS) line low, command data (A0) line high, and WR line low as indicated by the timing diagram. After a command is issued, the CPU can continue execution of its program concurrently with the 8231A command execution.

COMMAND COMPLETION

The 8231A signals the completion of each command execution by lowering the End Execution line (END). Simultaneously, the busy bit in the status register is cleared and the Service Request bit of the command register is checked. If it is a "1" the service request output level (SVREQ) is raised. END is cleared on receipt of an active low End Acknowledge (EACK) pulse. Similarly, the service request line is cleared by recognition of an active low Service Acknowledge (SVACK) pulse.

READY OPERATION

An active high ready (READY) is provided. This line is high in its quiescent state and is pulled low by the 8231A under the following conditions:

1. A previously initiated operation is in progress (device busy) and Command Entry has been attempted. In this case, the READY line will be pulled low and remain low until completion of the current command execution. It will then go high, permitting entry of the new command.

2. A previously initiated operation is in progress and stack access has been attempted. In this case, the READY line will be pulled low, will remain in that state until execution is complete, and will then be raised to permit completion of the stack access.

3. The 8231A is not busy, and data removal has been requested. READY will be pulled low for the length of time necessary to transfer the byte from the top of stack to the interface latch, and will then go high, indicating availability of the data.

4. The 8231A is not busy, and a data entry has been requested. READY will be pulled low for the length of time required to ascertain if the preceding data byte, if any, has been written to the stack. If so READY will immediately go high. If not, READY will remain low until the interface latch is free and will then go high.

5. When a status read has been requested, READY will be pulled low for the length of time necessary to transfer the status to the interface latch, and will then be raised to permit completion of the status read. Status may be read whether or not the 8231A is busy.

When READY goes low, the APU expects the bus control signals present at the time to remain stable until READY goes high.

DEVICE STATUS

Device status is provided by means of an internal status register whose format is shown below:

<table>
<thead>
<tr>
<th>BUSY</th>
<th>SIGN</th>
<th>ZERO</th>
<th>ERROR CODE</th>
<th>CARRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BUSY: Indicates that 8231A is currently executing a command (1 = Busy)
SIGN: Indicates that the value on the top of stack is negative (1 = Negative)
ZERO: Indicates that the value on the top of stack is zero (1 = Value is zero)
ERROR CODE: This field contains an indication of the validity of the result of the last operation. The error codes are:

- 0000 — No error
- 1000 — Divide by zero
- 0100 — Square root or log of negative number
- 1100 — Argument of inverse sine, cosine, or \( e^x \) too large
- XX10 — Underflow
- XX01 — Overflow

CARRY: Previous operation resulted in carry or borrow from most significant bit. (1 = Carry/Borrow, 0 = No Carry/No Borrow.)

If the BUSY bit in the status register is a one, the other status bits are not defined; if zero, indicating not busy, the operation is complete and the other status bits are defined as given above.

READ STATUS

The 8231A status register can be read by the CPU at any time (whether an operation is in progress or not) by bringing the chip select (CS) low, the command/data line (A0) high, and lowering RD. The status register is then gated onto the data bus and may be input by the CPU.

EXECUTION TIMES

Timing for execution of the 8231A command set is contained below. All times are given in terms of clock cycles. Where substantial variation of execution times
is possible, the minimum and maximum values are quoted; otherwise, typical values are given. Variations are data dependent.

Total execution times may require allowances for operand transfer into the APU, command execution, and result retrieval from the APU. Except for command execution, these times will be heavily influenced by the nature of the data, the control interface used, the speed of memory, the CPU used, the priority allotted to DMA and Interrupt operations, the size and number of operands to be transferred, and the use of chained calculations, etc.

### Table 5. Command Execution Times

<table>
<thead>
<tr>
<th>Command Mnemonic</th>
<th>Clock Cycles</th>
<th>Command Mnemonic</th>
<th>Clock Cycles</th>
<th>Command Mnemonic</th>
<th>Clock Cycles</th>
<th>Command Mnemonic</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>SADD</td>
<td>17</td>
<td>FADD</td>
<td>54-368</td>
<td>LN</td>
<td>4298-6956</td>
<td>POPF</td>
<td>12</td>
</tr>
<tr>
<td>SSUB</td>
<td>30</td>
<td>FSUB</td>
<td>70-370</td>
<td>EXP</td>
<td>3794-4878</td>
<td>XCHS</td>
<td>18</td>
</tr>
<tr>
<td>SMUL</td>
<td>84-94</td>
<td>FMUL</td>
<td>146-168</td>
<td>PWR</td>
<td>8290-12032</td>
<td>XCHD</td>
<td>26</td>
</tr>
<tr>
<td>SMUO</td>
<td>80-98</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDIV</td>
<td>84-94</td>
<td>FDIV</td>
<td>154-184</td>
<td>NOP</td>
<td>4</td>
<td>XCHF</td>
<td>26</td>
</tr>
<tr>
<td>DADD</td>
<td>21</td>
<td>SORT</td>
<td>800</td>
<td>CHSS</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSUB</td>
<td>38</td>
<td>SIN</td>
<td>4464</td>
<td>CHSD</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMUL</td>
<td>194-210</td>
<td>COS</td>
<td>4118</td>
<td>CHSF</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMUO</td>
<td>182-218</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDIV</td>
<td>208</td>
<td>TAN</td>
<td>5754</td>
<td>PTOS</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIXS</td>
<td>92-216</td>
<td>ASIN</td>
<td>7668</td>
<td>PTOD</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIXD</td>
<td>100-346</td>
<td>ACOS</td>
<td>7734</td>
<td>PTOF</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLT8</td>
<td>98-186</td>
<td>ATAN</td>
<td>6006</td>
<td>POPS</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLTD</td>
<td>98-378</td>
<td>LOG</td>
<td>4474-7132</td>
<td>POPD</td>
<td>12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### DERIVED FUNCTION DISCUSSION

Computer approximations of transcendental functions are often based on some form of polynomial equation, such as:

\[ F(X) = A_0 + A_1X + A_2X^2 + A_3X^3 + A_4X^4 \ldots \]  \hspace{1cm} (1-1)

The primary shortcoming of an approximation in this form is that it typically exhibits very large errors when the magnitude of \( |X| \) is large, although the errors are small when \( |X| \) is small. With polynomials in this form, the error distribution is markedly uneven over any arbitrary interval.

A set of approximating functions exists that not only minimizes the maximum error but also provides an even distribution of errors within the selected data representation interval. These are known as Chebyshev Polynomials and are based upon cosine functions. These functions are defined as follows:

\[ T_n(X) = \cos n\theta; \text{ where } n = 0,1,2 \ldots \]  \hspace{1cm} (1-2)

\[ \theta = \cos^{-1}X \]

The various terms of the Chebyshev series can be computed as shown below:

\[ T_0(X) = \cos (0 \cdot \theta) = \cos (0) = 1 \]  \hspace{1cm} (1-4)

\[ T_1(X) = \cos (\cos^{-1}X) = X \]  \hspace{1cm} (1-5)

\[ T_2(X) = \cos 2\theta = 2\cos^2 \theta - 1 = 2\cos^2 (\cos^{-1}X) - 1 \]  \hspace{1cm} (1-6)

In general, the next term in the Chebyshev series can be recursively derived from the previous term as follows:

\[ T_n(X) = 2X [T_{n-1}(X)] - T_{n-2}(X); n \geq 2 \]  \hspace{1cm} (1-7)

Common logarithms are computed by multiplication of the natural logarithm by the conversion factor 0.43429448 and the error function is therefore the same as that for natural logarithm. The power function is realized by combination of natural log and exponential functions according to the equation:

\[ XY = e^{Y \ln X} \]

The error for the power function is a combination of that for the logarithm and exponential functions.

Each of the derived functions is an approximation of the true function. Thus the result of a derived function will have an error. The absolute error is the difference between the function's result and the true result. A more useful measure of the function's error is relative error (absolute error/true result). This gives a measurement of the significant digits of algorithm accuracy. For the derived functions except LN, LOG, and PWR the relative error is typically \( 4 \times 10^{-7} \). For PWR the relative error is the summation of the EXP and LN errors, \( 7 \times 10^{-7} \). For LN and LOG, the absolute error is \( 2 \times 10^{-7} \).
APPLICATION INFORMATION

The diagram in Figure 4 shows the interface connections for the APU with operand transfers handled by an 8237 DMA controller, and CPU coordination handled by an Interrupt Controller. The APU interrupts the CPU to indicate that a command has been completed. When the performance enhancements provided by the DMA and Interrupt operations are not required, the APU interface can be simplified as shown in Figure 3. The 8231A APU is designed with a general purpose 8-bit data bus and interface control so that it can be conveniently used with any general 8-bit processor.

In many systems it will be convenient to use the microcomputer system clock to drive the APU clock input. In the case of 8080A systems it would be the 82TTL signal. Its cycle time will usually fall in the range of 250 ns to 1000 ns, depending on the system speed.
ABSOLUTE MAXIMUM RATINGS

Storage Temperature .................................. -65°C to +150°C
Ambient Temperature Under Bias ................. 0°C to 70°C
V_{DD} with Respect to V_{SS} ...................... -0.5V to +15.0V
V_{CC} with Respect to V_{SS} ...................... -0.5V to +7.0V
All Signal Voltages with Respect
to V_{SS} ........................................... -0.5V to +7.0V
Power Dissipation ................................... 2.0W

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may effect device reliability.

D.C. AND OPERATING CHARACTERISTICS (T_A = 0°C to 70°C, V_{SS} = 0V, V_{CC} = ±5V ± 10%, V_{DD} = ±12V ± 10%)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_{OH}</td>
<td>Output HIGH Voltage</td>
<td>3.7</td>
<td></td>
<td></td>
<td>Volts</td>
<td>I_{OH} = -200 μA</td>
</tr>
<tr>
<td>V_{OL}</td>
<td>Output LOW Voltage</td>
<td></td>
<td>0.4</td>
<td></td>
<td>Volts</td>
<td>I_{OL} = 3.2 mA</td>
</tr>
<tr>
<td>V_{IH}</td>
<td>Input HIGH Voltage</td>
<td>2.0</td>
<td></td>
<td>V_{CC}</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>V_{IL}</td>
<td>Input LOW Voltage</td>
<td>-0.5</td>
<td></td>
<td>0.8</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>I_{IL}</td>
<td>Input Load Current</td>
<td>±10</td>
<td></td>
<td></td>
<td>μA</td>
<td>V_{SS} ≤ V_{IN} ≤ V_{CC}</td>
</tr>
<tr>
<td>I_{OFL}</td>
<td>Data Bus Leakage</td>
<td>±10</td>
<td></td>
<td></td>
<td>μA</td>
<td>V_{SS} +0.45 ≤ V_{OUT} ≤ V_{CC}</td>
</tr>
<tr>
<td>I_{CC}</td>
<td>V_{CC} Supply Current</td>
<td>50</td>
<td>95</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>I_{DD}</td>
<td>V_{DD} Supply Current</td>
<td>50</td>
<td>95</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>C_O</td>
<td>Output Capacitance</td>
<td>8</td>
<td></td>
<td></td>
<td>pF</td>
<td>fc = 1.0 MHz, Inputs = 0V</td>
</tr>
<tr>
<td>C_I</td>
<td>Input Capacitance</td>
<td>5</td>
<td></td>
<td></td>
<td>pF</td>
<td></td>
</tr>
<tr>
<td>C_{IO}</td>
<td>I/O Capacitance</td>
<td>10</td>
<td></td>
<td></td>
<td>pF</td>
<td></td>
</tr>
</tbody>
</table>

A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

C_L = 150 pF
A.C. CHARACTERISTICS  \( (T_A = 0^\circ C \text{ to } 70^\circ C, \ V_{SS} = 0V, \ V_{CC} = +5V \pm 10\%, \ V_{DD} = +12V \pm 10\%) \)

READ OPERATION

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>( 8231A-8 )</th>
<th>( 8231A )</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{AR} )</td>
<td>( AO, CS ) Setup to ( RD )</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{RA} )</td>
<td>( AO, CS ) Hold from ( RD )</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{RY} )</td>
<td>READY ( \uparrow ) from ( RD ) ( \downarrow ) Delay (Note 2)</td>
<td>150</td>
<td>100</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{YR} )</td>
<td>READY ( \uparrow ) to ( RD ) ( \downarrow )</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{RRR} )</td>
<td>READY Pulse Width (Note 3)</td>
<td>3.5 ( t_{CY} ) + 50</td>
<td>3.5 ( t_{CY} ) + 50</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td>1.5 ( t_{CY} ) + 50</td>
<td>1.5 ( t_{CY} ) + 50</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>Status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{RDE} )</td>
<td>Data Bus Enable from ( RD ) ( \downarrow )</td>
<td>50</td>
<td>50</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{DRY} )</td>
<td>Data Valid to READY ( \uparrow )</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{DF} )</td>
<td>Data Float after ( RD ) ( \downarrow )</td>
<td>50</td>
<td>200</td>
<td>50</td>
</tr>
</tbody>
</table>

WRITE OPERATION

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>( 8231A-8 )</th>
<th>( 8231A )</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{AW} )</td>
<td>( AO, CS ) Setup to ( WR )</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{WA} )</td>
<td>( AO, CS ) Hold after ( WR )</td>
<td>60</td>
<td>25</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{WY} )</td>
<td>READY ( \uparrow ) from ( WR ) ( \downarrow ) Delay (Note 2)</td>
<td>150</td>
<td>100</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{YW} )</td>
<td>READY ( \uparrow ) to ( WR ) ( \downarrow )</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{RRW} )</td>
<td>READY Pulse Width (Note 4)</td>
<td>50</td>
<td>50</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{WI} )</td>
<td>Write Inactive Time (Note 4)</td>
<td>4 ( t_{CY} )</td>
<td>4 ( t_{CY} )</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>Command</td>
<td>5 ( t_{CY} )</td>
<td>5 ( t_{CY} )</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{DW} )</td>
<td>Data Setup to ( WR )</td>
<td>150</td>
<td>100</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{WD} )</td>
<td>Data Hold after ( WR )</td>
<td>20</td>
<td>20</td>
<td>ns</td>
</tr>
</tbody>
</table>

OTHER TIMINGS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>( 8231A-8 )</th>
<th>( 8231A )</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{CY} )</td>
<td>Clock Period</td>
<td>480</td>
<td>5000</td>
<td>250</td>
</tr>
<tr>
<td>( t_{CPH} )</td>
<td>Clock Pulse High Width</td>
<td>200</td>
<td>100</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{CPL} )</td>
<td>Clock Pulse Low Width</td>
<td>240</td>
<td>120</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{EE} )</td>
<td>END Pulse Width (Note 5)</td>
<td>400</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{EAE} )</td>
<td>EACK ( \uparrow ) to END ( \downarrow ) Delay</td>
<td>200</td>
<td>150</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{AA} )</td>
<td>EACK Pulse Width</td>
<td>100</td>
<td>50</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{SA} )</td>
<td>SVACK ( \uparrow ) to SVREQ ( \downarrow ) Delay</td>
<td>300</td>
<td>150</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{SS} )</td>
<td>SVACK Pulse Width</td>
<td>100</td>
<td>50</td>
<td>ns</td>
</tr>
</tbody>
</table>

NOTES:
1. Typical values are for \( T_A = 25^\circ C \), nominal supply voltages and nominal processing parameters.
2. READY is pulled low for both command and data operations.
3. Minimum values shown assume no previously entered command is being executed for the data access. If a previously entered command is being executed, READY low pulse width is the time to complete execution plus the time shown. Status may be read at any time without exceeding the time shown.
4. READY low pulse width is less than 50 ns when writing into the data port or the control port as long as the duty cycle requirement \( t_{WI} \) is observed and no previous command is being executed. \( t_{WI} \) may be safely violated as long as the extended \( t_{RRW} \) that results is observed. If a previously entered command is being executed, READY low pulse width is the time to complete execution plus the time shown. These timings refer specifically to the \( 8231A \).
5. END low pulse width is specified for EACK tied to VSS. Otherwise \( t_{EAE} \) applies.
The Intel® 8253 is a programmable counter/timer device designed for use as an Intel microcomputer peripheral. It uses nMOS technology with a single +5V supply and is packaged in a 24-pin plastic DIP.

It is organized as 3 independent 16-bit counters, each with a count rate of up to 2.6 MHz. All modes of operation are software programmable.

**Figure 1. Block Diagram**

**Figure 2. Pin Configuration**
FUNCTIONAL DESCRIPTION

General

The 8253 is a programmable interval timer/counter specifically designed for use with the Intel® Microcomputer systems. Its function is that of a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 8253 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in systems software, the programmer configures the 8253 to match his requirements, initializes one of the counters of the 8253 with the desired quantity, then upon command the 8253 will count out the delay and interrupt the CPU when it has completed its tasks. It is easy to see that the software overhead is minimal and that multiple delays can easily be maintained by assignment of priority levels.

Other counter/timer functions that are non-delay in nature but also common to most microcomputers can be implemented with the 8253.

• Programmable Rate Generator
• Event Counter
• Binary Rate Multiplier
• Real Time Clock
• Digital One-Shot
• Complex Motor Controller

Data Bus Buffer

This 3-state, bi-directional, 8-bit buffer is used to interface the 8253 to the system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput CPU instructions. The Data Bus Buffer has three basic functions.

1. Programming the MODES of the 8253.
2. Loading the count registers.
3. Reading the count values.

Read/Write Logic

The Read/Write Logic accepts inputs from the system bus and in turn generates control signals for overall device operation. It is enabled or disabled by CS so that no operation can occur to change the function unless the device has been selected by the system logic.

RD (Read)

A “low” on this input informs the 8253 that the CPU is inputting data in the form of a counters value.

WR (Write)

A “low” on this input informs the 8253 that the CPU is outputting data in the form of mode information or loading counters.

A0, A1

These inputs are normally connected to the address bus. Their function is to select one of the three counters to be operated on and to address the control word register for mode selection.

CS (Chip Select)

A “low” on this input enables the 8253. No reading or writing will occur unless the device is selected. The CS input has no effect upon the actual operation of the counters.

---

<table>
<thead>
<tr>
<th>CS</th>
<th>RD</th>
<th>WR</th>
<th>A1</th>
<th>A0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Control Word Register

The Control Word Register is selected when A0, A1 are 11. It then accepts information from the data bus buffer and stores it in a register. The information stored in this register controls the operational MODE of each counter, selection of binary or BCD counting and the loading of each count register.

The Control Word Register can only be written into; no read operation of its contents is available.

Counter #0, Counter #1, Counter #2

These three functional blocks are identical in operation so only a single Counter will be described. Each Counter consists of a single, 16-bit, pre-settable, DOWN counter. The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of MODES stored in the Control Word Register. The counters are fully independent and each can have separate Mode configuration and counting operation, binary or BCD. Also, there are special features in the control word that handle the loading of the count value so that software overhead can be minimized for these functions.

The reading of the contents of each counter is available to the programmer with simple READ operations for event counting applications and special commands and logic are included in the 8253 so that the contents of each counter can be read “on the fly” without having to inhibit the clock input.

8253 SYSTEM INTERFACE

The 8253 is a component of the Intel™ Microcomputer Systems and interfaces in the same manner as all other peripherals of the family. It is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A0, A1 connect to the A0, A1 address bus signals of the CPU. The CS can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel® 8205 for larger systems.
OPERATIONAL DESCRIPTION

General
The complete functional definition of the 8253 is programmed by the systems software. A set of control words must be sent out by the CPU to initialize each counter of the 8253 with the desired MODE and quantity information. Prior to initialization, the MODE, count, and output of all counters is undefined. These control words program the MODE, Loading sequence and selection of binary or BCD counting.

Once programmed, the 8253 is ready to perform whatever timing tasks it is assigned to accomplish.

The actual counting operation of each counter is completely independent and additional logic is provided on-chip so that the usual problems associated with efficient monitoring and management of external, asynchronous events or rates to the microcomputer system have been eliminated.

Programming the 8253
All of the MODES for each counter are programmed by the systems software by simple I/O operations.

Each counter of the 8253 is individually programmed by writing a control word into the Control Word Register (A0, A1 = 11)

Control Word Format

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1</td>
<td>SC0</td>
<td>RL1</td>
<td>RL0</td>
<td>M2</td>
<td>M1</td>
<td>M0</td>
<td>BCD</td>
</tr>
</tbody>
</table>

Definition of Control

SC — Select Counter:

<table>
<thead>
<tr>
<th>SC1</th>
<th>SC0</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Select Counter 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Select Counter 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Select Counter 2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Illegal</td>
</tr>
</tbody>
</table>

RL — Read/Load:

<table>
<thead>
<tr>
<th>RL1</th>
<th>RL0</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Counter Latching operation (see READ/WRITE Procedure Section)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Read/Load most significant byte only.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Read/Load least significant byte only.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Read/Load least significant byte first, then most significant byte.</td>
</tr>
</tbody>
</table>

M — MODE:

<table>
<thead>
<tr>
<th>M2</th>
<th>M1</th>
<th>M0</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Mode 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Mode 1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>Mode 2</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>Mode 3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Mode 4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Mode 5</td>
</tr>
</tbody>
</table>

BCD:

<table>
<thead>
<tr>
<th>0</th>
<th>Binary Counter 16-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Binary Coded Decimal (BCD) Counter (4 Decades)</td>
</tr>
</tbody>
</table>

Counter Loading
The count register is not loaded until the count value is written (one or two bytes, depending on the mode selected by the RL bits), followed by a rising edge and a falling edge of the clock. Any read of the counter prior to that falling clock edge may yield invalid data.

MODE Definition

MODE 0: Interrupt on Terminal Count. The output will be initially low after the mode set operation. After the count is loaded into the selected count register, the output will remain low and the counter will count. When terminal count is reached the output will go high and remain high until the selected count register is reloaded with the mode or a new count is loaded. The counter continues to decrement after terminal count has been reached.

Rewriting a counter register during counting results in the following:

1. Write 1st byte stops the current counting.
2. Write 2nd byte starts the new count.

MODE 1: Programmable One-Shot. The output will go low on the count following the rising edge of the gate input.

The output will go high on the terminal count. If a new count value is loaded while the output is low it will not affect the duration of the one-shot pulse until the succeeding trigger. The current count can be read at any time without affecting the one-shot pulse.

The one-shot is retriggereable, hence the output will remain low for the full count after any rising edge of the gate input.
MODE 2: Rate Generator. Divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. If the count register is reloaded between output pulses the present period will not be affected, but the subsequent period will reflect the new value.

The gate input, when low, will force the output high. When the gate input goes high, the counter will start from the initial count. Thus, the gate input can be used to synchronize the counter.

When this mode is set, the output will remain high until after the count register is loaded. The output then can also be synchronized by software.

MODE 3: Square Wave Rate Generator. Similar to MODE 2 except that the output will remain high until one half the count has been completed (for even numbers) and go low for the other half of the count. This is accomplished by decrementing the counter by two on the falling edge of each clock pulse. When the counter reaches terminal count, the state of the output is changed and the counter is reloaded with the full count and the whole process is repeated.

If the count is odd and the output is high, the first clock pulse (after the count is loaded) decrements the count by 1. Subsequent clock pulses decrement the clock by 2. After timeout, the output goes low and the full count is reloaded. The first clock pulse (following the reload) decrements the counter by 3. Subsequent clock pulses decrement the count by 2 until timeout. Then the whole process is repeated. In this way, if the count is odd, the output will be high for \((N + 1)/2\) counts and low for \((N - 1)/2\) counts.

In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following WR of a new count value.

MODE 4: Software Triggered Strobe. After the mode is set, the output will be high. When the count is loaded, the counter will begin counting. On terminal count, the output will go low for one input clock period, then will go high again.

If the count register is reloaded during counting, the new count will be loaded on the next CLK pulse. The count will be inhibited while the GATE input is low.

MODE 5: Hardware Triggered Strobe. The counter will start counting after the rising edge of the trigger input and will go low for one clock period when the terminal count is reached. The counter is retriggerable. The output will not go low until the full count after the rising edge of any trigger.

<table>
<thead>
<tr>
<th>Modes</th>
<th>Signal Status</th>
<th>Low Or Going Low</th>
<th>Rising</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disables</td>
<td>Enables counting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1) Initiates counting</td>
<td></td>
<td>2) Resets output after next clock</td>
<td>Enables counting</td>
</tr>
<tr>
<td>2</td>
<td>1) Disables counting</td>
<td></td>
<td>2) Sets output immediately high</td>
<td>Enables counting</td>
</tr>
<tr>
<td>3</td>
<td>1) Disables counting</td>
<td></td>
<td>2) Sets output immediately high</td>
<td>Enables counting</td>
</tr>
<tr>
<td>4</td>
<td>Disables</td>
<td>Enables counting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Initiates</td>
<td>Enables counting</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Gate Pin Operations Summary
Figure 7. 8253 Timing Diagrams
8253 READ/WRITE PROCEDURE

Write Operations

The systems software must program each counter of the 8253 with the mode and quantity desired. The programmer must write out to the 8253 a MODE control word and the programmed number of count register bytes (1 or 2) prior to actually using the selected counter.

The actual order of the programming is quite flexible. Writing out of the MODE control word can be in any sequence of counter selection, e.g., counter #0 does not have to be first or counter #2 last. Each counter's MODE control word register has a separate address so that its loading is completely sequence independent. (SC0, SC1)

The loading of the Count Register with the actual count value, however, must be done in exactly the sequence programmed in the MODE control word (RL0, RL1). This loading of the counter's count register is still sequence independent like the MODE control word loading, but when a selected count register is to be loaded it must be loaded with the number of bytes programmed in the MODE control word (RL0, RL1). The one or two bytes to be loaded in the count register do not have to follow the associated MODE control word. They can be programmed at any time following the MODE control word loading as long as the correct number of bytes is loaded in order.

All counters are down counters. Thus, the value loaded into the count register will actually be decremented. Loading all zeroes into a count register will result in the maximum count (2^6 for Binary or 10^4 for BCD). In MODE 0 the new count will not restart until the load has been completed. It will accept one of two bytes depending on how the MODE control words (RL0, RL1) are programmed. Then proceed with the restart operation.

<table>
<thead>
<tr>
<th>No. 1</th>
<th>MODE Control Word Counter 0</th>
<th>A1 A0</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. 2</td>
<td>MODE Control Word Counter 1</td>
<td>1 1</td>
</tr>
<tr>
<td>No. 3</td>
<td>MODE Control Word Counter 2</td>
<td>1 1</td>
</tr>
<tr>
<td>No. 4</td>
<td>LSB Count Register Byte Counter 1</td>
<td>0 1</td>
</tr>
<tr>
<td>No. 5</td>
<td>MSB Count Register Byte Counter 1</td>
<td>0 1</td>
</tr>
<tr>
<td>No. 6</td>
<td>LSB Count Register Byte Counter 2</td>
<td>1 0</td>
</tr>
<tr>
<td>No. 7</td>
<td>MSB Count Register Byte Counter 2</td>
<td>1 0</td>
</tr>
<tr>
<td>No. 8</td>
<td>LSB Count Register Byte Counter 0</td>
<td>0 0</td>
</tr>
<tr>
<td>No. 9</td>
<td>MSB Count Register Byte Counter 0</td>
<td>0 0</td>
</tr>
</tbody>
</table>

Note: The exclusive addresses of each counter's count register make the task of programming the 8253 a very simple matter, and maximum effective use of the device will result if this feature is fully utilized.

Figure 8. Programming Format

Figure 9. Alternate Programming Formats
**Read Operations**

In most counter applications it becomes necessary to read the value of the count in progress and make a computational decision based on this quantity. Event counters are probably the most common application that uses this function. The 8253 contains logic that will allow the programmer to easily read the contents of any of the three counters without disturbing the actual count in progress.

There are two methods that the programmer can use to read the value of the counters. The first method involves the use of simple I/O read operations of the selected counter. By controlling the A0, A1 inputs to the 8253, the programmer can select the counter to be read (remember that no read operation of the mode register is allowed; A0, A1 = 11). The only requirement with this method is that in order to assure a stable count reading the actual operation of the selected counter must be inhibited either by controlling the Gate input or by external logic that inhibits the clock input. The contents of the counter selected will be available as follows:

1. First I/O Read contains the least significant byte (LSB).
2. Second I/O Read contains the most significant byte (MSB).

Due to the internal logic of the 8253, it is absolutely necessary to complete the entire reading procedure. If two bytes are programmed to be read then two bytes must be read before any loading WR command can be sent to the same counter.

**Read Operation Chart**

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>RD</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Read Counter No. 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Read Counter No. 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Read Counter No. 2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Illegal</td>
</tr>
</tbody>
</table>

**Reading While Counting**

In order for the programmer to read the contents of any counter without effecting or disturbing the counting operation, the 8253 has special internal logic that can be accessed using simple WR commands to the MODE register. Basically, when the programmer wishes to read the contents of a selected counter "on the fly" he loads the MODE register with a special code which latches the present count value into a storage register so that its contents contain an accurate, stable quantity. The programmer then issues a normal read command to the selected counter and the contents of the latched register are available.

**MODE Register for Latching Count**

A0, A1 = 11

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1</td>
<td>SC0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

SC1,SC0 — specify counter to be latched
D5,D4 — 00 designates counter latching operation
X — don’t care.

The same limitation applies to this mode of reading the counter as the previous method. That is, it is mandatory to complete the entire read operation as programmed. This command has no effect on the counter's mode.

---

**Figure 10. MCS-85™ Clock Interface**

*If an 8085 clock output is to drive an 8253-5 clock input, it must be reduced to 2 MHz or less.*
**ABSOLUTE MAXIMUM RATINGS**

*NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

- **Ambient Temperature Under Bias**: 0°C to 70°C
- **Storage Temperature**: -65°C to +150°C
- **Voltage On Any Pin With Respect to Ground**: -0.5 V to +7 V
- **Power Dissipation**: 1 Watt

**D.C. CHARACTERISTICS** (TA = 0°C to 70°C, VCC = 5V ±10%)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>2.2</td>
<td>VCC +0.5 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td>V</td>
<td>Note 1</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td>V</td>
<td>Note 2</td>
<td></td>
</tr>
<tr>
<td>IIL</td>
<td>Input Load Current</td>
<td>±10</td>
<td>µA</td>
<td>V_IN = V_CC to 0V</td>
<td></td>
</tr>
<tr>
<td>IOFL</td>
<td>Output Float Leakage</td>
<td>±10</td>
<td>µA</td>
<td>V_OUT = V_CC to 0.45V</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>V_CC Supply Current</td>
<td>140</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CAPACITANCE** (TA = 25°C, VCC = GND = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_IN</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td>fc = 1 MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_I/O</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
<td>Unmeasured pins returned to VSS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**A.C. CHARACTERISTICS** (TA = 0°C to 70°C, VCC = 5.0V ±10%, GND = 0V)

*Bus Parameters (Note 3)*

**READ CYCLE**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8253 Min.</th>
<th>8253 Max.</th>
<th>8253-5 Min.</th>
<th>8253-5 Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_AR</td>
<td>Address Stable Before READ</td>
<td>50</td>
<td>30</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_RA</td>
<td>Address Hold Time for READ</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_RR</td>
<td>READ Pulse Width</td>
<td>400</td>
<td>300</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_RD</td>
<td>Data Delay From READ[4]</td>
<td>300</td>
<td>200</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_DF</td>
<td>READ to Data Floating</td>
<td>25</td>
<td>125</td>
<td>25</td>
<td>100</td>
<td>µs</td>
</tr>
<tr>
<td>t_RV</td>
<td>Recovery Time Between READ and Any Other Control Signal</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>µs</td>
</tr>
</tbody>
</table>
### A.C. CHARACTERISTICS (Continued)

#### WRITE CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8253</th>
<th>8253-5</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
</tr>
<tr>
<td>t_{AW}</td>
<td>Address Stable Before WRITE</td>
<td>50</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>t_{WA}</td>
<td>Address Hold Time for WRITE</td>
<td>30</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>t_{WW}</td>
<td>WRITE Pulse Width</td>
<td>400</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>t_{DW}</td>
<td>Data Set Up Time for WRITE</td>
<td>300</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>t_{WD}</td>
<td>Data Hold Time for WRITE</td>
<td>40</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>t_{RV}</td>
<td>Recovery Time Between WRITE and Any Other Control Signal</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

#### CLOCK AND GATE TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8253</th>
<th>8253-5</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
</tr>
<tr>
<td>t_{CLK}</td>
<td>Clock Period</td>
<td>380</td>
<td>dc</td>
<td>380</td>
</tr>
<tr>
<td>t_{PWH}</td>
<td>High Pulse Width</td>
<td>230</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>t_{PWL}</td>
<td>Low Pulse Width</td>
<td>150</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>t_{GW}</td>
<td>Gate Width High</td>
<td>150</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>t_{GL}</td>
<td>Gate Width Low</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>t_{GS}</td>
<td>Gate Set Up Time to CLK↑</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>t_{GH}</td>
<td>Gate Hold Time After CLK↑</td>
<td>50</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>t_{OD}</td>
<td>Output Delay From CLK↓[4]</td>
<td>400</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>t_{ODG}</td>
<td>Output Delay From Gate↓[4]</td>
<td>300</td>
<td>300</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**

1. I_{OL} = 2.2 mA.
2. I_{OH} = -400 μA.
3. AC timings measured at V_{OH} 2.2, V_{OL} = 0.8.
4. C_L = 150pF.
   * For Extended Temperature EXPRESS, use M8253 electrical parameters.

#### A.C. TESTING INPUT, OUTPUT WAVEFORM

![A.C. TESTING INPUT, OUTPUT WAVEFORM Diagram]

A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.4V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.2V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

#### A.C. TESTING LOAD CIRCUIT

![A.C. TESTING LOAD CIRCUIT Diagram]

C_L INCLUDES JIG CAPACITANCE
WAVEFORMS

WRITE TIMING

READ TIMING

CLOCK AND GATE TIMING
8254 PROGRAMMABLE INTERVAL TIMER

- Compatible with all Intel and most other microprocessors
- Handles Inputs from DC to 10 MHz
  - 5 MHz 8254-5
  - 8 MHz 8254
  - 10 MHz 8254-2
- Status Read-Back Command
- Six Programmable Counter Modes

The Intel® 8254 is a counter/timer device designed to solve the common timing control problems in microcomputer system design. It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz. All modes are software programmable. The 8254 is a superset of the 8253.

The 8254 uses HMOS technology and comes in a 24-pin plastic or CERDIP package.

Figure 1. 8254 Block Diagram

Figure 2. Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied. Information Contained herein Supersedes Previously Published Specifications On The Devices From Intel.
### Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7-D0</td>
<td>1-8</td>
<td>I/O</td>
<td>Data: Bi-directional three state data bus lines, connected to system data bus.</td>
</tr>
<tr>
<td>CLK 0</td>
<td>9</td>
<td>I</td>
<td>Clock 0: Clock input of Counter 0.</td>
</tr>
<tr>
<td>OUT 0</td>
<td>10</td>
<td>O</td>
<td>Output 0: Output of Counter 0.</td>
</tr>
<tr>
<td>GATE 0</td>
<td>11</td>
<td>I</td>
<td>Gate 0: Gate input of Counter 0.</td>
</tr>
<tr>
<td>GND</td>
<td>12</td>
<td></td>
<td>Ground: Power supply connection.</td>
</tr>
</tbody>
</table>

### Functional Description

**General**

The 8254 is a programmable interval timer/counter designed for use with Intel microcomputer systems. It is a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 8254 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in software, the programmer configures the 8254 to match his requirements and programs one of the counters for the desired delay. After the desired delay, the 8254 will interrupt the CPU. Software overhead is minimal and variable length delays can easily be accommodated.

Some of the other counter/timer functions common to microcomputers which can be implemented with the 8254 are:

- Real time clock
- Event counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller

**Block Diagram**

**DATA BUS BUFFER**

This 3-state, bi-directional, 8-bit buffer is used to interface the 8254 to the system bus (see Figure 3).
READ/WRITE LOGIC

The Read/Write Logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 8254. A1 and A0 select one of the three counters or the Control Word Register to be read from/written into. A "low" on the RD input tells the 8254 that the CPU is reading one of the counters. A "low" on the WR input tells the 8254 that the CPU is writing either a Control Word or an initial count. Both RD and WR are qualified by CS; RD and WR are ignored unless the 8254 has been selected by holding CS low.

CONTROL word REGISTER

The Control Word Register (see Figure 4) is selected by the Read/Write Logic when A1, A0 = 11. If the CPU then does a write operation to the 8254, the data is stored in the Control Word Register and is interpreted as a Control Word used to define the operation of the Counters. The Control Word Register can only be written to; status information is available with the Read-Back Command.

Figure 4. Block Diagram Showing Control Word Register and Counter Functions

COUNTER 0, COUNTER 1, COUNTER 2

These three functional blocks are identical in operation, so only a single Counter will be described. The internal block diagram of a single counter is shown in Figure 5.

The Counters are fully independent. Each Counter may operate in a different Mode.

The Control Word Register is shown in the figure; it is not part of the Counter itself, but its contents determine how the Counter operates.

Figure 5. Internal Block Diagram of a Counter

The status register, shown in the Figure, when latched, contains the current contents of the Control Word Register and status of the output and null count flag. (See detailed explanation of the Read-Back command.) The actual counter is labelled CE (for "Counting Element"). It is a 16-bit presettable synchronous down counter.

OLM and OLl are two 8-bit latches. OL stands for “Output Latch”; the subscripts M and L stand for “Most significant byte” and “Least significant byte” respectively. Both are normally referred to as one unit and called just OL. These latches normally follow” the CE, but if a suitable Counter Latch Command is sent to the 8254, the latches “latch” the present count until read by the CPU and then return to “following” the CE. One latch at a time is enabled by the counter's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that the CE itself cannot be read; whenever you read the count, it is the OL that is being read.

Similarly, there are two 8-bit registers called CRM and CRl (for “Count Register”). Both are normally referred to as one unit and called just CR. When a new count is written to the Counter, the count is stored in the CR and later transferred to the CE. The Control Logic allows one register at a time to be loaded from the internal bus. Both bytes are transferred to the CE simultaneously. CRM and CRl are cleared when the Counter is programmed. In this way, if the Counter has been programmed for one byte counts (either most significant byte only or least significant byte only) the other byte will be zero. Note that the CE cannot be written into; whenever a count is written, it is written into the CR.

The Control Logic is also shown in the diagram. CLK n, GATE n, and OUT n are all connected to the outside world through the Control Logic.
8254 SYSTEM INTERFACE

The 8254 is a component of the Intel Microcomputer Systems and interfaces in the same manner as all other peripherals of the family. It is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming. Basically, the select inputs A₀, A₁ connect to the A₀, A₁ address bus signals of the CPU. The CS can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.

OPERATIONAL DESCRIPTION

General

After power-up, the state of the 8254 is undefined. The Mode, count value, and output of all Counters are undefined.

How each Counter operates is determined when it is programmed. Each Counter must be programmed before it can be used. Unused counters need not be programmed.

Programming the 8254

Counters are programmed by writing a Control Word and then an initial count.

All Control Words are written into the Control Word Register, which is selected when A₁,A₀ = 11. The Control Word itself specifies which Counter is being programmed.

By contrast, initial counts are written into the Counters, not the Control Word Register. The A₁,A₀ inputs are used to select the Counter to be written into. The format of the initial count is determined by the Control Word used.

Control Word Format

A₁,A₀ = 11 CS = 0 RD = 1 WR = 0

<table>
<thead>
<tr>
<th>D₇</th>
<th>D₆</th>
<th>D₅</th>
<th>D₄</th>
<th>D₃</th>
<th>D₂</th>
<th>D₁</th>
<th>D₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC₁</td>
<td>SC₀</td>
<td>RW₁</td>
<td>RW₀</td>
<td>M₂</td>
<td>M₁</td>
<td>M₀</td>
<td>BCD</td>
</tr>
</tbody>
</table>

SC — Select Counter:

<table>
<thead>
<tr>
<th>SC₁</th>
<th>SC₀</th>
<th>Select Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Select Counter 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Select Counter 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Select Counter 2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Read-Back Command (See Read Operations)</td>
</tr>
</tbody>
</table>

RW — Read/Write:

<table>
<thead>
<tr>
<th>RW₁</th>
<th>RW₀</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Counter Latch Command (see Read Operations)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Read/Write least significant byte only.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Read/Write most significant byte only</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Read/Write least significant byte first, then most significant byte.</td>
</tr>
</tbody>
</table>

M — Mode:

<table>
<thead>
<tr>
<th>M₂</th>
<th>M₁</th>
<th>M₀</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Mode 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Mode 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Mode 2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Mode 3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Mode 4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Mode 5</td>
</tr>
</tbody>
</table>

BCD:

<table>
<thead>
<tr>
<th>BCD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Binary Counter 16-bits</td>
</tr>
<tr>
<td>1</td>
<td>Binary Coded Decimal (BCD) Counter (4 Decades)</td>
</tr>
</tbody>
</table>

NOTE: DON'T CARE BITS (X) SHOULD BE 0 TO INSURE COMPATIBILITY WITH FUTURE INTEL PRODUCTS.
Write Operations

The programming procedure for the 8254 is very flexible. Only two conventions need to be remembered:

1) For each Counter, the Control Word must be written before the initial count is written.
2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

Since the Control Word Register and the three Counters have separate addresses (selected by the A1,A0 inputs), and each Control Word specifies the Counter it applies to (SC0,SC1 bits), no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.

---

Read Operations

It is often desirable to read the value of a Counter without disturbing the count in progress. This is easily done in the 8254.

There are three possible methods for reading the counters: a simple read operation, the Counter Latch Command, and the Read-Back Command. Each is explained below. The first method is to perform a simple read operation. To read the Counter, which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result.

---

Figure 8. A Few Possible Programming Sequences

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>A1</th>
<th>A0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Word — Counter 0</td>
<td>1</td>
<td>1</td>
<td>Control Word — Counter 2</td>
</tr>
<tr>
<td>LSB of count — Counter 0</td>
<td>0</td>
<td>0</td>
<td>Control Word — Counter 1</td>
</tr>
<tr>
<td>MSB of count — Counter 0</td>
<td>0</td>
<td>0</td>
<td>Control Word — Counter 0</td>
</tr>
<tr>
<td>Control Word — Counter 1</td>
<td>1</td>
<td>1</td>
<td>LSB of count — Counter 2</td>
</tr>
<tr>
<td>LSB of count — Counter 1</td>
<td>0</td>
<td>1</td>
<td>MSB of count — Counter 2</td>
</tr>
<tr>
<td>MSB of count — Counter 1</td>
<td>0</td>
<td>1</td>
<td>LSB of count — Counter 1</td>
</tr>
<tr>
<td>Control Word — Counter 2</td>
<td>1</td>
<td>1</td>
<td>MSB of count — Counter 1</td>
</tr>
<tr>
<td>LSB of count — Counter 2</td>
<td>1</td>
<td>0</td>
<td>LSB of count — Counter 0</td>
</tr>
<tr>
<td>MSB of count — Counter 2</td>
<td>1</td>
<td>0</td>
<td>MSB of count — Counter 0</td>
</tr>
</tbody>
</table>

NOTE: IN ALL FOUR EXAMPLES, ALL COUNTERS ARE PROGRAMMED TO READWRITE TWO-BYTE COUNTS. THESE ARE ONLY FOUR OF MANY POSSIBLE PROGRAMMING SEQUENCES.
**COUNTER LATCH COMMAND**

The second method uses the "Counter Latch Command". Like a Control Word, this command is written to the Control Word Register, which is selected when \( A_1, A_0 = 11 \). Also like a Control Word, the SC0, SC1 bits select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.

\[
A_1, A_0 = 11; \; CS = 0; \; RD = 1; \; WR = 0
\]

\[
\begin{array}{ccccccc}
D_7 & D_6 & D_5 & D_4 & D_3 & D_2 & D_1 & D_0 \\
SC1 & SC0 & 0 & 0 & X & X & X & X \\
\end{array}
\]

SC1,SC0 — specify counter to be latched

<table>
<thead>
<tr>
<th>SC1</th>
<th>SC0</th>
<th>Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Read-Back Command</td>
</tr>
</tbody>
</table>

D5,D4 — 00 designates Counter Latch Command
X — don’t care

NOTE: DON'T CARE BITS (X) SHOULD BE 0 TO INSURE COMPATIBILITY WITH FUTURE INTEL PRODUCTS.

![Figure 9. Counter Latching Command Format](image)

The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed). The count is then unlatched automatically and the OL returns to "following" the counting element (CE). This allows reading the contents of the Counters “on the fly" without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one Counter. Each latched Counter's OL holds its count until it is read. Counter Latch Commands do not affect the programmed Mode of the Counter in any way.

If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.

With either method, the count must be read according to the programmed format; specifically, if the Counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other; read or write or programming operations of other Counters may be inserted between them.

Another feature of the 8254 is that reads and writes of the same Counter may be interleaved; for example, if the Counter is programmed for two byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.

**READ-BACK COMMAND**

The third method uses the Read-Back Command. This command allows the user to check the count value, programmed Mode, and current states of the OUT pin and Null Count flag of the selected counter(s).

The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D3,D2,D1 = 1.

![Figure 10. Read-Back Command Format](image)

The read-back command may be used to latch multiple counter output latches (OL) by setting the COUNT bit D5 = 0 and selecting the desired counter(s). This single command is functionally equivalent to several counter latch commands, one for each counter latched. Each counter's latched count is held until it is read (or the counter is reprogrammed). That counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple count read-back commands are issued to the same counter without reading the count, all but the first are ignored; i.e., the count which will be read is the count at the time the first read-back command was issued.

The read-back command may also be used to latch status information of selected counter(s) by setting STATUS bit D4 = 0. Status must be latched to be read; status of a counter is accessed by a read from that counter.
The counter status format is shown in Figure 11. Bits D5 through D0 contain the counter's programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D7 contains the current state of the OUT pin. This allows the user to monitor the counter's output via software, possibly eliminating some hardware from a system.

### Figure 11. Status Byte

NULL COUNT bit D6 indicates when the last count written to the counter register (CR) has been loaded into the counting element (CE). The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element (CE), it can't be read from the counter. If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

Both count and status of the selected counter(s) may be latched simultaneously by setting both COUNT and STATUS bits D5,D4=0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also. Specifically, if multiple count and/or status read-back commands are issued to the same counter(s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.

### Figure 12. Null Count Operation

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

Both count and status of the selected counter(s) may be latched simultaneously by setting both COUNT and STATUS bits D5,D4=0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also. Specifically, if multiple count and/or status read-back commands are issued to the same counter(s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.
If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched count.

### Table 1: Read/Write Operations Summary

<table>
<thead>
<tr>
<th>CS</th>
<th>RD</th>
<th>WR</th>
<th>A₁</th>
<th>A₀</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Write into Counter 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Write into Counter 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Write into Counter 2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Write Control Word</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Read from Counter 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Read from Counter 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Read from Counter 2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>No-Operation (3-State)</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>No-Operation (3-State)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>No-Operation (3-State)</td>
</tr>
</tbody>
</table>

**Figure 14. Read/Write Operations Summary**

### Mode Definitions

The following are defined for use in describing the operation of the 8254.

- **CLK pulse**: a rising edge, then a falling edge, in that order, of a Counter's CLK input.
- **trigger**: a rising edge of a Counter's GATE input.
- **Counter loading**: the transfer of a count from the CR to the CE (refer to the "Functional Description")

#### MODE 0: INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero. OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N + 1 CLK pulses after the initial count is written.

If a new count is written to the Counter, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte disables counting. OUT is set low immediately (no clock pulse required)
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go high until N + 1 CLK pulses after the new count of N is written.

If an initial count is written while GATE = 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later; no CLK pulse is needed to load the Counter as this has already been done.

**Figure 15. Mode 0**
MODE 1: HARDWARE RETRIGGERABLE ONE-SHOT

OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero. OUT will then go high and remain high until the CLK pulse after the next trigger.

After writing the Control Word and initial count, the Counter is armed. A trigger results in loading the Counter and setting OUT low on the next CLK pulse, thus starting the one-shot pulse. An initial count of N will result in a one-shot pulse N CLK cycles in duration. The one-shot is retrigerable, hence OUT will remain low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.

If a new count is written to the Counter during a one-shot pulse, the current one-shot is not affected unless the Counter is retrigged. In that case, the Counter is loaded with the new count and the one-shot pulse continues until the new count expires.

Figure 16. Mode 1

MODE 2: RATE GENERATOR

This Mode functions like a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt. OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated. Mode 2 is periodic; the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately. A trigger reloads the Counter with the initial count on the next CLK pulse; OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. OUT goes low N CLK Pulses after the initial count is written. This allows the Counter to be synchronized by software also.

Figure 17. Mode 2

NOTE: A GATE transition should not occur one clock prior to terminal count.
Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current period, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current counting cycle. In mode 2, a COUNT of 1 is illegal.

MODE 3: SQUARE WAVE MODE

Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high. When half the initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely. An initial count of N results in a square wave with a period of N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required. A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This allows the Counter to be synchronized by software also.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

Mode 3 is implemented as follows:

Even counts: OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. When the count expires, OUT changes value and the Counter is loaded with the initial count. The above process is repeated indefinitely.

Odd counts: OUT is initially high. The initial count minus one (an even number) is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. One CLK pulse after the count expires, OUT goes low and the Counter is loaded with the initial count minus one. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes high again and the Counter is loaded with the initial count minus one. The above process is repeated indefinitely. So for odd counts, OUT will be high for (N + 1)/2 counts and low for (N - 1)/2 counts.

MODE 4: SOFTWARE TRIGGERED STROBE

OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is "triggered" by writing the initial count.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after the initial count is written.

If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte has no effect on counting.
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the sequence to be "retriggered" by software. OUT strobes low N + 1 CLK pulses after the new count of N is written.
MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE)

OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again.

After writing the Control Word and initial count, the counter will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after a trigger.

A trigger results in the Counter being loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable. OUT will not strobe low for N + 1 CLK pulses after any trigger. GATE has no effect on OUT.

If a new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.
### Operation Common to All Modes

**PROGRAMMING**

When a Control Word is written to a Counter, all Control Logic is immediately reset and OUT goes to a known initial state; no CLK pulses are required for this.

#### GATE

The GATE input is always sampled on the rising edge of CLK. In Modes 0, 2, 3, and 4 the GATE input is level sensitive, and the logic level is sampled on the rising edge of CLK. In Modes 1, 2, 3, and 5 the GATE input is rising-edge sensitive. In these Modes, a GATE (trigger) sets an edge-sensitive flip-flop in the Counter. This flip-flop is then sampled on the next rising edge of CLK; the flip-flop is reset immediately after it is sampled. In this way, a trigger will be detected no matter when it occurs—a high logic level does not have to be maintained until the next rising edge of CLK. Note that in Modes 2 and 3, the GATE input is both edge- and level-sensitive. In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following WR of a new count value.

#### COUNTER

New counts are loaded and Counters are decremented on the falling edge of CLK.

The largest possible initial count is 0; this is equivalent to $2^{16}$ for binary counting and $10^4$ for BCD counting.

The Counter does not stop when it reaches zero. In Modes 0, 1, 4, and 5 the Counter “wraps around” to the highest count, either FFFF hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic; the Counter reloads itself with the initial count and continues counting from there.

---

**Table:** Minimum and Maximum Initial Counts

<table>
<thead>
<tr>
<th>Mode</th>
<th>Min Count</th>
<th>Max Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**NOTE:** 0 IS EQUIVALENT TO $2^{16}$ FOR BINARY COUNTING AND $10^4$ FOR BCD COUNTING.
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ............. 0°C to 70°C
Storage Temperature .................. −65°C to +150°C
Voltage on Any Pin with Respect to Ground ............. −0.5V to +7V
Power Dissipation ...................... 1 Watt

*NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to Absolute Maximum Rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = 5V ± 10%)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>−0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>VCC + 0.5V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td>V</td>
<td></td>
<td>IOL = 2.0 mA</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td>V</td>
<td></td>
<td>IOH = −400μA</td>
</tr>
<tr>
<td>IL</td>
<td>Input Load Current</td>
<td>±10</td>
<td>μA</td>
<td></td>
<td>VIN = VCC to 0V</td>
</tr>
<tr>
<td>IFL</td>
<td>Output Float Leakage</td>
<td>±10</td>
<td>μA</td>
<td></td>
<td>VOUT = VCC to 0.45V</td>
</tr>
<tr>
<td>ICC</td>
<td>VCC Supply Current</td>
<td>170</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CAPACITANCE (TA = 25°C, VCC = GND = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIN</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td></td>
<td>fc = 1 MHz</td>
</tr>
<tr>
<td>CIO</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
<td></td>
<td>Unmeasured pins returned to VSS</td>
</tr>
</tbody>
</table>

A.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = 5V ± 10%, GND = 0V)
Bus Parameters (Note 1)

READ CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8254-5</th>
<th>8254</th>
<th>8254-2</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>tAR</td>
<td>Address Stable Before RD</td>
<td>45</td>
<td>45</td>
<td>30</td>
<td>ns</td>
</tr>
<tr>
<td>tSR</td>
<td>CS Stable Before RD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>tRA</td>
<td>Address Hold Time After RD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>tRR</td>
<td>RD Pulse Width</td>
<td>150</td>
<td>150</td>
<td>95</td>
<td>ns</td>
</tr>
<tr>
<td>tRD</td>
<td>Data Delay from RD</td>
<td>120</td>
<td>120</td>
<td>85</td>
<td>ns</td>
</tr>
<tr>
<td>tAD</td>
<td>Data Delay from Address</td>
<td>220</td>
<td>220</td>
<td>185</td>
<td>ns</td>
</tr>
<tr>
<td>tDF</td>
<td>RD to Data Floating</td>
<td>5</td>
<td>90</td>
<td>5</td>
<td>65</td>
</tr>
<tr>
<td>tCR</td>
<td>Command Recovery Time</td>
<td>200</td>
<td>200</td>
<td>165</td>
<td>ns</td>
</tr>
</tbody>
</table>

Note 1: AC timings measured at VCC = 2.0V, VCC = 0.8V
### A.C. CHARACTERISTICS (Continued)

#### WRITE CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8254-5</th>
<th>8254</th>
<th>8254-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>t(_{AW})</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t(_{GW})</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t(_{WA})</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t(_{RW})</td>
<td>150</td>
<td>150</td>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td>t(_{D_W})</td>
<td>120</td>
<td>120</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>t(_{WD})</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t(_{RV})</td>
<td>200</td>
<td>200</td>
<td>165</td>
<td>165</td>
</tr>
</tbody>
</table>

#### CLOCK AND GATE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8254-5</th>
<th>8254</th>
<th>8254-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>t(_{CLK})</td>
<td>200</td>
<td>DC</td>
<td>125</td>
<td>DC</td>
</tr>
<tr>
<td>t(_{PW_H})</td>
<td>60(^{[3]})</td>
<td>60(^{[3]})</td>
<td>30(^{[3]})</td>
<td>30(^{[3]})</td>
</tr>
<tr>
<td>t(_{PW_L})</td>
<td>60(^{[3]})</td>
<td>60(^{[3]})</td>
<td>50(^{[3]})</td>
<td>50(^{[3]})</td>
</tr>
<tr>
<td>t(_{g})</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>t(_{f})</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>t(_{GW_H})</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>t(_{GW_L})</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>t(_{GS})</td>
<td>50</td>
<td>50</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>t(_{GH})</td>
<td>50(^{[2]})</td>
<td>50(^{[2]})</td>
<td>50(^{[2]})</td>
<td>50(^{[2]})</td>
</tr>
<tr>
<td>t(_{OD})</td>
<td>150</td>
<td>150</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>t(_{ODG})</td>
<td>120</td>
<td>120</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>t(_{WC})</td>
<td>0</td>
<td>55</td>
<td>0</td>
<td>55</td>
</tr>
<tr>
<td>t(_{WG})</td>
<td>-5</td>
<td>50</td>
<td>-5</td>
<td>50</td>
</tr>
<tr>
<td>t(_{WD})</td>
<td>260</td>
<td>260</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>t(_{CI})</td>
<td>-40</td>
<td>45</td>
<td>-40</td>
<td>45</td>
</tr>
</tbody>
</table>

Note 2: In Modes 1 and 5 triggers are sampled on each rising clock edge. A second trigger within 120 ns (70 ns for the 8254-2) of the rising clock edge may not be detected.

Note 3: Low-going glitches that violate \( t_{PW\_H} \) \( t_{PW\_L} \) may cause errors requiring counter repogramming.
A.C. TESTING INPUT, OUTPUT WAVEFORM

INPUT/OUTPUT

2.0  2.0
\[\text{TEST POINTS}\]

0.8  0.8

A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC 1" AND 0.45V FOR A LOGIC 0". TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC 1" AND 0.8V FOR A LOGIC 0".

A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

\[C_L = 150 \text{ pF}\]

\[C_L \text{ INCLUDES JIG CAPACITANCE}\]
The Intel 82C54 is a high-performance, CHMOS version of the industry standard 8254 counter/timer which is designed to solve the timing control problems common in microcomputer system design. It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz. All modes are software programmable. The 82C54 is pin compatible with the HCMOS 8254, and is a superset of the 8253.

Six programmable timer modes allow the 82C54 to be used as an event counter, elapsed time indicator, programmable one-shot, and in many other applications.

The 82C54 is fabricated on Intel's advanced CHMOS III technology which provides low power consumption with performance equal to or greater than the equivalent HCMOS product. The 82C54 is available in 24-pin DIP and 28-pin plastic leaded chip carrier (PLCC) packages.
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin Number</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DIP</td>
<td>PLCC</td>
<td></td>
</tr>
<tr>
<td>D7-D0</td>
<td>1-8</td>
<td>2-9</td>
<td>I/O</td>
</tr>
<tr>
<td>CLK 0</td>
<td>9</td>
<td>10</td>
<td>I</td>
</tr>
<tr>
<td>OUT 0</td>
<td>10</td>
<td>12</td>
<td>O</td>
</tr>
<tr>
<td>GATE 0</td>
<td>11</td>
<td>13</td>
<td>I</td>
</tr>
<tr>
<td>GND</td>
<td>12</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>OUT 1</td>
<td>13</td>
<td>16</td>
<td>O</td>
</tr>
<tr>
<td>GATE 1</td>
<td>14</td>
<td>17</td>
<td>I</td>
</tr>
<tr>
<td>CLK 1</td>
<td>15</td>
<td>18</td>
<td>I</td>
</tr>
<tr>
<td>GATE 2</td>
<td>16</td>
<td>19</td>
<td>I</td>
</tr>
<tr>
<td>OUT 2</td>
<td>17</td>
<td>20</td>
<td>O</td>
</tr>
<tr>
<td>CLK 2</td>
<td>18</td>
<td>21</td>
<td>I</td>
</tr>
<tr>
<td>A1, A0</td>
<td>20-19</td>
<td>23-22</td>
<td>I</td>
</tr>
<tr>
<td>CS</td>
<td>21</td>
<td>24</td>
<td>I</td>
</tr>
<tr>
<td>RD</td>
<td>22</td>
<td>26</td>
<td>I</td>
</tr>
<tr>
<td>WR</td>
<td>23</td>
<td>27</td>
<td>I</td>
</tr>
<tr>
<td>VCC</td>
<td>24</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>NC</td>
<td>1, 11, 15, 25</td>
<td></td>
<td>No Connect</td>
</tr>
</tbody>
</table>

FUNCTIONAL DESCRIPTION

General

The 82C54 is a programmable interval timer/counter designed for use with Intel microcomputer systems. It is a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 82C54 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in software, the programmer configures the 82C54 to match his requirements and programs one of the counters for the desired delay. After the desired delay, the 82C54 will interrupt the CPU. Software overhead is minimal and variable length delays can easily be accommodated.

Some of the other counter/timer functions common to microcomputers which can be implemented with the 82C54 are:

- Real time clock
- Even counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller
Block Diagram

DATA BUS BUFFER

This 3-state, bi-directional, 8-bit buffer is used to interface the 82C54 to the system bus (see Figure 3).

CONTROL WORD REGISTER

The Control Word Register (see Figure 4) is selected by the Read/Write Logic when A1, A0 = 11. If the CPU then does a write operation to the 82C54, the data is stored in the Control Word Register and is interpreted as a Control Word used to define the operation of the Counters.

The Control Word Register can only be written to; status information is available with the Read-Back Command.

READ/WRITE LOGIC

The Read/Write Logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 82C54. A1 and A0 select one of the three counters or the Control Word Register to be read from/written into. A "low" on the RD input tells the 82C54 that the CPU is reading one of the counters. A "low" on the WR input tells the 82C54 that the CPU is writing either a Control Word or an initial count. Both RD and WR are qualified by CS; RD and WR are ignored unless the 82C54 has been selected by holding CS low.

COUNTER 0, COUNTER 1, COUNTER 2

These three functional blocks are identical in operation, so only a single Counter will be described. The internal block diagram of a single counter is shown in Figure 5.

The Counters are fully independent. Each Counter may operate in a different Mode.

The Control Word Register is shown in the figure; it is not part of the Counter itself, but its contents determine how the Counter operates.
The status register, shown in the Figure, when latched, contains the current contents of the Control Word Register and status of the output and null count flag. (See detailed explanation of the Read-Back command.)

The actual counter is labelled CE (for "Counting Element"). It is a 16-bit presettable synchronous down counter.

OL_M and OL_L are two 8-bit latches. OL stands for "Output Latch"; the subscripts M and L stand for "Most significant byte" and "Least significant byte" respectively. Both are normally referred to as one unit and called just OL. These latches normally "follow" the CE, but if a suitable Counter Latch Command is sent to the 82C54, the latches "latch" the present count until read by the CPU and then return to "following" the CE. One latch at a time is enabled by the counter's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that the CE itself cannot be read; whenever you read the count, it is the OL that is being read.

Similarly, there are two 8-bit registers called CR_M and CR_L (for "Count Register"). Both are normally referred to as one unit and called just CR. When a new count is written to the Counter, the count is stored in the CR and later transferred to the CE. The Control Logic allows one register at a time to be loaded from the internal bus. Both bytes are transferred to the CE simultaneously. CR_M and CR_L are cleared when the Counter is programmed. In this way, if the Counter has been programmed for one byte counts (either most significant byte only or least significant byte only) the other byte will be zero. Note that the CE cannot be written into; whenever a count is written, it is written into the CR.

The Control Logic is also shown in the diagram. CLK n, GATE n, and OUT n are all connected to the outside world through the Control Logic.

82C54 SYSTEM INTERFACE

The 82C54 is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A_0, A_1 connect to the A_0, A_1 address bus signals of the CPU. The CS can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.
OPERATIONAL DESCRIPTION

General

After power-up, the state of the 82C54 is undefined. The Mode, count value, and output of all Counters are undefined.

How each Counter operates is determined when it is programmed. Each Counter must be programmed before it can be used. Unused counters need not be programmed.

Programming the 82C54

Counters are programmed by writing a Control Word and then an initial count. The control word format is shown in Figure 7.

All Control Words are written into the Control Word Register, which is selected when A1, A0 = 11. The Control Word itself specifies which Counter is being programmed.

By contrast, initial counts are written into the Counters, not the Control Word Register. The A1, A0 inputs are used to select the Counter to be written into. The format of the initial count is determined by the Control Word used.

Control Word Format

<table>
<thead>
<tr>
<th>A1, A0</th>
<th>CS = 0</th>
<th>RD = 1</th>
<th>WR = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
<td>D5</td>
<td>D4</td>
</tr>
<tr>
<td>SC1</td>
<td>SC0</td>
<td>RW1</td>
<td>RW0</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
<td>D1</td>
<td>D0</td>
</tr>
<tr>
<td>M2</td>
<td>M1</td>
<td>M0</td>
<td></td>
</tr>
<tr>
<td>BCD</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SC — Select Counter:

<table>
<thead>
<tr>
<th>SC1</th>
<th>SC0</th>
<th>M2</th>
<th>M1</th>
<th>M0</th>
<th>BCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Counter 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Counter 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Counter 2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>Read-Back Command (See Read Operations)</td>
</tr>
</tbody>
</table>

RW — Read/Write:

<table>
<thead>
<tr>
<th>RW1</th>
<th>RW0</th>
<th>M2</th>
<th>M1</th>
<th>M0</th>
<th>BCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Binary Counter 16-bits</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Read/Write least significant byte only.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Read/Write most significant byte only.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Read/Write least significant byte first, then most significant byte.</td>
</tr>
</tbody>
</table>

NOTE: Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 7. Control Word Format
Write Operations

The programming procedure for the 82C54 is very flexible. Only two conventions need to be remembered:

1) For each Counter, the Control Word must be written before the initial count is written.

2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

Since the Control Word Register and the three Counters have separate addresses (selected by the A1, A0 inputs), and each Control Word specifies the Counter it applies to (SC0, SC1 bits), no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>A1</th>
<th>A0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Word — Counter 0</td>
<td>1</td>
<td>1</td>
<td>Control Word — Counter 2</td>
</tr>
<tr>
<td>LSB of count — Counter 0</td>
<td>0</td>
<td>0</td>
<td>Control Word — Counter 1</td>
</tr>
<tr>
<td>MSB of count — Counter 0</td>
<td>0</td>
<td>0</td>
<td>Control Word — Counter 0</td>
</tr>
<tr>
<td>Control Word — Counter 1</td>
<td>1</td>
<td>1</td>
<td>LSB of count — Counter 2</td>
</tr>
<tr>
<td>LSB of count — Counter 1</td>
<td>0</td>
<td>1</td>
<td>MSB of count — Counter 2</td>
</tr>
<tr>
<td>MSB of count — Counter 1</td>
<td>0</td>
<td>1</td>
<td>LSB of count — Counter 1</td>
</tr>
<tr>
<td>Control Word — Counter 2</td>
<td>1</td>
<td>1</td>
<td>MSB of count — Counter 1</td>
</tr>
<tr>
<td>LSB of count — Counter 2</td>
<td>1</td>
<td>0</td>
<td>LSB of count — Counter 0</td>
</tr>
<tr>
<td>MSB of count — Counter 2</td>
<td>1</td>
<td>0</td>
<td>MSB of count — Counter 0</td>
</tr>
</tbody>
</table>

NOTE:
In all four examples, all counters are programmed to read/write two-byte counts. These are only four of many possible programming sequences.

Figure 8. A Few Possible Programming Sequences

Read Operations

It is often desirable to read the value of a Counter without disturbing the count in progress. This is easily done in the 82C54.

There are three possible methods for reading the counters: a simple read operation, the Counter Latch Command, and the Read-Back Command. Each is explained below. The first method is to perform a simple read operation. To read the Counter, which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result.
COUNTER LATCH COMMAND

The second method uses the "Counter Latch Command". Like a Control Word, this command is written to the Control Word Register, which is selected when A1, A0 = 11. Also like a Control Word, the SC0, SC1 bits select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.

READ-BACK COMMAND

The third method uses the Read-Back command. This command allows the user to check the count value, programmed Mode, and current state of the OUT pin and Null Count flag of the selected counter(s).

The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D3,D2,D1 = 1.

Figure 9. Counter Latching Command Format

The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed). The count is then unlatched automatically and the OL returns to "following" the counting element (CE). This allows reading the contents of the Counters "on the fly" without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one Counter. Each latched Counter's OL holds its count until it is read. Counter Latch Commands do not affect the programmed Mode of the Counter in any way.

If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.

With either method, the count must be read according to the programmed format; specifically, if the Counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other; read or write or programming operations of other Counters may be inserted between them.

Another feature of the 82C54 is that reads and writes of the same Counter may be interleaved; for example, if the Counter is programmed for two byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.

Figure 10. Read-Back Command Format

The read-back command may be used to latch multiple counter output latches (OL) by setting the COUNT bit D5 = 0 and selecting the desired counter(s). This single command is functionally equivalent to several counter latch commands, one for each counter latched. Each counter's latched count is held until it is read (or the counter is reprogrammed). That counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple count read-back commands are issued to the same counter without reading the
count, all but the first are ignored; i.e., the count which will be read is the count at the time the first read-back command was issued.

The read-back command may also be used to latch status information of selected counter(s) by setting STATUS bit D4 = 0. Status must be latched to be read; status of a counter is accessed by a read from that counter.

The counter status format is shown in Figure 11. Bits D5 through D0 contain the counter’s programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D7 contains the current state of the OUT pin. This allows the user to monitor the counter’s output via software, possibly eliminating some hardware from a system.

![Figure 11. Status Byte](image)

**Figure 11. Status Byte**

NULL COUNT bit D6 indicates when the last count written to the counter register (CR) has been loaded into the counting element (CE). The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element (CE), it can’t be read from the counter. If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

Both count and status of the selected counter(s) may be latched simultaneously by setting both COUNT and STATUS bits D5,D4 = 0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also. Specifically, if multiple count and/or status read-back commands are issued to the same counter(s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.

If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched count.

![Figure 12. Null Count Operation](image)

**Figure 12. Null Count Operation**

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

This action:
- Write to the control word register:[1]
- Write to the count register (CR);[2]
- New count is loaded into CE (CR → CE);

Causes:
- Null count = 1
- Null count = 0

[1] Only the counter specified by the control word will have its null count set to 1. Null count bits of other counters are unaffected.

[2] If the counter is programmed for two-byte counts (least significant byte then most significant byte) null count goes to 1 when the second byte is written.

![Figure 13. Read-Back Command Example](image)

**Figure 13. Read-Back Command Example**
### Mode Definitions

The following are defined for use in describing the operation of the 82C54.

- **CLK PULSE**: a rising edge, then a falling edge, in that order, of a Counter’s CLK input.
- **TRIGGER**: a rising edge of a Counter’s GATE input.
- **COUNTER LOADING**: the transfer of a count from the CR to the CE (refer to the “Functional Description”)

#### MODE 0: INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero. OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N + 1 CLK pulses after the initial count is written.

If a new count is written to the Counter, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1. Writing the first byte disables counting. OUT is set low immediately (no clock pulse required).
2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go high until N + 1 CLK pulses after the new count of N is written.

If an initial count is written while GATE = 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later; no CLK pulse is needed to load the Counter as this has already been done.

### Figure 15. Mode 0

**NOTE:**

The following conventions apply to all mode timing diagrams:

1. Counters are programmed for binary (not BCD) counting and for Reading/Writing least significant byte (LSB) only.
2. The counter is always selected (CS always low).
3. CW stands for “Control Word”; CW = 10 means a control word of 10, hex is written to the counter.
4. LSB stands for “Least Significant Byte” of count.
5. Numbers below diagrams are count values. The lower number is the least significant byte. The upper number is the most significant byte. Since the counter is programmed to Read/Write LSB only, the most significant byte cannot be read. N stands for an undefined count. Vertical lines show transitions between count values.
MODE 1: HARDWARE RETRIGGERABLE ONE-SHOT

OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero. OUT will then go high and remain high until the CLK pulse after the next trigger.

After writing the Control Word and initial count, the Counter is armed. A trigger results in loading the Counter and setting OUT low on the next CLK pulse, thus starting the one-shot pulse. An initial count of N will result in a one-shot pulse N CLK cycles in duration. The one-shot is retriggerable, hence OUT will remain low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.

If a new count is written to the Counter during a one-shot pulse, the current one-shot is not affected unless the Counter is retriggered. In that case, the Counter is loaded with the new count and the one-shot pulse continues until the new count expires.

Figure 16. Mode 1

MODE 2: RATE GENERATOR

This Mode functions like a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt. OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated. Mode 2 is periodic; the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately. A trigger reloads the Counter with the initial count on the next CLK pulse; OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. OUT goes low N CLK Pulses after the initial count is written. This allows the Counter to be synchronized by software also.

Figure 17. Mode 2

NOTE: A GATE transition should not occur one clock prior to terminal count.
Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current period, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current counting cycle. In mode 2, a COUNT of 1 is illegal.

**MODE 3: SQUARE WAVE MODE**

Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high. When half the initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely. An initial count of \( N \) results in a square wave with a period of \( N \) CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required. A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This allows the Counter to be synchronized by software also.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

Mode 3 is implemented as follows:

**Even counts:** OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. When the count expires OUT changes value and the Counter is reloaded with the initial count. The above process is repeated indefinitely.

**Odd counts:** OUT is initially high. The initial count minus one (an even number) is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. One CLK pulse after the count expires, OUT goes low and the Counter is reloaded with the initial count minus one. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes high again and the Counter is reloaded with the initial count minus one. The above process is repeated indefinitely. So for odd counts, OUT will be high for \( (N + 1)/2 \) counts and low for \( (N - 1)/2 \) counts.

**MODE 4: SOFTWARE TRIGGERED STROBE**

OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is "triggered" by writing the initial count.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of \( N \), OUT does not strobe low until \( N + 1 \) CLK pulses after the initial count is written.

If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

---

**NOTE:**

A GATE transition should not occur one clock prior to terminal count.

Figure 18. Mode 3
1) Writing the first byte has no effect on counting.
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the sequence to be "retriggered" by software. OUT strobos low \( N + 1 \) CLK pulses after the new count of \( N \) is written.

After writing the Control Word and initial count, the counter will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count, so for an initial count of \( N \), OUT does not strobe low until \( N + 1 \) CLK pulses after a trigger.

A trigger results in the Counter being loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable. OUT will not strobe low for \( N + 1 \) CLK pulses after any trigger. GATE has no effect on OUT.

If a new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.

**MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE)**

OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again.

**Figure 19. Mode 4**

**Figure 20. Mode 5**
Signal Status Modes

<table>
<thead>
<tr>
<th>Modes</th>
<th>Low Or Going Low</th>
<th>Rising</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disables counting</td>
<td>—</td>
<td>Enables counting</td>
</tr>
<tr>
<td>1</td>
<td>—</td>
<td>1) Initiates counting 2) Resets output after next clock</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>1) Disables counting 2) Sets output immediately high</td>
<td>Initiates counting</td>
<td>Enables counting</td>
</tr>
<tr>
<td>3</td>
<td>1) Disables counting 2) Sets output immediately high</td>
<td>Initiates counting</td>
<td>Enables counting</td>
</tr>
<tr>
<td>4</td>
<td>Disables counting</td>
<td>—</td>
<td>Enables counting</td>
</tr>
<tr>
<td>5</td>
<td>—</td>
<td>Initiates counting</td>
<td>—</td>
</tr>
</tbody>
</table>

Figure 21. Gate Pin Operations Summary

Operation Common to All Modes

Programming

When a Control Word is written to a Counter, all Control Logic is immediately reset and OUT goes to a known initial state; no CLK pulses are required for this.

GATE

The GATE input is always sampled on the rising edge of CLK. In Modes 0, 2, 3, and 4 the GATE input is level sensitive, and the logic level is sampled on the rising edge of CLK. In Modes 1, 2, 3, and 5 the GATE input is rising-edge sensitive. In these Modes, a rising edge of GATE (trigger) sets an edge-sensitive flip-flop in the Counter. This flip-flop is then sampled on the next rising edge of CLK; the flip-flop is reset immediately after it is sampled. In this way, a trigger will be detected no matter when it occurs—a high logic level does not have to be maintained until the next rising edge of CLK. Note that in Modes 2 and 3, the GATE input is both edge- and level-sensitive. In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following WR of a new count value.

COUNTER

New counts are loaded and Counters are decremented on the falling edge of CLK.

The largest possible initial count is 0; this is equivalent to $2^{16}$ for binary counting and $10^4$ for BCD counting.

The Counter does not stop when it reaches zero. In Modes 0, 1, 4, and 5 the Counter "wraps around" to the highest count, either FFFF hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic; the Counter reloads itself with the initial count and continues counting from there.
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ...................... 0°C to 70°C
Storage Temperature ............................. -65°C to +150°C
Supply Voltage .................................. -0.5 to +8.0V
Operating Voltage ................................. +4V to +7V
Voltage on any Input : GND -2V to +6.5V
Voltage on any Output : GND -0.5V to VCC + 0.5V
Power Dissipation ............................... 1 Watt

*Notice: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = 5V ± 10%, GND = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td>IOH = 2.0 mA</td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>2.4</td>
<td>VCC+0.5 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td>V</td>
<td></td>
<td>VCC = 0V</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td>V</td>
<td></td>
<td>VCC = 0.45V</td>
</tr>
<tr>
<td>IIL</td>
<td>Input Load Current</td>
<td>±10</td>
<td>μA</td>
<td></td>
<td>VIN = VCC</td>
</tr>
<tr>
<td>IOFL</td>
<td>Output Float Leakage Current</td>
<td>±10</td>
<td>μA</td>
<td></td>
<td>VOUT = VCC</td>
</tr>
<tr>
<td>ICC</td>
<td>VCC Supply Current</td>
<td>10</td>
<td>mA</td>
<td>8MHz 82C54</td>
<td>Clk Freq = 8MHz 82C54-2</td>
</tr>
<tr>
<td>ICCSB</td>
<td>VCC Supply Current-Standby</td>
<td>10</td>
<td>μA</td>
<td>10MHz 82C54</td>
<td>Clk Freq = DC</td>
</tr>
</tbody>
</table>

CAPACITANCE (TA = 25°C, VCC = GND = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIN</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td></td>
<td>fC = 1 MHz</td>
</tr>
<tr>
<td>CIO</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
<td></td>
<td>Unmeasured pins returned to GND</td>
</tr>
<tr>
<td>COUT</td>
<td>Output Capacitance</td>
<td>20</td>
<td>pF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = 5V ± 10%, GND = 0V)

BUS PARAMETERS (Note 1)

READ CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>82C54</th>
<th>82C54-2</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>tAR</td>
<td>Address Stable Before RD ↓</td>
<td>45</td>
<td>30</td>
<td>ns</td>
</tr>
<tr>
<td>tSR</td>
<td>CS Stable Before RD ↓</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>tRA</td>
<td>Address Hold Time After RD ↓</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>tRR</td>
<td>RD Pulse Width</td>
<td>150</td>
<td>95</td>
<td>ns</td>
</tr>
<tr>
<td>tRD</td>
<td>Data Delay from RD ↓</td>
<td>120</td>
<td>85</td>
<td>ns</td>
</tr>
<tr>
<td>tAD</td>
<td>Data Delay from Address</td>
<td>220</td>
<td>185</td>
<td>ns</td>
</tr>
<tr>
<td>tDF</td>
<td>RD ↑ to Data Floating</td>
<td>5</td>
<td>90</td>
<td>ns</td>
</tr>
<tr>
<td>tCV</td>
<td>Command Recovery Time</td>
<td>200</td>
<td>165</td>
<td>ns</td>
</tr>
</tbody>
</table>

NOTE:
1. AC timings measured at VOH = 2.0V, VOL = 0.8V.
### A.C. CHARACTERISTICS (Continued)

#### WRITE CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>82C54</th>
<th>82C54-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>t_{AW}</td>
<td>Address Stable Before WR ↓</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t_{SW}</td>
<td>CS Stable Before WR ↓</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t_{WA}</td>
<td>Address Hold Time After WR ↑</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t_{WW}</td>
<td>WR Pulse Width</td>
<td>150</td>
<td>95</td>
</tr>
<tr>
<td>t_{DW}</td>
<td>Data Setup Time Before WR ↑</td>
<td>120</td>
<td>95</td>
</tr>
<tr>
<td>t_{WD}</td>
<td>Data Hold Time After WR ↑</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t_{RV}</td>
<td>Command Recovery Time</td>
<td>200</td>
<td>165</td>
</tr>
</tbody>
</table>

#### CLOCK AND GATE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>82C54</th>
<th>82C54-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>t_{CLK}</td>
<td>Clock Period</td>
<td>125</td>
<td>DC</td>
</tr>
<tr>
<td>t_{PWH}</td>
<td>High Pulse Width</td>
<td>60^{[3]}</td>
<td>30^{[3]}</td>
</tr>
<tr>
<td>t_{PWL}</td>
<td>Low Pulse Width</td>
<td>60^{[3]}</td>
<td>50^{[3]}</td>
</tr>
<tr>
<td>t_R</td>
<td>Clock Rise Time</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>t_F</td>
<td>Clock Fall Time</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>t_{GW}</td>
<td>Gate Width High</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>t_{GL}</td>
<td>Gate Width Low</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>t_{GS}</td>
<td>Gate Setup Time to CLK ↑</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>t_{GH}</td>
<td>Gate Hold Time After CLK ↑</td>
<td>50^{[2]}</td>
<td>50^{[2]}</td>
</tr>
<tr>
<td>t_{OD}</td>
<td>Output Delay from CLK ↓</td>
<td>150</td>
<td>100</td>
</tr>
<tr>
<td>t_{ODG}</td>
<td>Output Delay from Gate ↓</td>
<td>120</td>
<td>100</td>
</tr>
<tr>
<td>t_{WC}</td>
<td>CLK Delay for Loading</td>
<td>0</td>
<td>55</td>
</tr>
<tr>
<td>t_{WG}</td>
<td>Gate Delay for Sampling</td>
<td>−5</td>
<td>50</td>
</tr>
<tr>
<td>t_{WO}</td>
<td>OUT Delay from Mode Write</td>
<td>260</td>
<td>240</td>
</tr>
<tr>
<td>t_{CL}</td>
<td>CLK Set Up for Count Latch</td>
<td>−4</td>
<td>45</td>
</tr>
</tbody>
</table>

**NOTES:**

2. In Modes 1 and 5 triggers are sampled on each rising clock edge. A second trigger within 120 ns (70 ns for the 8254-2) of the rising clock edge may not be detected.

3. Low-going glitches that violate $t_{PWH}$, $t_{PWL}$ may cause errors requiring counter reprogramming.
WAVEFORMS

WRITE

READ

RECOVERY
A.C. TESTING INPUT, OUTPUT WAVEFORM

INPUT/OUTPUT

2.4

0.45

2.0

-2.0

TEST POINTS

2.0

-0.8

A.C. Testing: Inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

\( C_L = 150 \text{ pF} \)

\( C_L \) includes jig capacitance

* Last byte of count being written
The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.

Figure 1. 8255A Block Diagram

Figure 2. Pin Configuration
8255A FUNCTIONAL DESCRIPTION

General
The 8255A is a programmable peripheral interface (PPI) device designed for use in Intel® microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 8255A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer
This 3-state bidirectional 8-bit buffer is used to interface the 8255A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic
The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

(CS)
Chip Select. A “low” on this input pin enables the communication between the 8255A and the CPU.

8255A BASIC OPERATION

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>RD</th>
<th>WR</th>
<th>CS</th>
<th>INPUT OPERATION (READ)</th>
<th>OUTPUT OPERATION (WRITE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PORT A → DATA BUS</td>
<td>DATA BUS → PORT A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>PORT B → DATA BUS</td>
<td>DATA BUS → PORT B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>PORT C → DATA BUS</td>
<td>DATA BUS → PORT C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>DATA BUS → CONTROL</td>
<td>DISABLE FUNCTION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>RD</th>
<th>WR</th>
<th>CS</th>
<th>INPUT OPERATION (READ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>DATA BUS → 3-STATE</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>ILLEGAL CONDITION</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>DATA BUS → 3-STATE</td>
</tr>
</tbody>
</table>

Figure 3. 8255A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions
(RESET)

Reset. A "high" on this input clears the control register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the system software. In essence, the CPU "outputs" a control word to the 8255A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A — Port A and Port C upper (C7-C4)
Control Group B — Port B and Port C lower (C3-C0)

The Control Word Register can only be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

The 8255A contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255A.

Port A. One 8-bit data output latch/buffer and one 8-bit data input latch.

Port B. One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C. One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.

Figure 4. 8225A Block Diagram Showing Group A and Group B Control Functions
8255A OPERATIONAL DESCRIPTION

Mode Selection
There are three basic modes of operation that can be selected by the system software:

- Mode 0 — Basic Input/Output
- Mode 1 — Strobed Input/Output
- Mode 2 — Bi-Directional Bus

When the reset input goes "high" all ports will be set to the input mode (i.e., all 24 lines will be in the high impedance state). After the reset is removed the 8255A can remain in the input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single output instruction. This allows a single 8255A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature
Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.
Operating Modes

MODE 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

- (BIT-SET) — INTE is SET — Interrupt enable
- (BIT-RESET) — INTE is RESET — Interrupt disable

Note: All Mask flip-flops are automatically reset during mode selection and device Reset.
MODE 0 Port Definition

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>GROUP A</th>
<th>GROUP B</th>
</tr>
</thead>
<tbody>
<tr>
<td>D4</td>
<td>D3</td>
<td>D1</td>
<td>D0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

MODE 0 Configurations

CONTROL WORD #0

CONTROL WORD #2

CONTROL WORD #1

CONTROL WORD #3
Operating Modes

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In mode 1, Port A and Port B use the lines on port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:
- Two Groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.
Input Control Signal Definition

**STB (Strobe Input).** A "low" on this input loads data into the input latch.

**IBF (Input Buffer Full F/F)**
A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by STB input being low and is reset by the rising edge of the RD input.

**INTR (Interrupt Request)**
A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the STB is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of RD. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

**INTE A**
Controlled by bit set/reset of PC4.

**INTE B**
Controlled by bit set/reset of PC2.

---

**Figure 8. MODE 1 Input**

**Figure 9. MODE 1 (Strobed Input)**
Output Control Signal Definition

**OBF (Output Buffer Full F/F).** The OBF output will go “low” to indicate that the CPU has written data out to the specified port. The OBF F/F will be set by the rising edge of the WR input and reset by ACK input being low.

**ACK (Acknowledge Input).** A “low” on this input informs the 8255A that the data from port A or port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.

**INTR (Interrupt Request).** A “high” on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when ACK is a “one”, OBF is a “one”, and INTE is a “one”. It is reset by the falling edge of WR.

**INTR (Interrupt Request).** A “high” on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when ACK is a “one”, OBF is a “one”, and INTE is a “one”. It is reset by the falling edge of WR.

**INTE A**
Controlled by bit set/reset of PC6.

**INTE B**
Controlled by bit set/reset of PC2.

---

**Figure 10. MODE 1 Output**

**Figure 11. Mode 1 (Strobed Output)**
Combinations of MODE 1

Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

![Diagram of Combinations of MODE 1](image)

**Figure 12. Combinations of MODE 1**

### Operating Modes

#### MODE 2 (Strobed Bidirectional Bus I/O)

This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). “Handshaking” signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

**MODE 2 Basic Functional Definitions:**
- Used in Group A only.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

### Bidirectional Bus I/O Control Signal Definition

#### INTR (Interrupt Request)

A high on this output can be used to interrupt the CPU for both input or output operations.

#### Output Operations

**OBF (Output Buffer Full).** The OBF output will go “low” to indicate that the CPU has written data out to port A.

**ACK (Acknowledge).** A “low” on this input enables the tri-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

**INTE 1 (The INTE Flip-Flop Associated with OBF).** Controlled by bit set/reset of PC6.

### Input Operations

**STB (Strobe Input).** A “low” on this input loads data into the input latch.

**IBF (Input Buffer Full F/F).** A “high” on this output indicates that data has been loaded into the input latch.

**INTE 2 (The INTE Flip-Flop Associated with IBF).** Controlled by bit set/reset of PC4.
Figure 13. MODE Control Word

Figure 14. MODE 2

Figure 15. MODE 2 (Bidirectional)

NOTE: Any sequence where WR occurs before ACK and STB occurs before RD is permissible.

(INTR = IBF • MASK • STB • RD + OBF • MASK • ACK • WR)

231308-001
Figure 16. MODE ¼ Combinations
## Mode Definition Summary

<table>
<thead>
<tr>
<th>MODE 0</th>
<th>MODE 1</th>
<th>MODE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA0</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA1</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA2</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA3</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA4</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA5</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA6</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA7</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB0</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB1</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB2</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB3</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB4</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB5</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB6</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB7</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC0</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC1</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC2</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC3</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC4</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC5</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC6</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC7</td>
<td>IN</td>
<td>OUT</td>
</tr>
</tbody>
</table>

### Special Mode Combination Considerations

There are several combinations of modes when not all of the bits in Port C are used for control or status. The remaining bits can be used as follows:

- **If Programmed as Inputs**
  - All input lines can be accessed during a normal Port C read.

- **If Programmed as Outputs**
  - Bits in C upper (PC\(_7\)-PC\(_4\)) must be individually accessed using the bit set/reset function.
  - Bits in C lower (PC\(_3\)-PC\(_0\)) can be accessed using the bit set/reset function or accessed as a threesome by writing into Port C.

### Source Current Capability on Port B and Port C

Any set of **eight** output buffers, selected randomly from Ports B and C can source 1mA at 1.5 volts. This feature allows the 8255 to directly drive Darlington type drivers and high-voltage displays that require such source current.

### Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 8255 is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly. There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

---

**Figure 17. MODE 1 Status Word Format**

**Figure 18. MODE 2 Status Word Format**
APPLICATIONS OF THE 8255A

The 8255A is a very powerful tool for interfacing peripheral equipment to the microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 8255A is programmed by the service routine and becomes an extension of the system software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word can easily be developed to initialize the 8255A to exactly "fit" the application. Figures 19 through 25 present a few examples of typical applications of the 8255A.

Figure 19. Printer Interface

Figure 20. Keyboard and Display Interface

Figure 21. Keyboard and Terminal Address Interface
Figure 22. Digital to Analog, Analog to Digital

Figure 23. Basic Floppy Disk Interface

Figure 24. Basic CRT Controller Interface

Figure 25. Machine Tool Controller Interface
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ............. 0°C to 70°C
Storage Temperature ...................... –65°C to +150°C
Voltage on Any Pin
  With Respect to Ground .............. –0.5V to +7V
Power Dissipation ....................... 1 Watt

*NOTICE: Stresses above those listed under "Absolute
  Maximum Ratings" may cause permanent damage to the
device. This is a stress rating only and functional opera­
tion of the device at these or any other conditions above
those indicated in the operational sections of this specifi­
cation is not implied. Exposure to absolute maximum
  rating conditions for extended periods may affect device
reliability.

D.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = +5V ± 10%, GND = 0V) *

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI_L</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VI_H</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>VCC</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VO_L(DB)</td>
<td>Output Low Voltage (Data Bus)</td>
<td>0.45*</td>
<td>V</td>
<td>IO_L = 2.5mA</td>
<td></td>
</tr>
<tr>
<td>VO_L(PER)</td>
<td>Output Low Voltage (Peripheral Port)</td>
<td>0.45*</td>
<td>V</td>
<td>IO_L = 1.7mA</td>
<td></td>
</tr>
<tr>
<td>VO_H(DB)</td>
<td>Output High Voltage (Data Bus)</td>
<td>2.4</td>
<td>V</td>
<td>IO_H = -400μA</td>
<td></td>
</tr>
<tr>
<td>VO_H(PER)</td>
<td>Output High Voltage (Peripheral Port)</td>
<td>2.4</td>
<td>V</td>
<td>IO_H = -200μA</td>
<td></td>
</tr>
<tr>
<td>ID_AR[1]</td>
<td>Darlington Drive Current</td>
<td>-1.0</td>
<td>-4.0</td>
<td>mA</td>
<td>R_EXT = 750Ω; V_EXT = 1.5V</td>
</tr>
<tr>
<td>I_C</td>
<td>Power Supply Current</td>
<td>120</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_L</td>
<td>Input Load Current</td>
<td>±10</td>
<td>μA</td>
<td></td>
<td>V_IN = V_CC to 0V</td>
</tr>
<tr>
<td>I_OFL</td>
<td>Output Float Leakage</td>
<td>±10</td>
<td>μA</td>
<td></td>
<td>V_OUT = V_CC to .45V</td>
</tr>
</tbody>
</table>

NOTE:
1. Available on any 8 pins from Port B and C.

CAPACITANCE (TA = 25°C, VCC = GND = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_IN</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td></td>
<td></td>
<td>tc = 1MHz</td>
</tr>
<tr>
<td>C_I/O</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
<td></td>
<td></td>
<td>Unmeasured pins returned to GND</td>
</tr>
</tbody>
</table>

A.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = +5V ± 10%, GND = 0V) *

Bus Parameters

READ

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8255A</th>
<th>8255A-5</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_AR</td>
<td>Address Stable Before READ</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>t_RA</td>
<td>Address Stable After READ</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>t_RR</td>
<td>READ Pulse Width</td>
<td>300</td>
<td>300</td>
<td>ns</td>
</tr>
<tr>
<td>t_RD</td>
<td>Data Valid From READ[1]</td>
<td></td>
<td>250</td>
<td>200</td>
</tr>
<tr>
<td>t_DF</td>
<td>Data Float After READ</td>
<td>10</td>
<td>150</td>
<td>10</td>
</tr>
<tr>
<td>t_RV</td>
<td>Time Between READs and/or WRITEs</td>
<td>850</td>
<td>850</td>
<td>ns</td>
</tr>
</tbody>
</table>
### A.C. CHARACTERISTICS (Continued)

#### WRITE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8255A Min.</th>
<th>8255A Max.</th>
<th>8255A-5 Min.</th>
<th>8255A-5 Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{AW} )</td>
<td>Address Stable Before WRITE</td>
<td>0</td>
<td>0</td>
<td>( t_{WA} )</td>
<td>Address Stable After WRITE</td>
<td>20</td>
</tr>
<tr>
<td>( t_{WW} )</td>
<td>WRITE Pulse Width</td>
<td>400</td>
<td>300</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{DW} )</td>
<td>Data Valid to WRITE (T.E.)</td>
<td>100</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{WD} )</td>
<td>Data Valid After WRITE</td>
<td>30</td>
<td>30</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### OTHER TIMINGS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8255A Min.</th>
<th>8255A Max.</th>
<th>8255A-5 Min.</th>
<th>8255A-5 Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{WB} )</td>
<td>WR = 1 to Output(^1)</td>
<td>350</td>
<td>350</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{IR} )</td>
<td>Peripheral Data Before RD</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{HR} )</td>
<td>Peripheral Data After RD</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{AK} )</td>
<td>ACK Pulse Width</td>
<td>300</td>
<td>300</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{ST} )</td>
<td>STB Pulse Width</td>
<td>500</td>
<td>500</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PS} )</td>
<td>Per. Data Before T.E. of STB</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PH} )</td>
<td>Per. Data After T.E. of STB</td>
<td>180</td>
<td>180</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{AD} )</td>
<td>ACK = 0 to Output(^1)</td>
<td>300</td>
<td>300</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{KD} )</td>
<td>ACK = 1 to Output Float</td>
<td>20</td>
<td>250</td>
<td>20</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{WOB} )</td>
<td>WR = 1 to OBF = 0(^1)</td>
<td>650</td>
<td>650</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{AOB} )</td>
<td>ACK = 0 to OBF = 1(^1)</td>
<td>350</td>
<td>350</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{SIB} )</td>
<td>STB = 0 to IBF = 1(^1)</td>
<td>300</td>
<td>300</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{RIB} )</td>
<td>RD = 1 to IBF = 0(^1)</td>
<td>300</td>
<td>300</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{RIT} )</td>
<td>RD = 0 to INTR = 0(^1)</td>
<td>400</td>
<td>400</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{SIT} )</td>
<td>STB = 1 to INTR = 1(^1)</td>
<td>300</td>
<td>300</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{AIT} )</td>
<td>ACK = 1 to INTR = 1(^1)</td>
<td>350</td>
<td>350</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{WIT} )</td>
<td>WR = 0 to INTR = 0(^1,3)</td>
<td>450</td>
<td>450</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**

1. Test Conditions: \( C_L = 150 \) pF.
2. Period of Reset pulse must be at least 50\( \mu \)s during or after power on. Subsequent Reset pulse can be 500 ns min.
3. INTR\(^1\) may occur as early as WR\(^1\).

* For Extended Temperature EXPRESS, use M8255A electrical parameters.

### A.C. TESTING INPUT, OUTPUT WAVEFORM

![A.C. TESTING INPUT, OUTPUT WAVEFORM](image)

A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC 1 AND 0.45V FOR A LOGIC 0. TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC 1 AND 0.8V FOR A LOGIC 0.

### A.C. TESTING LOAD CIRCUIT

![A.C. TESTING LOAD CIRCUIT](image)

\( V_{\text{EXT}} \) IS SET AT VARIOUS VOLTAGES DURING TESTING TO GUARANTEE THE SPECIFICATION \( C_L \) INCLUDES JUG CAPACITANCE.
WAVEFORMS (Continued)

MODE 1 (STROBED INPUT)

MODE 1 (STROBED OUTPUT)
WAVEFORMS (Continued)

MODE 2 (BIDIRECTIONAL)

DATA FROM 8080 TO 8255

WR

OBF

INTR

ACK

STB

IBF

PERIPHERAL BUS

DATA FROM PERIPHERAL TO 8255

DATA FROM 8255 TO PERIPHERAL

DATA FROM 8255 TO 8080

NOTE: Any sequence where WR occurs before ACK and STB occurs before RD is permissible.
(INTR = IBF · MASK · STB · RD + OBF · MASK · ACK · WR)

WRITE TIMING

READ TIMING

A0-1, CS

DATA BUS

WR

A0-1, CS

RD

DATA BUS: HIGH IMPEDANCE

VALID: HIGH IMPEDANCE

tAW

tAD

tRD

tWW

tWR

tAS

tAD

tRD

tWW

tWR

5-293
The Intel 82C55A is a high-performance, CHMOS version of the industry standard 8255A general purpose programmable I/O device which is designed for use with all Intel and most other microprocessors. It provides 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. The 82C55A is pin compatible with the NMOS 8255A and 8255A-5.

In MODE 0, each group of 12 I/O pins may be programmed in sets of 4 to be inputs or outputs. In MODE 1, each group may be programmed to have 8 lines of input or output. 3 of the remaining 4 pins are used for handshaking and interrupt control signals. MODE 2 is a bi-directional bus, and 5 lines, borrowing one from the other group, for handshaking.

The 82C55A is fabricated on Intel's advanced CHMOS III technology which provides low power consumption with performance equal to or greater than the equivalent NMOS product. The 82C55A is available in 40-pin DIP and 44-pin plastic leaded chip carrier (PLCC) packages.
## Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin Number</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA3–0</td>
<td>1–4</td>
<td>I/O</td>
<td>PORT A, PINS 0–3: Lower nibble of an 8-bit data output latch/ buffer and an 8-bit data input latch.</td>
</tr>
<tr>
<td>RD</td>
<td>5</td>
<td>I</td>
<td>READ CONTROL: This input is low during CPU read operations.</td>
</tr>
<tr>
<td>CS</td>
<td>6</td>
<td>I</td>
<td>CHIP SELECT: A low on this input enables the 82C55A to respond to RD and WR signals. RD and WR are ignored otherwise.</td>
</tr>
<tr>
<td>GND</td>
<td>7</td>
<td></td>
<td>System Ground</td>
</tr>
<tr>
<td>A1–0</td>
<td>8–9</td>
<td>I</td>
<td>ADDRESS: These input signals, in conjunction RD and WR, control the selection of one of the three ports or the control word registers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Input Operation (Read)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A1  A0  RD  WR  CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0    0    0    1    0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0    1    0    1    0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1    0    0    1    0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1    1    0    1    0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Output Operation (Write)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0    0    1    0    0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0    1    1    0    0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1    0    1    0    0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1    1    1    0    0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Disable Function</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X    X    X    X    1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X    X    1    1    0</td>
</tr>
<tr>
<td>PC7–4</td>
<td>10–13</td>
<td>I/O</td>
<td>PORT C, PINS 4–7: Upper nibble of an 8-bit data output latch/ buffer and an 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.</td>
</tr>
<tr>
<td>PC0–3</td>
<td>14–17</td>
<td>I/O</td>
<td>PORT C, PINS 0–3: Lower nibble of Port C.</td>
</tr>
<tr>
<td>PB0–7</td>
<td>18–25</td>
<td>I/O</td>
<td>PORT B, PINS 0–7: An 8-bit data output latch/buffer and an 8-bit data input buffer.</td>
</tr>
<tr>
<td>VCC</td>
<td>26</td>
<td></td>
<td>SYSTEM POWER: + 5V Power Supply.</td>
</tr>
<tr>
<td>D7–0</td>
<td>27–34</td>
<td>I/O</td>
<td>DATA BUS: Bi-directional, tri-state data bus lines, connected to system data bus.</td>
</tr>
<tr>
<td>RESET</td>
<td>35</td>
<td>I</td>
<td>RESET: A high on this input clears the control register and all ports are set to the input mode.</td>
</tr>
<tr>
<td>WR</td>
<td>36</td>
<td>I</td>
<td>WRITE CONTROL: This input is low during CPU write operations.</td>
</tr>
<tr>
<td>NC</td>
<td>8, 28</td>
<td></td>
<td>No Connect</td>
</tr>
</tbody>
</table>

**Remark:**
- Data Bus - 3 - State
- Data Bus - 3 - State
82C55A FUNCTIONAL DESCRIPTION

General

The 82C55A is a programmable peripheral interface device designed for use in Intel microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 82C55A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 82C55A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 82C55A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A - Port A and Port C upper (C7-C4)  
Control Group B - Port B and Port C lower (C3-C0)

The control word register can be both written and read as shown in the address decode table in the pin descriptions. Figure 6 shows the control word format for both Read and Write operations. When the control word is read, bit D7 will always be a logic "1", as this implies control word mode information.

Ports A, B, and C

The 82C55A contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 82C55A.

Port A. One 8-bit data output latch/buffer and one 8-bit input latch. Both "pull-up" and "pull-down" bus hold devices are present on Port A.

Port B. One 8-bit data input/output latch/buffer and one 8-bit data input buffer. Only "pull-up" bus hold devices are present on Port B.

Port C. One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B. Only "pull-up" bus hold devices are present on Port C.

See Figure 4 for the bus-hold circuit configuration for Port A, B, and C.
Figure 3. 82C55A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions

Figure 4. Port A, B, C, Bus-Mode Configuration
82C55A OPERATIONAL DESCRIPTION

Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0 — Basic input/output
Mode 1 — Strobed Input/output
Mode 2 — Bi-directional Bus

When the reset input goes "high" all ports will be set to the input mode with all 24 port lines held at a logic "one" level by the internal bus hold devices. After the reset is removed the 82C55A can remain in the input mode with no additional initialization required. This eliminates the need for pullup or pulldown devices in "all CMOS" designs. During the execution of the system program, any of the other modes may be selected by using a single output instruction. This allows a single 82C55A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 82C55A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.
Interrupt Control Functions

When the 82C55A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET)—INTE is SET—Interrupt enable
(BIT-RESET)—INTE is RESET—Interrupt disable

Note:
All Mask flip-flops are automatically reset during mode selection and device Reset.
Operating Modes

Mode 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:
- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.

MODE 0 (BASIC INPUT)

MODE 0 (BASIC OUTPUT)
MODEL 0 Port Definition

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>GROUP A</th>
<th>GROUP B</th>
</tr>
</thead>
<tbody>
<tr>
<td>D4</td>
<td>D3</td>
<td>D1</td>
<td>D0</td>
</tr>
<tr>
<td>PORT A</td>
<td>PORT C (UPPER)</td>
<td>PORT B</td>
<td>PORT C (LOWER)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

MODE 0 Configurations

CONTROL WORD #0

CONTROL WORD #1

CONTROL WORD #2

CONTROL WORD #3

231256–10
Operating Modes

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or “handshaking” signals. In mode 1, Port A and Port B use the lines on Port C to generate or accept these “handshaking” signals.

Mode 1 Basic functional Definitions:
- Two Groups (Group A and Group B).
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output.
- Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.
Input Control Signal Definition

STB (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by STB input being low and is reset by the rising edge of the RD input.

INTR (Interrupt Request)

A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the STB is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of RD. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A
Controlled by bit set/reset of PC4.

INTE B
Controlled by bit set/reset of PC2.

Figure 8. MODE 1 Input

Figure 9. MODE 1 (Strobed Input)
Output Control Signal Definition

OBF (Output Buffer Full F/F). The OBF output will go “low” to indicate that the CPU has written data out to the specified port. The OBF F/F will be set by the rising edge of the WR input and reset by ACK Input being low.

ACK (Acknowledge Input). A “low” on this input informs the 82C55A that the data from Port A or Port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.

INTR (Interrupt Request). A “high” on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when ACK is a “one”, OBF is a “one” and INTE is a “one”. It is reset by the falling edge of WR.

INTE A
Controlled by bit set/reset of PC6.

INTE B
Controlled by bit set/reset of PC2.
Combinations of MODE 1

Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

Operating Modes

MODE 2 (Strobed Bidirectional Bus I/O). This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). “Handshaking” signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

MODE 2 Basic Functional Definitions:
- Used in Group A only.
- One 8-bit, bi-directional bus port (Port A) and a 5-bit control port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bidirectional Bus I/O Control Signal Definition

INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for input or output operations.

Output Operations

ÖBF (Output Buffer Full). The ÖBF output will go “low” to indicate that the CPU has written data out to port A.

ACK (Acknowledge). A “low” on this input enables the tri-state output buffer of Port A to send out the data. Otherwise, the output buffer will be in the high impedance state.


Input Operations

STB (Strobe Input). A “low” on this input loads data into the input latch.

IBF (Input Buffer Full F/F). A “high” on this output indicates that data has been loaded into the input latch.

Figure 13. MODE Control Word

Figure 14. MODE 2

Figure 15. MODE 2 (Bidirectional)

NOTE:
Any sequence where WR occurs before ACK, and STB occurs before RD is permissible.
(INTR = IBF • MASK • STB • RD + OBF • MASK • ACK • WR)
Figure 16. MODE ¼ Combinations
### Mode Definition Summary

<table>
<thead>
<tr>
<th>MODE 0</th>
<th>MODE 1</th>
<th>MODE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA0</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA1</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA2</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA3</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA4</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA5</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA6</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PA7</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB0</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB1</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB2</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB3</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB4</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB5</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB6</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PB7</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC0</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC1</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC2</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC3</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC4</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC5</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC6</td>
<td>IN</td>
<td>OUT</td>
</tr>
<tr>
<td>PC7</td>
<td>IN</td>
<td>OUT</td>
</tr>
</tbody>
</table>

### Special Mode Combination Considerations

There are several combinations of modes possible. For any combination, some or all of the Port C lines are used for control or status. The remaining bits are either inputs or outputs as defined by a "Set Mode" command.

During a read of Port C, the state of all the Port C lines, except the ACK and STB lines, will be placed on the data bus. In place of the ACK and STB line states, flag status will appear on the data bus in the PC2, PC4, and PC6 bit positions as illustrated by Figure 18.

Through a "Write Port C" command, only the Port C pins programmed as outputs in a Mode 0 group can be written. No other pins can be affected by a "Write Port C" command, nor can the interrupt enable flags be accessed. To write to any Port C output programmed as an output in a Mode 1 group or to change an interrupt enable flag, the "Set/Reset Port C Bit" command must be used.

With a "Set/Reset Port C Bit" command, any Port C line programmed as an output (including INTR, IBF and OBF) can be written, or an interrupt enable flag can be either set or reset. Port C lines programmed as inputs, including ACK and STB lines, associated with Port C are not affected by a "Set/Reset Port C Bit" command. Writing to the corresponding Port C bit positions of the ACK and STB lines with the "Set/Reset Port C Bit" command will affect the Group A and Group B interrupt enable flags, as illustrated in Figure 18.

### Current Drive Capability

Any output on Port A, B or C can sink or source 2.5 mA. This feature allows the 82C55A to directly drive Darlington type drivers and high-voltage displays that require such sink or source current.
Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 82C55A is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias.....0°C to + 70°C
Storage Temperature .......... -65°C to + 150°C
Supply Voltage .................. -0.5 to + 8.0V
Operating Voltage .............. + 4V to + 7V
Voltage on any Input .......... GND−2V to + 6.5V
Voltage on any Output .. GND−0.5V to VCC + 0.5V
Power Dissipation ................ 1 Watt

*Notice: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS
$T_A = 0°C to 70°C, V_{CC} = +5V \pm 10\%, \ GND = 0V$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>V_{CC}</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>0.4</td>
<td></td>
<td>V</td>
<td>$I_{OL} = 2.5 mA$</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>3.0</td>
<td>V_{CC}−0.4</td>
<td>V</td>
<td>$I_{OH} = -2.5 mA$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$I_{OH} = -100 \mu A$</td>
</tr>
<tr>
<td>IIL</td>
<td>Input Leakage Current</td>
<td>±10</td>
<td></td>
<td>\mu A</td>
<td>$V_{IN} = V_{CC}$ to 0V</td>
</tr>
<tr>
<td>IOL</td>
<td>Output Float Leakage Current</td>
<td>±10</td>
<td>\mu A</td>
<td>$V_{IN} = V_{CC}$ to 0V</td>
<td></td>
</tr>
<tr>
<td>IDAR</td>
<td>Darlington Drive Current</td>
<td>-2.0</td>
<td></td>
<td>mA</td>
<td>Ports A, B, C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$R_{ext} = 750 \Omega$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$V_{ext} = 1.5V$</td>
</tr>
<tr>
<td>IBHL</td>
<td>Bus Hold Low Leakage Current</td>
<td>+50</td>
<td>+300</td>
<td>\mu A</td>
<td>$V_{OUT} = 1.0V$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Port A only</td>
</tr>
<tr>
<td>IBHH</td>
<td>Bus Hold High Leakage Current</td>
<td>-50</td>
<td>-300</td>
<td>\mu A</td>
<td>$V_{OUT} = 3.0V$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Ports A, B, C</td>
</tr>
<tr>
<td>ICC</td>
<td>V_{CC} Supply Current</td>
<td>10</td>
<td></td>
<td>mA</td>
<td>Outputs Open</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>See Note</td>
</tr>
<tr>
<td>ICCSB</td>
<td>V_{CC} Supply Current-Standby</td>
<td>10</td>
<td>\mu A</td>
<td>$V_{CC} = 5.5V$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$V_{IN} = V_{CC}$ or GND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Outputs Open</td>
</tr>
</tbody>
</table>

Note:
Average $I_{CC}$ with all outputs open is 100 \mu A.
## Capacitance

$T_A = 25°C, V_{CC} = GND = 0V$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{IN}$</td>
<td>Input Capacitance</td>
<td>10</td>
<td></td>
<td>pF</td>
<td>Unmeasured pins returned to GND</td>
</tr>
<tr>
<td>$C_{I/O}$</td>
<td>I/O Capacitance</td>
<td>20</td>
<td></td>
<td>pF</td>
<td></td>
</tr>
</tbody>
</table>

## A.C. Characteristics

$T_A = 0°$ to $70°C, V_{CC} = +5V ±10%, GND = 0V$

## Bus Parameters

### Read Cycle

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>$82C55A$</th>
<th>$82C55A$-2</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{AR}$</td>
<td>Address Stable Before RD ↓</td>
<td>Min</td>
<td>Max</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>$t_{RA}$</td>
<td>Address Hold Time After RD ↑</td>
<td>0</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>$t_{RR}$</td>
<td>RD Pulse Width</td>
<td>150</td>
<td>150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>$t_{RD}$</td>
<td>Data Delay from RD ↓</td>
<td>120</td>
<td>120</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>$t_{DF}$</td>
<td>RD ↑ to Data Floating</td>
<td>10</td>
<td>75</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>$t_{RV}$</td>
<td>Recovery Time between RD/WR</td>
<td>300</td>
<td>300</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

### Write Cycle

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>$82C55A$</th>
<th>$82C55A$-2</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{AW}$</td>
<td>Address Stable Before WR ↓</td>
<td>Min</td>
<td>Max</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>$t_{WA}$</td>
<td>Address Hold Time After WR ↑</td>
<td>20</td>
<td>20</td>
<td></td>
<td>Ports A &amp; B</td>
</tr>
<tr>
<td>$t_{WW}$</td>
<td>WR Pulse Width</td>
<td>100</td>
<td>100</td>
<td></td>
<td>Port C</td>
</tr>
<tr>
<td>$t_{DW}$</td>
<td>Data Setup Time Before WR ↑</td>
<td>100</td>
<td>100</td>
<td></td>
<td>Ports A &amp; B</td>
</tr>
<tr>
<td>$t_{WD}$</td>
<td>Data Hold Time After WR ↑</td>
<td>30</td>
<td>30</td>
<td></td>
<td>Port C</td>
</tr>
</tbody>
</table>
## OTHER TIMINGS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>82C55A</th>
<th>82C55A-2</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>tWB</td>
<td>WR = 1 to Output</td>
<td>350</td>
<td></td>
<td>350</td>
<td>ns</td>
</tr>
<tr>
<td>tR</td>
<td>Peripheral Data Before RD</td>
<td>0</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tHR</td>
<td>Peripheral Data After RD</td>
<td>0</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tAK</td>
<td>ACK Pulse Width</td>
<td>100</td>
<td>100</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tST</td>
<td>STB Pulse Width</td>
<td>100</td>
<td>100</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tPS</td>
<td>Per. Data Before STB High</td>
<td>20</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tPH</td>
<td>Per. Data After STBHigh</td>
<td>50</td>
<td>50</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tAD</td>
<td>ACK = 0 to Output</td>
<td>175</td>
<td>175</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tKD</td>
<td>ACK = 1 to Output Float</td>
<td>20</td>
<td>250</td>
<td>20</td>
<td>250</td>
</tr>
<tr>
<td>tWOB</td>
<td>WR = 1 to OBF = 0</td>
<td>150</td>
<td>150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tAOB</td>
<td>ACK = 0 to OBF = 1</td>
<td>150</td>
<td>150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tSIB</td>
<td>STB = 0 to IBF = 1</td>
<td>150</td>
<td>150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tRIB</td>
<td>RD = 1 to IBF = 0</td>
<td>150</td>
<td>150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tRIT</td>
<td>RD = 0 to INTR = 0</td>
<td>200</td>
<td>200</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tSIT</td>
<td>STB = 1 to INTR = 1</td>
<td>150</td>
<td>150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tAIT</td>
<td>ACK = 1 to INTR = 1</td>
<td>150</td>
<td>150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>tWIT</td>
<td>WR = 0 to INTR = 0</td>
<td>200</td>
<td>200</td>
<td></td>
<td>see note 1</td>
</tr>
<tr>
<td>tRES</td>
<td>Reset Pulse Width</td>
<td>500</td>
<td>500</td>
<td></td>
<td>see note 2</td>
</tr>
</tbody>
</table>

**NOTE:**

1. INTR † may occur as early as WR ‡.
2. Pulse width of initial Reset pulse after power on must be at least 50 µSec. Subsequent Reset pulses may be 500 ns minimum.
WAVEFORMS

MODE 0 (BASIC INPUT)

MODE 0 (BASIC OUTPUT)
WAVEFORMS (Continued)

MODE 1 (STROBED INPUT)

MODE 1 (STROBED OUTPUT)
WAVEFORMS (Continued)

MODE 2 (BIDIRECTIONAL)

Note:
Any sequence where WR occurs before ACK AND STB occurs before RD is permissible.
(INTR = IBF • MASK • STB • RD + OBS • MASK • ACK • WR)

WRITE TIMING

READ TIMING

A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING LOAD CIRCUIT

A.C. Testing Inputs Are Driven At 2.4V For A Logic 1 And 0.45V For A Logic 0. Timing Measurements Are Made At 2.0V For A Logic 1 And 0.8 For A Logic 0.
**8256AH**

**MULTIFUNCTION MICROPROCESSOR SUPPORT CONTROLLER**

- Programmable Serial Asynchronous Communications Interface for 5, 6, 7, or 8-Bit Characters, 1, 1½, or 2 Stop Bits, and Parity Generation
- Two 8-Bit Programmable Parallel I/O Ports; Port 1 Can Be Programmed for Port 2 Handshake Controls and Event Counter Inputs
- On-Board Baud Rate Generator Programmable for 13 Common Baud Rates up to 19.2K Bits/second, or an External Baud Clock Maximum of 1M Bit/second
- Eight-Level Priority Interrupt Controller Programmable for 8085 or iAPX 86, iAPX 88 Systems, and for Fully Nested Interrupt Capability
- Five 8-Bit Programmable Timer/Counters; Four Can Be Cascaded to Two 16-Bit Timer/Counters
- Programmable System Clock to 1 x, 2 x, 3 x, or 5 x 1.024 MHz

The Intel® 8256AH Multifunction Universal Asynchronous Receiver-Transmitter (MUART) combines five commonly used functions into a single 40-pin device. It is designed to interface to the 8086/88, iAPX 186/188, and 8051 to perform serial communications, parallel I/O, timing, event counting, and priority interrupt functions. All of these functions are fully programmable through nine internal registers. In addition, the five timer/counters and two parallel I/O ports can be accessed directly by the microprocessor.

---

**Figure 1. MUART Block Diagram**

**Figure 2. MUART Pin Configuration**
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADO-AD4 DB5-DB7</td>
<td>1-5</td>
<td>I/O</td>
<td>ADDRESS/DATA: Three-state address/data lines which interface to the lower 8 bits of the microprocessor’s multiplexed address/data bus. The 5-bit address is latched on the falling edge of ALE. In the 8-bit mode, ADO-AD3 are used to select the proper register, while AD1-AD4 are used in the 16-bit mode. AD4 in the 8-bit mode is ignored as an address, while AD0 in the 16-bit mode is used as a second chip select, active low.</td>
</tr>
<tr>
<td>ALE</td>
<td>9</td>
<td>I</td>
<td>ADDRESS LATCH ENABLE: Latches the 5 address lines on ADO-AD4 and CS on the falling edge.</td>
</tr>
<tr>
<td>RD</td>
<td>10</td>
<td>I</td>
<td>READ CONTROL: When this signal is low, the selected register is gated onto the data bus.</td>
</tr>
<tr>
<td>WR</td>
<td>11</td>
<td>I</td>
<td>WRITE CONTROL: When this signal is low, the value on the data bus is written into the selected register.</td>
</tr>
<tr>
<td>RESET</td>
<td>12</td>
<td>I</td>
<td>RESET: An active high pulse on this pin forces the chip into its initial state. The chip remains in this state until control information is written.</td>
</tr>
<tr>
<td>CS</td>
<td>13</td>
<td>I</td>
<td>CHIP SELECT: A low on this signal enables the MUART. It is latched with the address on the falling edge of ALE, and RD and WR have no effect unless CS was latched low during the ALE cycle.</td>
</tr>
<tr>
<td>INTA</td>
<td>14</td>
<td>I</td>
<td>INTERRUPT ACKNOWLEDGE: If the MUART has been enabled to respond to interrupts, this signal informs the MUART that its interrupt request is being acknowledged by the microprocessor. During this acknowledgement the MUART puts an RSTn instruction on the data bus for the 8-bit mode or a vector for the 16-bit mode.</td>
</tr>
<tr>
<td>INT</td>
<td>15</td>
<td>O</td>
<td>INTERRUPT REQUEST: A high signals the microprocessor that the MUART needs service.</td>
</tr>
<tr>
<td>EXTINT</td>
<td>16</td>
<td>I</td>
<td>EXTERNAL INTERRUPT: An external device can request interrupt service through this input. The input is level sensitive (high), therefore it must be held high until an INTA occurs or the interrupt address register is read.</td>
</tr>
<tr>
<td>CLK</td>
<td>17</td>
<td>I</td>
<td>SYSTEM CLOCK: The reference clock for the baud rate generator and the timers.</td>
</tr>
<tr>
<td>RxC</td>
<td>18</td>
<td>I/O</td>
<td>RECEIVE CLOCK: If the baud rate bits in the Command Register 2 are all 0, this pin is an input which clocks serial data into the RxD pin on the rising edge of RxC. If baud rate bits in Command Register 2 are programmed from 1-0FH, this pin outputs a square wave whose rising edge indicates when the data on RxD is being sampled. This output remains high during start, stop, and parity bits.</td>
</tr>
<tr>
<td>RxD</td>
<td>19</td>
<td>I</td>
<td>RECEIVE DATA: Serial data input.</td>
</tr>
<tr>
<td>GND</td>
<td>20</td>
<td>PS</td>
<td>GROUND: Power supply and logic ground reference.</td>
</tr>
</tbody>
</table>
Table 1. Pin Description (continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTS</td>
<td>21</td>
<td>I</td>
<td>CLEAR TO SEND: This input enables the serial transmitter. If 1, 1.5, or 2 stop bits are selected, CTS is level sensitive. As long as CTS is low, any character loaded into the transmitter buffer register will be transmitted serially. A single negative going pulse causes the transmission of a single character previously loaded into the transmitter buffer register. If a baud rate from 1-OFH is selected, CTS must be low for at least 1/32 of a bit, or it will be ignored. If the transmitter buffer is empty, this pulse will be ignored. If this pulse occurs during the transmission of a character up to the time where ½ the first (or only) stop bit is sent out, it will be ignored. If it occurs afterwards, but before the end of the stop bits, the next character will be transmitted immediately following the current one. If CTS is still high when the transmitter register is sending the last stop bit, the transmitter will enter its idle state until the next high-to-low transition on CTS occurs. If 0.75 stop bits is chosen, the CTS input is edge sensitive. A negative edge on CTS results in the immediate transmission of the next character. The length of the stop bits is determined by the time interval between the beginning of the first stop bit and the next negative edge on CTS. A high-to-low transition has no effect if the transmitter buffer is empty or if the time interval between the beginning of the stop bit and next negative edge is less than 0.75 bits. A high or a low level or a low-to-high transition has no effect on the transmitter for the 0.75 stop bit mode.</td>
</tr>
<tr>
<td>TxC</td>
<td>22</td>
<td>I/O</td>
<td>TRANSMIT CLOCK: If the baud rate bits in command register 2 are all set to 0, this input clocks data out of the transmitter on the falling edge. If baud rate bits are programmed for 1 or 2, this input permits the user to provide a 32x or 64x clock which is used for the receiver and transmitter. If the baud rate bits are programmed for 3-0FH, the internal transmitter clock is output. As an output it delivers the transmitter clock at the selected bit rate. If 1½ or 0.75 stop bits are selected, the transmitter divider will be asynchronously reset at the beginning of each start bit, immediately causing a high-to-low transition on TxC. TxC makes a high-to-low transition at the beginning of each serial bit, and a low-to-high transition at the center of each bit.</td>
</tr>
<tr>
<td>TxD</td>
<td>23</td>
<td>O</td>
<td>TRANSMIT DATA: Serial data output.</td>
</tr>
<tr>
<td>P27-P20</td>
<td>24-31</td>
<td>I/O</td>
<td>PARALLEL I/O PORT 2: Eight bit general purpose I/O port. Each nibble (4 bits) of this port can be either an input or an output. The outputs are latched whereas the input signals are not. Also, this port can be used as an 8-bit input or output port when using the two-wire handshake. In the handshake mode both inputs and outputs are latched.</td>
</tr>
<tr>
<td>P17-P10</td>
<td>32-39</td>
<td>I/O</td>
<td>PARALLEL I/O PORT 1: Each pin can be programmed as an input or an output to perform general purpose I/O. All outputs are latched whereas inputs are not. Alternatively these pins can serve as control pins which extend the functional spectrum of the chip.</td>
</tr>
<tr>
<td>Vcc</td>
<td>40</td>
<td>PS</td>
<td>POWER: +5V power supply.</td>
</tr>
</tbody>
</table>

5-319
FUNCTIONAL DESCRIPTION

The 8256AH Multi-Function Universal Asynchronous Receiver-Transmitter (UART) combines five commonly used functions into a single 40-pin device. The UART performs asynchronous serial communications, parallel I/O, timing, event counting, and interrupt control. For detailed application information, see Intel Ap Note #153, Designing with the 8256.

Serial Communications

The serial communications portion of the UART contains a full-duplex asynchronous receiver-transmitter (UART). A programmable baud rate generator is included on the UART to permit a variety of operating speeds without external components. The UART can be programmed by the CPU for a variety of character sizes, parity generation and detection, error detection, and start/stop bit handling. The receiver checks the start and stop bits in the center of the bit, and a break halts the reception of data. The transmitter can send breaks and can be controlled by an external enable pin.

Parallel I/O

The UART includes 16 bits of general purpose parallel I/O. Eight bits (Port 1) can be individually changed from input to output or used for special I/O functions. The other eight bits (Port 2) can be used as nibbles (4 bits) or as bytes. These eight bits also include a handshaking capability using two pins on Port 1.

Counter/Timers

There are five 8-bit counter/timers on the UART. The timers can be programmed to use either a 1 kHz or 16 kHz clock generated from the system clock. Four of the 8-bit counter/timers can be cascaded to two 16-bit counter/timers, and one of the 8-bit counter/timers can be reset to its initial value by an external signal.

Interrupts

An eight-level priority interrupt controller can be configured for fully nested or normal interrupt priority. Seven of the eight interrupts service functions on the UART (counter/timers, UART), and one external interrupt is provided which can be used for a particular function or for chaining interrupt controllers or more UARTs. The UART will support 8086 and 8086/88 systems with direct interrupt vectoring, or the UART can be polled to determine the cause of the interrupt. If additional interrupt control capability is needed, the UART's interrupt controller can be cascaded into another UART, into an Intel 8259A Programmable Interrupt Controller, or into the interrupt controller of the iAPX 186/188 High-Integration Microprocessor.

INITIALIZATION

In general the UART's functions are independent of each other and only the registers and bits associated with a particular function need to be initialized, not the entire chip. The command sequence is arbitrary since every register is directly addressable; however, Command Byte 1 must be loaded first. To put the device into a fully operational condition, it is necessary to write the following commands:

- Command byte 1
- Command byte 2
- Command byte 3
- Mode byte
- Port 1 control
- Set Interrupts

The modification register may be loaded if required for special applications; normally this operation is not necessary. The UART should be reset before initialization. (Either a hardware or a software reset will do.)

INTERFACING

This section describes the hardware interface between the 8256 UART and the 80186 microprocessor. Figure 3 displays the block diagram for this interface. The UART can be interfaced to many other microprocessors using these basic principles.

In all cases the 8256 will be connected directly to the CPU's multiplexed address/data bus. If latches or data bus buffers are used in a system, the UART should be on the microprocessor side of the address/data bus. The UART latches the address internally on the falling edge of ALE. The address consists of Chip Select (CS) and four address lines. For 8-bit microprocessors, AD0-AD3 are the address lines. For 16-bit microprocessors, AD1-AD4 are the address lines; AD0 is used as a second chip select which is active low. Since chip select is internally latched along with the address, it does not have to remain active during the entire instruction cycle. As long as the chip select setup and hold times are met, it can be derived from multiplexed address/data lines or multiplexed address/status lines. When the 8256 is in the 16-bit mode, A0 serves as a second chip select. As a result the UART's internal registers will all have even addresses since A0 must be zero to select the device. Normally the UART will be placed on the lower data byte. If the UART is placed on the upper data byte.
8086 — 8086 Mode Enable

This bit selects between 8085 mode and 8086/8088 mode. In 8085 mode (8086 = 0), A0 to A3 are used to address the internal registers, and an RSTn instruction is generated in response to the first INTA. In 8086 mode (8086 = 1), A1 to A4 are used to address the internal registers, and A0 is used as an extra chip select (A0 must equal zero to be enabled). The response to INTA is for 8086 interrupts where the first INTA is ignored, and an interrupt vector (40H to 47H) is placed on the bus in response to the second INTA.

BITI — Interrupt on Bit Change

This bit selects between one of two interrupt sources on Priority Level 1, either Counter/Timer 2 or Port 1 P17 interrupt. When this bit equals 0, Counter/Timer 2 will be mapped into Priority Level 1. If BITI equals 0 and Level 1 interrupt is enabled, a transition from 1 to 0 in Counter/Timer 2 will generate an interrupt request on Level 1. When BITI equals 1, Port 1 P17 external edge triggered interrupt source is mapped into Priority Level 1. In this case if Level 1 is enabled, a low-to-high transition on P17 generates an interrupt request on Level 1.
Table 2. MUART Registers

<table>
<thead>
<tr>
<th>Read Registers</th>
<th>Write Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8085 Mode:</strong></td>
<td><strong>8086 Mode:</strong></td>
</tr>
<tr>
<td>AD3</td>
<td>AD2</td>
</tr>
<tr>
<td>AD4</td>
<td>AD3</td>
</tr>
<tr>
<td>L1</td>
<td>L0</td>
</tr>
<tr>
<td>Command 1</td>
<td>Command 1</td>
</tr>
<tr>
<td>L1</td>
<td>L0</td>
</tr>
<tr>
<td>Command 2</td>
<td>Command 2</td>
</tr>
<tr>
<td>0</td>
<td>RxE</td>
</tr>
<tr>
<td>Command 3</td>
<td>Command 3</td>
</tr>
<tr>
<td>T35</td>
<td>T24</td>
</tr>
<tr>
<td>Mode</td>
<td>Mode</td>
</tr>
<tr>
<td>P17</td>
<td>P16</td>
</tr>
<tr>
<td>Port 1 Control</td>
<td>Port 1 Control</td>
</tr>
<tr>
<td>L7</td>
<td>L6</td>
</tr>
<tr>
<td>Interrupt Enable</td>
<td>Set Interrupts</td>
</tr>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>Receiver Buffer</td>
<td>Transmitter Buffer</td>
</tr>
<tr>
<td>Port 1</td>
<td>Port 1</td>
</tr>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>Timer 1</td>
<td>Timer 1</td>
</tr>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>Timer 2</td>
<td>Timer 2</td>
</tr>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>Timer 3</td>
<td>Timer 3</td>
</tr>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>Timer 4</td>
<td>Timer 4</td>
</tr>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>Timer 5</td>
<td>Timer 5</td>
</tr>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>Status</td>
<td>Status</td>
</tr>
<tr>
<td>INT</td>
<td>RBF</td>
</tr>
</tbody>
</table>
BRKI — Break-In Detect Enable

If this bit equals 0, Port 1 P16 is a general purpose I/O port. When BRKI equals 1, the Break-In Detect feature is enabled on Port 1 P16. A Break-In condition is present on the transmission line when it is forced to the start bit voltage level by the receiving station. Port 1 P16 must be connected externally to the transmission line in order to detect a Break-In. A Break-In is polled by the MUART during the transmission of the last or only stop bit of a character.

A Break-In Detect is OR-ed with Break Detect in Bit 3 of the Status Register. The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on Level 5, the transmit interrupt, while Break will generate an interrupt on Level 4, the receive interrupt.

S0, S1 — Stop Bit Length

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>Stop Bit Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.75</td>
</tr>
</tbody>
</table>

The relationship of the number of stop bits and the function of input CTS is discussed in the Pin Description section under "CTS".

L0, L1 — Character Length

<table>
<thead>
<tr>
<th>L1</th>
<th>L0</th>
<th>Character Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Command Register 2

<table>
<thead>
<tr>
<th>PEN</th>
<th>EP</th>
<th>C1</th>
<th>C0</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1R)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1W)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Programming bits 0...3 with values from 3H to FH enables the internal baud rate generator as a common clock source for the transmitter and receiver and determines its divider ratio.

Programming bits 0...3 with values of 1H or 2H enables input TxC as a common clock source for the transmitter and receiver. The external clock must provide a frequency of either 32x or 64x the baud rate. The data transmission rates range from 0...32 Kbaud.

If bits 0...3 are set to 0, separate clocks must be input to pin RxC for the receiver and pin TxC for the transmitter. Thus, different baud rates can be used for transmission and reception. In this case, prescalers are disabled and the input serial clock frequency must match the baud rate. The input serial clock frequency can range from 0 to 1.024 MHz.

B0, B1, B2, B3 — Baud Rate Select

These four bits select the bit clock's source, sampling rate, and serial rate for the internal baud rate generator.

<table>
<thead>
<tr>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
<th>Baud Rate</th>
<th>Sampling Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TxC, RxC</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>TxC/64</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>TxC/32</td>
<td>32</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>19200</td>
<td>32</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>9600</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4800</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2400</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1200</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>600</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>300</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>200</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>150</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>110</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>75</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>50</td>
<td>64</td>
</tr>
</tbody>
</table>

The following table gives an overview of the function of pins TxC and RxC:

<table>
<thead>
<tr>
<th>Bits 3 to 0 (Hex.)</th>
<th>TxC</th>
<th>RxC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Input: 1 x baud rate clock for the transmitter</td>
<td>Input: 1 x baud rate clock for the receiver</td>
</tr>
<tr>
<td>1, 2</td>
<td>Input: 32 x or 64 x baud rate for transmitter and receiver</td>
<td>Output: receiver bit clock with a low-to-high transition at data bit sampling time. Otherwise: high level</td>
</tr>
<tr>
<td>3 to F</td>
<td>Output: baud rate clock of the transmitter</td>
<td>Output: as above</td>
</tr>
</tbody>
</table>
As an output, RxC outputs a low-to-high transition at sampling time of every data bit of a character. Thus, data can be loaded, e.g., into a shift register externally. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

**C0, C1 — System Clock Prescaler (Bits 4, 5)**

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

<table>
<thead>
<tr>
<th>C1</th>
<th>C0</th>
<th>Divider Ratio</th>
<th>Clock at Pin CLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5.12 MHz</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3.072 MHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2.048 MHz</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.024 MHz</td>
</tr>
</tbody>
</table>

**EP — Even Parity (Bit 6)**
EP = 0: Odd parity  
EP = 1: Even parity

**PEN — Parity Enable (Bit 7)**
Bit 7 enables parity generation and checking.

PEN = 0: No parity bit  
PEN = 1: Enable parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

**Command Register 3**

<table>
<thead>
<tr>
<th>SET</th>
<th>RxE</th>
<th>IAE</th>
<th>NIW</th>
<th>END</th>
<th>SBRK</th>
<th>TBRK</th>
<th>RST</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2R)</td>
<td>(2W)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Command Register 3 is different from the first two registers because it has a bit set/reset capability. Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0-6 is low, no change occurs to that bit. When Command Register 3 is read, bits 0, 3, and 7 will always be zero.

**RST — Reset**
If RST is set, the following events occur:

1. All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.
2. The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be cancelled. Interrupt signal INT will go low.
3. The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.
4. If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does not alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has no effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

**TBRK — Transmit Break**
The transmission data output TxD will be set low as soon as the transmission of the previous character has been finished. It stays low until TBRK is cleared. The state of CTS is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be re-enabled.

**SBRK — Single Character Break**
This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle (marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.
**END — End of Interrupt**

If fully nested interrupt mode is selected, this bit reset the currently served interrupt level in the Interrupt Service Register. This command must occur at the end of each interrupt service routine during fully nested interrupt mode. END is automatically cleared when the Interrupt Service Register (internal) is cleared. END is ignored if nested interrupts are not enabled.

**NIE — Nested Interrupt Enable**

When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section of AP-153 under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

**IAE — Interrupt Acknowledge Enable**

This bit enables an automatic response to INTA. The particular response is determined by the 8086 bit in Command Register 1.

**RxE — Receive Enable**

This bit enables the serial receiver and its associated status bits in the status register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

**SET — Bit Set/Reset**

If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

**Mode Register**

<table>
<thead>
<tr>
<th>T35</th>
<th>T24</th>
<th>T5C</th>
<th>CT3</th>
<th>CT2</th>
<th>P2C2</th>
<th>P2C1</th>
<th>P2C0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3R)</td>
<td>(3W)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3 - B0 with a value $\geq 3H$.

**P2C2, P2C1, P2C0 — Port 2 Control**

<table>
<thead>
<tr>
<th>P2C2</th>
<th>P2C1</th>
<th>P2C0</th>
<th>Mode</th>
<th>Direction</th>
<th>Upper</th>
<th>Lower</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Nibble</td>
<td>Input</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Nibble</td>
<td>Input</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Nibble</td>
<td>Output</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Nibble</td>
<td>Output</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Byte</td>
<td>Handshake</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Byte</td>
<td>Handshake</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>DO NOT USE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Test</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:**

If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshake.

**CT2, CT3 — Counter/Timer Mode**

Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 36). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

**T5C — Timer 5 Control**

If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with an initial value. The first trigger pulse causes the initial value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.
T35, T24 — Cascade Timers

These two bits cascade Timers 3 and 5 or 2 and 4. Timers 2 and 3 are the lower bytes, while Timers 4 and 5 are the upper bytes. If T5C is set, then both Timers 3 and 5 can be preset and started by an external pulse.

When a high-to-low transition occurs, Timer 5 is preset to its saved value, but Timer 3 is always preset to all ones. If either CT2 or CT3 is set, then the corresponding timer pair is a 16-bit event counter.

A summary of the counter/timer control bits is given in Table 3.

**NOTE:** Interrupt levels assigned to single counters are partly not occupied if event counters/timers are cascaded. Level 2 will be vacated if event counters/timers 2 and 4 are cascaded. Likewise, Level 7 will be vacated if event counters/timers 3 and 5 are cascaded.

Single event counters/timers generate an interrupt request on the transition from 01H to 00H, while cascaded ones generate it on the transition from 0001H to 0000H.

---

**Port 1 Control Register**

<table>
<thead>
<tr>
<th>P17</th>
<th>P16</th>
<th>P15</th>
<th>P14</th>
<th>P13</th>
<th>P12</th>
<th>P11</th>
<th>P10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(4W)</td>
</tr>
</tbody>
</table>

Each bit in the Port 1 Control Register configures the direction of the corresponding pin. If the bit is high, the pin is an output, and if it low the pin is an input. Every Port 1 pin has another function which is controlled by other registers. If that special function is disabled, the pin functions as a general I/O pin as specified by this register. The special functions for each pin are described below.

**Port 10, 11 — Handshake Control**

If byte handshake control is enabled for Port 2 by the Mode Register, then Port 10 is programmed as STB/ACK handshake control input, and Port 11 is programmed as I/BF/OBF handshake control output.

If byte handshake mode is enabled for output on Port 2 IBF indicates that a character has been loaded.

---

Table 3. Event Counters/Timers Mode of Operation

<table>
<thead>
<tr>
<th>Event Counter/Timer</th>
<th>Function</th>
<th>Programming (Mode Word)</th>
<th>Clock Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8-bit timer</td>
<td>—</td>
<td>Internal clock</td>
</tr>
<tr>
<td>2</td>
<td>8-bit timer</td>
<td>T24=0, CT2=0</td>
<td>Internal clock</td>
</tr>
<tr>
<td></td>
<td>8-bit event counter</td>
<td>T24=0, CT2=1</td>
<td>P12 pin 37</td>
</tr>
<tr>
<td>2</td>
<td>8-bit timer</td>
<td>T35=0, CT3=0</td>
<td>Internal clock</td>
</tr>
<tr>
<td></td>
<td>8-bit event counter</td>
<td>T35=0, CT3=1</td>
<td>P13 pin 36</td>
</tr>
<tr>
<td>4</td>
<td>8-bit timer</td>
<td>T24=0</td>
<td>Internal clock</td>
</tr>
<tr>
<td>5</td>
<td>8-bit timer, normal mode</td>
<td>T35=0, T5C=0</td>
<td>Internal clock</td>
</tr>
<tr>
<td></td>
<td>8-bit timer, retriggerable mode</td>
<td>T35=0, T5C=1</td>
<td>Internal clock</td>
</tr>
<tr>
<td>2 and 4 cascaded</td>
<td>16-bit timer</td>
<td>T24=1, CT2=0</td>
<td>Internal clock</td>
</tr>
<tr>
<td></td>
<td>16-bit event counter</td>
<td>T24=1, CT2=1</td>
<td>P12 pin 37</td>
</tr>
<tr>
<td>3 and 5 cascaded</td>
<td>16-bit timer, normal mode</td>
<td>T35=1, T5C=0, CT3=0</td>
<td>Internal clock</td>
</tr>
<tr>
<td></td>
<td>16-bit event counter, normal mode</td>
<td>T35=1, T5C=0, CT3=1</td>
<td>P13 pin 36</td>
</tr>
<tr>
<td></td>
<td>16-bit timer, retriggerable mode</td>
<td>T35=1, T5C=1, CT3=0</td>
<td>Internal clock</td>
</tr>
<tr>
<td></td>
<td>16-bit event counter, retriggerable mode</td>
<td>T35=1, T5C=1, CT3=1</td>
<td>P13 pin 36</td>
</tr>
</tbody>
</table>
into the Port 2 output buffer. When an external device reads the data, it acknowledges this operation by driving ACK low. OBF is set low by writing to Port 2 and is reset by ACK.

If byte handshake mode is enabled for input on Port 2, STB is an input. IBF is driven low after STB goes low. On the rising edge of STB the data from Port 2 is latched.

IBF is reset high when Port 2 is read.

**Port 12, 13 — Counter 2, 3 Input**

If Timer 2 or Timer 3 is programmed as an event counter by the Mode Register, then Port 12 or Port 13 is the counter input for Event Counter 2 or 3, respectively.

**Port 14 — Baud Rate Generator Output Clock**

If test mode is enabled by the Mode Register and Command Register 2 baud rate select is greater than 2, then Port 14 is an output from the internal baud rate generator.

P14 in Port 1 control register must be set to 1 for the baud rate generator clock to be output. The baud rate generator clock is 64 x the serial bit rate except at 19.2Kbps when it is 32 x the bit rate.

**Port 15 — Timer 5 Trigger**

If T5C is set in the Mode Register enabling a retriggerable timer, then Port 15 is the input which starts and reloads Timer 5.

A high-to-low transition on P15 (Pin 34) loads the timer with the save register and starts the timer.

**Port 16 — Break-In Detect**

If Break-In Detect is enabled by BRKI in Command Register 1, then this input is used to sense a Break-In. If Port 16 is low while the serial transmitter is sending the last stop bit, then a Break-In condition is signaled.

**Port 17 — Port Interrupt Source**

If BITI in Command Register 1 is set, then a low-to-high transition on Port 17 generates an interrupt request on Priority Level 1.

Port 17 is edge triggered.

### Interrupt Enable Register

<table>
<thead>
<tr>
<th>L7</th>
<th>L6</th>
<th>L5</th>
<th>L4</th>
<th>L3</th>
<th>L2</th>
<th>L1</th>
<th>L0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5W)</td>
<td>(5W=enable, (6W=disable)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Interrupts are enabled by writing to the Set Interrupts Register (5W). Interrupts are disabled by writing to the Reset Interrupts Register (6W). Each bit set by the Set Interrupts Register (5W) will enable that level interrupt, and each bit set in the Reset Interrupts Register (6W) will disable that level interrupt. The user can determine which interrupts are enabled by reading the Interrupt enable Register (5R).

#### Priority Source

<table>
<thead>
<tr>
<th>Priority</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td>L0 Timer 1</td>
</tr>
<tr>
<td></td>
<td>L1 Timer 2 or Port Interrupt</td>
</tr>
<tr>
<td></td>
<td>L2 External Interrupt (EXTINT)</td>
</tr>
<tr>
<td></td>
<td>L3 Timer 3 or Timers 3 &amp; 5</td>
</tr>
<tr>
<td></td>
<td>L4 Receiver Interrupt</td>
</tr>
<tr>
<td></td>
<td>L5 Transmitter Interrupt</td>
</tr>
<tr>
<td></td>
<td>L6 Timer 4 or Timers 2 &amp; 4</td>
</tr>
<tr>
<td>Lowest</td>
<td>L7 Timer 5 or Port 2 Handshaking</td>
</tr>
</tbody>
</table>

### Interrupt Address Register

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>
| (6R) | (Interrupt Level Indication)

Reading the interrupt address register transfers an identifier for the currently requested interrupt level on the system data bus. This identifier is the number of the interrupt level multiplied by 4. It can be used by the CPU as an offset address for interrupt handling. Reading the interrupt address register has the same effect as a hardware interrupt acknowledge INTA; it clears the interrupt request pin (INT) and indicates an interrupt acknowledgement to the interrupt controller.

### Receiver and Transmitter Buffer

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(7R)</td>
<td>(7W)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Both the receiver and transmitter in the MUART are double buffered. This means that the transmitter and receiver have a shift register and a buffer register. The buffer registers are directly addressable by reading or writing to register seven. After the receiver buffer is full, the RBF bit in the status register is set.
Reading the receive buffer clears the RBF status bit. The transmit buffer should be written to only if the TBE bit in the status register is set. Bytes written to the transmit buffer are held there until the transmit shift register is empty, assuming CTS is low. If the transmit buffer and shift register are empty, writing to the transmit buffer immediately transfers the byte to the transmit shift register. If a serial character length is less than 8 bits, the unused most significant bits are set to zero when reading the receive buffer, and are ignored when writing to the transmit buffer.

Port 1

```
D7  D6  D5  D4  D3  D2  D1  D0  
(8R) (8W)
```

Writing to Port 1 sets the data in the Port 1 output latch. Writing to an input pin does not affect the pin, but the data is stored and will be output if the direction of the pin is changed later. If the pin is used as a control signal, the pin will not be affected, but the data is stored. Reading Port 1 transfers the data in Port 1 onto the data bus.

Port 2

```
D7  D6  D5  D4  D3  D2  D1  D0  
(9R) (9W)
```

Writing to Port 2 sets the data in the Port 2 output latch. Writing to an input pin does not affect the pin, but it does store the data in the latch. Reading Port 2 puts the input pins onto the bus or the contents of the output latch for output pins.

Timer 1-5

```
D7  D6  D5  D4  D3  D2  D1  D0  
(0A_{16} - 0E_{16} R) (0A_{16} - 0E_{16} W)
```

Reading Timer N puts the contents of the timer onto the data bus. If the counter changes while RD is low, the value on the data bus will not change. If two timers are cascaded, reading the high-order byte will cause the low-order byte to be latched. Reading the low-order byte will un latch them both. Writing to either timer or decasing them also clears the latch condition. Writing to a timer sets the starting value of that timer. If two timers are cascaded, writing to the high-order byte presets the low-order byte to all ones. Loading only the high-order byte with a value of X leads to a count of X * 256 + 255. Timers count down continuously. If the interrupt is enabled, it occurs when the counter changes from 1 to 0.

The timer/counter interrupts are automatically disabled when the interrupt request is generated.

Status Register

```
INT  RBF  TBE  TRE  BD  PE  OE  FE  
(0F_{16} R)
```

Reading the status register gates its contents onto the data bus. It holds the operational status of the serial interface as well as the status of the interrupt pin INT. The status register can be read at any time. The flags are stable and well defined at all instants.

**FE — Framing Error, Transmission Mode**

Bit 0 can be used in two modes. Normally, FE indicates framing error which can be changed to transmission mode indication by setting the TME bit in the modification register.

If transmission mode is disabled (in Modification Register), then FE indicates a framing error. A framing error is detected during the first stop bit. The error is reset by reading the Status Register or by a chip reset. A framing error does not inhibit the loading of the Receiver Buffer. If RxD remains low, the receiver will assemble the next character. The false stop bit is treated as the next start bit, and no high-to-low transition on RxD is required to synchronize the receiver.

When the TME bit in the Modification Register is set, FE is used to indicate that the transmitter was active during the reception of a character, thus indicating that the character received was transmitted by its own transmitter. FE is reset when the transmitter is not active during the reception of character. Reading the status register will not reset the FE bit in the transmission mode.

**OE — Overrun Error**

If the user does not read the character in the Receiver Buffer before the next character is received and transferred to this register, then the OE bit is set. The OE flag is set during the reception of the first stop bit and is cleared when the Status Register is read or when a hardware or software reset occurs. The first character received in this case will be lost.
PE — Parity Error

This bit indicates that a parity error has occurred during the reception of a character. A parity error is present if value of the parity bit in the received character is different from the one expected according to command word 2 bits 6 EP. The parity bit is expected and checked only if it is enabled by command word 2 bit 7 PEN.

A parity error is set during the first stop bit and is reset by reading the Status Register or by a chip reset.

BD — Break/Break-In

The BD bit flags whether a break character has been received, or a Break-In condition exists on the transmission line. Command Register 1 Bit 3 (BRKI) enables the Break-In Detect function.

Whenever a break character has been received, Status Register Bit 3 will be set and in addition an interrupt request on Level 4 is generated. The receiver will be idled. It will be started again with the next high-to-low transition at pin RxD.

The break character received will not be loaded into the receiver buffer register.

If Break-In Detection is enabled and a Break-In condition occurs, Status Register Bit 3 will be set and in addition an interrupt request on Level 5 is generated.

The BD status bit will be reset on reading the status register or on a hardware or software reset. For more information on Break/Break-In, refer to the "Serial Asynchronous Communication" section of AP-153 under "Receive Break Detect" and "Break-In Detect."

TRE — Transmit Register Empty

When TRE is set the transmit register is empty and an interrupt request is generated on Level 5 if enabled. When TRE equals 0 the transmit register is in the process of sending data. TRE is set by a chip reset and when the last stop bit has left the transmitter. It is reset when a character is loaded into the Transmitter Register. If CTS is low, the Transmitter Register will be loaded during the transmission of the start bit. If CTS is high at the end of a character, TRE will remain high and no character will be loaded into the Transmitter Register until CTS goes low. If the transmitter was inactive before a character is loaded into the Transmitter Buffer, the Transmitter Register will be empty temporarily while the buffer is full. However, the data in the buffer will be transferred to the transmitter register immediately and TRE will be cleared while TBE is set.

TBE — Transmitter Buffer Empty

TBE indicates the Transmitter Buffer is empty and is ready to accept a character. TBE is set by a chip reset or the transfer of data to the Transmitter Register, and is cleared when a character is written to the transmitter buffer. When TBE is set, an interrupt request is generated on Level 5 if enabled.

RBF — Receiver Buffer Full

RBF is set when the Receiver Buffer has been loaded with a new character during the sampling of the first stop bit. RBF is cleared by reading the receiver buffer or by a chip reset.

INT — Interrupt Pending

The INT bit reflects the state of the INT Pin (Pin 15) and indicates an interrupt is pending. It is reset by INTA or by reading the Interrupt Address Register if only one interrupt is pending and by a chip reset.

FE, OE, PE, RBF, and Break Detect all generate a Level 4 interrupt when the receiver samples the first stop bit. TRE, TBE, and Break-In Detect generate a Level 5 interrupt. TRE generates an interrupt when TBE is set and the Transmitter Register finished transmitting. The Break-In Detect interrupt is issued at the same time as TBE or TRE.

Modification Register

DSC — Disable Start Bit Check

DSC disables the receiver's start bit check. In this state the receiver will not be reset if RxD is not low at the center of the start bit.

TME — Transmission Mode Enable

TME enables transmission mode and disables framing error detection. For information on transmission mode see the description of the framing error bit in the Status Register.

RS0, RS1, RS2, RS3, RS4 — Receiver Sample Time

The number in RSn alters when the receiver samples RxD. The receiver sample time can be modified only if the receiver is not clocked by RxC.
NOTE: The modification register cannot be read. Reading from address 0FH, 8086: 1EH gates the contents of the status register onto the data bus.

A hardware reset (reset, Pin 12) resets all modification register bits to 0, i.e.:  
- The start bit check is enabled.  
- Status Register Bit 0 (FE) indicates framing error.  
- The sampling time of the serial receiver is the bit center.

A software reset (Command Word 3, RST) does not affect the modification register.

**Hardware Reset**  

A reset signal on pin RESET (HIGH level) forces the device 8256 into a well-defined initial state. This state is characterized as follows:

1. Command registers 1, 2 and 3, mode register, Port 1 control register, and modification register are reset. Thus, all bits of the parallel interface are set to be inputs and event counters/timers are configured as independent 8-bit timers.

2. Status register bits are reset with the exception of bits 4 and 5. Bits 4 and 5 are set indicating that both transmitter register and transmitter buffer register are empty.

3. The interrupt mask, interrupt request, and interrupt service register bits are reset and disable all requests. As a consequence, interrupt signal INT IS INACTIVE (LOW).

4. The transmit data output is set to the marking state (HIGH) and the receiver section is disabled until it is enabled by Command Register 3 Bit 6.

5. The start bit will be checked at sampling time. The receiver will return to start bit search mode if input RxD is not LOW at this time.

6. Status Register Bit 0 implies framing error.

7. The receiver samples input RxD at bit center.

<table>
<thead>
<tr>
<th>RS4</th>
<th>RS3</th>
<th>RS2</th>
<th>RS1</th>
<th>RS0</th>
<th>Point of time between start of bit and end of bit measured in steps of 1/32 bit length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1 (Start of Bit)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16 (Bit center)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32 (End of Bit)</td>
</tr>
</tbody>
</table>
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias: 0°C to 70°C
Storage Temperature: -65°C to +150°C
Voltage On Any Pin With Respect to ground: -0.5V to +7V
Power Dissipation: 1 Watt

*NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_a = 0°C to 70°C, V_cc = +5.0V ± 10%)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_il</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_ih</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>V_Cc + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>v_oL</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td>V</td>
<td></td>
<td>I_OH = 2.5 mA</td>
</tr>
<tr>
<td>V_oh</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td>V</td>
<td></td>
<td>I_OH = -400 μA</td>
</tr>
<tr>
<td>I_il</td>
<td>Input Leakage</td>
<td>10</td>
<td>-10</td>
<td>μA</td>
<td>V_IN = V_Cc</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>V_IN = 0V</td>
</tr>
<tr>
<td>I_oL</td>
<td>Output Leakage</td>
<td>10</td>
<td>-10</td>
<td>μA</td>
<td>V_OUT = V_Cc</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>V_OUT = 0.45V</td>
</tr>
<tr>
<td>I_cc</td>
<td>V_CC Supply Current</td>
<td>160</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CAPACITANCE (T_a = 25°C, V_cc = GND = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_in</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td></td>
<td>f_C = 1 MHz</td>
</tr>
<tr>
<td>C_iO</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
<td></td>
<td>Unmeasured pins returned to V_Ss</td>
</tr>
</tbody>
</table>
### A.C. CHARACTERISTICS

(T<sub>A</sub> = 0°C to 70°C, V<sub>cc</sub> = +5.0V ± 10%, GND = 0V)

**BUS PARAMETERS**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8256AH</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>tLL</td>
<td>ALE Pulse Width</td>
<td>50</td>
<td>ns</td>
</tr>
<tr>
<td>tCSL</td>
<td>CS to ALE Setup Time</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>tAL</td>
<td>Address to ALE Setup Time</td>
<td>20</td>
<td>ns</td>
</tr>
<tr>
<td>tLA</td>
<td>Address Hold Time After ALE</td>
<td>25</td>
<td>ns</td>
</tr>
<tr>
<td>tLC</td>
<td>ALE to RD/WR</td>
<td>20</td>
<td>ns</td>
</tr>
<tr>
<td>tCC</td>
<td>RD, WR, INTA Pulse Width</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>tRD</td>
<td>Data Valid from RD (1)</td>
<td>120</td>
<td>ns</td>
</tr>
<tr>
<td>tDF</td>
<td>Data Float After RD (2)</td>
<td>50</td>
<td>ns</td>
</tr>
<tr>
<td>tDW</td>
<td>Data Valid to WR</td>
<td>150</td>
<td>ns</td>
</tr>
<tr>
<td>tWD</td>
<td>Data Valid After WR</td>
<td>50</td>
<td>ns</td>
</tr>
<tr>
<td>tCL</td>
<td>RD/WR Control to Latch Enable</td>
<td>25</td>
<td>ns</td>
</tr>
<tr>
<td>tLDR</td>
<td>ALE to Data Valid</td>
<td>150</td>
<td>ns</td>
</tr>
<tr>
<td>tRST</td>
<td>Reset Pulse Width</td>
<td>300</td>
<td>ns</td>
</tr>
<tr>
<td>tRV</td>
<td>Recovery Time Between RD/WR</td>
<td>500</td>
<td>ns</td>
</tr>
</tbody>
</table>

**TIMER/COUNTER PARAMETERS**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8256AH</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>tCPI</td>
<td>Counter Input Cycle Time (P12, P13)</td>
<td>2.2</td>
<td>µs</td>
</tr>
<tr>
<td>tCPWH</td>
<td>Counter Input Pulse Width High</td>
<td>1.1</td>
<td>µs</td>
</tr>
<tr>
<td>tCPWL</td>
<td>Counter Input Pulse Width Low</td>
<td>1.1</td>
<td>µs</td>
</tr>
<tr>
<td>tTPI</td>
<td>Counter Input* to INTI at Terminal Count</td>
<td>2.75</td>
<td>µs</td>
</tr>
<tr>
<td>tTIH</td>
<td>LOAD Pulse High Time Counter 5</td>
<td>1.1</td>
<td>µs</td>
</tr>
<tr>
<td>tTIL</td>
<td>LOAD Pulse Low Time Counter 5</td>
<td>1.1</td>
<td>µs</td>
</tr>
<tr>
<td>tPP</td>
<td>Counter 5 Load Before Next Clock Pulse on P13</td>
<td>1.1</td>
<td>µs</td>
</tr>
<tr>
<td>tCR</td>
<td>External Count Clock* to RD† to Ensure Clock is Reflected in Count</td>
<td>2.2</td>
<td>µs</td>
</tr>
<tr>
<td>tRC</td>
<td>RD† to External Count Clock* to Ensure Clock not Reflected in Count</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>tCW</td>
<td>External Count Clock† to WR† to Ensure Count Written is Not Decremented</td>
<td>2.2</td>
<td>µs</td>
</tr>
<tr>
<td>tWC</td>
<td>WR† to External Count Clock to Ensure Count Written is Decremented</td>
<td>0</td>
<td>ns</td>
</tr>
</tbody>
</table>

**INTERRUPT PARAMETERS**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8256AH</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>tDEX</td>
<td>EXTINT† to INT†</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>tDPI</td>
<td>Interrupt request on P17† to INT†</td>
<td>2tCY+</td>
<td>ns</td>
</tr>
<tr>
<td>tPI</td>
<td>Pulse Width of Interrupt Request on P17</td>
<td>tCY+</td>
<td>ns</td>
</tr>
<tr>
<td>tHEA</td>
<td>INTA† or RD† to EXTINT†</td>
<td>30</td>
<td>ns</td>
</tr>
<tr>
<td>tHIA</td>
<td>INTA† or RD† to INT†</td>
<td>300</td>
<td>ns</td>
</tr>
</tbody>
</table>
A.C. CHARACTERISTICS (continued)

SERIAL INTERFACE AND CLOCK PARAMETERS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8256AH</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>tCY</td>
<td>Clock Period</td>
<td>195</td>
<td>1000</td>
</tr>
<tr>
<td>tCLKH</td>
<td>Clock High Pulse Width</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>tCLKL</td>
<td>Clock Low Pulse Width</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>tR</td>
<td>Clock Rise Time</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>tF</td>
<td>Clock Fall Time</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>tSCY</td>
<td>Serial Clock Period (4)</td>
<td>975</td>
<td></td>
</tr>
<tr>
<td>tSPD</td>
<td>Serial Clock High (4)</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>tSPW</td>
<td>Serial Clock Low (4)</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>tSTD</td>
<td>Internal Status Update Delay From Center of Stop Bit (5)</td>
<td>300</td>
<td>ns</td>
</tr>
<tr>
<td>tDTX</td>
<td>TxC to TxD Data Valid</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>tRBF</td>
<td>INT Delay From Center of First Stop Bit</td>
<td>2tCY+500</td>
<td></td>
</tr>
<tr>
<td>tTBE</td>
<td>INT Delay From Falling Edge of Transmit Clock at end of Start Bit</td>
<td>2tCY+500</td>
<td>ns</td>
</tr>
<tr>
<td>tCTS</td>
<td>Pulse Width for Single Character Transmission</td>
<td>(6)</td>
<td></td>
</tr>
</tbody>
</table>

PARALLEL I/O PORT PARAMETERS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8256AH</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWP</td>
<td>WR \uparrow to P1/P2 Data Valid</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>tPR</td>
<td>P1/P2 Data Stable Before RD \downarrow (7)</td>
<td>300</td>
<td>ns</td>
</tr>
<tr>
<td>tRP</td>
<td>P1/P2 Data Hold Time</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>tAK</td>
<td>ACK Pulse Width</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>tST</td>
<td>Strobe Pulse Width</td>
<td>tSIB</td>
<td></td>
</tr>
<tr>
<td>tPS</td>
<td>Data Setup to STB \uparrow</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>tPH</td>
<td>Data Hold After STB \uparrow</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>tWOB</td>
<td>WR \uparrow to OBF \uparrow</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>tAOB</td>
<td>ACK \uparrow to OBF \downarrow</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>tSIB</td>
<td>STB \uparrow to IBF \downarrow</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>tRI</td>
<td>RD \uparrow to IBF \uparrow</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>tSIT</td>
<td>STB \uparrow to INT \uparrow</td>
<td>2tCY+500</td>
<td></td>
</tr>
<tr>
<td>tAIT</td>
<td>ACK \uparrow to INT \uparrow</td>
<td>2tCY+500</td>
<td></td>
</tr>
<tr>
<td>tAED</td>
<td>OBF \downarrow to ACK \downarrow</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

NOTES:

1. $C_L = \mu F$ all outputs.
2. Measured from logic "one" or "zero" to 1.5V at $C_L = 150 \mu F$.
3. P12, P13 are external clock inputs.
4. Note that RXC may be used as an input only in 1X mode, otherwise it will be an output.
5. The center of the Stop Bit will be the receiver sample time, as programmed by the modification register.
6. 1/16th bit length for 32X, 64X; 100 ns for 1X.
7. To ensure $t_{RB}$ spec is met.
WAVEFORMS

A.C. TESTING INPUT, OUTPUT WAVEFORM

INPUT/OUTPUT

2.4

2.0

2.0

TEST POINTS

0.45

0.8

0.8

NOTES:
A.C. testing: inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0". timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

C_L = 150 pF

NOTES:
C_L includes jig capacitance

SYSTEM CLOCK

WRITE CYCLE

READ CYCLE
WAVEFORMS (Continued)

PARALLEL PORT HANDSHAKING - INPUT MODE

PARALLEL PORT HANDSHAKING - OUTPUT MODE
COUNT PULSE TIMINGS

P12 - P13 (COUNTER INPUT)

INT

LOADING TIMER (OR CASCADED COUNTER/TIMER 3 AND 5)

P13 (COUNTER INPUT)
P15 (COUNTER INPUT)

INT

TRIGGER PULSE FOR TIMER 5 (CASCADED EVENT COUNTER/TIMER 3 AND 5)

P15 (TRIGGER INPUT)

COUNTER TIMER TIMING

EXTERNAL CLOCK (P12, P13)

RD

WR

OUTPUT FROM PORT 1 AND PORT 2

DB0-7

A0-3

WR

OUTPUT

P10-17, P20-27

DATA VALID

5-336
INPUT FROM PORT 1 AND PORT 2

INPUT
P10-17, P20-27

RD

DB_0-7
A_0-3

DATA VALID

INTERRUPT TIMING

INTERRUPT FROM P17

EXTINT

INT

INTA OR RD

DB_0-7
A_0-3

DATA

CTS FOR SINGLE CHARACTER TRANSMISSION

CTS

RESET TIMING

RESET

EXTERNAL BAUD RATE CLOCK FOR SERIAL INTERFACE

TxC

(64 x AND 32
BAUD RATE INPUT

t_spw

SPD

t_scv

5-337
230759-002
TRANSMITTER AND RECEIVER CLOCK FROM INTERNAL CLOCK SOURCE

\[ T_{CCY} = \frac{1}{2} \left( \frac{1}{TAU} \right) \]

TRANSMISSION OF CHARACTERS ON SERIAL INTERFACE

NOTES:
1. Load transmitter buffer register.
2. Transmitter buffer register is empty.
3. Transmitter register is empty.
4. Character format for this example: 7 Data Bits with Parity Bit and 2 Stop Bits.
5. Loading of transmitter buffer register must be complete before CTS goes low.
6. Interrupt due to transmitter buffer register empty.
7. Interrupt due to transmitter register empty.

No Status bits are altered when RD is active.

DATA BIT OUTPUT ON SERIAL INTERFACE
CONTINUOUS RECEPTION OF CHARACTERS ON SERIAL INTERFACE WITHOUT ERROR CONDITION

NOTES:
1. Character format for this example: 6 data bits with parity bit and one stop bit.
2. Set or reset bit 6 of command register 3 (enable receiver).
3. Receiver buffer located.
4. Read receiver buffer register.

ERROR CONDITIONS DURING RECEPTION OF CHARACTERS ON THE SERIAL INTERFACE

NOTES:
1. Character format for this example: 6 data bits without parity and one stop bit.
2. Receiver buffer register loaded.
3. Overrun error.
4. Framing error.
5. Interrupt from receiver buffer register loading.
6. Interrupt from overrun error.
7. Interrupt from framing error and loading receiver buffer register.

No status bits are altered when RD is active.
8279/8279-5
PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

- Simultaneous Keyboard Display Operations
- Scanned Keyboard Mode
- Scanned Sensor Mode
- Strobed Input Entry Mode
- 8-Character Keyboard FIFO
- 2-Key Lockout or N-Key Rollover with Contact Debounce
- Dual 8- or 16-Numerical Display
- Single 16-Character Display
- Right or Left Entry 16-Byte Display RAM
- Mode Programmable from CPU
- Programmable Scan Timing
- Interrupt Output on Key Entry
- Available in EXPRESS
  — Standard Temperature Range
  — Extended Temperature Range

The Intel® 8279 is a general purpose programmable keyboard and display I/O interface device designed for use with Intel® microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard portion will also interface to an array of sensors or a strobed interface keyboard, such as the hall effect and ferrite variety. Key depressions can be 2-key lockout or N-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO. If more than 8 characters are entered, overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16X8 display RAM which can be organized into dual 16X4. The RAM can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto-increment of the display RAM address.

Figure 1. Logic Symbol

Figure 2. Pin Configuration
HARDWARE DESCRIPTION

The 8279 is packaged in a 40 pin DIP. The following is a functional description of each pin.

### Table 1. Pin Descriptions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB0-DB7</td>
<td>19-12</td>
<td>Bi-directional data bus: All data and commands between the CPU and the 8279 are transmitted on these lines.</td>
</tr>
<tr>
<td>CLK</td>
<td>3</td>
<td>Clock: Clock from system used to generate internal timing.</td>
</tr>
<tr>
<td>RESET</td>
<td>9</td>
<td>Reset: A high signal on this pin resets the 8279. After being reset the 8279 is placed in the following mode: 1) 16 8-bit character display —left entry. 2) Encoded scan keyboard —2 key lockout. Along with this the program clock prescaler is set to 31.</td>
</tr>
<tr>
<td>CS</td>
<td>22</td>
<td>Chip Select: A low on this pin enables the interface functions to receive or transmit.</td>
</tr>
<tr>
<td>A0</td>
<td>21</td>
<td>Buffer Address: A high on this line indicates the signals in or out are interpreted as a command or status. A low indicates that they are data.</td>
</tr>
<tr>
<td>RD, WR</td>
<td>10-11</td>
<td>Input/Output Read and Write: These signals enable the data buffers to either send data to the external bus or receive it from the external bus.</td>
</tr>
<tr>
<td>IRQ</td>
<td>4</td>
<td>Interrupt Request: In a keyboard mode, the interrupt line is high when there is data in the FIFO/Sensor RAM. The interrupt line goes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor mode, the interrupt line goes high whenever a change in a sensor is detected.</td>
</tr>
<tr>
<td>VSS, VCC</td>
<td>20, 40</td>
<td>Ground and power supply pins.</td>
</tr>
<tr>
<td>SL0-SL3</td>
<td>32-35</td>
<td>Scan Lines: Scan lines which are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 of 16) or decoded (1 of 4).</td>
</tr>
<tr>
<td>RL0-RL7</td>
<td>38, 39, 1, 2, 5-8</td>
<td>Return Line: Return line inputs which are connected to the scan lines through the keys or sensor switches. They have active internal pullups to keep them high until a switch closure pulls one low. They also serve as an 8-bit input in the Strobed Input mode.</td>
</tr>
</tbody>
</table>

### FUNCTIONAL DESCRIPTION

Since data input and display are an integral part of many microprocessor designs, the system designer needs an interface that can control these functions without placing a large load on the CPU. The 8279 provides this function for 8-bit microprocessors.

The 8279 has two sections: keyboard and display. The keyboard section can interface to regular typewriter style keyboards or random toggle or thumb switches. The display section drives alphanumeric displays or a bank of indicator lights. Thus the CPU is relieved from scanning the keyboard or refreshing the display.

The 8279 is designed to directly connect to the microprocessor bus. The CPU can program all operating modes for the 8279. These modes include:
Input Modes

- Scanned Keyboard — with encoded (8 x 8 key keyboard) or decoded (4 x 8 key keyboard) scan lines. A key depression generates a 6-bit encoding of key position. Position and shift and control status are stored in the FIFO. Keys are automatically debounced with 2-key lockout or N-key rollover.
- Scanned Sensor Matrix — with encoded (8 x 8 matrix switches) or decoded (4 x 8 matrix switches) scan lines. Key status (open or closed) stored in RAM addressable by CPU.
- Strobed Input — Data on return lines during control line strobe is transferred to FIFO.

Output Modes

- 8 or 16 character multiplexed displays that can be organized as dual 4-bit or single 8-bit (B0 = D0, A3 = D7).
- Right entry or left entry display formats.

Other features of the 8279 include:

- Mode programming from the CPU.
- Clock Prescaler
- Interrupt output to signal CPU when there is keyboard or sensor data available.
- An 8 byte FIFO to store keyboard information.
- 16 byte internal Display RAM for display refresh. This RAM can also be read by the CPU.

PRINCIPLES OF OPERATION

The following is a description of the major elements of the 8279 Programmable Keyboard/Display interface device. Refer to the block diagram in Figure 3.

I/O Control and Data Buffers

The I/O control section uses the CS, A0, RD and WR lines to control data flow to and from the various internal registers and buffers. All data flow to and from the 8279 is enabled by CS. The character of the information, given or desired by the CPU, is identified by A0. A logic one means the information is a command or status. A logic zero means the information is data. RD and WR determine the direction of data flow through the Data Buffers. The Data Buffers are bi-directional buffers that connect the internal bus to the external bus. When the chip is not selected (CS = 1), the devices are in a high impedance state. The drivers input during WR • CS and output during RD • CS.

Control and Timing Registers and Timing Control

These registers store the keyboard and display modes and other operating conditions programmed by the CPU. The modes are programmed by presenting the proper command on the data lines with A0 = 1 and then sending a WR. The command is latched on the rising edge of WR.
SOFTWARE OPERATION

8279 commands

The following commands program the 8279 operating modes. The commands are sent on the Data Bus with CS low and A0 high and are loaded to the 8279 on the rising edge of WR.

Keyboard/Display Mode Set

<table>
<thead>
<tr>
<th>Code:</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0</td>
<td>D D K K</td>
</tr>
</tbody>
</table>

Where DD is the Display Mode and KKK is the Keyboard Mode.

Keyboard - 2 Key Lockout

<table>
<thead>
<tr>
<th>Code:</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>P</td>
<td>P P P P</td>
</tr>
</tbody>
</table>

All timing and multiplexing signals for the 8279 are generated by an internal prescaler. This prescaler divides the external clock (pin 3) by a programmable integer. Bits PPPP determine the value of this integer which ranges from 2 to 31. Choosing a divisor that yields 100 kHz will give the specified scan and debounce times. For instance, if Pin 3 of the 8279 is being clocked by a 2 MHz signal, PPPP should be set to 10100 to divide the clock by 20 to yield the proper 100 kHz operating frequency.

Read FIFO/Sensor RAM

<table>
<thead>
<tr>
<th>Code:</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0</td>
<td>A I X A A A</td>
<td>X = Don’t Care</td>
</tr>
</tbody>
</table>

The CPU sets up the 8279 for a read of the FIFO/Sensor RAM by first writing this command. In the Scan Key-
board Mode, the Auto-Increment flag (AI) and the RAM address bits (AAA) are irrelevant. The 8279 will automatically drive the data bus for each subsequent read (A0 = 0) in the same sequence in which the data first entered the FIFO. All subsequent reads will be from the FIFO until another command is issued.

In the Sensor Matrix Mode, the RAM address bits AAA select one of the 8 rows of the Sensor RAM. If the AI flag is set (AI = 1), each successive read will be from the subsequent row of the sensor RAM.

Read Display RAM

Code: 0 1 1 A A A A

The CPU sets up the 8279 for a read of the Display RAM by first writing this command. The address bits AAAA select one of the 16 rows of the Display RAM. If the AI flag is set (AI = 1), this row address will be incremented after each following read or write to the Display RAM.

Since the same counter is used for both reading and writing, this command sets the next read or write address and the sense of the Auto-Increment mode for both operations.

Write Display RAM

Code: 1 0 0 A A A A

The CPU sets up the 8279 for a write to the Display RAM by first writing this command. After writing the command with A0 = 1, all subsequent writes with A0 = 0 will be to the Display RAM. The addressing and Auto-Increment functions are identical to those for the Read Display RAM. However, this command does not affect the source of subsequent Data Reads; the CPU will read from whichever RAM (Display or FIFO/Sensor) which was last specified. If, indeed, the Display RAM was last specified, the Write Display RAM will, nevertheless, change the next Read location.

Display Write Inhibit/Blanking

A B A B

Code: 1 0 1 X IW IW BL BL

The IW Bits can be used to mask nibble A and nibble B in applications requiring separate 4-bit display ports. By setting the IW flag (IW = 1) for one of the ports, the port becomes marked so that entries to the Display RAM from the CPU do not affect that port. Thus, if each nibble is input to a BCD decoder, the CPU may write a digit to the Display RAM without affecting the other digit being displayed. It is important to note that bit B7 corresponds to bit D0 on the CPU bus, and that bit A3 corresponds to bit D7.

If the user wishes to blank the display, the BL flags are available for each nibble. The last Clear command issued determines the code to be used as a "blank." This code defaults to all zeros after a reset. Note that both BL flags must be set to blank a display formatted with a single 8-bit port.

Clear

Code: 1 1 0 C0 C0 C0 CF CA

The C0 bits are available in this command to clear all rows of the Display RAM to a selectable blanking code as follows:

<table>
<thead>
<tr>
<th>C0</th>
<th>C0</th>
<th>C0</th>
<th>CF</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>X</td>
<td>All Zeros (X = Don’t Care)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>AB = Hex 20 (0010 0000)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>AB = Hex 20 (0010 0000)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>AB = Hex 20 (0010 0000)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

During the time the Display RAM is being cleared (∼180 µs), it may not be written to. The most significant bit of the FIFO status word is set during this time. When the Display RAM becomes available again, it automatically resets.

If the CF bit is asserted (CF = 1), the FIFO status is cleared and the interrupt output line is reset. Also, the Sensor RAM pointer is set to row 0.

CA: the Clear All bit, has the combined effect of C0 and CF: it uses the C0 clearing code on the Display RAM and also clears FIFO status. Furthermore, it resynchronizes the internal timing chain.

End Interrupt/Error Mode Set

Code: 1 1 1 E X X X X

X = Don’t care.

For the sensor matrix modes this command lowers the IRQ line and enables further writing into RAM. (The IRQ line would have been raised upon the detection of a change in a sensor value. This would have also inhibited further writing into the RAM until reset).

For the N-key rollover mode — if the E bit is programmed to “1” the chip will operate in the special Error mode. (For further details, see Interface Considerations Section.)

Status Word

The status word contains the FIFO status, error, and display unavailable signals. This word is read by the CPU when A0 is high and CS and RD are low. See Interface Considerations for more detail on status word.

Data Read

Data is read when A0, CS and RD are all low. The source of the data is specified by the Read FIFO or Read Display commands. The trailing edge of RD will cause the address of the RAM being read to be incremented if the Auto-Increment flag is set. FIFO reads always increment (if no error occurs) independent of AI.

Data Write

Data that is written with A0, CS and WR low is always written to the Display RAM. The address is specified by the latest Read Display or Write Display command. Auto-Incrementing on the rising edge of WR occurs if AI set by the latest display command.

5-344
INTERFACE CONSIDERATIONS

Scanned Keyboard Mode, 2-Key Lockout

There are three possible combinations of conditions that can occur during debounce scanning. When a key is depressed, the debounce logic is set. Other depressed keys are looked for during the next two scans. If none are encountered, it is a single key depression and the key position is entered into the FIFO along with the status of CNTL and SHIFT lines. If the FIFO was empty, IRQ will be set to signal the CPU that there is an entry in the FIFO. If the FIFO was full, the key will not be entered and the error flag will be set. If another closed switch is encountered, no entry to the FIFO can occur. If all other keys are released before this one, then it will be entered to the FIFO. If this key is released before any other, it will be entirely ignored. A key is entered to the FIFO only once per depression, no matter how many keys were pressed along with it or in what order they were released. If two keys are depressed within the debounce cycle, it is a simultaneous depression. Neither key will be recognized until one key remains depressed alone. The last key will be treated as a single key depression.

Scanned Keyboard Mode, N-Key Rollover

With N-key Rollover each key depression is treated independently from all others. When a key is depressed, the debounce circuit waits 2 keyboard scans and then checks to see if the key is still down. If it is, the key is entered into the FIFO. Any number of keys can be depressed and another can be recognized and entered into the FIFO. If a simultaneous depression occurs, the keys are recognized and entered according to the order the keyboard scan found them.

Scanned Keyboard — Special Error Modes

For N-key rollover mode the user can program a special error mode. This is done by the "End Interrupt/Error Mode Set" command. The debounce cycle and key-validity check are as in normal N-key mode. If during a single debounce cycle, two keys are found depressed, this is considered a simultaneous multiple depression, and sets an error flag. This flag will prevent any further writing into the FIFO and will set interrupt (if not yet set). The error flag could be read in this mode by reading the FIFO STATUS word. (See "FIFO STATUS" for further details.) The error flag is reset by sending the normal CLEAR command with CF = 1.

Sensor Matrix Mode

In Sensor Matrix mode, the debounce logic is inhibited. The status of the sensor switch is input directly to the Sensor RAM. In this way the Sensor RAM keeps an image of the state of the switches in the sensor matrix. Although debouncing is not provided, this mode has the advantage that the CPU knows how long the sensor was closed and when it was released. A keyboard mode can only indicate a validated closure. To make the software easier, the designer should functionally group the sensors by row since this is the format in which the CPU will read them. The IRQ line goes high if any sensor value change is detected at the end of a sensor matrix scan. The IRQ line is cleared by the first data read operation if the Auto-Increment flag is set to zero, or by the End Interrupt command if the Auto-Increment flag is set to one.

Note: Multiple changes in the matrix Addressed by (SL0-3 = 0) may cause multiple interrupts. (SL0 = 0 in the Decoded Mode). Reset may cause the 8279 to see multiple changes.

Data Format

In the Scanned Keyboard mode, the character entered into the FIFO corresponds to the position of the switch in the keyboard plus the status of the CNTL and SHIFT lines (non-inverted). CNTL is the MSB of the character and SHIF is the next most significant bit. The next three bits are from the scan counter and indicate the row the key was found in. The last three bits are from the column counter and indicate to which return line the key was connected.

In Sensor Matrix mode, the data on the return lines is entered directly in the row of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each switch position maps directly to a Sensor RAM position. The SHIF and CNTL inputs are ignored in this mode. Note that switches are not necessarily the only thing that can be connected to the return lines in this mode. Any logic that can be triggered by the scan lines can enter data to the return line inputs. Eight multiplexed input ports could be tied to the return lines and scanned by the 8279.

Display

Left Entry

Left Entry mode is the simplest display format in that each display position directly corresponds to a byte (or nibble) in the Display RAM. Address 0 in the RAM is the left-most display character and address 15 (or address 7 in 8 character display) is the right most display character. Entering characters from position zero causes the display to fill from the left. The 17th (9th) character is entered back in the left most position and filling again proceeds from there.
Right Entry

Right entry is the method used by most electronic calculators. The first entry is placed in the right most display character. The next entry is also placed in the right most character after the display is shifted left one character. The left most character is shifted off the end and is lost.

Left Entry Mode (Auto Increment)

<table>
<thead>
<tr>
<th>Left Entry</th>
<th>Right Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st entry</td>
<td>1st entry</td>
</tr>
<tr>
<td>2nd entry</td>
<td>2nd entry</td>
</tr>
<tr>
<td>16th entry</td>
<td>16th entry</td>
</tr>
<tr>
<td>17th entry</td>
<td>17th entry</td>
</tr>
<tr>
<td>18th entry</td>
<td>18th entry</td>
</tr>
</tbody>
</table>

Note that now the display position and register address do not correspond. Consequently, entering a character to an arbitrary position in the Auto Increment mode may have unexpected results. Entry starting at Display RAM address 0 with sequential entry is recommended.

Auto Increment

In the Left Entry mode, Auto Incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non-Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Auto Increment mode has no undesirable side effects and the result is predictable.

Right Entry Mode (Auto Increment)

Starting at an arbitrary location operates as shown below:

<table>
<thead>
<tr>
<th>Command 10010101</th>
<th>1st entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter next at Location 5 Auto Increment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command 10010101</th>
<th>2nd entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter next at Location 5 Auto Increment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command 10010101</th>
<th>3rd entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter next at Location 5 Auto Increment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command 10010101</th>
<th>4th entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter next at Location 5 Auto Increment</td>
<td></td>
</tr>
</tbody>
</table>

5-346
Entry appears to be from the initial entry point.

8/16 Character Display Formats
If the display mode is set to an 8 character display, the on duty-cycle is double what it would be for a 16 character display (e.g., 5.1 ms scan time for 8 characters vs. 10.3 ms for 16 characters with 100 kHz internal frequency).

G. FIFO Status
FIFO status is used in the Keyboard and Strobed Input modes to indicate the number of characters in the FIFO and to indicate whether an error has occurred. There are two types of errors possible: overrun and underrun. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO.

The FIFO status word also has a bit to indicate that the Display RAM was unavailable because a Clear Display or Clear All command had not completed its clearing operation.

In a Sensor Matrix mode, a bit is set in the FIFO status word to indicate that at least one sensor closure indication is contained in the Sensor RAM.

In Special Error Mode the S/E bit is showing the error flag and serves as an indication to whether a simultaneous multiple closure error has occurred.

FIFO STATUS WORD

<table>
<thead>
<tr>
<th>D0</th>
<th>S/E</th>
<th>O</th>
<th>F</th>
<th>N</th>
<th>N</th>
</tr>
</thead>
</table>

- Number of characters in FIFO
- Error-Underrun
- Error-Overrun
- Sensor Closure/Error Flag for Multiple Closures
- Display unavailable

*Do not drive the keyboard decoder with the MSB of the scan lines.

Figure 4. System Block Diagram
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature .......................... 0°C to 70°C
Storage Temperature ......................... -65°C to 125°C
Voltage on any Pin with Respect to Ground ........... -0.5V to +7V
Power Dissipation .............................. 1 Watt

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS \([T_A = 0^\circ C \text{ to } 70^\circ C, V_{SS} = 0V, \text{(Note 3)}]\)*

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V_{IL1})</td>
<td>Input Low Voltage for Return Lines</td>
<td>-0.5</td>
<td>1.4</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>(V_{IL2})</td>
<td>Input Low Voltage for All Others</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>(V_{IH1})</td>
<td>Input High Voltage for Return Lines</td>
<td>2.2</td>
<td></td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>(V_{IH2})</td>
<td>Input High Voltage for All Others</td>
<td>2.0</td>
<td></td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>(V_{OL})</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td></td>
<td>V</td>
<td>Note 1</td>
</tr>
<tr>
<td>(V_{OH1})</td>
<td>Output High Voltage on Interrupt Line</td>
<td>3.5</td>
<td></td>
<td>V</td>
<td>Note 2</td>
</tr>
<tr>
<td>(I_{IL1})</td>
<td>Input Current on Shift, Control and Return Lines</td>
<td>+10</td>
<td></td>
<td>(\mu A)</td>
<td>(V_{IN} = V_{CC})</td>
</tr>
<tr>
<td>(I_{IL2})</td>
<td>Input Leakage Current on All Others</td>
<td>±10</td>
<td></td>
<td>(\mu A)</td>
<td>(V_{IN} = V_{CC}) to 0V</td>
</tr>
<tr>
<td>(I_{OFL})</td>
<td>Output Float Leakage</td>
<td>±10</td>
<td></td>
<td>(\mu A)</td>
<td>(V_{OUT} = V_{CC}) to 0.45V</td>
</tr>
<tr>
<td>(I_{CC})</td>
<td>Power Supply Current</td>
<td>120</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
</tbody>
</table>

CAPACITANCE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C_{IN})</td>
<td>Input Capacitance</td>
<td>5</td>
<td>10</td>
<td>pF</td>
<td>(f_C = 1 \text{ MHz Unmeasured pins returned to } V_{SS})</td>
</tr>
<tr>
<td>(C_{OUT})</td>
<td>Output Capacitance</td>
<td>10</td>
<td>20</td>
<td>pF</td>
<td></td>
</tr>
</tbody>
</table>

A.C. CHARACTERISTICS \([T_A = 0^\circ C \text{ to } 70^\circ C, V_{SS} = 0V, \text{(Note 3)}]\) *

Bus Parameters

READ CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>(8279)</th>
<th>(8279-5)</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
</tr>
<tr>
<td>(t_{AR})</td>
<td>Address Stable Before READ</td>
<td>50</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{RA})</td>
<td>Address Hold Time for READ</td>
<td>5</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{RR})</td>
<td>READ Pulse Width</td>
<td>420</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{RD}[4])</td>
<td>Data Delay from READ</td>
<td>300</td>
<td>150</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{AD}[4])</td>
<td>Address to Data Valid</td>
<td>450</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{DF})</td>
<td>READ to Data Floating</td>
<td>10</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>(t_{RCY})</td>
<td>Read Cycle Time</td>
<td>1</td>
<td>1</td>
<td>(\mu s)</td>
</tr>
</tbody>
</table>
A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>8279</strong></td>
<td><strong>8279-5</strong></td>
</tr>
<tr>
<td>tAW</td>
<td>Address Stable Before WRITE</td>
<td>50 ns</td>
</tr>
<tr>
<td>tWA</td>
<td>Address Hold Time for WRITE</td>
<td>20 ns</td>
</tr>
<tr>
<td>tWW</td>
<td>WRITE Pulse Width</td>
<td>400 ns</td>
</tr>
<tr>
<td>tDW</td>
<td>Data Set Up Time for WRITE</td>
<td>300 ns</td>
</tr>
<tr>
<td>tWD</td>
<td>Data Hold Time for WRITE</td>
<td>40 ns</td>
</tr>
<tr>
<td>tWCY</td>
<td>Write Cycle Time</td>
<td>1 μs</td>
</tr>
</tbody>
</table>

OTHER TIMINGS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>8279</strong></td>
<td><strong>8279-5</strong></td>
</tr>
<tr>
<td>tCW</td>
<td>Clock Pulse Width</td>
<td>230 nsec</td>
</tr>
<tr>
<td>tCY</td>
<td>Clock Period</td>
<td>500 nsec</td>
</tr>
</tbody>
</table>

Keyboard Scan Time ......................... 5.1 msec
Keyboard Debounce Time ..................... 10.3 msec
Key Scan Time .............................. 80 μsec
Display Scan Time ........................... 10.3 msec
Digit-on Time .............................. 480 μsec
Blanking Time .............................. 160 μsec
Internal Clock Cycle[6] ..................... 10 μsec

NOTES:
1. 8279, IOL = 1.6mA; 8279-5, IOL = 2.2mA.
2. IOH = -100 μA
3. 8279, VCC = +5V ±5%; 8279-5, VCC = +5V ±10%.
4. 8279, Cl = 100pF; 8279-5, Cl = 150pF.
5. The Prescaler should be programmed to provide a 10 μs internal clock cycle.
   * For Extended Temperature EXPRESS, use M8279A electrical parameters.

A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

C_L = 120 pF

C_L includes JIG capacitance
WAVEFORMS

**READ OPERATION**

- **A0, CS**
- **RD**
- **DATA BUS (OUTPUT)**

**WRITE OPERATION**

- **A0, CS**
- **WR**
- **DATA BUS (INPUT)**

**CLOCK INPUT**

- **tCW**
- **tCY**

(System's Address Bus)

(Read Control)

(May Change)

(Data Valid)

(High Impedance)

(System's Address Bus)

(Write Control)

(Data May Change)

(Data Valid)

(Data May Change)
WAVEFORMS (Continued)

SCAN

So

S1

ENCODED SCAN

S2

S3

DECODED SCAN

S4

S5

DISPLAY

PRESCALER PROGRAMMED FOR INTERNAL FREQUENCY = 100 kHz SO

tCY = 10 µs

NOTE: SHOWN IS ENCODED SCAN LEFT ENTRY

S6-S9 ARE NOT SHOWN BUT THEY ARE SIMPLY S, DIVIDED BY 2 AND 4

RETURN LINES ARE SAMPLED ONE AT A TIME AS SHOWN.
Designing with the 8256

Charles T. Yager
Applications Engineer
Designing with the 8256

Contents

INTRODUCTION

DESCRIPTION OF THE MUART
   Microprocessor Bus Interface
   Command and Status Registers
   Clock Circuitry
      System Clock Prescaler
      Timer Prescaler
   Asynchronous Serial Interface
      Receiver Section of the UART
      Receive Break Detect
      Transmitter Section of the UART
      Transmit Break Features
      Modification Register
   Parallel I/O
      Two Wire Byte Handshake
   Event Counter/Timers
   Interrupt Controller
      MCS-85/8256 Interrupt Operation
      MCS-86/88/8256 Interrupt Operation
      Using the Interrupt Controller
         Without INTA
      Interrupt Registers
      Interrupt Modes
      Edge Triggering
      Level Triggering
      Cascading the MUART's Interrupt Controller
      Polling the MUART

PIN DESCRIPTIONS

DESCRIPTION OF REGISTERS
   Hardware Reset

INTERFACING

PROGRAMMING
   Initialization
   Operating the Serial Interface
      Transmitting
      Receiving
   Operating the Parallel Interface
      Loading Port 1 and 2
      Reading Port 1 and 2
   Operating the Event Counter/Timers
      Loading Event Counter/Timers
      Reading Event Counter/Timers
Contents (cont.)

APPLICATION EXAMPLE
Description of the Line Printer Multiplexer
Description of the Hardware
Description of the Software
Buffer Management
Using the LPM with the MDS SERIES II OR SERIES III

APPENDIX
Listing of the Line Printer Multiplexer Software
Listing of the WRITE Program
MUART Registers
INTRODUCTION

The INTEL 8256 MUART is a Multifunction Universal Asynchronous Receiver Transmitter designed to be used for serial asynchronous communication while also providing hardware support for parallel I/O, timing, counting and interrupt control. Its versatile design allows it to be directly connected to the MCS®-85, iAPX-86, iAPX-88, iAPX-186, and iAPX-188 microcomputer systems plus the MCS-48 and MCS-51 family of single-chip microcomputers.

The four commonly used peripheral functions contained in the MUART are:

1) Full-duplex, double-buffered serial asynchronous Receiver/Transmitter with an on-chip Baud Rate Generator
2) Two - 8-bit parallel I/O ports
3) Five - 8-bit counters/timers
4) 8-level priority interrupt controller

This manual can be divided into two parts. The first part describes the MUART in detail, including its functions, registers and pins. This section also describes the interface between the MUART and Intel CPUs plus a discussion on programming considerations. The second section provides an application example: a MUART-based line printer multiplexer. The Appendix contains software listings for the line printer multiplexer and some useful reference information.

DESCRIPTION OF THE MUART

The MUART can be logically partitioned into seven sections: the microprocessor bus interface, the command and status registers, clocking circuitry, asynchronous serial communication, parallel I/O, timer/event counters, and the interrupt controller. This can be seen from the block diagram of the 8256 MUART as shown in Figure 1. The MUART’s pin configuration can be seen in Figure 2.

Microprocessor Bus Interface

The microprocessor bus interface is the hardware section of the MUART which allows a μP to communicate with the MUART. It consists of tristate bi-directional data-bus buffers, an address latch, a chip select (CS) latch and bus control logic. In order to provide all of the MUART’s functions in a 40-pin DIP while retaining direct register addressing, a multiplexed address/data bus is used.

Address/Data Bus

The MUART contains 16 internal directly addressable read/write registers. Four of the eight address/data lines are used to generate the address. When using 8-bit microprocessors such as MCS-85, MCS-48 and MCS-51, AD0 – AD3 are used to address the 16 internal registers while Address/Data line 4 (AD4) is not used for addressing. For 16-bit systems, AD1 – AD4 are used to generate the address for the internal data registers and AD0 is used as a second active low chip select.

RD, WR, CS

The 8256 bus interface uses the standard bus control signals which are compatible with all Intel peripherals and microprocessors. The chip select signal (CS), typically derived from an address decoder, is latched along with the address on the falling edge of ALE. As a result, chip select does not have to remain low for the entire bus cycle. However, the data bus buffers will remain tristated unless an RD or a WR signal becomes active while chip select has been latched in low.

INT, INTA

The INT and INTA signals are used to interrupt the CPU and receive the CPU’s acknowledgment to the interrupt request. The MUART can vector the CPU to the appropriate service routine depending on the source of the interrupt.

RESET

When a high level occurs on the RESET pin, the MUART is placed in a known initial state. This initial state is described under “Hardware Reset.”

Command and Status Register

There are three command registers and one status register as shown in Figure 1. The three command registers are read/write registers while the status register is a read only. The command registers configure the MUART for its operating environment (i.e., 8 or 16 bits CPU, system clock frequency). In addition, they direct its higher level functions such as controlling the UART, selecting modes of operation for the interrupt controller, and choosing the fundamental frequency for the timers. Command Register 3 is the only register in the MUART which is a bit set/reset register, allowing the programmer to simply perform one write to set or reset any of the bits.
Figure 1. Block Diagram of the 8258 UART

The status register provides all of the information about the status of the UART's transmitter and receiver as well as the status of the interrupt pin. The status register is the only read-only register in the UART.

CLOCK CIRCUITRY

The clock for the five timers and baud rate generator is derived from the system clock. The system clock, pin 17 (CLK), is fed into a system clock prescaler which in turn feeds the five timers and the baud rate generator. The UART's system clock can be asynchronous to the microprocessor's clock.

System Clock Prescaler

The system clock prescaler is a programmable divider which normalizes the internal clocking frequency for the timers and baud rate generator to 1.024MHz. It divides the system clock (CLK) by 1, 2, 3, or 5, allowing clock frequencies of 1.024MHz, 2.048MHz, 3.072MHz, or 5.12MHz. (The commonly used 6.144MHz crystal frequency for the 8085 results in a 3.072MHz frequency from the 8085's CLK pin.) If the system clock is not one of the four frequencies mentioned above, then the frequency of the baud rate generator and the timers will be nonstandard;
however, the MUART will still run as long as the system clock meets the data sheet tcry spec.

**Timer Prescaler**

The timer prescaler permits the user to select one of two fundamental timing frequencies for all of the MUART's timers, either 1KHz or 16KHz. The frequency selection is made via Command Register 0.

**Asynchronous Serial Interface**

The asynchronous serial interface of the MUART is a full-duplex double-buffered transmitter and receiver with separate control registers. The standard asynchronous format is used as shown in Figure 3. The operation of the UART section of the MUART is very similar to the operation of the 8251A USART.

**Receiver Section of the UART**

The serial asynchronous receiver section contains a serial shift register, a receiver buffer register and receiver control logic. The serial input data is clocked into the receive shift register from the RxD pin at the specified baud rate. The sampling actually takes place at the rising edge of RxC, assuming an external clock, or at the rising edge of the internal baud clock. When the receiver is enabled but inactive, the receive logic is sampling RxD at either 32 or 64 times the bit rate, looking for a change from the Mark (high) to the Space (low) state. This is commonly referred to as the start bit search mode. When this state change occurs, the receive logic waits one half of a bit time and then samples RxD again. If RxD is still in the Space state, the receive logic begins to clock in the receive data beginning one bit period later. If RxD has returned to the Mark state (i.e., false start bit), the receive logic will return to the start bit search mode. Normally the received data is sampled in the center of each bit, however it is possible to adjust the location where the bit is sampled. This feature is controlled by the modification register.

The bit rate of the serial receive data is derived from either the internal baud rate generator or an external clock. When using an external clock, the programmer has a choice of three sampling rates: 1x, 32x, or 64x. Using the internal baud rate generator, the sampling rates are all 64x except for 19.2 Kbps which is 32x.

When the serial shift register clocks in the stop bit, an internal load pulse is generated which transfers the contents of the shift register into the receive buffer. This transfer takes place during the first half of the first stop bit. The load pulse also triggers several other signals relevant to the receive section including Receive Buffer Full (RBF), Parity Error (PE), Overrun Error (OE), and Framing Error (FE). These four status bits are updated after the middle of the first stop bit when the receive buffer has already been latched. Each one of these four status bits are latched. They are reset on the rising edge of the first read pulse (RD) addressed to the status register. A complete description of the status register is given in the section "Description of the Registers."

When the serial receiver is disabled (via bit 6 of Command Register 3) the load pulse is suppressed. The result is that the receive buffer is not loaded with the contents of the shift register and the RBF, PE, OE, and FE bits in the status register are not updated. Even though the receiver is disabled, the serial shift register will still be clocking in the data from RxD, if any. This means that the receiver will still be synchronized with the start and stop bits. For example, if the receiver is enabled via Command Register 3 in the middle of receiving a serial character, the character will still be assembled correctly. When the receiver is disabled the last character received will remain in the receive buffer. On power-up the value in the receive buffer is undefined.

![Figure 3. Asynchronous Format](image-url)
Whenever a character length of fewer than 8 bits is programmed, the most significant bits of a received character will read as zero. Also, the receiver will only check the first stop bit of any character, regardless of how many stop bits are programmed into the device.

**Receive Break Detect**

A Receive Break occurs when RxD remains in the space state for one character time, including the parity bit (if any) and the first stop bit. The MUART will set the Break Detect status bit (BD) when it receives a break. The Break Detect status bit is set after the middle of the first stop bit. If the MUART detects a break it will inhibit the receive buffer load pulse, thus the receive buffer will not be loaded with the null character, and none of the four status bits (PE, OE, FE, and RBF) will be updated. The last character received will remain in the receive buffer. A break detect state has the same effect as disabling the receiver—they both inhibit the load pulse—therefore one can think of the break status as disabling the receiver.

The Break Detect status bit is latched. It is cleared by the rising edge of the read pulse addressed to the status register. If a break occurs, and then the RxD data line returns to the Mark state before the status register is read, the BD status bit will remain set until it is read. If RxD returns to the Mark state after the BD status bit has been read true, the BD status bit will be reset automatically without reading the status register.

The receive break detect logic of the MUART is independent of whether the receiver is enabled or disabled; therefore even if the receiver is disabled the MUART will recognize a break. When the RxD line returns to the Mark state after a break, the 8256 will be in the start bit search mode.

If the receiver interrupt level is enabled, break will generate an interrupt request regardless of whether the receiver is enabled. Another receive interrupt will not be generated until the RxD pin returns to the Mark state.

**Transmitter Section of the UART**

The serial asynchronous transmitter section of the MUART consists of a transmit buffer, a transmit (shift) register, and the associated control logic. There are two bits in the status register which indicate the status of the transmit buffer and transmit register: TBE (transmit buffer empty) and TRE (transmit register empty).

To transmit a character, a byte is written to the transmit buffer. The transmit buffer should only be written to when TBE = 1. When the transmit register is empty and CTS = 0, the character will be automatically transferred from the transmit buffer into the transmit register. The data transfer from the transmit buffer to the transmit register takes place during the transmission of the start bit. After this transfer takes place, sometime at the beginning of the transmission of the first data bit, TBE is set to 1.

When the transmitter is idle, both TBE and TRE will be set to 1. After a character is written to the transmit buffer, TBE = 0 and TRE = 1. This state will remain for a short period of time, then the character will be transferred into the transmit register and the status bits will read TBE = 1 and TRE = 0. At this point a second character may be written to the transmit buffer after which TBE = 0 and TRE = 0. TBE will not be set to 1 again until the transmit register becomes empty and is reloaded with the byte in the transmit buffer.

The transmitter can be disabled only one way—using the CTS pin. When CTS = 0 the transmitter is enabled, and when CTS = 1 the transmitter is disabled. If the transmitter is idle and CTS goes from 0 to 1, disabling the transmitter, TBE and TRE will remain set to 1. Since TBE = 1 a character can be written into the transmit buffer. The character will be stored in the transmit buffer but it will not be transferred to the transmit register until CTS goes low.

If CTS goes from low to high during transmission of a character, the character in transmission will be completed and TxD will return to the Mark state. If the transmitter is full (i.e., TBE and TRE = 0), the transmit shift register will be emptied but the transmit buffer will not; therefore TBE = 0 and TRE = 1.

**Transmitter Break Features**

The MUART has three transmit break features: Break-In Detect, Transmit Break (TBRK), and Single Character Break (SBRK).

Break-In Detect - A Break-In condition occurs when the MUART is sending a serial message and the transmission line is forced to the space state by the receiving station. Break-In is usually used with half-duplex transmission so that the receiver can signal a break to the transmitter. Port 16 must be connected externally to the transmission line in order to detect a Break-In. If transmission voltage levels other than TTL are used, then proper buffering must be provided so that Port 16 on the MUART will receive the correct polarity and voltage levels.
When Break-In Detect is enabled, Port 16 is polled internally during the transmission of the last or only stop bit of a character. If this pin is low during transmission of the stop bit, the Break Detect status bit (BD) will be set. Break-In Detect and receive Break Detect are OR-ed to set the BD status bit. (Either one can set this bit.) The distinction cannot be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on level 5, the transmit interrupt, while Break will generate an interrupt on level 4, the receive interrupt. If RxC and TxC are used for the serial bit rates, Break-In cannot be detected.

Transmit Break – This causes the TxD pin to be forced low for as long as the TBRK bit in Command Register 3 is set. While Transmit Break is active, data transfers from the Transmit Buffer to the Transmit register will be inhibited.

If both the Transmit Buffer and the Transmit Register are full, and a Transmit Break command is issued (command register 3, TBRK = 1), the entire character in the Transmit register is sent including the stop bits. TxD is then driven low and the character in the Transmit Buffer remains there until Transmit Break is disabled (command register 3, TBRK = 0). At this time TxD will go high for one bit time and then send the character in the Transmit Buffer.

Single Character Break – This causes TxD to be set low for as long as the TBRK bit in Command Register 3 is set. While Transmit Break is active, data transfers from the Transmit Buffer to the Transmit register will be inhibited.

If both the Transmit Buffer and the Transmit Register are full and a Send Break command is issued (command register 3, SBRK = 1) the entire character in the Transmit Register is sent including the stop bits. TxD is driven low for one complete character time followed by a high for two bit times after which the character in the Transmit Buffer is sent.

Modification Register

The modification register is used to alter two standard functions of the receiver (start bit check, and sampling time) and to enable a special indicator flag for half-duplex operation (transmitter status). Disabling start bit check means that the receiver will not return to the start bit search mode if RxD has returned to the Mark state in the center of the start bit. It will simply proceed to assemble a character from the RxD pin regardless of whether it received a false start bit or not. The modification register also allows the user to define where within the receive data bits the MUART will sample.

Parallel I/O

The MUART contains 16 parallel I/O pins which are divided into two 8-bit ports. These two parallel I/O ports (Port 1 and Port 2) can be used for basic digital I/O such as setting a bit high or low, or for byte transfers using a two-wire handshake. Port 1 is bit programmable for input or output, so any combination of the eight bits in Port 1 can be selected as either an input or an output. Port 2 is nibble programmable, which means that all four bits in the upper or lower nibble have to be selected as either inputs or outputs. For byte transfers using the two-wire handshake, Port 2 can either input or output the byte while two bits in Port 1 are used for the handshaking signals.

All of the bits in Port 1 have alternate functions other than I/O ports. As mentioned above, when using the byte handshake mode, two bits on Port 1 are used for the handshaking signals. As a result, these two bits cannot be used for general purpose I/O. The other six bits in Port 1 also have alternate functions if they are not used as I/O ports. Table 1 lists each bit from Port 1 and its corresponding alternate function.

The bits in the Port 1 Control Register select whether the pins on Port 1 are inputs or outputs. The pins on Port 1 are selected as control pins through the other programming registers which are relevant to the control signal. Configuring a bit in Port 1 as a control function overrides its definition in the Port 1 Control Register. If the pins on Port 1 are redefined as control signals, the definition of whether the pin is an input or an output in the Port 1 Control Register remains unchanged. If the pins on Port 1 are converted back to I/O pins, they assume the state which was defined in the Port 1 Control Register.

Each parallel I/O port has a latch and drivers. When the port is in the output mode, the data written to the port is latched and driven on the pins. The data which is latched in the I/O ports remains unchanged unless the port is written to again. Reading the ports, whether the port is an input or output, gates the state at the pins onto the data bus. Writing to an input port has no effect on the pin, but the data is stored in the latch and will be output if the direction on the pin is changed later. Writing to a control pin on Port 1 has the same effect as writing to an input pin. If pins 2, 3, 5, and 6 in Port 1 are used for control signals, the contents of the respective output latches will be read, not the state of the control signals. If pins 0, 1, and 7 on
Port 1 are used for control signals. The state of the control signals will be read. If pin 4 on Port 1 is used as a test output for the internal baud rate, this clock signal will be output through the output latch, thus the information in the output latch will be lost.

The Two-Wire Byte Handshake

The 8256 can be programmed, via the Mode Register, to implement an input or output two-wire byte handshake. When the Mode Register is programmed for the byte handshake, Port 2 is used to transmit or receive the byte, and pins P10 and P11 are used for the two handshake control signals. Figures 4 and 5 on pages 7 through 10 show a block diagram and timing signals for the two-wire handshake input and output.

To set up the two-wire handshake output using interrupts one must first program the Mode Register, and then enable the interrupt via the interrupt mask register. An interrupt will not occur immediately after the two-wire handshake interrupt is enabled. The interrupt is triggered by the rising edge of ACK. There are two ways to generate the first interrupt. Either the first data byte must be written to Port 2 and completely transferred before an interrupt will occur, or the two-wire handshake interrupt is enabled while ACK is low, and then ACK goes high.

Event Counters/Timers

The MUART's five 8-bit programmable counters/timers are binary presettable down counters. The distinction between timer and counter is determined by the clock source. A timer measures an absolute time interval, and its input clock frequency is derived from the MUART's system clock. A counter's input clock frequency is derived from a pulse applied to an external pin. The counter is decremented on the rising edge of this pulse.

When the counters/timers are configured as timers their clock source passes through two dividers: the system clock prescaler, and the timer prescaler. As mentioned before, the system clock prescaler normalizes the internal system clock to 1.024 MHz. The timer prescaler receives this normalized system clock and divides it down to either 1 kHz or 16 kHz, depending on the configuration.
on how Command Register 1 is programmed. If more timing resolution is needed the clock frequency can be input externally through the I/O ports.

By programming the Mode Register, four of the 8-bit counters/timers can be cascaded to form two 16-bit counters. Counters/timers 3 and 5 can be cascaded together, and counters/timers 2 and 4 can be cascaded together. Counters/timers 2 and 3 are the lower bytes, while counters/timers 4 and 5 are the upper bytes in the cascaded mode.

Each counter can be loaded with an arbitrary initial value. Timer 5 is the only timer which has a special save register which holds its initial value. Whenever Timer 5 is loaded with an initial value the special save register is also loaded with this value. Timer 5 can be reloaded to its initial value from the detection of a high-to-low transition on Port P15.

The counters are decremented on the first rising edge of the clock after the initial value has been loaded. The setup time for loading the counter when using an external clock is specified in the data sheet. When using internal clocks, the user has no way of knowing the phase relationship of the clock to the write pulse; therefore the timing accuracy is one clock period.

The timers are counting continuously, and an interrupt request is issued any time a single counter or pair of cascaded counters reaches zero. If the timers are going to be used with interrupts, then the programmer should first load the timer with the initial value, then enable the interrupt. If the programmer enables the interrupt first, it is possible that the interrupt will occur before the initial value is loaded. When an interrupt from any one of the timers occurs, the corresponding bit in the interrupt mask register is automatically reset, preventing further interrupt requests from occurring.

The event counters/timers can be used in the following modes of operation:

**Timer 1**
- Serves as an 8-bit timer.

**Event Counter/Timer 2**
- Serves as an 8-bit timer or event counter, or cascaded with Timer 4 as a 16-bit timer or event counter.

**Event Counter/Timer 3**
- Serves as an 8-bit timer or event counter, or cascaded with Timer 5 as a 16-bit timer or event counter, with the additional modes of operation selectable for Timer 5.

**Timer 4**
- Serves as an 8-bit timer, or cascaded with Event Counter/Timer 2 as a 16-bit timer or event counter.

**Timer 5**
1) Non-retriggerable 8-bit timer
2) Retriggerable 8-bit timer whose initial value is loaded from a save register which starts following the negative transition of an external signal. Subsequent transitions of this signal after the counting has started, reloads the initial value and restarts the counting.
3) Cascaded with Event Counter/Timer 3, non-retriggerable 16-bit timer, which can be loaded with an initial value by two write operations.
The 8256 signals with INT that the equipment has accepted the last character and that the output latches are empty again.

Thereupon, the microprocessor transfers the next data to the 8256.

The rising edge of WR latches the data into port 2 (P20...P27) and "Output Buffer Full" (OBF) is set which indicates that a new byte is available.

The equipment acknowledges with the falling edge of ACK that it recognized OBF.

Thereupon, the 8256 releases OBF.

The equipment acknowledges the data transfer with a rising edge of ACK which causes the 8256 to set INT.
4) Cascaded with event counter/timer 3, non-retriggerable 16-bit event counter, which can be loaded with an initial value by two write operations.

5) Cascaded with Event Counter/Timer 3, retriggerable 16-bit timer. The most significant byte (Timer 5) will be loaded with its initial value from the save register, while the least significant byte (Event Counter/Timer 3) will be set to OFFH automatically. Loading, starting, and retriggering operations follow the same pattern as in 2).

6) Cascaded with Event Counter/Timer 3, retriggerable 16-bit event counter. The most significant byte (Timer 5) will be loaded with its initial value from the save register, while the least significant byte (Event Counter/Timer 3) will be set to OFFH automatically. Loading, starting, and retriggering operations follow the same pattern as in 2).

**Interrupt Controller**

In a microcomputer system there are several ways for the CPU to recognize that a peripheral device needs service. Two of the most common ways are the polling method and the interrupt service method.

In the polling method the CPU reads the status of each peripheral to determine whether it needs service. If the peripheral does not need service, the time the CPU spends polling is wasted; therefore this overhead results in increasing the execution time. Some systems must meet a specific request to response time such as a real time signal. In this case the programmer must guarantee that the peripheral is polled at a certain frequency. This polling frequency cannot always be met when the CPU must execute a main program as well as subroutines. Usually each peripheral has its own request to response time requirements; therefore the user must establish a priority scheme.

The interrupt method provides certain advantages over the polling method. When a peripheral device needs service it signals the CPU through hardware asynchronously, thus reducing the overhead of polling a device which does not need service. The CPU would typically finish the instruction it is executing, save the important registers, and acknowledge the peripheral's interrupt request. During the acknowledgment, the CPU reads a vector which directs the CPU to the starting location of the appropriate interrupt service routine. If several interrupt requests occur at the same time, special logic can prioritize the requests so that when the CPU acknowledges the interrupt, the highest priority request is vectored to the CPU.

An interrupt driven system requires additional hardware to control the interrupt request signal, priority, and vectoring. The 8256 integrates this additional hardware onto the chip. The interrupt controller on the UART is directly compatible with the MCS-85, iAPX-86, iAPX-88, iAPX-186, iAPX-188 family of microcomputer systems, and it can also be used with other microprocessors as well. It contains eight priority levels, however, there are a total of 12 interruptable sources: 10 internal and 2 external. Since there are eight priority levels, only eight interrupts can be used at one time. The assignment of the interrupts used is selected by Command Register 1 and by the mode register. The UART's interrupt sources have a fixed priority. Table 2 displays how the 12 interrupt sources are mapped into the 8 priority levels.
The equipment indicates with the falling edge of STB (Strobe) that a new character is available at port 2. The 8256 acknowledges the indication by activating IBF (Input Buffer Full).

Thereupon, the equipment releases STB and the 8256 latches the character.

The 8256 informs the microprocessor through INT that a new character is ready for transfer.

The microprocessor reads the character.

The rising edge of signal RD resets signal IBF.

This action signals to the equipment that the input latches of the 8256 are empty and the next character can be transferred.
The 8256 is compatible with the 8085 interrupt vectoring method when the 8086 bit in Command Register 1 of the MUART is set to 0. This is the default condition after a hardware reset. The 8085 has five hardware interrupt pins: INTR, RST 7.5, RST 6.5, RST 5.5, and TRAP. When the MUART’s interrupt acknowledge feature is enabled (IAE bit 5 Command Register 3 = 1) the MUART’s INT Pin 15 should be tied to the 8085’s INTR, and both the 8085 and the MUART’s INTA pins should be tied together. All of the interrupt pins on the 8085 except INTR automatically vector the program counter to a specified location in memory. When the INTR pin becomes active (HIGH), assuming the 8085 has interrupts enabled, the 8085 fetches the next instruction from the data bus where it has been placed by the 8256 or some other interrupt controller. This instruction is usually a Call or an RST0 through RST7. Figure 6 shows the memory locations where the 8085 will vector to based on which type of interrupt occurred.

The 8085 can receive an interrupt request any time, since its INTR input is asynchronous. The 8085, however, doesn’t always acknowledge an interrupt request immediately. It can accept or disregard requests under software control using the EI (Enable Interrupt) or DI (Disable Interrupt) instructions.

At the end of each instruction cycle, the 8085 examines the state of its INTR pin. If an interrupt request is present and interrupts are enabled, the 8085 enters an interrupt machine cycle. During the interrupt machine cycle the 8085 automatically disables further interrupts until the EI instruction is executed. Unlike normal machine cycles, the interrupt machine cycle doesn’t increment the program counter. This ensures that the 8085 can return to the pre-interrupt program location after the interrupt service is completed. The 8085 issues an INTA pulse indicating that it is honoring the request and is ready to process the interrupt.

The 8256 can now vector program execution to the corresponding service routine. This is done during the first and only INTA pulse. Upon receiving the INTA pulse, the 8256 places the opcode RSTn on the data bus; where n equals 0 through 7 based on the level of the interrupt requested. The RSTn instruction causes the contents of the program counter to be pushed onto the stack, then transfers control to the instruction whose address is eight times n, as shown in Figure 6.

Note that because interrupts are disabled during the interrupt acknowledge sequence, the EI instruction must be executed in either the service routine or the main program before further interrupts can be processed.

For additional information on the 8085 interrupt operation and the RSTn instruction, refer to the MCS-85 User’s Manual.
IAPX-86/88 - 8256 Interrupt Operation

The MUART is compatible with the 8086/8088 method of interrupt vectoring when the 8086 bit in Command Register 1 is set to 1. The MUART's INT pin is tied to the 8086/8088 INTR pin, and its INTA pin connected to the 8086/88's INTA pin. Like the 8085, the 8086/8088's INTR pin is also asynchronous so that an interrupt request can occur at any time. The 8086/8088 can accept or disregard requests on the INTR pin under software control instructions. These instructions set or clear the interrupt-enabled flag IF. When the 8086/8088 is powered-on or reset, the IF flag is cleared, disabling external interrupts on INTR.

Although there are some basic similarities, the actual processing of interrupts with an 8086/8088 is different from the 8085. When an interrupt request is present and interrupts are enabled, the 8086/8088 enters its interrupt acknowledge machine cycle. The interrupt acknowledge machine cycle pushes the flag registers onto the stack (as in PUSHF instruction). It then clears the IF flag, which disables interrupts. Finally, the contents of both the code segment register and the instruction pointer are pushed onto the stack. Thus, the stack retains the pre-interrupt flag status and program location which are used to return from the service routine. The 8086/8088 then issues the first of two INTA pulses which signals the 8256 that the 8086/8088 has honored its interrupt request.

The 8256 is now ready to vector program execution to the appropriate service routine. Unlike the 8085 where the first INTA pulse is used to place an instruction on the data bus, the first INTA pulse from the 8086/8088 is used only to signal the 8256 of the honored request. The second INTA pulse causes the 8256 to place a single interrupt vector byte onto the data bus. The 8256 places the interrupt vector bytes 40H through 47H corresponding to the level of the interrupt to be serviced. Not used as a direct address, this interrupt vector byte pertains to one of 256 interrupt "types" supported by the 8086/8088 memory. Program execution is vectored to the corresponding service routine by the contents of a specified interrupt type.

All 256 interrupt types are located in absolute memory locations 0 through 3FFH which make up the 8086/8088's interrupt vector table. Each type in the interrupt vector table requires 4 bytes of memory and stores a code segment address and an instruction pointer address. Figure 7 shows a block diagram of the interrupt vector table. When the 8086/8088 receives an interrupt vector byte, it multiplies its value by four to acquire the address of the interrupt type.

![Interrupt Vector Table Diagram](image-url)

Figure 7. 8086/8088 Interrupt Vector Table
Once the service routine is completed the main program may be reentered by using an IRET (Interrupt Return) instruction. The IRET instruction will pop the pre-interrupt instruction pointer, code segment and flags off the stack. Thus the main program will resume where it was interrupted with the same flag status regardless of changes in the service routine. Note especially that this includes the state of the IF flag; thus interrupts are re-enabled automatically when returning from the service routine. For further information refer to the iAPX 86,88 User's Manual.

Using the 8256's Interrupt Controller Without INTA

There are several configurations where the 8256 will not have an INTA signal connected to it. Some examples are when using the 8256 with an 8051 or 8048, or when connecting the INT pin on the 8256 to the 8085's RST 7.5, RST 6.5, or RST 5.5 inputs. In these configurations the IAE bit in Command Register 3 is set to 0, and the INTA pin on the 8256 is tied high. When the interrupt occurs the CPU should branch to a service routine which reads the interrupt address register to determine which interrupt request level occurred. The interrupt address register contains the level of the interrupt multiplied by four. Reading the interrupt address register is equivalent in effect to the rINT signal; it clears the INT pin and indicates to the MUART that the interrupt request has been acknowledged. After the CPU reads the value in the interrupt address register, it can add an offset to this value and branch to an interrupt vector table which contains jump instructions to the appropriate interrupt service routines. An 8085 program which demonstrates this routine is given is Figure 8.

Table 3 summarizes the priority levels and the interrupt vectors which the 8256 sends back to the CPU. Note that when using Timer 1 there is a conflict present between RST0 in the 8085 mode and a hardware reset, because both expect instructions starting at address 0H. However, there is a way to distinguish between the two. A hardware reset, all control registers are reset to a value of 0H; therefore when using Timer 1, Reset and RST0 can be distinguished by reading one of the control registers of the 8256 which has not been programmed with a value of 0H. The control registers will contain the previously programmed values if RST0 occurs.

Interrupt Registers

The 8256's interrupt controller has several registers associated with it: an Interrupt Mask Register, an Interrupt Address Register, an Interrupt Request Register, an Interrupt Service Register, and a Priority Controller. Only the Interrupt Mask Registers and the Interrupt Address Register can be accessed by the user.

Interrupt Mask Registers

The Interrupt Mask Registers consist of two write registers — the Set Interrupts Register and Reset Interrupts Register, and one read register — the Interrupt Enable Register. Each one of the eight levels of interrupts may be individually enabled or disabled through these registers. Writing a one to any of the bits in the Set Interrupts Register enables the corresponding interrupt level, while writing a one to a bit in the Reset Interrupts Register disables the corresponding interrupt level. Reading the Interrupt Enable Register allows the user to determine which interrupt levels are enabled. The bits which are set to one in the Interrupt Enable Register correspond to the levels which are enabled. All of the interrupt levels will remain enabled until disabled by the Reset Interrupts Register except the counter/timer interrupts which automatically disable themselves when they reach zero.

### INTA: IN INTADD ;Read the Interrupt Address Register
MOV L, A ;Put the interrupt address in HL
XRA A
MOV H, A
LXI B, TABLE ;Load BE with the interrupt table offset
DAD B ;Add the offset to the interrupt address
PCHL ;Jump to the interrupt vector table

**Figure 8. Software Interrupt Acknowledge Routine**
Table 3. Assignment of Interrupt Levels to Interrupt Sources

<table>
<thead>
<tr>
<th>Interrupt Level</th>
<th>Restart Command 8085 mode</th>
<th>Interrupt Vector 8086 mode</th>
<th>Interrupt Address</th>
<th>Trigger Mode</th>
<th>Sources (Only one source can be assigned at any time)</th>
<th>Selection by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest Priority 0</td>
<td>RST0</td>
<td>40H</td>
<td>0H</td>
<td>edge</td>
<td>Timer 1</td>
<td>–</td>
</tr>
<tr>
<td>1</td>
<td>RST1</td>
<td>41H</td>
<td>4H</td>
<td>edge</td>
<td>Event Counter/Timer 2 or external interrupt request on Port 1 P17</td>
<td>Command word 1 BIT1 (bit 2)</td>
</tr>
<tr>
<td>2</td>
<td>RST2</td>
<td>42H</td>
<td>8H</td>
<td>level</td>
<td>Input EXTINT</td>
<td>–</td>
</tr>
<tr>
<td>3</td>
<td>RST3</td>
<td>43H</td>
<td>CH</td>
<td>edge</td>
<td>Event Counter/Timer 3 or cascaded event counters/timers 3 and 5</td>
<td>Mode word T35 (bit 7)</td>
</tr>
<tr>
<td>4</td>
<td>RST4</td>
<td>44H</td>
<td>10H</td>
<td>edge</td>
<td>Serial receiver</td>
<td>–</td>
</tr>
<tr>
<td>5</td>
<td>RST5</td>
<td>45H</td>
<td>14H</td>
<td>edge</td>
<td>Serial transmitter</td>
<td>–</td>
</tr>
<tr>
<td>6</td>
<td>RST6</td>
<td>46H</td>
<td>28H</td>
<td>edge</td>
<td>Timer 4 or cascaded event counters/timers 2 and 4</td>
<td>Mode word T24 (bit 6)</td>
</tr>
<tr>
<td>7 Lowest Priority</td>
<td>RST7</td>
<td>47H</td>
<td>1CH</td>
<td>edge</td>
<td>Timer 5 or Port 2 with handshaking interrupt request</td>
<td>Mode word P2C2 – P2C0 (bits 2…0)</td>
</tr>
</tbody>
</table>

Note:
If no interrupt requests are pending and INTA cycle occurs, interrupt level 2 will be the default value vectored to the CPU.

Interrupt requests occurring when the corresponding interrupt level is disabled are lost. An interrupt will only occur if the interrupt is enabled before the interrupt request occurs.

**Interrupt Address Register**
The Interrupt Address Register contains an identifier for the currently requested interrupt level. The numerical value in this register is equal to the interrupt level multiplied by four. It can be used in lieu of an INTA signal to vector the CPU to the appropriate interrupt service routine. Reading this register has the same effect as the INTA pulse: it clears the INT pin and indicates an interrupt acknowledgement to the MUART. If the Interrupt Address Register is read while no interrupts are pending, the external interrupt EXTINT will be the default value, 08H.

**Interrupt Request Register**
The Interrupt Request Register latches all pending interrupt requests unless they are masked off. The request is set whenever the associated event occurs.

**Interrupt Service Register**
In the fully nested mode of operation, every interrupt request which is granted service is entered into this register. The appropriate bit will be set whenever the interrupt is acknowledged by INTA or by reading the Interrupt Address Register. At the same time, the corresponding bit in the Interrupt Request Register is reset. The Interrupt Service Register bit remains set until the microcomputer transfers the End Of Interrupt command (EOI) to the device by writing it into Command Register 3. In the normal mode the bits in the Interrupt Service Register are never set.
Priority Controller

The priority controller selects the highest priority request in the Interrupt Request Register from up to eight requests pending. If the INTA signal is enabled and becomes active, the priority controller will cause the highest priority level in the Interrupt Request Register to be vectored back to the CPU, regardless of whether the 8256 is in the normal mode or the nested mode. In the normal mode, if any bits are set in the Interrupt Request Register, the INT pin is activated. The highest priority level in the Interrupt Request register will be transferred to the Interrupt Address Register at the same time the interrupt request occurs. In the Fully Nested mode, the priorities of all pending requests are compared to the priorities in the Interrupt Service Register. If there is a higher priority in the Interrupt Request Register than in the Interrupt Service Register, the INT signal will be activated and the new interrupt level will be loaded into the Interrupt Address Register.

Interrupt Modes

There are two modes of operation for the interrupt controller: a normal mode and a fully nested mode. In the normal mode the CPU should only be a maximum of one interrupt level deep; therefore, the CPU can be interrupted only while in the main program and not while in an interrupt service routine. In the fully nested mode it is possible for the CPU to be nested up to eight interrupt levels deep. Using the fully nested mode, the MUART will activate the INT pin only when a higher priority than the one in service is requested. The fully nested mode is used to protect high priority interrupt service routines from being interrupted by equal or lower priority requests.

Normal Mode

In the normal mode of operation the 8256 will activate the INT pin whenever any of the bits in the Interrupt Request Register are set. The bits in the Interrupt Request Register can be set only if the corresponding interrupts are enabled. If more than one interrupt request bit is set, the MUART will always place the highest priority level in the Interrupt Address Register and vector this level to the CPU during an INTA cycle. When the CPU acknowledges the interrupt request, using either the INTA signal or by reading the Interrupt Address Register, the corresponding Interrupt Request Register bit is reset. Since the Interrupt Service Register bits are never set, there is no indication in the MUART that an interrupt service routine is in progress. Therefore, the priority controller will interrupt the CPU again if any of the interrupt request bits are set, regardless of whether the next request is a higher, lower, or equal priority.

The implied way to design a program using the normal mode is to have the CPU's interrupt flag enabled during portions of the main program, but to leave the interrupt flag disabled while the CPU is executing code in an interrupt service routine. This way, the CPU can never be interrupted in an interrupt service routine. Upon completion of an interrupt service routine the program can enable the CPU's interrupt flag, then return to the main program.

Figure 9 shows an example of how the normal mode of interrupts may operate. As the CPU begins executing code in the main program, certain I/O ports, variables, and arrays need to be initialized. During this time the CPU's interrupt flag is disabled. Once the program has completed the initialization routine and can accept an interrupt, the interrupt flag is enabled. In the 8085 this is done with the assembly language instruction EI, and on the 8086 with STI.

A short time later, an interrupt request comes in on Level 4. Since the CPU's interrupt flag is enabled, the interrupt acknowledge signal is activated and the CPU branches off to Interrupt Service Routine 4. While the CPU is executing code in Interrupt Service Routine 4, an interrupt request comes in on Level 6 and then a short time later on Level 2. The 8256 activates the INT signal; however, the CPU ignores this because its interrupt flag is disabled. Upon returning to the main program the interrupt flag is enabled. When the interrupt acknowledge signal is activated, the MUART places the highest priority interrupt request on the data bus regardless of the order in which the requests came in. Therefore, during the interrupt acknowledge the MUART vectors the indirect address for Interrupt Level 2. The INT signal is not cleared after the acknowledge because there is still a pending interrupt.

The normal mode of operation is advantageous in that it simplifies programming and lowers code requirements within interrupt routines; however, there are also several disadvantages. One disadvantage is that the interrupt response time for higher priority interrupts may be excessive. For example, if the CPU is executing code in an interrupt service routine during a higher priority request, the CPU will not branch off to the higher priority service routine until the current interrupt service routine is completed. This delay time may not be acceptable for interrupts such as the serial receiver or a real time signal. For these cases the MUART provides the nested mode.

Nested Mode

In the nested mode of operation, whenever a bit in the Interrupt Request Register is set, the Priority Con-
controller compares the Interrupt Request Register to the Interrupt Service Register. If the bit set in the Request Register is of a higher priority than the highest priority bit set in the Service Register, the MUART will activate the INT signal and update the Interrupt Address Register. If the bit in the Request Register is of equal or lower priority than the highest priority bit set in the Service Register, the INT signal will not be activated. When an INTA signal is activated or the Interrupt Address Register is read, the corresponding bit in the Request Register which caused the INT signal to be asserted is reset and set in the Service Register. When an EOI (End Of Interrupt) command is issued, the highest priority bit in the Service Register is reset.

Figure 10 shows an example of the program flow using the nested mode of interrupts. During the main program an interrupt request is generated from Level 4. Since the interrupt flag is enabled, the interrupt acknowledge signal is activated, and the microprocessor is vectored to Service Routine 4. During Service Routine 4, Level 2 requests an interrupt. Since Level 2 is a higher priority than Level 4, the 8256 activates its INT signal. An interrupt
acknowledge is not generated because the interrupt flag is disabled. This section of code in Service Routine 4 is protected and cannot be interrupted. A protected section of code may reinitialize a timer, take a sample, or update a global variable. When the interrupt flag is enabled the microprocessor acknowledges the interrupt and vectors into Service Routine 2. Service Routine 2 immediately enables the interrupt flag because it does not have a protected section of code. During Service Routine 2, Interrupt Request 6 is generated. However, the MUART will not interrupt the microprocessor until service routines 2 and 4 have issued the EOI command.

**Edge Triggering**

The MUART has a maximum of two external interrupts—EXTINT and P17. EXTINT is a dedicated interrupt pin which is level triggered, where P17 is either an I/O port or an edge triggered interrupt. If P17 is selected as an interrupt through Command Register 1 and its interrupt level is enabled, it will generate an interrupt when the level on this pin changes from low to high. The edge triggered mode incorporates an edge lockout feature. This means that after the rising edge of an interrupt request and the acknowledgment of the request, the positive level on
P17 won’t generate further interrupts. Before another interrupt can be generated P17 must return low.

External devices which generate a pulse for an interrupt request can use the edge triggered mode as long as the minimum high time specified in the data sheet is met.

Level Triggering

The external interrupt (EXTINT pin 16) is the only level triggered interrupt on the MUART. The 8256 will recognize any active (high) level on the EXTINT as an interrupt request. The EXTINT pin must stay high until a short time after the rising edge of the first INTA pulse. If the voltage level on the EXTINT pin is high then goes low, the bit in the interrupt request register corresponding to EXTINT will be reset.

In the normal mode of operation if EXTINT is still high after the INTA pulse has been activated, the INT signal will remain active. If the microprocessor’s interrupt flag is immediately reenabled, another interrupt will occur. Unless repeated interrupt generation is desired, the programmer should not reenable the CPU’s interrupt flag until EXTINT has gone low.

In the nested mode of operation, if EXTINT is still high after the INTA pulse has been activated, the INT signal will not be reactivated. This is because in the nested mode only a higher priority interrupt than the one being serviced can activate the INT signal. The EXTINT pin should go inactive (low) before the EOI command is issued if an immediate interrupt is not desired.

Depending upon the particular design and application, the EXTINT pin has a number of uses. For example, it can provide repeated interrupt generation in the normal mode. This is useful in cases when a service routine needs to be continually executed until the interrupt request goes inactive. Another use of the EXTINT pin is that a number of external interrupt requests can be wire-ORed. This can’t be done using P17, for if a device makes an interrupt request while P17 is high (from another request), its transition will be shadowed. Note that when a wire-OR’ed scheme is used, the actual requesting device has to be determined by the software in the service routine.

Cascading the MUART’s Interrupt Controller

Cascading the MUART’s interrupt controller is necessary in an interrupt driven system which contains more than one interrupt controller, such as a system using more than one MUART, or using a MUART with another interrupt controller like the 8259A. For a system which uses several MUART’s, one of them is tied directly to the microprocessor’s INT and INTA pins, while the remaining MUARTs are daisy-chained using the EXTINT and INT pins. This is shown in Figure 11.

Figure 11. Cascading the MUART’s Interrupt Controller
Using the configuration in Figure 11, when the microprocessor receives an interrupt, it generates an interrupt acknowledge and branches into an interrupt service routine. For the interrupt service routine of the external interrupt, EXTINT Level 2, the microprocessor will read the next MUART's interrupt address register and branch to the appropriate service routine. In effect, this would be a software interrupt acknowledge. An example of this type of interrupt acknowledge is given in Figure 8. If the last MUART in the chain indicated an external interrupt, the microprocessor would simply return to the main program; however, this would be an error condition caused by a spurious interrupt. A flow chart of the software to handle cascaded interrupts is given in Figure 12.

![Flow Chart to Resolve Interrupt Request When Cascading MUART Interrupt Controllers](image-url)
Some consideration should be given to the priority of the interrupts when cascading MUARTs. If all of the MUART's Level 0 and Level 1 interrupts are disabled, the highest priority interrupt is the EXTINT. In this case the last MUART in the chain would have the highest priority; however, it would take the longest time to propagate back to the CPU. If, however, Level 0 or Level 1 interrupts were enabled, the closer to the microprocessor the MUART is, the higher the priority these two levels would have.

When using the 8256 interrupt controller along with some other interrupt controller, such as the 8259A, the MUART's INT signal would simply be tied to one of the interrupt controller's request inputs. The service routine for the MUART's interrupt request would initially perform the software interrupt acknowledge before servicing the MUART's interrupt request. A block diagram of this configuration is given in Figure 13.

### Polling the MUART

If interrupts are not used, the only other way to control the MUART is to poll it. It is still possible to use the priority structure of the MUART with polling. In this mode of operation the MUART's INT signal (Pin 15) is not used, and the INTA pin is tied high. Since the INT pin's level is duplicated in the MSB of the Status Register, a program can poll this bit. When it becomes set, the program could read the Interrupt Address Register to determine the cause. Either the normal or nested mode of operation can be used. Note that the functions used with this polled method must have their interrupts enabled.

It is also possible to poll the counters/timers, parallel I/O, and UART separately. To control the UART, one could poll the Status Register. Byte handshakes with the parallel I/O can be controlled by polling Port 1. Finally, each counter/timer has its own register which can be polled.

---

**Figure 13. Connecting the 8256 to the 8259A Interrupt Controller**
### PIN DESCRIPTIONS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD0-AD4</td>
<td>1-5</td>
<td>I/O</td>
<td><strong>Address/Data:</strong> Three-state address/data lines which interface to the lower 8 bits of the microprocessor's multiplexed address/data bus. The 5-bit address is latched on the falling edge of ALE. In the 8-bit mode, AD0-AD3 are used to select the proper register, while AD1-AD4 are used in the 16-bit mode. AD4 in the 8-bit mode is ignored as an address, while AD0 in the 16-bit mode is used as a second chip select, active low.</td>
</tr>
<tr>
<td>DB5-DB7</td>
<td>6-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALE</td>
<td>9</td>
<td>I</td>
<td><strong>Address Latch Enable:</strong> Latches the 5 address lines on AD0-AD4 and CS on the falling edge.</td>
</tr>
<tr>
<td>RD</td>
<td>10</td>
<td>I</td>
<td><strong>Read Control:</strong> When this signal is low, the selected register is gated onto the data bus.</td>
</tr>
<tr>
<td>WR</td>
<td>11</td>
<td>I</td>
<td><strong>Write Control:</strong> When this signal is low, the value on the data bus is written into the selected register.</td>
</tr>
<tr>
<td>RESET</td>
<td>12</td>
<td>I</td>
<td><strong>Reset:</strong> An active high pulse on this pin forces the chip into its initial state. The chip remains in this state until control information is written.</td>
</tr>
<tr>
<td>CS</td>
<td>13</td>
<td>I</td>
<td><strong>Chip Select:</strong> A low on this signal enables the MUART. It is latched with the address on the falling edge of ALE, and RD and WR have no effect unless CS was latched low during the ALE cycle.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>14</td>
<td>I</td>
<td><strong>Interrupt Acknowledge:</strong> If the MUART has been enabled to respond to interrupts, this signal informs the MUART that its interrupt request is being acknowledged by the microprocessor. During this acknowledgement the MUART puts an RSTn instruction on the data bus for the 8-bit mode or a vector for the 16-bit mode.</td>
</tr>
<tr>
<td>EXTINT</td>
<td>15</td>
<td>O</td>
<td><strong>Interrupt Request:</strong> A high signals the microprocessor that the MUART needs service.</td>
</tr>
<tr>
<td>RxC</td>
<td>16</td>
<td>I</td>
<td><strong>External Interrupt:</strong> An external device can request interrupt service through this input. The input is level sensitive (high), therefore it must be held high until an INTA occurs or the interrupt address register is read.</td>
</tr>
<tr>
<td>CLK</td>
<td>17</td>
<td>I</td>
<td><strong>System Clock:</strong> The reference clock for the baud rate generator and the timers.</td>
</tr>
</tbody>
</table>
| RxC    | 18     | I/O  | **Receive Clock:** If the baud rate bits in Command Register 2 are all 0, this pin is an input which clocks serial data into the RxD pin on the rising edge of RxC. If baud rate bits in Command Register 2 are programmed from 1-0FH, this pin outputs a square wave whose rising
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RxD 19</td>
<td>I</td>
<td></td>
<td>Receive Data: Serial data input.</td>
</tr>
<tr>
<td>CTS 21</td>
<td>I</td>
<td></td>
<td>Clear To Send: This input enables the serial transmitter. If 1, 1.5, or 2 stop bits are selected, CTS is level sensitive. As long as CTS is low, any character loaded into the transmitter buffer register will be transmitted serially. A single negative going pulse causes the transmission of a single character previously loaded into the transmitter buffer register. If a baud rate from 1-0FH is selected, CTS must be low for at least 1/32 of a bit, or it will be ignored. If the transmitter buffer is empty, this pulse will be ignored. If this pulse occurs during the transmission of a character up to the time where 1/2 of the first (or only) stop bit is sent out, it will be ignored. If it occurs afterwards, but before the end of the stop bits, the next character will be transmitted immediately following the current one. If CTS is still high when the transmitter register is sending the last stop bit, the transmitter will enter its idle state until the next high-to-low transition on CTS occurs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxC 22</td>
<td>I/O</td>
<td></td>
<td>Transmit Clock: If the baud rate bits in command register 2 are all set to 0, this input clocks data out of the transmitter on the falling edge. If baud rate bits are programmed for 1 or 2, this input permits the user to provide a 32x or 64x clock which is used for the receiver and transmitter. If the baud rate bits are programmed for 3-0FH, the internal transmitter clock is output. As an output it delivers the transmitter clock at the selected bit rate. If 1½ or 0.75 stop bits are selected, the transmitter divider will be asynchronously reset at the beginning of each character.</td>
</tr>
</tbody>
</table>

If 0.75 stop bits is chosen, the CTS input is edge sensitive. A negative edge on CTS results in the immediate transmission of the next character. The length of the stop bits is determined by the time interval between the beginning of the first stop bit and the next negative edge on CTS. A high-to-low transition has no effect if the transmitter buffer is empty or if the time interval between the beginning of the stop bit and next negative edge is less than 0.75 bits. A high or a low level or a low-to-high transition has no effect on the transmitter for the 0.75 stop bit mode. |
## PIN DESCRIPTIONS (CONTINUED)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxD</td>
<td>23</td>
<td>O</td>
<td>Transmit Data: Serial data output.</td>
</tr>
<tr>
<td>P27-P20</td>
<td>24-31</td>
<td>I/O</td>
<td>Parallel I/O Port 2: Eight bit general purpose I/O port. Each nibble (4 bits) of this port can be either an input or an output. The outputs are latched whereas the input signals are not. Also, this port can be used as an 8-bit input or output port when using the two-wire handshake. In the handshake mode both inputs and outputs are latched.</td>
</tr>
<tr>
<td>P17-P10</td>
<td>32-39</td>
<td>I/O</td>
<td>Parallel I/O Port 1: Each pin can be programmed as an input or an output to perform general purpose I/O. All outputs are latched whereas inputs are not. Alternatively these pins can serve as control pins which extend the functional spectrum of the chip.</td>
</tr>
<tr>
<td>GND</td>
<td>20</td>
<td>PS</td>
<td>Ground: Power supply and logic ground reference.</td>
</tr>
<tr>
<td>Vcc</td>
<td>40</td>
<td>PS</td>
<td>Power: +5V power supply.</td>
</tr>
</tbody>
</table>

## DESCRIPTION OF THE REGISTERS

The following section will provide a description of the registers and define the bits within the registers where appropriate. Table 4 lists the registers and their addresses.

### Command Register 1

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>LO</td>
<td>Level</td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>Source</td>
<td></td>
</tr>
<tr>
<td>S0</td>
<td>Source</td>
<td></td>
</tr>
<tr>
<td>BRKI</td>
<td>Break-In</td>
<td>Detect Enable</td>
</tr>
<tr>
<td>BITI</td>
<td>Interrupt on Bit Change</td>
<td></td>
</tr>
<tr>
<td>8086</td>
<td>8086 Mode Enable</td>
<td></td>
</tr>
<tr>
<td>FRQ</td>
<td>Timer Frequency Select</td>
<td></td>
</tr>
</tbody>
</table>

### FRQ — Timer Frequency Select

This bit selects between two frequencies for the five timers. If FRQ = 0, the timer input frequency is 16KHz (62.5us). If FRQ = 1, the timer input frequency is 1 KHz (1ms). The selected clock frequency is shared by all the counter/timers enabled for timing; thus, all timers must run with the same time base.

### 8086 — 8086 Mode Enable

This bit selects between 8085 mode and 8086/8088 mode. In 8085 mode (8086 = 0), A0 to A3 are used to address the internal registers, and an RSTn instruction is generated in response to the first INTA. In 8086 mode (8086 = 1), A1 to A4 are used to address the internal registers, and A0 is used as an extra chip select (A0 must equal zero to be enabled). The response to INTA is for 8086 interrupts where the first INTA is ignored, and an interrupt vector (40H to 47H) is placed on the bus in response to the second INTA.

### BITI — Interrupt on Bit Change

This bit selects between one of two interrupt sources on Priority Level 1, either Counter/Timer 2 or Port 1 P17 interrupt. When this bit equals 0, Counter/Timer 2 will be mapped into Priority Level 1. If BITI equals 1 and Level 1 interrupt is enabled, a transition from 1 to 0 in Counter/Timer 2 will generate an interrupt request on Level 1. When BITI equals 1, Port 1 P17 external edge triggered interrupt source is mapped into Priority Level 1. In this case if Level 1 is enabled, a low-to-high transition on P17 generates an interrupt request on Level 1.

### BRKI — Break-In Detect Enable

If this bit equals 0, Port 1 P16 is a general purpose I/O port. When BRKI equals 1, the Break-In Detect feature is enabled on Port 1 P16. A Break-In condition is present on the transmission line when it is forced to the start bit voltage level by the receiving station. Port 1 P16 must be connected externally to the transmission line in order to detect a Break-In.
<table>
<thead>
<tr>
<th>Read Registers</th>
<th>Write Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command 1</strong></td>
<td><strong>Command 1</strong></td>
</tr>
<tr>
<td>L1 L0 S1 S0 BRK</td>
<td>L1 L0 S1 S0 BRK</td>
</tr>
<tr>
<td>PEN EP C1 C0</td>
<td>PEN EP C1 C0</td>
</tr>
<tr>
<td>B3 B2 B1 B0</td>
<td>B3 B2 B1 B0</td>
</tr>
<tr>
<td><strong>Command 2</strong></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 1 1</td>
</tr>
<tr>
<td><strong>Command 3</strong></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 1 1</td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td><strong>Mode</strong></td>
</tr>
<tr>
<td>T35 T24 TSC</td>
<td>T35 T24 TSC</td>
</tr>
<tr>
<td>P17 P15 P14</td>
<td>P17 P15 P14</td>
</tr>
<tr>
<td>P12 P11 P10</td>
<td>P12 P11 P10</td>
</tr>
<tr>
<td><strong>Port 1 Control</strong></td>
<td></td>
</tr>
<tr>
<td>L7 L6 L5 L4</td>
<td>L7 L6 L5 L4</td>
</tr>
<tr>
<td>L3 L2 L1 L0</td>
<td>L3 L2 L1 L0</td>
</tr>
<tr>
<td><strong>Interrupt Enable</strong></td>
<td></td>
</tr>
<tr>
<td>D7 D6 D5 D4</td>
<td>D7 D6 D5 D4</td>
</tr>
<tr>
<td>D3 D2 D1 D0</td>
<td>D3 D2 D1 D0</td>
</tr>
<tr>
<td><strong>Interrupt Address</strong></td>
<td></td>
</tr>
<tr>
<td>D7 D6 D5 D4</td>
<td>D7 D6 D5 D4</td>
</tr>
<tr>
<td>D3 D2 D1 D0</td>
<td>D3 D2 D1 D0</td>
</tr>
<tr>
<td><strong>Receiver Buffer</strong></td>
<td></td>
</tr>
<tr>
<td>D7 D6 D5 D4</td>
<td>D7 D6 D5 D4</td>
</tr>
<tr>
<td>D3 D2 D1 D0</td>
<td>D3 D2 D1 D0</td>
</tr>
<tr>
<td><strong>Port 1</strong></td>
<td><strong>Port 1</strong></td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 1 1</td>
</tr>
<tr>
<td><strong>Port 2</strong></td>
<td><strong>Port 2</strong></td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 1 1</td>
</tr>
<tr>
<td><strong>Timer 1</strong></td>
<td><strong>Timer 1</strong></td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 1 1</td>
</tr>
<tr>
<td><strong>Timer 2</strong></td>
<td><strong>Timer 2</strong></td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 1 1</td>
</tr>
<tr>
<td><strong>Timer 3</strong></td>
<td><strong>Timer 3</strong></td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 1 1</td>
</tr>
<tr>
<td><strong>Timer 4</strong></td>
<td><strong>Timer 4</strong></td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 1 1</td>
</tr>
<tr>
<td><strong>Timer 5</strong></td>
<td><strong>Timer 5</strong></td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 1 1</td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td><strong>Modification</strong></td>
</tr>
<tr>
<td>INT RBF TBE</td>
<td>0 RS4 RS3</td>
</tr>
<tr>
<td>TRE BD PE</td>
<td>RS2 RS1</td>
</tr>
<tr>
<td>OE FE</td>
<td>RS0 TME</td>
</tr>
</tbody>
</table>
| | DSC | **Table 4. MUART Registers**
Break-In is polled by the UART during the transmission of the last or only stop bit of a character.

A Break-In Detect is OR-ed with Break Detect in Bit 3 of the Status Register. The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on Level 5, the transmit interrupt, while Break will generate an interrupt on Level 4, the receive interrupt.

**S0, S1 — Stop Bit Length**

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>Stop Bit Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.75</td>
</tr>
</tbody>
</table>

The relationship of the number of stop bits and the function of input CTS is discussed in the Pin Description section under “CTS”.

**L0, L1 — Character Length**

<table>
<thead>
<tr>
<th>L1</th>
<th>L0</th>
<th>Character Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

**Command Register 2**

<table>
<thead>
<tr>
<th>PEN</th>
<th>EP</th>
<th>C1</th>
<th>C0</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(IR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(1W)</td>
</tr>
</tbody>
</table>

Programming bits 0...3 with values from 3H to FH enables the internal baud rate generator as a common clock source for the transmitter and receiver and determines its divider ratio.

Programming bits 0...3 with values of 1H or 2H enables input TxC as a common clock source for the transmitter and receiver. The external clock must provide a frequency of either 32x or 64x the baud rate. The data transmission rates range from 0...32 Kbaud.

If bits 0...3 are set to 0, separate clocks must be input to pin RxC for the receiver and pin TxC for the transmitter. Thus, different baud rates can be used for transmission and reception. In this case, prescalers are disabled and the input serial clock frequency must match the baud rate. The input serial clock frequency can range from 0 to 1.024 MHz.

**B0, B1, B2, B3 — Baud Rate Select**

These four bits select the bit clock’s source, sampling rate, and serial bit rate for the internal baud rate generator.

<table>
<thead>
<tr>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
<th>Baud Rate</th>
<th>Sampling Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TxC, RxC</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>TxC/64</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>TxC/32</td>
<td>32</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>19200</td>
<td>32</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>9600</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4800</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2400</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1200</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>600</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>300</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>200</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>150</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>110</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>75</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>50</td>
<td>64</td>
</tr>
</tbody>
</table>

The following table gives an overview of the function of pins TxC and RxC:

<table>
<thead>
<tr>
<th>Bits 3 to 0 (Hex.)</th>
<th>TxC</th>
<th>RxC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Input: 1 x baud rate clock for the transmitter</td>
<td>Input: 1 x baud rate clock for the receiver</td>
</tr>
<tr>
<td>1, 2</td>
<td>Input 32 x or 64 x baud rate for transmitter and receiver</td>
<td>Output: receiver bit clock with a low-to-high transition at data bit sampling time. Otherwise: high level</td>
</tr>
<tr>
<td>3 to F</td>
<td>Output: baud rate clock of the transmitter</td>
<td>Output: as above</td>
</tr>
</tbody>
</table>

As an output, RxC outputs a low-to-high transition at sampling time of every data bit of a character. Thus, data can be loaded, e.g., into a shift register external-
ly. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

**C0, C1 — System Clock Prescaler**
(Bits 4, 5)

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

<table>
<thead>
<tr>
<th>C1</th>
<th>C0</th>
<th>Divider Ratio</th>
<th>Clock at Pin CLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5.12 MHz</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3.072 MHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2.048 MHz</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.024 MHz</td>
</tr>
</tbody>
</table>

**EP — Even Parity (Bit 6)**

EP = 0: Odd parity
EP = 1: Even parity

**PEN — Parity Enable (Bit 7)**

Bit 7 enables parity generation and checking.

PEN = 0: No parity bit
PEN = 1: Enable parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

**Command Register 3**

<table>
<thead>
<tr>
<th>SET</th>
<th>RxE</th>
<th>IAE</th>
<th>NIE</th>
<th>END</th>
<th>SBRK</th>
<th>TBRK</th>
<th>RST</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2R)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(2W)</td>
</tr>
</tbody>
</table>

Command Register 3 is different from the first two registers because it has a bit set/reset capability.

Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0-6 is low, no change occurs to that bit. When Command Register 3 is read, bits 0, 3, and 7 will always be zero.

**RST — Reset**

If RST is set, the following events occur:

1) All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.

2) The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be cancelled. Interrupt signal INT will go low.

3) The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.

4) If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does not alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has no effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

**TBRK — Transmit Break**

The transmission data output TxD will be set low as soon as the transmission of the previous character has been finished. It stays low until TBRK is cleared. The state of CTS is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be re-enabled.

**SBRK — Single Character Break**

This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle state.
(marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.

END — End of Interrupt
If fully nested interrupt mode is selected, this bit resets the currently served interrupt level in the Interrupt Service Register. This command must occur at the end of each interrupt service routine during fully nested interrupt mode. END is automatically cleared when the Interrupt Service Register (internal) is cleared. END is ignored if nested interrupts are not enabled.

NIE — Nested Interrupt Enable
When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

IAE — Interrupt Acknowledge Enable
This bit enables an automatic response to INTA. The particular response is determined by the 8086 bit in Command Register 1.

RxE — Receive Enable
This bit enables the serial receiver and its associated status bits in the status register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

SET — Bit Set/Reset
If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

Mode Register

<table>
<thead>
<tr>
<th>T35</th>
<th>T24</th>
<th>T5C</th>
<th>CT3</th>
<th>CT2</th>
<th>P2C2</th>
<th>P2C1</th>
<th>P2C0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3R)</td>
<td>(3W)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P2C2, P2C1, P2C0 — Port 2 Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2C2</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3 – B0 with a value ≥ 3H.

Note:
If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshaking.

CT2, CT3 — Counter/Timer Mode
Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 36). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

T5C — Timer 5 Control
If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with
an initial value. The first trigger pulse causes the initial value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.

**T35, T24 — Cascade Timers**

These two bits cascade Timers 3 and 5 or 2 and 4. Timers 2 and 3 are the lower bytes, while Timers 4 and 5 are the upper bytes. If T5C is set, then both Timers 3 and 5 can be preset and started by an external pulse.

When a high-to-low transition occurs, Timer 5 is preset to its saved value. But Timer 3 is always preset to all ones. If either CT2 or CT3 is set, then the corresponding timer pair is a 16-bit event counter.

A summary of the counter/timer control bits is given in Table 5.

**Note:**

Interrupt levels assigned to single counters are partly not occupied if event counters/timers are cascaded. Level 2 will be vacated if event counters/timers 2 and 4 are cascaded. Likewise, Level 7 will be vacated if event counters/timers 3 and 5 are cascaded.

Single event counters/timers generate an interrupt request on the transition from 01H to 00H, while cascaded ones generate it on the transition from 0001H to 0000H.

**Table 5. Event Counters/Timers Mode of Operation**

<table>
<thead>
<tr>
<th>Event Counter/Timer</th>
<th>Function</th>
<th>Programming (Mode Word)</th>
<th>Clock Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8-bit timer</td>
<td>T24 = 0, CT2 = 0</td>
<td>internal clock</td>
</tr>
<tr>
<td>2</td>
<td>8-bit timer</td>
<td>T24 = 0, CT2 = 1</td>
<td>P12 pin 37</td>
</tr>
<tr>
<td>3</td>
<td>8-bit timer</td>
<td>T35 = 0, CT3 = 0</td>
<td>internal clock</td>
</tr>
<tr>
<td>4</td>
<td>8-bit timer</td>
<td>T35 = 0, CT3 = 1</td>
<td>P13 pin 36</td>
</tr>
<tr>
<td>5</td>
<td>8-bit timer, normal mode</td>
<td>T35 = 0, T5C = 0</td>
<td>internal clock</td>
</tr>
<tr>
<td>2 and 4 cascaded</td>
<td>16-bit timer, normal mode</td>
<td>T35 = 0, T5C = 1</td>
<td>internal clock</td>
</tr>
<tr>
<td>3 and 5 cascaded</td>
<td>16-bit event counter, normal mode</td>
<td>T35 = 1, T5C = 0, CT3 = 0</td>
<td>P13 pin 36</td>
</tr>
<tr>
<td></td>
<td>16-bit event counter, normal mode</td>
<td>T35 = 1, T5C = 0, CT3 = 1</td>
<td>P13 pin 36</td>
</tr>
<tr>
<td></td>
<td>16-bit timer, Retriggerable mode</td>
<td>T35 = 1, T5C = 1, CT3 = 0</td>
<td>internal clock</td>
</tr>
<tr>
<td></td>
<td>16-bit event counter, Retriggerable mode</td>
<td>T35 = 1, T5C = 1, CT3 = 1</td>
<td>P13 pin 36</td>
</tr>
</tbody>
</table>
Each bit in the Port 1 Control Register configures the direction of the corresponding pin. If the bit is high, the pin is an output, and if it is low the pin is an input. Every Port 1 pin has another function which is controlled by other registers. If that special function is disabled, the pin functions as a general I/O pin as specified by this register. The special functions for each pin are described below.

### Port 10, 11 — Handshake Control

If byte handshake control is enabled for Port 2 by the Mode Register, then Port 10 is programmed as STB/ACK handshake control input, and Port 11 is programmed as IBF/OBF handshake control output.

If byte handshake mode is enabled for output on Port 2, OBF indicates that a character has been loaded into the Port 2 output buffer. When an external device reads the data, it acknowledges this operation by driving ACK low. OBF is set low by writing to Port 2 and is reset high by ACK.

If byte handshake mode is enabled for input on Port 2, STB is an input. IBF is driven low after STB goes low. On the rising edge of STB the data from Port 2 is latched.

IBF is reset high when Port 2 is read.

### Port 12, 13 — Counter 2, 3 Input

If Timer 2 or Timer 3 is programmed as an event counter by the Mode Register, then Port 12 or Port 13 is the counter input for Event Counter 2 or 3, respectively.

### Port 14 — Baud Rate Generator Output Clock

If test mode is enabled by the Mode Register and Command Register 2 baud rate select is greater than 2, then Port 14 is an output from the internal baud rate generator.

P14 in Port 1 control register must be set to 1 for the baud rate generator clock to be output. The baud rate generator clock is 64 x the serial bit rate except at 19.2Kbps when it is 32 x the bit rate.

### Port 15 — Timer 5 Trigger

If T5C is set in the Mode Register enabling a retriggerable timer, then Port 15 is the input which starts and reloads Timer 5.

A high-to-low transition on P15 (Pin 34) loads the timer with the save register and starts the timer.

### Port 16 — Break-In Detect

If Break-In Detect is enabled by BRKI in Command Register 1, then this input is used to sense a Break-In. If Port 16 is low while the serial transmitter is sending the last stop bit, then a Break-In condition is signaled.

### Port 17 — Port Interrupt Source

If BITI in Command Register 1 is set, then a low-to-high transition on Port 17 generates an interrupt request on Priority Level 1.

Port 17 is edge triggered.

### Interrupt Enable Register

<table>
<thead>
<tr>
<th>L7</th>
<th>L6</th>
<th>L5</th>
<th>L4</th>
<th>L3</th>
<th>L2</th>
<th>L1</th>
<th>L0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5R)</td>
<td>(5W = enable, 6W = disable)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Interrupts are enabled by writing to the Set Interrupts Register (5W). Interrupts are disabled by writing to the Reset Interrupts Register (6W). Each bit set by the Set Interrupts Register (5W) will enable that level interrupt, and each bit set in the Reset Interrupts Register (6W) will disable that level interrupt. The user can determine which interrupts are enabled by reading the Interrupt Enable Register (5R).

### Priority Source

<table>
<thead>
<tr>
<th>Priority</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td>L0</td>
</tr>
<tr>
<td></td>
<td>L1</td>
</tr>
<tr>
<td></td>
<td>L2</td>
</tr>
<tr>
<td></td>
<td>L3</td>
</tr>
<tr>
<td></td>
<td>L4</td>
</tr>
<tr>
<td></td>
<td>L5</td>
</tr>
<tr>
<td></td>
<td>L6</td>
</tr>
<tr>
<td>Lowest</td>
<td>L7</td>
</tr>
<tr>
<td></td>
<td>Timer 1</td>
</tr>
<tr>
<td></td>
<td>Timer 2 or Port Interrupt</td>
</tr>
<tr>
<td></td>
<td>External Interrupt (EXTINT)</td>
</tr>
<tr>
<td></td>
<td>Timer 3 or Timers 3 &amp; 5</td>
</tr>
<tr>
<td></td>
<td>Receiver Interrupt</td>
</tr>
<tr>
<td></td>
<td>Transmitter Interrupt</td>
</tr>
<tr>
<td></td>
<td>Timer 4 or Timers 2 &amp; 4</td>
</tr>
<tr>
<td></td>
<td>Timer 5 or Port 2 Handshaking</td>
</tr>
</tbody>
</table>

### Interrupt Address Register

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(6R)</td>
<td>Interrupt Level Indication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reading the interrupt address register transfers an identifier for the currently requested interrupt level on the system data bus. This identifier is the number of the interrupt level multiplied by 4. It can be used by the CPU as an offset address for interrupt handling. Reading the interrupt address register has the same effect as a hardware interrupt acknowledge INTA; it clears the interrupt request pin (INT) and indicates an interrupt acknowledgement to the interrupt controller.

Receiver and Transmitter Buffer

Both the receiver and transmitter in the UART are double buffered. This means that the transmitter and receiver have a shift register and a buffer register. The buffer registers are directly addressable by reading or writing to register seven. After the receiver buffer is full, the RBF bit in the status register is set. Reading the receive buffer clears the RBF status bit. The transmit buffer should be written to only if the TBE bit in the status register is set. Bytes written to the transmit buffer are held there until the transmit shift register is empty, assuming CTS is low. If the transmit buffer and shift register are empty, writing to the transmit buffer immediately transfers the byte to the transmit shift register. If a serial character length is less than 8 bits, the unused most significant bits are set to zero when reading the receive buffer, and are ignored when writing to the transmit buffer.

Port 2

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(9R)</td>
<td>(9W)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Writing to Port 2 sets the data in the Port 2 output latch. Writing to an input pin does not affect the pin, but it does store the data in the latch. Reading Port 2 puts the input pins onto the bus or the contents of the output latch for output pins.

Timer 1-5

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0A_{16},0E_{16}R)</td>
<td>(0A_{16},0E_{16}W)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reading Timer N puts the contents of the timer onto the data bus. If the counter changes while RD is low, the value on the data bus will not change. If two timers are cascaded, reading the high-order byte will cause the low-order byte to be latched. Reading the low-order byte will un latch them both. Writing to either timer or decascading them also clears the latch condition. Writing to a timer sets the starting value of that timer. If two timers are cascaded, writing to the high-order byte presets the low-order byte to all ones. Loading only the high-order byte with a value of X leads to a count of X 256 + 255. Timers count down continuously. If the interrupt is enabled, it occurs when the counter changes from 1 to 0.

The timer/counter interrupts are automatically disabled when the interrupt request is generated.

Status Register

<table>
<thead>
<tr>
<th>INT</th>
<th>RBF</th>
<th>TBE</th>
<th>TRE</th>
<th>BD</th>
<th>PE</th>
<th>OE</th>
<th>FE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0F_{16}R)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reading the status register gates its contents onto the data bus. It holds the operational status of the serial interface as well as the status of the interrupt pin INT. The status register can be read at any time. The flags are stable and well defined at all instants.

FE — Framing Error, Transmission Mode

Bit 0 can be used in two modes. Normally, FE indicates framing error which can be changed to transmission mode indication by setting the TME bit in the modification register.
If transmission mode is disabled (in Modification Register), then FE indicates a framing error. A framing error is detected during the first stop bit. The error is reset by reading the Status Register or by a chip reset. A framing error does not inhibit the loading of the Receiver Buffer. If RxD remains low, the receiver will assemble the next character. The false stop bit is treated as the next start bit, and no high-to-low transition on RxD is required to synchronize the receiver.

When the TME bit in the Modification Register is set, FE is used to indicate that the transmitter was active during the reception of a character, thus indicating that the character received was transmitted by its own transmitter. FE is reset when the transmitter is not active during the reception of character. Reading the status register will not reset the FE bit in the transmission mode.

**OE — Overrun Error**

If the user does not read the character in the Receiver Buffer before the next character is received and transferred to this register, then the OE bit is set. The OE flag is set during the reception of the first stop bit and is cleared when the Status Register is read or when a hardware or software reset occurs. The first character received in this case will be lost.

**PE — Parity Error**

This bit indicates that a parity error has occurred during the reception of a character. A parity error is present if value of the parity bit in the received character is different from the one expected according to command word 2 bits 6 EP. The parity bit is expected and checked only if it is enabled by command word 2 bit 7 PEN.

A parity error is set during the first stop bit and is reset by reading the Status Register or by a chip reset.

**BD — Break/Break-In**

The BD bit flags whether a break character has been received, or a Break-In condition exists on the transmission line. Command Register 1 Bit 3 (BRKI) enables the Break-In Detect function.

Whenever a break character has been received, Status Register Bit 3 will be set and in addition an interrupt request on Level 4 is generated. The receiver will be idled. It will be started again with the next high-to-low transition at pin RxD.

The break character received will not be loaded into the receiver buffer register.

If Break-In Detection is enabled and a Break-In condition occurs, Status Register Bit 3 will be set and in addition an interrupt request on Level 5 is generated.

The BD status bit will be reset on reading the status register or on a hardware or software reset. For more information on Break/Break-In, refer to the “Serial Asynchronous Communication” section under “Receive Break Detect” and “Break-In Detect.”

**TRE — Transmit Register Empty**

When TRE is set the transmit register is empty and an interrupt request is generated on Level 5 if enabled. When TRE equals 0 the transmit register is in the process of sending data. TRE is set by a chip reset and when the last stop bit has left the transmitter. It is reset when a character is loaded into the Transmitter Register. If CTS is low, the Transmitter Register will be loaded during the transmission of the start bit. If CTS is high at the end of a character, TRE will remain high and no character will be loaded into the Transmitter Register until CTS goes low. If the transmitter was inactive before a character is loaded into the Transmitter Buffer, the Transmitter Register will be empty temporarily while the buffer is full. However, the data in the buffer will be transferred to the transmitter register immediately and TRE will be cleared while TBE is set.

**TBE — Transmitter Buffer Empty**

TBE indicates the Transmitter Buffer is empty and is ready to accept a character. TBE is set by a chip reset or the transfer of data to the Transmitter Register, and is cleared when a character is written to the transmitter buffer. When TBE is set, an interrupt request is generated on Level 5 if enabled.

**RBF — Receiver Buffer Full**

RBF is set when the Receiver Buffer has been loaded with a new character during the sampling of the first stop bit. RBF is cleared by reading the receiver buffer or by a chip reset.

**INT — Interrupt Pending**

The INT bit reflects the state of the INT Pin (Pin 15) and indicates an interrupt is pending. It is reset by INTA or by reading the Interrupt Address Register if only one interrupt is pending and by a chip reset.
FE, OE, PE, RBF, and Break Detect all generate a Level 4 interrupt when the receiver samples the first stop bit. TRE, TBE, and Break-In Detect generate a Level 5 interrupt. TRE generates an interrupt when TBE is set and the Transmitter Register finished transmitting. The Break-In Detect interrupt is issued at the same time as TBE or TRE.

**Modification Register**

<table>
<thead>
<tr>
<th>0</th>
<th>RS4</th>
<th>RS3</th>
<th>RS2</th>
<th>RS1</th>
<th>RS0</th>
<th>TME</th>
<th>DSC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0F16)</td>
</tr>
</tbody>
</table>

DSC — Disable Start Bit Check

DSC disables the receiver's start bit check. In this state the receiver will not be reset if RxD is not low at the center of the start bit.

TME — Transmission Mode Enable

TME enables transmission mode and disables framing error detection. For information on transmission mode see the description of the framing error bit in the Status Register.

RS0, RS1, RS2, RS3, RS4 — Receiver Sample Time

The number in RSn alters when the receiver samples RxD. The receiver sample time can be modified only if the receiver is not clocked by RxC.

**Note:**
The modification register cannot be read. Reading from address 0FH, 8086: IEH gates the contents of the status register onto the data bus.

- A hardware reset (reset, Pin 12) resets all modification register bits to 0, i.e.:
  - The start bit check is enabled.
  - Status Register Bit 0 (FE) indicates framing error.
  - The sampling time of the serial receiver is the bit center.

A software reset (Command Word 3, RST) does not affect the modification register.

**Hardware Reset**

A reset signal on pin RESET (HIGH level) forces the device 8256 into a well-defined initial state. This state is characterized as follows:

<table>
<thead>
<tr>
<th>RS4</th>
<th>RS3</th>
<th>RS2</th>
<th>RS1</th>
<th>RS0</th>
<th>Point of time between start of bit and end of bit measured in steps of 1/32 bit length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1 (Start of Bit)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16 (Bit center)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32 (End of Bit)</td>
</tr>
</tbody>
</table>

1) Command registers 1, 2 and 3, mode register, Port 1 control register, and modification register are reset. Thus, all bits of the parallel interface are set to be inputs and event counters/timers are configured as independent 8-bit timers.

2) Status register bits are reset with the exception of bits 4 and 5. Bits 4 and 5 are set indicating that both transmitter register and transmitter buffer register are empty.
3) The interrupt mask, interrupt request, and interrupt service register bits are reset and disable all requests. As a consequence, interrupt signal INT is inactive (LOW).

4) The transmit data output is set to the marking state (HIGH) and the receiver section is disabled until it is enabled by Command Register 3 Bit 6.

5) The start bit will be checked at sampling time. The receiver will return to start bit search mode if input RxD is not LOW at this time.

6) Status Register Bit 0 implies framing error.

7) The receiver samples input RxD at bit center.

Reset has no effect on the contents of receiver buffer register, transmitter buffer register, the intermediate latches of parallel ports, and event counters/timers, respectively.

INTERFACING

This section describes the hardware interface between the 8256 MUART and the 8085, 8086, 8088, and 80186 microprocessors. Figures 14 through 19 display the block diagrams for these interfaces. The MUART can be interfaced to many other microprocessors using these basic principles.

In all cases the 8256 will be connected directly to the CPU's multiplexed address/data bus. If latches or data bus buffers are used in a system, the MUART should be on the microprocessor side of the address/data bus. The MUART latches the address internally on the falling edge of ALE. The address consists of Chip Select (CS) and four address lines. For 8-bit microprocessors, AD0-AD3 are the address lines. For 16-bit microprocessors, AD1-AD4 are the address lines; AD0 is used as a second chip select which is active low. Since chip select is internally latched along with the address, it does not have to remain active during the entire instruction cycle. As long as the chip select setup and hold times are met, it can be derived from multiplexed address/data lines or multiplexed address/status lines.

In Figure 15, the 8088 min mode, the 8205 chip select decoder is connected to the 8088's address bus lines A8-A15. These address lines are stable throughout the entire instruction cycle. However, the MUART's chip select signal could have been derived from A16/S3-A19/S6.

Figure 16 shows the 8256 interfaced with an 8086 in the min mode. When the 8256 is in the 16-bit mode, A0 serves as a second chip select. As a result the MUART's internal registers will all have even addresses since A0 must be zero to select the device. Normally the MUART will be placed on the lower data byte. If the MUART is placed on the upper data byte internal registers will be 512 address locations apart and the chip would occupy an 8 K word address space. Figure 16A shows a table and a diagram of how the 8256 may be selected in an 8086 system where the MUART is I/O mapped and used on the lower byte of the address/data bus.

PROGRAMMING

Initialization

In general the MUART's functions are independent of each other and only the registers and bits associated with a particular function need to be initialized, not the entire chip. The command sequence is arbitrary since every register is directly addressable; however, Command Word 1 must be loaded first. To put the device into a fully operational condition, it is necessary to write the following commands:

- Command byte 1
- Command byte 2
- Command byte 3
- Mode byte
- Port 1 control
- Set Interrupts

The modification register may be loaded if required for special applications; normally this operation is not necessary. It is a good idea to reset the part before initialization. (Either a hardware or a software reset will do.)

Operating the Serial Interface

The microprocessor transfers data to the serial interface by writing bytes to the Transmit Buffer Register. Receive characters are transferred by reading the Receiver Buffer Register. The Status Register provides all of the necessary information to operate the serial I/O, including when to write to the Transmit Buffer, and when to read the Receive Buffer and error information.

Transmitting

The transmitter and the receiver may be operated by using either polling or interrupts. If polling is used then the software may poll the Status Register and write a byte to the Transmit Buffer whenever TBE = 1. Writing a byte to the Transmit Buffer clears the TBE
status bit. If the CTS pin is low, then the Transmit Buffer will transfer the data to the Transmit Register when it becomes empty. When this transfer takes place the TRE bit is reset, and the TBE bit is set indicating the next byte may be written to the Transmit Buffer. If CTS is high, disabling the transmitter, the data byte will remain in the Transmit Buffer and TBE will remain low until CTS goes low. The transmitter can only buffer one byte if it is disabled.

There is no way of knowing that the transmitter is disabled unless the CTS signal is fed into one of the I/O ports. Using the transmitter interrupt will free up the CPU to perform other functions while the transmitter is disabled or while the Transmit Buffer is full.

To enable the transmit interrupt feature Bit L5 in the Set Interrupt Register must be set. An interrupt request will not occur immediately after this bit has been set. Before any transmit interrupt request will occur a byte must be written to the Transmit Buffer. After the first byte has been written to the Transmit Buffer, a transmit interrupt request will occur, providing the transmitter is enabled.

There are three sources of transmitter interrupt requests: TBE=1, TRE=1, and Break-In Detect. Assuming the Break-In Detect feature is disabled, after the transmit interrupt is enabled and the first byte is written, a transmit interrupt request will be generated by TBE going active. The microprocessor can immediately write a byte to the Transmit Buffer without reading any status. However if Break-In Detect is enabled, the Status Register must be read to determine whether the transmit interrupt request was generated by Break-In Detect or TBE.

The TRE interrupt request can be used to indicate when the transmitter has completely sent all of the data. For example, using half-duplex communica-
Figure 15. 8088 Min Mode/8256 Interface Multiplexed Bus

tions, all of the data written to the UART must be transmitted before the line can be turned around. After the last byte is written, an interrupt request will be generated by TBE. If this interrupt is acknowledged without writing another byte, then the next transmitter interrupt request, TRE = 1, will indicate that the transmitter is empty and the line may be turned around.

RECEIVING

Valid data may be read from the Receive Buffer whenever the RBF bit in the Status Register is set. Reading the Receive Buffer resets the RBF status bit. The RBF bit in the Status Register can be used for polling. When the RBF bit is set, the three receive status bits, PE, OE, and FE are updated. These three status bits are reset when they are read. Therefore when the status register is read with RBF set, the three error status bit should be tested too.

If interrupts are used for serial receive data, the receiver must be enabled by setting the RxEn bit in Command Register 3, and Bit L4 must be set in the Set Interrupt Register. When the receive interrupt request occurs the Receive Buffer may be read, but the status register should also be read since the receive interrupt could have been generated by the Break Detect. Also, reading the status register will indicate whether there were any errors in the received character.

Operating the Parallel Interface

Data can be transferred to or read from Port 1 and Port 2 by using the appropriate write and read operations.

LOADING PORT 1 and PORT 2

Writing to the ports transfers the data present on the data bus into the output latches. This operation is independent of the programmed I/O characteristics of the individual port pins. Writing to control or input ports has no effect on the state of the pins. Pins defined as outputs immediately assume the state which is associated with the transferred data. If inputs or control pins are reprogrammed into outputs, they assume the states stored in their output latches which were transferred by the most recent port write operation.
Figure 16. 8086 Min Mode/8256 Interface

<table>
<thead>
<tr>
<th>BHE</th>
<th>A_n</th>
<th>CHARACTERISTICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>WHOLE WORD</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>UPPER BYTE FROM/ODD ADDRESS</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>LOWER BYTE FROM/EVEN ADDRESS</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>NONE</td>
</tr>
</tbody>
</table>

Figure 16a. Technique for Generating the MUART's Chip Select
READING PORT 1 AND PORT 2

Reading the ports gates the state at the pins onto the data bus if they are defined as I/O pins. A read operation transfers the contents of the associated output latches of pins P12, P13, P15, and P16, which are defined as control function pins. Reading control pins P10, P11, and P17 delivers the state of these pins.

Operating the Event Counters/Timers

The event counters/timers can be loaded with an initial value at any time. Reading event counters/timers is possible without interfering with the counting process.

LOADING EVENT COUNTERS/TIMERS

Loading event counters/timers 1-5 under their respective addresses transfers the data present on the data bus as an initial value into the addressed event counter/timer. The event counter/timer counts from the new initial value immediately following the data transfer (exception: retriggerable mode of Timer 5, or 3 and 5).

Cascaded counters/timers can be loaded with an initial value using one of two procedures:
1) Only the event counter/timer representing the most significant byte will be loaded. The event counter/timer representing the least significant byte is set to 0FFH automatically. Counting is started immediately after the data transfer.
2) The event counter/timer representing the most significant byte will be loaded, causing the least significant byte to be set to 0FFH automatically. Counting is started immediately following the data transfer. Next, the counter representing the least significant byte will be loaded and counting is started.
again, but this time with a complete 16-bit initial value. The least significant byte of the initial value must be transferred before the counter representing the least significant byte exhibits its zero transition to prevent the most significant byte of the initial value from being decremented improperly.

In the case of an 8-bit initial value for Timer 5 or for cascaded Event Counter/Timer 3 and 5, the initial value for Timer 5 is loaded from a save register, if it is operated in retriggerable counting mode. Counting is started after an initial value has been transferred whenever a high-to-low transition occurs on Port P15.

Cascaded Event Counter/Timer 3 and 5 operating in retriggerable counting mode can be loaded directly with an initial value for Timer 5 representing the most significant byte; Event Counter/Timer 3 will be set to 0FFH automatically.

**READING EVENT COUNTERS/TIMERS**

Reading event counters/timers 1-5 from their respective addresses gates the counter contents onto the data bus. The counter contents gated onto the data bus remain stable during the read operation while the counter just being read continues to count. The minimum time between the two read operations from the same counter is 1 usec.

The procedure to be followed when reading cascaded event counters/timers is:
1) The event counter/timer representing the most significant byte will be read first. At this time, the least significant byte is latched into read latches.
2) When the event counter/timer representing the least significant byte is addressed, the byte stored in the read latches will be gated onto the data bus. The value stored in the read latches remains valid until it is read, the cascading condition is removed, or a write
operation affecting one of the two event counters/timers is executed.

The time between reading the most significant byte and the least significant byte must be at least 1 usec.

Note:
For cascaded event counters/timers the least significant counter/timer is latched after reading the most significant counter/timer. If the lower byte changes from 00H to 0FFH between the reading of the MSB and the latching of the LSB, the carry from the most significant event counter/timer to the least significant event counter/timer is lost.

Therefore, it is necessary to repeat the whole reading once if the value of the least significant event counter/timer is 0FFH. Doing this will avoid working with a wrong value (correct value + 255).

**APPLICATION EXAMPLE**

This section describes how the 8256 was designed into a Line Printer Multiplexer (LPM). This application example was chosen because it employs a majority of the MUART’s features. The information in this section will be applicable to many other designs since it describes some common software and hardware aspects of using the MUART.

**Description of the Line Printer Multiplexer (LPM)**

The Line Printer Multiplexer allows up to eight workstations to share one printer. The workstations transmit serial asynchronous data to the LPM. The LPM receives the serial data, buffers it, then transmits.
Figure 20. Using the Line Printer Multiplexer to Share a Line Printer

The buffer size on the LPM was chosen to complement the disk access time on the workstations. Figure 21 illustrates the buffer size calculation. The line printer can print up to 300 lines per minute, or approximately 660 characters per second. This corresponds to a serial transmission rate of 6,600bps (assuming ASCII character codes and a parity bit) as shown in equation 1.

\[
\text{Serial bit rate} = \frac{(300 \text{ lines/min}) \times (132 \text{ char/line}) \times (10 \text{ bits/char})}{(60 \text{ sec/min})}
\]

The bottleneck in this data transfer is the line printer since the MUART and the workstations can both transmit and receive at 19.2Kbps. To realize the maximum data transfer rate of this system the LPM must guarantee that the average transfer rate to the line printer is 660 characters per second. The maximum amount of dead time that the serial port on the workstation is not transmitting, multiplied by 660 is the number of bytes which the LPM should buffer. It was determined through experimentation that it takes about 3 seconds to load 40K bytes of data from the disk into the workstation's RAM. During these 3 seconds no serial data is being sent; therefore the buffer size on the LPM should be 2K bytes. (Note: even though only a 2K byte FIFO is required, this design used an 8 Kbyte FIFO.)

To keep the LPM's buffer full the serial data rate must be greater than 6.6Kbps. The two bit rates which the
workstations use 9.6Kbps and 19.2Kbps. The CTS signal is used to control the flow of the serial data so that the LPM buffer will not overflow.

Each serial port on the LPM can have a different bit rate, character length, and parity format. These parameters are programmable through the serial port. When the LPM powers up, or is reset, it expects a bit rate of 9600 bps, 7 bit characters, and odd parity.

When a serial port receives an ASCII ESC character (1BH), it puts that port in the program mode. The next two bytes will program these three parameters. Only the lower nibbles of these two bytes are used, and the upper nibbles are discarded. The format of these programming words is given in Figure 22. If the word following the ESC is an ASCII NUL (0), the LPM will exit from the programming mode and not change any of its parameters.

### Figure 22. Programming Words Format for LPM

<table>
<thead>
<tr>
<th>FIRST BYTE</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>PEN</th>
<th>EP</th>
<th>L1</th>
<th>L0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PARITY ENABLE 1</td>
<td>PARITY DISABLE 0</td>
<td>EVEN PARITY 1</td>
<td>ODD PARITY 0</td>
<td>CHARACTER LENGTH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1 L0</td>
<td>0</td>
<td>0</td>
<td>8-BIT</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SECOND BYTE</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>B3 B2 B1 B0</td>
<td>BAUD RATE SELECT</td>
<td>BIT RATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>DO NOT USE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>DO NOT USE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>9600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 2</td>
<td>1200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 2 2</td>
<td>2400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 2 1 2</td>
<td>4800</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When a serial port receives an ASCII ESC character (1BH), it puts that port in the program mode. The next two bytes will program these three parameters. Only the lower nibbles of these two bytes are used, and the upper nibbles are discarded. The format of these programming words is given in Figure 22. If the word following the ESC is an ASCII NUL (0), the LPM will exit from the programming mode and not change any of its parameters.
Description of the Hardware

Figure 23 shows a block diagram of the LPM. In addition to the standard components of most microprocessor systems such as CPU, RAM, and ROM this particular design requires a UART, timers, parallel I/O and an interrupt controller. The MUART is the ideal choice for this design since it integrates these four functions onto one device.

The eight serial I/O ports use four signals: Transmit Data (TxD), Receive Data (RxD), Request To Send (RTS), and Clear To Send (CTS). These four signals, controlled by the MUART, are connected to one port at a time using TTL multiplexers. The TTL multiplexers are interfaced to RS-232 transceivers to be electrically compatible with the RS-232 spec. The serial port select address is derived from three bits of the MUART's parallel I/O port (Port 1). Two more bits from Port 1 control CTS and RTS, and another bit lights up an LED to indicate when the LPM's buffer is full. Parallel Port 2 and two bits from Port 1 are connected to the line printer implementing a two-wire byte handshake transfer. These signals are passed through a line driver so that they can reliably drive a long cable.

There are three timing functions needed for the LPM: a scan timer, a debounce timer, and a receive timeout. The Scan timer determines the amount of time spent sampling RTS on each port before the next port is addressed. By using one of the MUART's timers to do this function, the CPU is free to perform other functions instead of implementing the timer in software. If RTS is recognized as true, the CPU branches into a debounce procedure. This procedure uses another one of the MUART's timers to wait 10 msec then sample RTS again, thus preventing any glitches from registering as a false RTS. The receive timeout timer uses two 8-bit timers in the cascaded mode to measure an 18-second interval. After a valid RTS is recognized,
the LPM sends back a CTS and initializes the receive timeout timer for 18 seconds. Each time a character is received by the LPM, this timer is reinitialized. If this timer times out, the LPM considers the transmission complete and returns to scanning.

The schematic diagram of the LPM is shown in Figure 24. The CPU is an 8088 used in the min mode. It is interfaced directly to the 8256. An 8282 latch is employed in the system so that nonmultiplexed bus memory can be used. A 2716 holds the entire program, and six 2016s (2K x 8 static RAMs) are used to store the buffer, temporary data, stack area, and interrupt vector table. The 2716 is located in the upper 2K of the 8088 address space (FF800-FFFFFH) so that the reset vectors can be stored starting at location FFF0H. The RAM address space spans 0-2FFFH so that the interrupt vector table can be stored starting at location 0. The MUART is I/O mapped and its registers occupy even addresses from 0 to 1EH. Using an 8088 CPU the MUART must be placed in the 8086 mode since the INTA signal is used; hence the register addresses are all even numbers.

The line printer used provides a choice of two standard parallel interfaces: Centronics or Dataproducts. The Centronics interface uses a two-wire handshake pulsed strobe where the transmitter asserts a complete strobe pulse before an acknowledge is received. The Dataproducts interface is an interlocking two-wire handshake. The Dataproducts interface was chosen since it is directly compatible with the MUART’s two-wire byte handshake. The MUART could also be connected to the Centronics interface; however, additional hardware would be necessary to generate the pulsed strobe for correct interrupt operation. Figure 25 shows the timing of the Dataproducts interface and Table 6 lists the connector pin configuration.

### Table 6. Dataproducts Interface Line Functions

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
<th>Connector Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Request</td>
<td>Sent by printer to synchronize data transmission. When true, requests a character. Remains true until Data strobe is received, then goes false within 100 nsec.</td>
<td>E(return C)</td>
</tr>
<tr>
<td>Data Strobe</td>
<td>Sent by user system to cause printer to accept information on data lines. Should remain true until printer drops Data Request line. Data lines must stabilize for at least 50 nsec before Data Strobe is sent.</td>
<td>j(return m)</td>
</tr>
<tr>
<td>Data Bit 1</td>
<td>Bit 8 controls optional character set Refer to Commands and Formats.</td>
<td>B(return D)</td>
</tr>
<tr>
<td>Data Bit 2</td>
<td></td>
<td>F(return J)</td>
</tr>
<tr>
<td>Data Bit 3</td>
<td></td>
<td>L(return N)</td>
</tr>
<tr>
<td>Data Bit 4</td>
<td></td>
<td>R(return T)</td>
</tr>
<tr>
<td>Data Bit 5</td>
<td></td>
<td>V(return X)</td>
</tr>
<tr>
<td>Data Bit 6</td>
<td></td>
<td>Z(return b)</td>
</tr>
<tr>
<td>Data Bit 7</td>
<td></td>
<td>n(return k)</td>
</tr>
<tr>
<td>Data Bit 8</td>
<td></td>
<td>h(return e)</td>
</tr>
<tr>
<td>VFU Control (PI)</td>
<td>Optional control from user system. Used for VFU control. Data Request/Strobe timing is same as for data lines.</td>
<td>p(return s)</td>
</tr>
<tr>
<td>Ready</td>
<td>Sent to user system by printer. True when no Check condition exists.</td>
<td>CC(return EE)</td>
</tr>
<tr>
<td>On Line</td>
<td>Sent to user system by printer. True when Ready line is true and operator has activated ON LINE Pushbutton. Enables interface activity.</td>
<td>y(return AA)</td>
</tr>
<tr>
<td>Interface Verify</td>
<td>Jumper in printer connector. Continuity informs user system that connector is properly seated.</td>
<td>x to v</td>
</tr>
<tr>
<td>+5V</td>
<td>Supply voltage for Exerciser only.</td>
<td>HH</td>
</tr>
</tbody>
</table>
Figure 24. Schematic of LPM
Figure 24. Schematic of LPM (Continued)
Only ten signals are used to interface the LPM to the line printer: Data Request, Data Strobe, and the eight data lines. The most significant data line is not used since the character code is 7-bit ASCII. Data Strobe connects to OBF on the MUART; however, for the Dataproducts interface this signal must be inverted. Data Request is connected to ACK on the MUART. When the line printer is ready to accept data, the Data Request signal goes high. The 8256 will not interrupt the CPU to transmit parallel data unless this signal is high.

The Dataproducts interface is slightly different from the MUART's two-wire handshake in that it latches the data on the leading edge of the strobe signal. When the MUART receives bytes it latches the data on the trailing edge. As a result the Dataproducts interface has a 50 nsec setup time for data stable to the leading edge of Data Strobe. In the LPM hardware a delay line was used to realize this setup time.

**Description of the Software**

The software is written in PL/M and is broken up into four separate modules, each containing several procedures. A block diagram of the software structure is given in Figure 26. The modules are identified by the dotted boxes, and the procedures are identified by the solid boxes. Two or more procedures connected by a solid line means the procedure above calls the procedure below. The procedures without any solid lines connected above are interrupt procedures. They are entered when the MUART interrupts the CPU and vectors an indirect address to it.

The LPM program uses nested interrupts; the priority of the interrupt procedures is given in Table 7.

**Table 7. Line Printer Multiplexers' Interrupt Priority**

<table>
<thead>
<tr>
<th>Priority</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td>Debounce timer</td>
</tr>
<tr>
<td></td>
<td>Not Used</td>
</tr>
<tr>
<td></td>
<td>Not Used</td>
</tr>
<tr>
<td></td>
<td>Receive timer</td>
</tr>
<tr>
<td></td>
<td>RxD Interrupt</td>
</tr>
<tr>
<td></td>
<td>TxD Interrupt</td>
</tr>
<tr>
<td></td>
<td>Scan timer</td>
</tr>
<tr>
<td></td>
<td>LP Interrupt</td>
</tr>
</tbody>
</table>

The priority of the interrupts is not programmable but they are logically oriented so that for this application the priority is correct. In the steady state of the LPM's operation the UART will be receiving data, and the parallel port will be transmitting data. The serial receiver should be the highest priority since it can have overrun errors. This is the case because the debounce timer will be disabled, and the receive timeout interrupt will only occur when serial reception has ended. Therefore the RxD request can interrupt any other service routine, thus preventing any possibility of an overrun error.
On power-up the CPU branches from 0FFFF0H to the INITCODE routine which is included in the machine code by the MDS locator utility. INITCODE initializes the 8088's segment registers, stack pointer, and instruction pointer, then it disabled interrupts and jumps into MAIN_MOD. The first executable instruction in MAIN_MOD calls POWERSON, which initializes the UART, flags, variables, and arrays. The MAIN_MOD calls LOAD$INT$TABLE, which initializes the interrupt vector table. The CPU's interrupt is then enabled and the program enters into a DO FOREVER loop which scans the eight serial ports for an RTS.

There are three software functions which employ the UART's timers and interrupt controller to measure time intervals: SCAN, debounce, and INIT$RECEIVER. DEBOUNCE and INIT$RECEIVER procedures, employ the UART's timers and interrupt controller to measure time intervals. The CPU remains in a loop for a specific amount of time before it proceeds with the next section of code. In this loop the CPU is waiting for a global status flag to change while servicing any interrupts which may occur. When the appropriate timer interrupt occurs, the interrupt service routine will set the global flag which causes the CPU to exit the loop and proceed to the next section of code. An example can be seen from the scan flow chart in Figure 27.

The first thing the program does before entering the loop is set the flag (in this case SCAN$DELAY) TRUE. The timer is initialized and the loop is entered. As long as SCAN$DELAY is TRUE the CPU will continue to sample RTS. If RTS remains false for more than 100 msec, the timer interrupts the CPU and the interrupt service routine sets SCAN$DELAY FALSE. This causes the CPU to exit the loop and address the next port. The process is then repeated. If RTS becomes true while it is being sampled, the DEBOUNCE procedure is called.

DEBOUNCE does nothing more than wait 10 msec and sample RTS again using the same technique discussed above. If RTS is still valid INIT$RECEIVER is called, otherwise the CPU returns to scan.
INIT$RECEIVER calls CONFIGURE which programs the MUART for the bit rate, number of bits in a character, and parity format. This information is stored in an array called SERIAL$FORMAT, which contains a byte for each port. The bytes in the SERIAL$FORMAT array have the same bit definition as the two nibbles in the programming words in Figure 22. Upon returning to INIT$RECEIVER the receiver is enabled, the receive timeout timer is initialized, and the timer and receiver interrupts are enabled. CTS on the serial port is then set true, and the CPU enters a loop which does nothing except wait for 18 seconds. If no characters are received within 18 seconds, the receive timeout interrupt occurs and the loop flag is set false, which causes the CPU to exit the loop. If a character is received, a receive interrupt occurs, and the CPU vectors into the RxD interrupt service routine.

Figure 28 shows a flow chart of the RxD interrupt service routine. This routine begins by reading the receive buffer and reinitializing the receive timeout timer. There are two conditions to check for before the character can be inserted into the FIFO. First, if there are any errors in the received character, an ERROR procedure is called which reports back to the serial port what the error condition was. The character in error is discarded and the routine returns. The other condition is that if the received character is an ASCII ESC, the PROGRAM procedure is called. If neither one of these conditions occurs, the character is placed in the FIFO by the BUFF$IN procedure.

The LP interrupt routine is entered when the byte handshake interrupt request is acknowledged. This routine simply calls the BUFF$OUT procedure, which extracts a byte out of the FIFO. BUFF$OUT returns the byte to the LP interrupt procedure, which then writes it to Port 2. One small problem with getting the handshake interrupt going is that the first byte has to be written to Port 2 before the first handshake interrupt will occur. The problem is that the line printer may not be ready for the first byte. This would be indicated by DATA REQUEST being low. If the byte was written to the LP while DATA REQUEST is low, it would be lost. Note that if the handshake interrupt is enabled while DATA REQUEST is low, then DATA REQUEST goes high, the interrupt will occur without

Figure 27. Scan Flow Chart

Figure 28. RxD Interrupt Procedure Flow Chart
writing the first byte. There are several ways to solve this problem. Port 1 can be read to find out what the state of the DATA REQUEST line is. If DATA REQUEST is low, the CPU can simply wait for the interrupt without writing the first byte. If DATA REQUEST is high, then the first data byte may be written. Another solution would be to write a NUL character as the first byte to Port 2. If DATA REQUEST is low, then a worthless character is lost. If DATA REQUEST is high, the NUL character would be sent to the line printer; however, it is not printed since NUL is a nonprintable character. The LPM program uses the NUL character solution.

BUFFER MANAGEMENT

The FIFO implementation uses an 8K byte array to store the characters. There are two pointers used as indexes in the array to address the characters: IN$POINTER and OUT$POINTER. IN$POINTER points to the location in the array which will store the next byte of data inserted. OUT$POINTER points to the next byte of data which will be removed from the array. Both IN$POINTER and OUT$POINTER are declared as words. Figure 29 illustrates the FIFO in a block diagram.

The BUFF$IN procedure receives a byte from the RxD interrupt routine and stores it in the array location pointed to by IN$POINTER, then IN$POINTER is incremented. Similarly, when BUFF$OUT is called by the LP interrupt routine, the byte in the array pointed to by OUT$POINTER is read. OUT$POINTER is incremented, and the byte which was read is passed back to the LP interrupt routine. Since IN$POINTER and OUT$POINTER are always incremented, they must be able to roll over when they hit the top of the 8K byte address space. This is done by clearing the upper three bits of each pointer after it is incremented.

IN$POINTER and OUT$POINTER not only point to the locations in the FIFO, they also indicate how many bytes are in the FIFO and whether the FIFO is full or empty. When a character is placed into the FIFO and IN$POINTER is incremented, the FIFO is full if IN$POINTER equals OUT$POINTER. When a character is read from the FIFO and OUT$POINTER is incremented, the FIFO is empty if OUT$POINTER equals IN$POINTER. If the buffer is neither full nor empty, then it is in use. A byte called BUFFER$STATUS is used to indicate one of these three conditions.

The software uses the buffer status information to control the flow into and out of the FIFO. When the FIFO is empty the handshake interrupt must be turned off. When the FIFO is full, CTS must be sent false so that no more data will be received. If the buffer status is in use, CTS is true and the handshake interrupt is enabled.

Figure 30 shows the flow chart of the BUFF$IN procedure. The BUFF$IN procedure begins by checking the BUFFER$STATUS. If it is empty and the character to be inserted into the FIFO is a CR or LF, the handshake interrupt is enabled, a NUL character is output, and the BUFFER$STATUS is set to IN-USE. The character passed to BUFF$IN from RxD is put into the FIFO. If the FIFO is now full, the BUFFER$STATUS is set to FULL, CTS is set false, and the buffer full LED is turned on.

Figure 31 shows the flow chart of the BUFF$OUT procedure. After the character is read from the FIFO, the FIFO is tested to determine if it is empty. If it is not empty, the BUFFER$STATUS is FULL and there are 200 bytes available in the FIFO, serial data reception is reenabled, and the FIFO fills again. While data is being received from the workstation, CTS toggles high and low, filling up and emptying the last 200 bytes in the FIFO. Referring to the top of the flow chart (FIFO empty test) if it's empty, the BUFFER$STATUS is set to EMPTY, and the handshake interrupt is disabled. During this time all interrupts
are disabled at the CPU. (Remember that the RxD interrupt routine can interrupt the LP and BUFF$OUT procedures since it has a higher priority, and the UART is in the nested mode.)

If the CPU interrupt was not disabled during this time, the following events could occur which would cause the LPM to crash. Assume that the RxD interrupt occurred where the asterisk is in the flow chart, after BUFFER$STATUS is set to EMPTY. The BUFF$IN procedure would set BUFFER$STATUS to INUSE and enable the handshake interrupt. When the RxD interrupt routine returned to BUFF$OUT, the handshake interrupt is disabled, but the BUFFER$STATUS is INUSE. The handshake interrupt could never be reenabled, and the FIFO would fill up.

This is known as a critical section of code. Suspicion should arise for a critical section of code when two or more nested interrupt routines can affect the same status. One solution is to disable the interrupt flag at the CPU while the status and conditional operations are being modified.

The flow chart for the TxD interrupt procedure is given in Figure 32. For this program five different messages can be transmitted, and they are stored in ROM. It is possible to download the messages into a dedicated RAM buffer; however, the RAM buffer would have to be as large as the largest message. A more efficient way to transmit the messages is to read them from ROM. In this case the address of the first byte of the message would have to be accessible by the transmit interrupt procedure. Since parameters cannot be passed to interrupt procedures, this message pointer is declared PUBLIC in one module and EXTERNAL in the other modules.

To get the transmit interrupt started, the first byte of the message must be written to the transmit buffer. When a section of code decides to transmit a message serially, it loads the global message pointer with the address of the first byte of the message, enables the transmit interrupt, and calls the TxD interrupt procedure. Calling the TxD interrupt procedure writes the first byte to the transmit buffer to initiate transmit interrupts. This can be done by calling PL/M's built-in procedure CAUSE$INTERRUPT.

The transmit interrupt routine checks each byte before it writes it to the transmit buffer. The last character in each message is a 0, so if the character fetched is 0, the transmit interrupt is disabled and the character is ignored.

**USING THE LPM WITH THE INTELLEC® MICROCOMPUTER DEVELOPMENT SYSTEM, SERIES II OR SERIES III**

A special driver program was written for the MDS to communicate to the LPM. This program, called WRITE, reads a specified file from the disk, expands any TAB characters, and transmits the data through Serial Channel 2 to the LPM. Serial Channel 2 was chosen because CTS and RTS are brought out to the RS-232 connector. The WRITE program is listed in appendix B. It was also necessary to modify the boot ROM of the development system so that Serial Channel 2 initializes with RTS false and a bit rate of 9600 bps.
Figure 31. Flow Chart of the BUFF$OUT Procedure

Figure 32. Flow Chart for TxD Interrupt Procedure
APPENDIX A
LISTING OF THE LINE PRINTER MULTIPLEXER SOFTWARE
SERIES-III PL/M-86 VI 0 COMPILATION OF MODULE MAINMOD
OBJECT MODULE PLACED IN /MAIN OBJ
COMPILER INVOKED BY PLM86 86 F1 MAIN SRC

/*============================================================*/
/* MAIN MODULE FOR THE LINE PRINTER MULTIPLEXER */
/*============================================================*/

$DEBUG MAIN$MOD DD.

/**************************** PORT 1 BIT CONFIGURATION */
/* * BUFFER FULL CTS ADDRESS RTS TWO WIRE HANDSHAKE */
/* * B7 B6 B5 B4 B3 B2 B1 B0 */

DECLARE LIT LITERALLY 'LITERALLY', TRUE LIT 'OFFH', FALSE LIT '0', FOREVER LIT 'WHILE 1',
CMD$1 LIT '0', /*8256 REGISTERS*/
CMD$2 LIT '2',
CMD$3 LIT '4',
MODE LIT '6',
PORT$1%CTRL LIT '8',
SET$INT LIT '0AH',
INT$EN LIT '0AH',
RST$INT LIT '0CH',
INT$ADDR LIT '0CH',
TX$BUFF LIT '0EH',
RX$BUFF LIT '0EH',
PORT$1 LIT '10H',
PORT$2 LIT '12H',
DEBOUNCE$TIMER LIT '14H',
SCAN$TIMER LIT '1AH',
RECEIVE$TIMER LIT '1CH',
STATUS$REG LIT '1EH',
SCAN$INT LIT '40H',
DEBOUNCE$INT LIT '01H',
RECEIVER$INT LIT '10H',
TIME$OUT$INT LIT '08H',
TRANSMIT$INT LIT '20H',
EMPTY LIT '0',
INUSE LIT '1',
FULL LIT '2',
RTS LIT '((INPUT(PORT$1) AND 04H)'.

5-407
BEGIN         LABEL         PUBLIC,

          TEMP       BYTE,        PUBLIC,
          SCAN*DELAY BYTE,        PUBLIC,
          DEBOUNCE*DELAY BYTE,    PUBLIC,
          RECEIVE*DELAY BYTE,    PUBLIC,
          PORTS PTR BYTE,        PUBLIC, /* PEN EP L1 L0 B3 B2 B1 B0 */
          SERIAL*FORMAT(8) BYTE PUBLIC,
          MESSAGE*PTR POINTER EXTERNAL,
          J BYTE EXTERNAL,
          OK(1) BYTE EXTERNAL,
          BUFFER*STATUS BYTE EXTERNAL,

/*************************************************************************/

BEGIN LABEL PUBLIC,

BEGIN         LABEL         PUBLIC,

          TEMP       BYTE,        PUBLIC,
          SCAN*DELAY BYTE,        PUBLIC,
          DEBOUNCE*DELAY BYTE,    PUBLIC,
          RECEIVE*DELAY BYTE,    PUBLIC,
          PORTS PTR BYTE,        PUBLIC, /* PEN EP L1 L0 B3 B2 B1 B0 */
          SERIAL*FORMAT(8) BYTE PUBLIC,
          MESSAGE*PTR POINTER EXTERNAL,
          J BYTE EXTERNAL,
          OK(1) BYTE EXTERNAL,
          BUFFER*STATUS BYTE EXTERNAL,

/*************************************************************************/

POWER$ON PROCEDURE EXTERNAL;
END POWER$ON;

LOAD$INT$TABLE PROCEDURE EXTERNAL.
END LOAD$INT$TABLE.

*************************************************************************/

CONFIGURE PROCEDURE, /* Initialize bit rate and data format*/
TEMP=SERIAL$FORMAT(SHR(PORT$PTR,3»,
OUTPUT(TRANSMIT$INT)=SHL(TEMP,;/)
AND 0EOH) OR 03H),
OUTPUT(RECEIVE$TIMER)=70,
OUTPUT(PORTS1)=(INPUT(PORTS1)
AND 0BFH), I*Send CTS TRUE*
DO WHILE RECEIVE$DELAY=TRUE. /* Wait here while receiving serial data */
END,

*************************************************************************/

INITIALIZE SERIAL RECEIVER
*************************************************************************/

SET THE BIT RATE AND DATA FORMAT FOR THE SERIAL PORT
*************************************************************************/

INITIALIZE SERIAL RECEIVER
*************************************************************************/

CALL CONFIGURE, /* Initialize 8256 serial port*/
OUTPUT(CMD$3)=0COH, /*Enable serial receiver*/
OUTPUT(RECEIVE$TIMER)=70, /*Enable RECEIVE and TIME$OUT interrupts*/
OUTPUT(PORTS1)=(INPUT(PORTS1)
AND 0BFH), /*Send CTS TRUE*/
DO WHILE RECEIVE$DELAY=TRUE. /* Wait here while receiving serial data */
END,

*************************************************************************/

IF (BUFFER*STATUS<>FULL) THEN
*************************************************************************/

OUTPUT(PORT$1)=(INPUT(PORT$1) AND 0BFH), /*Send CTS TRUE*/
DO WHILE RECEIVE$DELAY=TRUE. /* Wait here while receiving serial data */
END,

*************************************************************************/

/* After 16 seconds of not receiving a character, proceed */

OUTPUT(SET$INT)=TRANSMIT$INT, /* Send the terminating message */
J=0,
MESSAGE*PTR= OK(0),
CAUSE$INTERRUPT (45H),

5-408 210907-002
AP-153

PL/M-86 Compiler MAINMOD

```
26 2 OUTPUT(PORT$1)=(INPUT(PORT$1) OR 40H), */Send CTS FALSE*/
27 2 OUTPUT(RST$INT)=16H,  */Clear RECEIVER and TIMER Intorrupts*/
28 2 OUTPUT(CMD$3)=40H,  */Disable serial receiver*/
29 2 END INIT$RECEIVER,

/***************************************************************************/
* DEBOUNCE RTS *
***************************************************************************/
30 1 DEBOUNCE PROCEDURE,
31 2 DEBOUNCE$DELAY=TRUE,
32 2 OUTPUT(DEBOUNCE$TIMER)=10, */ 10 msec debounce time delay */
33 2 OUTPUT(SET$INT)=DEBOUNCE$INT,
34 2 DO WHILE DEBOUNCE$DELAY=TRUE;
35 3 END.
36 2 IF RTS=0 THEN CALL INIT$RECEIVER,
38 2 END DEBOUNCE,

/***************************************************************************/
* BEGIN MAIN PROGRAM *
***************************************************************************/
39 1 BEGIN CALL POWER$ON,
40 1 CALL LOAD$INT$TABLE,
41 1 ENABLE,
42 1 DO FOREVER,
43 2 SCAN$DELAY=TRUE,
44 2 OUTPUT(SCAN$TIMER)=100,  */Spend 100 msec on each serial port sampling RTS*/
45 2 OUTPUT(SET$INT)=SCAN$INT,
46 2 DO WHILE SCAN$DELAY=TRUE,  */Sample RTS*/
47 3 IF RTS=0 THEN CALL DEBOUNCE,
48 3 END;
49 3 END;
50 2 TEMP=INPUT(PORT$1),  */Increment PORT$PTR*/
51 2 PORT$PTR=TEMP AND 38H,
52 2 TEMP=TEMP AND NOT 38H,
53 2 PORT$PTR=(PORT$PTR+8) AND 38H,
54 2 OUTPUT(PORT$1)=TEMP OR PORT$PTR,  */Look at next serial port*/
55 2 END,  */DO FOREVER*/
56 1 END MAIN$MOD.
```

MODULE INFORMATION.

CODE AREA SIZE = 011CH 284D

END OF PL/M-86 COMPILATION
PL/M-B6 COMPILER

SERIES-III PL/M-B6 V1.0 COMPILENOF MODULE INTMOD
OBJECT MODULE PLACED IN F1 INT OBJ
COMMAND INVOKED BY PL/M-B6 F1 INT SRC

/* ************************************************** */
/* INTERRUPT MODULE CONTAINS ALL INTERRUPT Routines */
/* PLUS LOAD INTERRUPT TABLE PROCEDURE */
/* ************************************************** */

#define DEBUG
#define NOLIST

procedure INTMOD DO,
#define

DECLARE
ESC LIT '1B',
SCANDELAY BYTE EXTERNAL,
DEBOUNCEDELAY BYTE EXTERNAL,
RECEIVEDelay BYTE EXTERNAL,
MESSAGESPTR POINTER EXTERNAL,
J BYTE EXTERNAL,

/* ************************************************** */
/* MESSAGES SENT TO SERIAL PORTS */
/* ************************************************** */
OK (*) BYTE PUBLIC DATA ('TRANSMISSION COMPLETE', OAH, OAH, 00),
BREAK (*) BYTE PUBLIC DATA ('BREAK DETECT ERROR', OAH, ODH, 00),
PARITY (*) BYTE PUBLIC DATA ('PARITY ERROR DETECTED', OAH, ODH, 00),
FRAME (*) BYTE PUBLIC DATA ('FRAMING ERROR DETECTED', OAH, ODH, 00),
OVERRUN(*) BYTE PUBLIC DATA ('OVER RUN ERROR DETECTED', OAH, ODH, 00),

/* ************************************************** */
/* EXTERNAL PROCEDURES CALLED BY THE INTERRUPT ROUTINES */
/* ************************************************** */
ERROR PROCEDURE (STATUS) EXTERNAL,
DECLARE STATUS BYTE;

END ERROR,

PROGRAM PROCEDURE EXTERNAL,
DECLARE CHAR BYTE;

END PROGRAM,

BUFFIN PROCEDURE (CHAR) EXTERNAL,
DECLARE CHAR BYTE,
END BUFFIN,

BUFFOUT PROCEDURE BYTE EXTERNAL,
END BUFFOUT,

/* ************************************************** */
/* LOAD THE INTERRUPT TABLE */
/* ************************************************** */
LOADINTSTABLE PROCEDURE PUBLIC.
CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

END

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.

CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
CALL SET$INTERRUPT (43H, RECEIVE$TIME),
CALL SET$INTERRUPT (44H, RXD),
CALL SET$INTERRUPT (45H, TXD),
CALL SET$INTERRUPT (46H, SCAN$TIME),
CALL SET$INTERRUPT (47H, LP),
END LOAD$INT$TABLE.
IF STATUS = 0 THEN CALL ERROR (STATUS), ELSE IF CHAR = ESC THEN CALL PROGRAM, ELSE CALL BUFF$IN (CHAR), OUTPUT(CMD$3) = BBH. END RXD.

****************************************************
***************
* SEND A BYTE TO THE LINE PRINTER *
****************************************************

LP: PROCEDURE INTERRUPT 47H, ENABLE, OUTPUT(PORTS21 = BUFF$OUT, OUTPUT(CMD$3) = BBH, END LP,

TXD: PROCEDURE INTERRUPT 45H, DECLARE MESSAGE BASED MESSAGE$PTR (I) BYTE, I BYTE, ENABLE, I = MESSAGE (J), IF I > 0 THEN OUTPUT(TX$BUFF) = I, ELSE OUTPUT(RST$INT) = TRANSMIT$INT, J = J + 1, OUTPUT(CMD$3) = BBH. END TXD.

END INT$MOD.

MODULE INFORMATION

CODE AREA SIZE = 01BDH 445D
CONSTANT AREA SIZE = 007BH 120D
VARIABLE AREA SIZE = 0003H 3D
MAXIMUM STACK SIZE = 0022H 34D
181 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
PL/M-86 COMPILER

BUFFMOD

SERIES-III PL/M-86 V1 0 COMPIlATION OF MODULE BUFFMOD
OBJECT MODULE PLACED IN F1 BUFF OBJ
COMPIlER INVOKED BY PLM86 86 F1 BUFF SRC

******************************************************************************
* BUFFER MODULE INSERTS AND REMOVES CHARACTERS FROM FIFO *
* REPORTS SERIAL RECEIVE ERRORS AND *
* RE-PROGRAMS SERIAL PORTS *
******************************************************************************

$DEBUG
BUFFMOD DO.
$NOLIST

3 1 DECLARE MESSAGE$PTR POINTER PUBLIC.
   J BYTE PUBLIC.
   OK(I) BYTE EXTERNAL.
   BREAK(1) BYTE EXTERNAL.
   PARITY(1) BYTE EXTERNAL.
   FRAME(1) BYTE EXTERNAL.
   OVER$RUN(1) BYTE EXTERNAL.
   SERIAL$FORMAT(1) BYTE EXTERNAL.
   PORT$PTR BYTE EXTERNAL.
   FIFO(192) BYTE.
   IN$POINTER WORD PUBLIC.
   OUT$POINTER WORD PUBLIC.
   BUFFER$STATUS BYTE PUBLIC.

******************************************************************************
* INSERT CHARACTER INTO FIFO *
******************************************************************************

4 1 BUFF$IN PROCEDURE (CHAR) PUBLIC.
5 2 DECLARE CHAR BYTE.
6 2 IF (((BUFFER$STATUS=EMPTY) AND ((CHAR=LF) OR (CHAR=CR)))
7 2 THEN
8 3 DO.
9 3 OUTPUT(SET$INT)=HANDSHAKE$INT. /* Enable two-wire handshake interrupt */
10 3 BUFFER$STATUS=INUSE. /* Output NULL character to get */
11 3 OUTPUT(PORT$2)=O. /* the interrupt started */
12 3 END.
13 2 FIFO(IN$POINTER)=CHAR. /* Put CHAR into FIFO and increment pointer */
14 2 IN$POINTER=((IN$POINTER+I) AND 1FFFF).
15 2 IF (((IN$POINTER+4) AND 1FFFF)=OUT$POINTER) /* If the buffer is full stop reception */
16 2 THEN
17 2 DO. /* Send CTS FALSE, and light up buffer full LED */
BUFFMOD

16 3  OUTPUT(PORT$1)=((INPUT(PORT$1) OR 40H) AND 7FH),
17 3  BUFFER$STATUS=FULL,
18 3  END,
19 2  END BUFF$IN,

/******************************************************************************************
* REMOVE CHARACTER FROM FIFO
* ******************************************************************************************/
20 1  BUFF$OUT PROCEDURE BYTE PUBLIC,
21 2  DECLARE CHAR BYTE;
22 2  CHAR=FIFO(OUT$POINTER),
23 2  OUT$POINTER=((OUT$POINTER+1) AND 1FFH),
24 2  IF OUT$POINTER=IN$POINTER /* If the buffer is empty disable the output to LP */
25 2  THEN
26 3  DISABLE,
27 3  BUFFER$STATUS=EMPTY,
28 3  OUTPUT(RST$INT)=HANDSHAKE$INT,
29 3  ENABLE,
30 3  END,

/* If the buffer is ready to fill up again then send CTS TRUE */
31 2  ELSE IF ((BUFFER$STATUS=FULL) AND (((OUT$POINTER-200) AND 1FFH)=IN$POINTER))
32 2  THEN
33 3  DO; /* Turn off buffer-full LED and turn on CTS */
34 3  OUTPUT(PORT$1)=((INPUT(PORT$1) OR 8FH) OR 00H),
35 3  BUFFER$STATUS=INUSE,
36 3  END,
37 2  RETURN CHAR,
38 2  END BUFF$OUT,

/******************************************************************************************
* SEND ERROR MESSAGE TO SERIAL PORT
* ******************************************************************************************/
39 1  ERROR PROCEDURE (STATUS) PUBLIC,
40 2  DECLARE STATUS BYTE;
41 2  MESSAGE BASED MESSAGE$PTR(I) BYTE,
42 2  IF (STATUS AND 02H)<>0
43 3  THEN
44 3  STATUS=2,
45 3  ELSE IF (STATUS AND 04H)<>0
46 3  THEN
47 3  STATUS=3,
48 3  ELSE IF (STATUS AND 08H)<>0
49 3  THEN
50 3  STATUS=4,
51 3  ELSE IF (STATUS AND 01H)<>0
52 3  THEN
53 3  STATUS=1,
54 3  DO CASE STATUS,
MESSAGESPTR=~OVERSRUN(O),
MESSAGESPTR=~PARITY(O),
MESSAGESPTR=~BREAK(O),
END,
RESET SERIAL PORT CONFIGURE BYTE
********************************************************************************
PROGRAM PROCEDURE PUBLIC,
DECLARE TEMP BYTE,
    CHAR BYTE,
DO WHILE (INPUT(STATUSSREG) AND 40H)=0, /* Wait for next byte */
END,
CHAR=INPUT(RX$BUFF),
IF CHAR=O /* If second byte is 0, exit program mode */
THEN
    OUTPUT(RECEIVESTIMER)=70,
    CALL BUFF$IN (CHAR),
    RETURN,
END;
TEMP=(CHAR AND OFH),
DO WHILE (INPUT(STATUSSREG) AND 40H)=0,
END,
TEMP=(INPUT(RX$BUFF) AND OFH) OR SHL(TEMP,4),
SERIAL$FORMAT (SHR(PORT$PTR,3))=TEMP,
END PROGRAM.
END BUFF$MOD.

MODULE INFORMATION:
CODE AREA SIZE = 01E4H  4940
CONSTANT AREA SIZE = 0000H  0D
VARIABLE AREA SIZE = 2008H  8203D
MAXIMUM STACK SIZE = 000AH  10D
189 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS
END OF PL/M-86 COMPILATION
POWER ON INITIALIZATION OF THE LINE PRINTER MULTIPLEXER

DECLARE BUFFERSSTATUS BYTE EXTERNAL,
IN$POINTER WORD EXTERNAL,
OUT$POINTER WORD EXTERNAL,
PORTSPTR BYTE EXTERNAL,
SERIAL$FORMAT(8)BYTE EXTERNAL.

POWER$ON PROCEDURE PUBLIC.
DECLARE I BYTE.
DISABLE.

/* INITIALIZE THE MUART */
OUTPUT(CMD$1)=0100001IB, /*8086 MODE, FREQ=1KH z, 1 STOP BIT, &
7 BITS/CHARACTER*/
OUTPUT(CMD$2)=10110100B, /*ODD PARITY, SYSTEM CLOCK=1 024 MH z, &
9600 bps*/
OUTPUT(CMD$3)=0111111IB, /*CLEAR C MDA $ REGISTER*/
OUTPUT(CMD$3)=1011000IB, /*RESET, INTERRUPT ACKNOWLEDGE ENABLE, &
NESTED INTERRUPT MODE*/
OUTPUT(PORT$1)=10000101B, /*CASCADE TIMERS 35 FOR THE
RECEIVE*TIME*OUT TIMER, BYTE OUTPUT MODE*/
OUTPUT(PORT$1)=01111111B, /*PORT 1 RTS=INPUT, THE REST ARE OUTPUTS*/
OUTPUT(PORT$1)=11000000B, /*POINT TO THE FIRST PORT, CTS IS Fz*,
AND BUFFER IS NOT FULL*/

/* INITIALIZE FLAGS, VARIABLES, AND ARRAYS */
BUFFERSSTATUS=EMPTY,
IN$POINTER=0, OUT$POINTER=0,
PORTSPTR=0.
DO I=0 TO 7,
PL/M-86 COMPILER

19 3 SERIAL_FORMAT(I)=10010100B. /* ON POWER-UP ALL EIGHT SERIAL PORTS
END TO 9600 bps, ODD PARITY, AND
7 BITS/CHARACTER*/

20 3 END.

21 2 END POWER_ON.

22 1 END PON_MOD.

MODULE INFORMATION

<table>
<thead>
<tr>
<th>CODE AREA SIZE</th>
<th>CONSTANT AREA SIZE</th>
<th>VARIABLE AREA SIZE</th>
<th>MAXIMUM STACK SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0058H</td>
<td>0000H</td>
<td>0001H</td>
<td>0002H</td>
</tr>
</tbody>
</table>

98 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
APPENDIX B
LISTING OF THE WRITE PROGRAM
ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE WRITE MOD
OBJECT MODULE PLACED IN F1 WRITE OBJ
COMPILER INVOKED BY F2 PLM80 F1 WRITE SRC

*DEBUG
WRITE*MOD DO.

iscriminated

WRITE PROGRAM READS A FILE FROM A DISK AND COPIES
IT TO SERIAL CHANNEL 2 ON THE MDS

DECLARE LIT LITERALLY 'LITERALLY',
USART*DATA LIT 'OF6H',
USART*STATUS LIT 'OF7H',
RTS LIT '20H',
TXEN LIT '01H',
RXE LIT '04H',
CR LIT '0DH',
LF LIT '0AH',
TAB LIT '09H',
SP LIT '20H',
ESC LIT '1BH',
FORM$FEED LIT '0CH',

DECLARE AFT$IN ADDRESS,
FILENAME(15) BYTE,
STATUS ADDRESS,
BUFFER(32000) BYTE,
ACTUAL ADDRESS,
CHAR$COUNT ADDRESS,
BYE(42) BYTE

(INITIAL
'WROTE ',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 TO THE LINE PRINTER', OAH, ODH),
I ADDRESS,
J BYTE.

EXTERNAL SYSTEM LIB PROCEDURES

OPEN
PROCEDURE (AFTNPTR, FILE, ACCESS, MODE, STATUS) EXTERNAL;
DECLARE (AFTNPTR, FILE, ACCESS, MODE, STATUS) ADDRESS.
END OPEN.

READ
PROCEDURE (AFTN, BUFFER, COUNT, ACTUAL, STATUS) EXTERNAL;
DECLARE (AFTN, BUFFER, COUNT, ACTUAL, STATUS) ADDRESS;
END READ.

WRITE
PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL,
DECLARE (AFTN, BUFFER, COUNT, STATUS) ADDRESS;
END WRITE,
CLOSE
PROCEDURE (AFTN, STATUS) EXTERNAL,
DECLARE (AFTN, STATUS) ADDRESS;
END CLOSE,
ERROR
PROCEDURE (ERRNUM) EXTERNAL,
DECLARE (ERRNUM) ADDRESS,
END ERROR,
EXIT
PROCEDURE EXTERNAL,
END EXIT,
PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL,
DECLARE (AFTN, BUFFER, COUNT, STATUS) ADDRESS;
END WRITE,
CLOSE
PROCEDURE (AFTN, STATUS) EXTERNAL,
DECLARE (AFTN, STATUS) ADDRESS;
END CLOSE,
ERROR
PROCEDURE (ERRNUM) EXTERNAL,
DECLARE (ERRNUM) ADDRESS,
END ERROR,
EXIT
PROCEDURE EXTERNAL,
END EXIT,
PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL,
DECLARE (AFTN, BUFFER, COUNT, STATUS) ADDRESS;
END WRITE,
CLOSE
PROCEDURE (AFTN, STATUS) EXTERNAL,
DECLARE (AFTN, STATUS) ADDRESS;
END CLOSE,
ERROR
PROCEDURE (ERRNUM) EXTERNAL,
DECLARE (ERRNUM) ADDRESS,
END ERROR,
EXIT
PROCEDURE EXTERNAL,
END EXIT,
PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL,
DECLARE (AFTN, BUFFER, COUNT, STATUS) ADDRESS;
END WRITE,
CLOSE
PROCEDURE (AFTN, STATUS) EXTERNAL,
DECLARE (AFTN, STATUS) ADDRESS;
END CLOSE,
ERROR
PROCEDURE (ERRNUM) EXTERNAL,
DECLARE (ERRNUM) ADDRESS,
END ERROR,
EXIT
PROCEDURE EXTERNAL,
END EXIT,
IF BUFFER(0)=FORM$FEED /* If the first character is a form feed remove it. Form feeds are inserted at the end of a file */
THEN
DO,
BUFFER(0)=00H,
CHAR$COUNT=-1,
END.
DO I = 0 TO (ACTUAL - 1).
IF (BUFFER(I)=TAB) /* Replace TAB characters with the appropriate number of spaces */
THEN
DO,
CALL TXRDY,
OUTPUT(USART$DATA)=SP,
CHAR$COUNT=CHAR$COUNT+1,
END.
ELSE /* If the character is not an ESC or TAB then output it */
DO,
CALL TXRDY,
OUTPUT(USART$DATA)=BUFFER(I),
IF (BUFFER(I)=CR OR BUFFER(I)=LF)
THEN /* Only increment CHAR$COUNT for printable characters */
CHAR$COUNT=CHAR$COUNT+1,
ELSE /* If outputting ESC, then output a 0 next so the LPM does not get re-programmed */
DO J=0 TO 1.
CALL TXRDY,
OUTPUT(USART$DATA)=O.
END,
ELSE
DO,
CALL TXRDY,
OUTPUT(USART$DATA)=BUFFER(I),
IF (BUFFER(I):IFH AND BUFFER(I):7FH)
THEN /* Only increment CHAR$COUNT for CR or LF */
CHAR$COUNT=O,
ENDIF.
IF ACTUAL = 32000 /* If the file is more than 32K, get some more data */
THEN
GO TO REPEAT,
ELSE /* Terminate file with CR, LF, and FF */
CALL TXRDY,
OUTPUT(USART$DATA)=CR,
CALL TXRDY.
PL/M-80 COMPILER

73 1 OUTPUT(USART$DATA)=LF.
74 1 CALL TXRDY.
75 1 OUTPUT(USART$DATA)=FORM*FEED.
76 1 OUTPUT(USART$STATUS)=RXE OR TXEN. /* Shut off RTS */
77 1 CALL CLOSE (AFT$IN. STATUS).
78 1 DO I=0 TO 14; /* Output sign off message */
79 2 IF FILENAME(I)=CR
80 2 THEN
81 2 GO TO SKIP.
82 2 END.
83 1 SKIP.
84 1 END.
85 1 DONE.
86 1 NEXT: CALL ERROR(STATUS).
87 1 END WRITE$MOD.

MODULE INFORMATION.

CODE AREA SIZE = 0209H 521D
VARIABLE AREA SIZE = 7D44H 32060D
MAXIMUM STACK SIZE = 0006H 6D
191 LINES READ
0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
APPENDIX C
MUART REGISTERS
8085 Mode: AD3 AD2 AD1 AD0
8086 Mode: AD4 AD3 AD2 AD1

0000

L1 L0 S1 S0 BRKI BITI 8086 FRQ

Command 1

Timer Frequency Select
8086 Mode Enable
Interrupt on Bit Change
Break-in Detect Enable
Stop Bit Length
Character Bit Length

0001

PEN EP C1 C0 B3 B2 B1 B0

Command 2

Baud Rate Select
System Clock Divider
Even Parity
Parity Enable

0010

SET RxE IAE NIE END SBRK TBRK RST

Command 3

Software Reset
Transmit Break
Single Character Break
End of Interrupt
Nested Interrupt Enable
Interrupt Acknowledge Enable
Receiver Enable
Bit Set/Reset

0011

T35 T24 T5C CT3 CT2 P2C2 P2C1 P2C0

Mode

Port 2 Control
Counter/Timer 2
Counter/Timer 3
Timer 5 Retriggerable
Cascade Counter/Timer 2 & 4
Cascade Counter/Timer 3 & 5
### Port 1 Control

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Output/Input of Port 1 pins</td>
</tr>
</tbody>
</table>

#### Set Interrupts

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L7</td>
<td>L6</td>
<td>L5</td>
<td>L4</td>
<td>L3</td>
<td>L2</td>
<td>L1</td>
<td>L0</td>
<td>Enable</td>
</tr>
</tbody>
</table>

#### Reset Interrupts

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L7</td>
<td>L6</td>
<td>L5</td>
<td>L4</td>
<td>L3</td>
<td>L2</td>
<td>L1</td>
<td>L0</td>
<td>Disable</td>
</tr>
</tbody>
</table>

#### Interrupt Enable

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L7</td>
<td>L6</td>
<td>L5</td>
<td>L4</td>
<td>L3</td>
<td>L2</td>
<td>L1</td>
<td>L0</td>
<td>Interrupt Levels Enabled</td>
</tr>
</tbody>
</table>

#### Interrupt Address

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>D4</td>
<td>D3</td>
<td>D2</td>
<td>0</td>
<td>0</td>
<td>Interrupt Level in Service</td>
</tr>
</tbody>
</table>

#### Modification

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RS4</td>
<td>RS3</td>
<td>RS2</td>
<td>RS1</td>
<td>RS0</td>
<td>TME</td>
<td>DSC</td>
<td>Disable Start Bit Check</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RS4</td>
<td>RS3</td>
<td>RS2</td>
<td>RS1</td>
<td>RS0</td>
<td>TME</td>
<td>DSC</td>
<td>Transmitter Mode Enable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Receiver Sampling Point</td>
</tr>
</tbody>
</table>
Status Register (Read only)

1111  INT  RBF  TBE  TRE  BD  PE  OE  FE

- Framing Error/Transmission Mode Indication
- Overrun Error
- Parity Error
- Break Detect or Break-in Detect
- Transmitter Register Empty
- Transmitter Buffer Empty
- Receiver Buffer Full
- Interrupt Pending

Response to INTA

8085-Mode (RST-instruction in response to INTA)

\[
\begin{array}{ccccccc}
1 & 1 & D5 & D4 & D3 & 1 & 1 & 1
\end{array}
\]

Interrupt Level

8086-Mode (Interrupt Vector in response to second INTA)

\[
\begin{array}{ccccccc}
0 & 1 & 0 & 0 & 0 & D2 & D1 & D0
\end{array}
\]

Interrupt Level
8256AH Multifunction Peripheral Simplifies Microcomputer I/O Design

CHRISTOPHER SCOTT
8256AH Multifunction Peripheral Simplifies Microcomputer I/O Design

CONTENTS

INTRODUCTION
Description of the 8256AH

HARDWARE DESCRIPTION
8256AH/80186 System Design
RS-232C Hardware Interface
Parallel I/O with Handshaking

SOFTWARE DESCRIPTION
Serial RS-232C Interface
RS-232C Control Signals Interrupt Structure

CONCLUSION

APPENDIX A.
Software Listing

FIGURES
1a. System Block Diagram Without 8256AH
1b. System Block Diagram With 8256AH
2. 8256AH Internal Block Diagram
3. 8256AH / 80186 Schematic
4. Block Diagram of the 8256AH Serial RS-232C Interface Software Structure
5. 8256AH Interrupt Source To Priority Level Map
6. Port 1 RS-232C Pin Definition
7. Receive Data Interrupt Service Routine Software Flowchart
8. Transmit Data Interrupt Service Routine Software Flowchart

Additional Sources of Information
Ap Note 153 Designing with the 8256AH
INTRODUCTION
A primary goal of microcomputer system design is to provide the required functionality and flexibility with the fewest number of components. The 8256AH Multifunction Peripheral is designed specifically to meet these conflicting requirements. Four of the most common microcomputer system functions, previously requiring up to four separate MSI or LSI devices, are combined into one LSI device. The 8256AH incorporates a serial asynchronous communication channel, two 8-bit parallel I/O ports, five 8-bit timer/counters and an eight level priority interrupt controller in one 40 pin package. Its flexible design allows it to directly interface to most microprocessors, including Intel's MCS-85, iAPX-86, iAPX-88, iAPX-186 and iAPX-188, and the MCS-48 and MCS-51 family of single-chip microcomputers.

This application note describes using the 8256AH to implement a Data Terminal Equipment (DTE) RS-232C serial asynchronous communication link with the control signals necessary to interface to a Bell 103/212A modem. The interface requires a total of nine interface signals. Three of these signals, TxD, RxD and CTS, are provided by the UART section of the 8256AH. The balance of the RS-232C interface signals are implemented using six of the independently programmable parallel PORT 1 lines. In addition, the application design provides an eight bit parallel I/O port with handshaking signals. The on-chip priority interrupt controller enables the RS-232C serial interface an the parallel interface to operate on an interrupt basis. The 8256AH uniquely addresses the complexities of implementing an RS-232C communications interface. By utilizing the built-in hardware and software features of the 8256AH, the design achieves flexibility with simplicity, qualities often exclusive of one another.

Previous solutions required four components to implement the same interface. Figure 1 illustrates the basic system block diagrams for the two solutions. In Figure 1a the 8251A Programmable Communications Interface provides the UART serial communications interface. The 8254 Programmable Interval Timer provides baud rate generation and other timing functions, such as time-out loops, needed for software support of an RS-232C interface. These are especially needed if the RS-232C channel is to operate in an interrupt system environment. The 8255A Programmable Peripheral Interface provides parallel I/O with one port dedicated to the RS-232C control signals. The 8259A Priority Interrupt Controller provides an eight level priority interrupt structure. This represents a total of 120 device pins compared to the single 40 pin 8256AH, and 465 mA current requirement verses a 160 mA current requirement. Figure 1b represents the 8256AH solution incorporating the four functions in one package.

In some data communication applications only three lines - ground, Transmit Data and Receive Data - are used for serial communication. An example is communication between an ASCII terminal or printer and a personal computer. These devices are usually located close to one another and in general do not require the additional control signals of the EIA RS-232C serial communications standard. In other data communications applications, this same equipment requires that the integrity of the serial communications link be constantly monitored. This enables the host system to control the data transmission at all times, whether it be a host computer or intelligence local to a communications device, such as an ASCII terminal. The need for control and monitoring of the serial line is particularly important when the communications link is over telephone lines using a modem. In a Switched Network, where a number of serial devices share the same communications line, the control signals are crucial to the system's multiplexing the single line.

![Figure 1a. System Block Diagram Without the 8256AH](image-url)
This Application Note assumes that the reader is familiar with the 8256AH Data Sheet and with the RS-232C communication protocol and terminology. A complete software listing is provided in Appendix A. A complete description and definition of the RS-232C interface standard may be found in the book “Data Communications: A Users Guide” by Kenneth Sherman, Reston Publishing 1981.

DESCRIPTION OF THE 8256AH

The 8256AH combines four commonly used peripheral functions into one device (see Figure 2);

1. A full-duplex, double-buffered serial asynchronous Receiver/Transmitter (UART) with an on-chip Baud Rate Generator.

2. Two 8-bit parallel I/O ports; One bit programmable, One nibble programmable.

3. Five 8-bit timer/counters; 4 can be cascaded to form 2 16-bit timer/counters

4. An 8-level priority interrupt controller.

The 8256AH uses the standard bus control signals compatible with Intel's family of peripherals and microprocessors. The microprocessor interface utilizes a multiplexed address/data bus. Four of the eight address/data lines are used to generate the register address. This enables all of the 8256AH's functionality to be contained in a 40 pin package while retaining direct register addressing.

The sixteen directly addressable internal read/write registers provide control for all of the 8256AH's various functions. Fourteen of the registers are read/write, one, the Status Register, is read only and one, the Modification Register, is write only. Three Command Registers configure the operating environment including the type of CPU, 8 or 16 bit, and system clock frequency. Command Register Three provides bit set-reset capability for control of such functions as End of Interrupt, Nested Interrupts, Interrupt Acknowledge and UART Receive Enable. The Status Register provides all information about the UART's transmitter and receiver, and the state of the interrupt (INT) output pin to the microprocessor. The Mode Register defines the configuration of the two parallel ports and the five timer/counters. The write only Modification Register is used to alter two standard functions of the receiver, start bit sampling and to enable a special indicator flag for half-duplex operation. In addition, six registers control the two parallel ports. Two registers provide for UART Transmit and Receive Buffers. Ten registers are used for timer/counter interface, and four registers provide for Priority Interrupt Controller support.

The UART section of the 8256AH features a full-duplex double-buffered transmitter and receiver with separate control registers. The internal baud rate generator provides the thirteen common sampling rates from 50 bps to 19.2 kbps. An external baud rate clock can also be used, with programmable choice of 1X, 32X or 64X sampling rates.

The two parallel I/O ports can be configured as two independent 8-bit parallel I/O ports, or as one 8-bit
parallel port with ACK/OBF and STB/IBF two wire handshake signals. In the latter configuration, the six remaining I/O lines may be used as either independently programmable I/O lines, or as predefined special function inputs and/or outputs, such as a second external interrupt input or timer/counter inputs.

The five 8-bit programmable timer/counters are binary presettable downcounters. In addition, an independent on-chip Baud Rate Generator is provided for the UART. The clock sources for the timers/counters may be either internal or external - via programmed parallel port pins - depending upon whether they are configured as timers or counters. Four of the timer/counters may be cascaded to form two 16-bit timer/counters. Each of the five timer/counters has its own read/write register.

The eight level priority interrupt controller has twelve possible interrupt sources. Ten of the sources are internal and two are external. One of the external interrupt sources is a fixed pin; EXTINT. The second is one of the parallel Port 1 pins which can be programmed as an external interrupt source. The twelve interrupt sources are internally mapped to the eight interrupt priority levels.

The interrupt controller may be programmed to operate in either a Normal or Nested Interrupt Mode. In Normal Mode any interrupt may interrupt any other interrupt based upon the enable/disable bits in the Interrupt Enable, or Mask, Register. In the Nested Mode only an interrupt of higher priority may interrupt one of lower priority, again based upon the bits in the Enable Register.

The 8256AH interrupt structure supports both 8085 and 8086 interrupt vectoring methods via the INTR and INTA signals. In vectored interrupt operation the 8256AH places the interrupt vector address on the data bus during the INTA sequence. In addition the 8256AH supports non-vectored interrupt interfaces, such as MCS-51 and MCS-48 systems. In non-vectored interrupt applications the host system simply reads the interrupt vector address from the Interrupt Address Register of the 8256AH. Reading the interrupt address register clears the INT pin and acknowledges that the interrupt has been serviced. This is the functional equivalent to an INTA sequence generated by the host processor.
DESIGN DESCRIPTION

Hardware Description

Figure 3 shows a block diagram of this application's system design. The microprocessor used is an iAPX-186 with two 8256AH's for parallel and serial I/O, as well as for providing a variety of system support functions. One 8256AH is used to implement both the RS-232C modem interface and provide multiplexed parallel I/O. The system uses the Intel 957B System Monitor for control of the system hardware and software development support. The second 8256AH is used for basic serial communication between an ASCII terminal and the Intel 957B System Monitor residing in 16K bytes of EPROM. The two 8256AHs provide a total of six I/O channels - two UARTs and four parallel I/O ports.

When one of the 8256AHs is configured for the serial RS-232C interface, one of its parallel ports, Port 1 pins 2-7, provides control signals for the serial interface. Four of the RS-232C control signals (CTS, DSRS, DSR, and CD) are OR'd to the EXTINT pin of the 8256AH. If any of these signals change from their defined state, an interrupt is generated to the 8256AH. The modem driver software then responds to the interrupt by reading the Port 1 register, determines the signal generating the interrupt and responds accordingly (see the software listing; INT-MOD). In addition to the RS-232C control signals, the communications software can support all of the standard UART error conditions such as framing errors, underrun, overrun and parity, if parity is enabled.

Parallel I/O With Handshaking

The remaining two Port 1 lines, not used for the RS-232C control signals, provide ACK/OWF and STB/IBF handshaking signals for parallel Port 2. In an environment which utilized the second parallel port, while implementing the above described RS-232C channel, both would operate on an interrupt basis. The interrupt software algorithm depends upon whether the parallel port is configured as input or output, and whether Nested or Normal interrupt mode is programmed. If Nested Interrupt Mode is used, the software flow would default to parallel input or output (as programmed) with Port 2 handshaking the lowest priority interrupt. The serial channel would then interrupt parallel Port 2 transmission whenever the serial channel transmitted or received a character. The RS-232C control signals, OR'd to the External Interrupt (EXTINT) pin, would have the highest interrupt controller priority. The Software Description below describes this in greater detail.

SOFTWARE DESCRIPTION

Serial RS—232C Interface

The software is written in PL/M and is broken up into four separate modules, each containing several procedures. A block diagram of the software structure is given in Figure 4. The modules are identified by the dotted boxes, and the procedures are identified by the solid boxes. Two or more procedures connected by a solid line means the procedure above calls the procedure below. The procedures without any solid lines connecting them are interrupt procedures. They are entered when the 8256AH interrupts the 80186 and vectors an indirect address to the 80186.

The Serial RS-232C Interface software uses nested interrupts. The priority of the interrupt procedures is given in Figure 5.

The priority of the interrupts is not programmable but they are logically oriented so that for this application the priority is correct. The serial receiver should have the highest priority since it could have overrun errors. Therefore the RxD request can interrupt any other interrupt service routine thus preventing any possibility of an overrun error.

The Serial RS-232C Interface software is entered via a GO instruction from the 957B System Monitor console. The software first calls POWR—ON—INIT which initializes the 8256AH. This sets the 8256AH to 8086 Mode with parallel Port 2 in two wire handshake mode using Port 1 pin 0-1 for Port 2 handshaking. The initialization configures six of the Port 1 lines, pins 2-7, for RS-232C handshaking—input or output depending upon the specific signal tied to the pin. Figure 6 illustrates the definition of each Port 1 RS-232C handshaking line and its direction.

Both the Serial RS-232C Interface and the parallel interface with handshaking operate on an interrupt basis. Following initialization the software enters an endless loop and awaits an interrupt from one of three sources; Receive Data (RxD), Transmit Data (TxD) or the parallel interface. In the serial interface idle state, neither transmitting nor receiving data, the software is constantly responding to TxD interrupts; a result of the Transmit Buffer (TBE) and/or Transmit Register (TRE) being continuously empty. When data is received by the RS-232C channel the RxD interrupt, being of higher priority, asserts its interrupt.

AP-183
Figure 3. 8256AH / 80186 Schematic
Figure 4. Block Diagram of the 8256AH Serial RS-232C Interface Software Structure

<table>
<thead>
<tr>
<th>Priority</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td>0: Not Used</td>
</tr>
<tr>
<td></td>
<td>1: Not Used</td>
</tr>
<tr>
<td></td>
<td>2: External Interrupt (EXTINT)</td>
</tr>
<tr>
<td></td>
<td>3: Not Used</td>
</tr>
<tr>
<td></td>
<td>4: RxD Interrupt</td>
</tr>
<tr>
<td></td>
<td>5: TxD Interrupt</td>
</tr>
<tr>
<td></td>
<td>6: Timer 2 or 2 &amp; 4 (16 bit)</td>
</tr>
<tr>
<td></td>
<td>7: Port 2 Handshaking</td>
</tr>
</tbody>
</table>

Figure 5. 8256AH Interrupt Source To Priority Level Map

<table>
<thead>
<tr>
<th>Port 1 Pin No.</th>
<th>Circuit</th>
<th>I/O</th>
<th>Abrev.</th>
<th>Signal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>STB/ACK</td>
<td>Parallel Port 2</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>IBF/OBF</td>
<td>Handshaking Signals</td>
</tr>
<tr>
<td>2</td>
<td>CG</td>
<td>I</td>
<td>CTS</td>
<td>Clear To Send</td>
</tr>
<tr>
<td>3</td>
<td>CE</td>
<td>I</td>
<td>RI</td>
<td>Ring Indicator</td>
</tr>
<tr>
<td>4</td>
<td>CD</td>
<td>O</td>
<td>DTR</td>
<td>Data Terminal Ready</td>
</tr>
<tr>
<td>5</td>
<td>CI</td>
<td>I</td>
<td>DSRS</td>
<td>Data Signal Rate Selector</td>
</tr>
<tr>
<td>6</td>
<td>CF</td>
<td>I</td>
<td>RLSD</td>
<td>Receive Line Signal Detector</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(or CD)</td>
<td>(Carrier Detect)</td>
</tr>
<tr>
<td>7</td>
<td>CC</td>
<td>I</td>
<td>DSR</td>
<td>Data Set Ready</td>
</tr>
</tbody>
</table>

Figure 6. Port 1 RS-232C Pin Definition
Although the parallel interface software is not implemented in the software listing of Appendix A, the algorithm for implementing multiplexed parallel and serial I/O is to input or output data on the parallel port during the relatively lengthy time required for serial communication overhead. The algorithm differs slightly during the serial channel idle state when the software responds to repetitive TxD interrupts. In this case the endless loop would detect the idle state repetitive TxD interrupts and disable the TxD interrupt for a short time while the parallel inputs or outputs data. This would require using one of the 8256AH timers to time out repetitive TxD interrupts. The timer used has to be lower in priority than the RxD interrupt to guarantee protection against overrun errors. Timer 2, or 2 and 4 cascaded if longer time delays are desired, provides the proper interrupt level as shown in Figure 5.

Figure 7 shows the Receive Data (RxD) interrupt service routine software flowchart. Since two conditions can generate an RxD Interrupt the Software first reads the Status Register and checks for the Break Detect (DB) bit being set. If the BD bit is clear, no Break condition being present, the data byte is read, stripped to seven bits, for an ASCII character, and sent to the system console via a call to the 957B System Monitor Console Output (CO) routine. Upon return from the 957B monitor call an End Of Interrupt (EOI) is sent to the 8256AH to reset the currently served interrupt level bit in the Interrupt Service Register.

Figure 8 shows the Transmit Data (TxD) interrupt service routine software flowchart. There are three conditions which may cause a TxD Interrupt; TBE, TRE and Break-In Detect. The TxD service routine first reads the Status Register to determine if the interrupt source is the TBE (Transmit Buffer Empty), if not then the interrupt service routine returns to the MAIN—MOD loop. If TBE = 1 (true) then a data byte is read from the 957B System Monitor Console Input (CI) routine. If the data byte is an ASCII character it is written to the 8256AH Transmit Buffer. The software exists via an EOI (End Of Interrupt) command to the 8256AH then returns to the MAIN—MOD Rx—Tx—Loop.

![Figure 7. Receive Data Interrupt Service Routine Software Flowchart](image-url)
Figure 8. Transmit Data Interrupt Service Routine
Software Flowchart

RS-232C Control Signals Interrupt Structure

The overall interrupt scheme is such that a change in a RS-232C handshake line causes an interrupt via the EXTINT pin on the 8256AH (see Figure 3 8256AH/ 80186 Schematic). The EXTINT interrupt is of higher priority than either the RxD or TxD interrupt. This enables the RS-232C handshake signals to manage the receipt or transmission of data via the nested interrupt mode of the 8256AH. The EXTINT interrupt service routine first reads the Port 1 pins 2-7 data and compares it to default state for the signal requiring service. The EXTINT interrupt service routine then calls the appropriate handshake signal service procedure as shown in the bottom module of Figure 4 Software Structure Block Diagram.

Each of the individual RS-232C control signal service procedures displays a message on the 957B monitor console device indicating the signal requiring a response. The service procedure then either initiates predefined actions or prompts the user with options. In a system which utilized file storage, such as a personal computer, the RS-232C software driver could pass a flag to the communications software, rather than a message. The communications software would in turn perform the same types of action but could also protect disk buffering files which might be open at the time of the interrupt. Two examples of the RS-232C Control Signal interrupt service routines, CTS and DSRS, are described below;

If Clear To Send (CTS) changes state, the UART automatically disables the transmitter. The CTS interrupt service procedure initializes the 8256AH's internal Timer. If the timer times out before CTS goes active again an interrupt is generated, a second message is displayed at the 957B monitor console prompting the user that the CTS line remains inactive. The options available at this point are to wait again, re-initializing Timer 1, or to disconnect the RS-232C channel.
If Data Signal Rate Selector (DSRS) changes state, the software prompts the user with a message that the Data Rates of the two RS-232C channels are not the same and the user is given the option of altering the data rate. This application example was interfaced to a 103A/212 Bell modem and as such prompts the user to select between 300 or 1200 bps data rates. In the case of a non-modem interface the routine could prompt the user for one of the thirteen standard data rates. The software then returns to the TxD/RxD software loop. The balance of the interrupt service procedures for the RS-232C handshaking signals function in a similar manner.

Depending upon the specific system design and software requirements, a variety of enhancements could be added to the system design. These could include interrupt traps that initiate specific corrective options or cascading multiple 8256AHs each with an RS-232C interfaces as described above. An example of an interrupt trap might be auto redial upon time out for lack of Carrier Detect (CD) upon initiating a communications link, or automatic disk file update when a receive buffer approaches overflow.

The ability of the 8256AH to be reprogrammed to meet the changing requirements of a system simplifies the overall system design and multiplies its capabilities. A simple reinitialization sequence could reconfigure the 8256AH as a UART with two parallel ports or utilize any of the various special functions of the parallel Port 1; e.g., an external timer input or an additional external interrupt input, etc. The reinitialization could also configure the 8256AH Multifunction Peripheral for a variety of custom applications.

CONCLUSION

The functional integration of the 8256AH makes it ideal for designs which require maximum flexibility and simplicity of implementation. The implementation of the RS-232C serial channel modem interface and multiplexed parallel I/O described in this application note represent a level of efficiency in peripheral performance and design previously unavailable. The 8256AH Multi-function Peripheral represents a savings of two-thirds the board space and power required by the previous four chip solution, with the added benefit of increased system reliability. The application note demonstrates the ease of implementing the variety of I/O capabilities and system support functions of the 8256AH. The integration of four common microprocessor system functions into one VLSI device enables the designer to devote valuable resources to adding features to enhance the system design, adding performance and flexibility, and reducing the system’s overhead.
APPENDIX A.

SOFTWARE LISTING

PL/M-8b COMPILER MAINMOD

SERIES-III PL/M-8b V2.3 COMPIILATION OF MODULE MAINMOD
OBJECT MODULE PLACED IN: F2:56.OBJ
COMPILER INVOKED BY: PLMB6.86 : F2:56

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * 8256AH MULTIFUNCTION PERIPHERAL SIMPLIFIES  *
* * MICROCOMPUTER I/O DESIGN  *
* * Intel Corporation  *
* * 3065 Bowers Avenue  *
* * Santa Clara, Ca. 95051  *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

$MOD186 DEBUG
MAINMOD:
DO.

/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
/ * 8256AH Register / Value / Constant Definitions  * / * * * * * * * * * *
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

Declare    Literally     'literally',
DCL      lit          'Declare',

True     lit          'Offh',
False    lit          'Oh',
Forever  lit          'while 1',
Pcs1      lit          '80h',
Cmd1reg   lit          'pcs1 + 0',
Cmd2reg   lit          'pcs1 + 2',
Cmd3reg   lit          'pcs1 + 4',
Modereg   lit          'pcs1 + 6',

Port1Ctrl1Reg lit     'pcs1 + 8',
SetIntReg  lit     'pcs1 + 0ah',
EnIntReg   lit     'pcs1 + 0ah',
RstIntReg  lit     'pcs1 + 0ch',
IntAddrReg lit     'pcs1 + 0ch',
TxBufferReg lit     'pcs1 + Oeh',
RxBufferReg lit     'pcs1 + Och',

Port1Reg   lit     'pcs1 + 10h',
Port2Reg   lit     'pcs1 + 12h',
Timer1Reg  lit     'pcs1 + 14h',
Timer2Reg  lit     'pcs1 + 1eh',
Timer3Reg  lit     'pcs1 + 1ch',
StatReg    lit     'pcs1 + 1eh',

Intr1      lit     'pcs1 + 40h',
Intr2      lit     'pcs1 + 01h',
Intr3      lit     'pcs1 + 10h',

231125-8
Intr4 lit 'pcsl + 08h',
Int_Reset lit '08h',
SioTxEn lit '10h',
SioTxRdy lit '20h',
SioRxRdy lit '40h',
Break lit '04h',
DisIntr lit '00h',
StripTo7fh lit '7fh',
Port1 Strip lit '0fh',
Cmd1 lit '43h',
Cmd2A lit '07h',
Cmd2B lit '09h',
Cmd3Clr lit '7fh',
Cmd3 lit '01h',
Mode lit '00h',
EnRcvr lit '0ch',
A lit '41h',
B lit '42h',
DSR lit '80h',
DSR_Flag lit '80h',
CD lit '40h',
CD_Flag lit '40h',
DSRS lit '20h',
DSRS_Flag lit '20h',
DTR lit '10h',
RI lit '05h',
CTS lit '04h',
CTS_Flag lit '04h',
(Status,
Hndshk_Pins,
J) Byte,
Char Byte External,
Message_Ptr Pointer;

/* Message Declarations */
DCL CTS_MSG (S) Byte Public Data ('CTS Disabled Receive Data stopped.', OAH, ODH, 0).
DSR_MSG (S) Byte Public Data ('DSR Disabled', OAH, ODH, 0).
CD_MSG (S) Byte Public Data ('CD Disabled', OAH, ODH, 0).
DSRS_MSG (S) Byte Public Data ('Enter Baud Rate. A 300 B 1200 (A/B) : ', 00).
CTS2_MSG (S) Byte Public Data ('CTS Disabled Receive Data stopped', OAH, ODH, 0).
Break_MSG (S) Byte Public Data ('Break in Receive Data', OAH, ODH, 0).

/* External Procedures */
/*  MCO:  957B Monitor Console Output Routine */
/MCO: Procedure(Char) External;
DCL Char Byte;
End MCO;
/MCI: 957B Monitor Console Input Routine */
/MCI: Procedure Byte External;
End MCI;
/* Initialize 8256AH Procedure */
Init56: Procedure;
Disable;
/* Output 8256AH Init Data */
Output(Cmd1Reg)=Cmd1; /* 8086 mode: freq=1kHz, 1 stop bit, and 7 bit char */
Output(Cmd1Reg)=Cmd2A; /* odd parity, system clk=1.024MHz, and 1200 bps */
Output(Cmd1Reg)=Cmd3Clr; /* clear cmd reg 3 */
Output(Cmd1Reg)=Cmd3; /* reset, itr ack enabled, nested intr mode */
Output(Cmd1Reg)=EnRcvr; /* enable receiver */
Output(Cmd1Reg)=Mode; /* cascade timers 3&5, for the receiver timer & out timer, byte & output mode */
Call Load_Int_Table;
Enable;
End Init56;
/* */
/* Procedure:  Load Interrupt Address Vectors */
Load_Int_Table: Procedure Public;
Call SetInterrupt(42H,EXTINT);
Call SetInterrupt(44H,Receive_Char);
Call SetInterrupt(43H,Transmit_Char);
Call SetInterrupt(46H,Timer_2_4);
End Load_Int_Table;
/* */
/* EXTINT Interrupt Procedure: */
/ Service routine reads the Port 1 RS232 handshake signals and sets the message pointer corresponding to the signal detected.
26

EXTINT. Procedure Interrupt 42H:
27

Enable;
28

HndShk_Pins=Input(PortRegReg) and PortRegStrip;
29

If CTS_Flag = HndShk_Pins and CTS Then
30

Do,
31

Message_Ptr=0CTS_MSG(0);
32

Output(TimerReg)=100;
33

End;
34

Else
35

If DSR_Flag = HndShk_Pins and DSR Then
36

Message_Ptr=0DSR_MSG(0);
37

Else
38

If CD_Flag = HndShk_Pins and CD Then
39

Message_Ptr=0CD_MSG(0);
40

Else
41

If DSRS_Flag = HndShk_Pins and DSRS Then
42

Do;
43

Message_Ptr=0DSRS_MSG(0);
44

If MCI = A Then
45

Output(CmdReg)=Cmd2A;
46

Else
47

If MCI = B Then
48

Output(CmdReg)=Cmd2B;
49

End;
50

End EXTINT;
51

End EXTINT;
52

End EXTINT;
53

Procedure Receive a character:
54

Receive_Char: Procedure Interrupt 44H:
55

Enable;
56

Status=(Input(StatReg) and SioRxRdy),
57

If Status AND Break Then
58

Do;
59

Message_Ptr=0Break_MSG(0);
60

Call Send_Msg;
61

End;
62

Else
63

Do;
64

Char=Input(RxBuffReg) and Stripto7fh;
65

Call MCO(Char);
66

End;
67

End Receive_Char;
68

End Receive_Char;
*/

Procedure. Write character to 8256AH UART */
63 1 Transmitchar: Procedure Interrupt 45H;

64 2 Status=Input(StatReg) and SioRxdy
65 2 If Status and SioRxdy Then
66 2 Char=(MCI And Stripto7FH); /* strip to 7 bits */
67 2 If Char >= 20H And Char <= 7FH Then /* if char is ASCII output it */
68 2 Output(TxBuffReg)=Char;
69 2 Output(RstIntReg)=Int_Reset;
70 2 End Transmitchar;

/*

Procedure. Write character to 856AH UART */
71 1 Timer2_4: Procedure Interrupt 46H;
72 2 MessagePtr=ECTS2_MSG(0);
73 2 Call Send_Msg;
74 2 Output(RstIntReg)=Int_Reset;
75 2 End Timer2_4;

/*

Message Output Procedure */
76 1 Send_Msg: Procedure;
77 2 DCL Message Based Message_Ptr (1) Byte;
78 2 J=0;
79 2 Do While Message(J) <> 0;
80 3 Char=Message(J);
81 3 Call MCO(Char);
82 3 J=J+1;
83 3 End;
84 2 Return;
85 2 End Send_Msg;

/*

Main Program Body */
86 1 Call Init56;
87 1 Do Forever;
88 2 End;

231125-12
PL/M-86 COMPILER MAINMOD

89 1 End MainMod;

MODULE INFORMATION:

<table>
<thead>
<tr>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE AREA SIZE</td>
<td>0235H</td>
</tr>
<tr>
<td>CONSTANT AREA SIZE</td>
<td>00BEH</td>
</tr>
<tr>
<td>VARIABLE AREA SIZE</td>
<td>0007H</td>
</tr>
<tr>
<td>MAXIMUM STACK SIZE</td>
<td>0034H</td>
</tr>
<tr>
<td>280 LINES READ</td>
<td></td>
</tr>
<tr>
<td>0 PROGRAM WARNINGS</td>
<td></td>
</tr>
<tr>
<td>0 PROGRAM ERRORS</td>
<td></td>
</tr>
</tbody>
</table>

DICTIONARY SUMMARY:

31KB MEMORY AVAILABLE
6KB MEMORY USED (19%)
OKB DISK SPACE USED

END OF PL/M-86 COMPILATION
The 8272A is an LSI Floppy Disk Controller (FDC) Chip, which contains the circuitry and control functions for interfacing a processor to 4 Floppy Disk Drives. It is capable of supporting either IBM 3740 single density format (FM), or IBM System 34 Double Density format (MFM) including double sided recording. The 8272A provides control signals which simplify the design of an external phase locked loop and write precompensation circuitry. The FDC simplifies and handles most of the burdens associated with implementing a Floppy Disk Drive Interface. The 8272A is a pin-compatible upgrade to the 8272.
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Connection To</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>1</td>
<td>I</td>
<td>μP</td>
<td>Reset: Places FDC in idle state. Resets output lines to FDD to &quot;0&quot; (low). Does not clear the last specify command.</td>
</tr>
<tr>
<td>RD</td>
<td>2</td>
<td>I</td>
<td>μP</td>
<td>Read: Control signal for transfer of data from FDC to Data Bus, when &quot;0&quot; (low).</td>
</tr>
<tr>
<td>WR</td>
<td>3</td>
<td>I</td>
<td>μP</td>
<td>Write: Control signal for transfer of data to FDC via Data Bus, when &quot;0&quot; (low).</td>
</tr>
<tr>
<td>CS</td>
<td>4</td>
<td>I</td>
<td>μP</td>
<td>Chip Select: IC selected when &quot;0&quot; (low), allowing RD and WR to be enabled.</td>
</tr>
<tr>
<td>A0</td>
<td>5</td>
<td>I</td>
<td>μP</td>
<td>Data/Status Register Select: Selects Data Reg (A0 = 1) or Status Reg (A0 = 0) contents to be sent to Data Bus.</td>
</tr>
<tr>
<td>DB6-DB7</td>
<td>6-13</td>
<td>I</td>
<td>μP</td>
<td>Data Bus: Bidirectional 8-Bit Data Bus.</td>
</tr>
<tr>
<td>DRQ</td>
<td>14</td>
<td>O</td>
<td>DMA</td>
<td>Data DMA Request: DMA Request is being made by FDC when DRQ &quot;1&quot;.</td>
</tr>
<tr>
<td>DACK</td>
<td>15</td>
<td>I</td>
<td>DMA</td>
<td>DMA Acknowledge: DMA cycle is active when &quot;0&quot; (low) and Controller is performing DMA transfer.</td>
</tr>
<tr>
<td>TC</td>
<td>16</td>
<td>I</td>
<td>DMA</td>
<td>Terminal Count: Indicators the termination of a DMA transfer when &quot;1&quot; (high).</td>
</tr>
<tr>
<td>IDX</td>
<td>17</td>
<td>O</td>
<td>FDD</td>
<td>Index: Indicates the beginning of a disk track.</td>
</tr>
<tr>
<td>INT</td>
<td>18</td>
<td>O</td>
<td>μP</td>
<td>Interrupt: Interrupt Request Generated by FDC.</td>
</tr>
<tr>
<td>CLK</td>
<td>19</td>
<td>I</td>
<td></td>
<td>Clock: Single Phase 8 MHz (4 MHz for mini floppies) Squarewave Clock.</td>
</tr>
<tr>
<td>GND</td>
<td>20</td>
<td></td>
<td></td>
<td>Ground: D.C. Power Return.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Connection To</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>40</td>
<td></td>
<td></td>
<td>D.C. Power: +5V</td>
</tr>
<tr>
<td>RW/SEEK</td>
<td>39</td>
<td>O</td>
<td>FDD</td>
<td>Read Write / SEEK: When &quot;1&quot; (high) Seek mode selected and when &quot;0&quot; (low) Read/Write mode selected.</td>
</tr>
<tr>
<td>ΥCT/DIR</td>
<td>38</td>
<td>O</td>
<td>FDD</td>
<td>Low Current/Direction: Lowers Write current on inner tracks in Read/Write mode, determines direction head will step in Seek mode.</td>
</tr>
<tr>
<td>FR/STP</td>
<td>37</td>
<td>O</td>
<td>FDD</td>
<td>Fault Reset/Step: Resets fault FF in FDD in Read/Write mode, provides step pulses to move head to another cylinder in Seek mode.</td>
</tr>
<tr>
<td>HDL</td>
<td>36</td>
<td>O</td>
<td>FDD</td>
<td>Head Load: Command which causes read/write head in FDD to contact diskette.</td>
</tr>
<tr>
<td>RDY</td>
<td>35</td>
<td>I</td>
<td>FDD</td>
<td>Ready: Indicates FDD is ready to send or receive data. Must be tied high (gated by the index pulse) for mini floppies which do not normally have a Ready line.</td>
</tr>
<tr>
<td>WP/TS</td>
<td>34</td>
<td>I</td>
<td>FDD</td>
<td>Write Protect / Two-Side: Senses Write Protect status in Read/Write mode, and Two Side Media in Seek mode.</td>
</tr>
<tr>
<td>FLT/TRK</td>
<td>33</td>
<td>I</td>
<td>FDD</td>
<td>Fault/Track 0: Senses FDD fault condition in Read/Write mode and Track 0 condition in Seek mode.</td>
</tr>
<tr>
<td>PS1,PS0</td>
<td>31,32</td>
<td>O</td>
<td>FDD</td>
<td>Precompensation (pre-shift): Write precompensation status during MFM mode. Determines early, late, and normal times.</td>
</tr>
<tr>
<td>WR DATA</td>
<td>30</td>
<td>O</td>
<td>FDD</td>
<td>Write Data: Serial clock and data bits to FDD.</td>
</tr>
<tr>
<td>DS1,DS0</td>
<td>28,29</td>
<td>O</td>
<td>FDD</td>
<td>Drive Select: Selects FDD unit.</td>
</tr>
<tr>
<td>HDSEL</td>
<td>27</td>
<td>O</td>
<td>FDD</td>
<td>Head Select: Head 1 selected when &quot;1&quot; (high) Head 0 selected when &quot;0&quot; (low).</td>
</tr>
</tbody>
</table>

Note 1: Disabled when CS=1
Note 2: TC must be activated to terminate the Execution Phase of any command
Note 3: DRQ is also an input for certain test modes; it should have a 4.7kΩ pull-up resistor to prevent activation.

5-445
**Table 1. Pin Description (Continued)**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Connection To</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFM</td>
<td>26</td>
<td>O</td>
<td>PLL</td>
<td>MFM Mode: MFM mode when &quot;1&quot;, FM mode when &quot;0&quot;</td>
</tr>
<tr>
<td>WE</td>
<td>25</td>
<td>O</td>
<td>FDD</td>
<td>Write Enable: Enables write data into FDD.</td>
</tr>
<tr>
<td>VCO</td>
<td>24</td>
<td>O</td>
<td>PLL</td>
<td>VCO Sync: Inhibits VCO in PLL when &quot;0&quot; (low), enables VCO when &quot;1&quot;</td>
</tr>
<tr>
<td>RD DATA</td>
<td>23</td>
<td>I</td>
<td>FDD</td>
<td>Read Data: Read data from FDD, containing clock and data bits.</td>
</tr>
</tbody>
</table>

**FEATURES**

Address mark detection circuitry is internal to the FDC which simplifies the phase locked loop and read electronics. The track stepping rate, head load time, and head unload time may be programmed by the user. The 8272A offers many additional features such as multiple sector transfers in both read and write modes with a single command, and full IBM compatibility in both single (FM) and double density (MFM) modes.

**8272A ENHANCEMENTS**

On the 8272A, after detecting the Index Pulse, the VCO Sync output stays low for a shorter period of time. See Figure 4A. On the 8272 there can be a problem reading data when Gap 4A is 00 and there is no IAM. This occurs on some older floppy formats. The 8272A cures this problem by adjusting the VCO Sync timing so that it is not low during the data field. See Figure 4B.
8272A REGISTERS — CPU INTERFACE

The 8272A contains two registers which may be accessed by the main system processor; a Status Register and a Data Register. The 8-bit Main Status Register contains the status information of the FDC, and may be accessed at any time. The 8-bit Data Register (actually consists of several registers in a stack with only one register presented to the data bus at a time), stores data, commands, parameters, and FDD status information. Data bytes are read out of, or written into, the Data Register in order to program or obtain the results after execution of a command. The Status Register may only be read and is used to facilitate the transfer of data between the processor and 8272A.

The relationship between the Status/Data registers and the signals RD, WR, and Ao is shown in Table 2.

Table 2. Ao, RD, WR decoding for the selection of Status/Data register functions.

<table>
<thead>
<tr>
<th>Ao</th>
<th>RD</th>
<th>WR</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Read Main Status Register</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Illegal (see note)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Illegal (see note)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Read from Data Register</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Write into Data Register</td>
</tr>
</tbody>
</table>

Note: Design must guarantee that the 8272A is not subjected to illegal inputs.

The Main Status bits are defined in Table 3.

Table 3. Main Status Register bit description.

<table>
<thead>
<tr>
<th>BIT NUMBER</th>
<th>NAME</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>FDD Busy</td>
<td>D0B</td>
<td>FDD number 0 is in the Seek mode.</td>
</tr>
<tr>
<td>D1</td>
<td>FDD 1 Busy</td>
<td>D1B</td>
<td>FDD number 1 is in the Seek mode.</td>
</tr>
<tr>
<td>D2</td>
<td>FDD 2 Busy</td>
<td>D2B</td>
<td>FDD number 2 is in the Seek mode.</td>
</tr>
<tr>
<td>D3</td>
<td>FDD 3 Busy</td>
<td>D3B</td>
<td>FDD number 3 is in the Seek mode.</td>
</tr>
<tr>
<td>D4</td>
<td>FDC Busy</td>
<td>CB</td>
<td>A read or write command is in process</td>
</tr>
<tr>
<td>D5</td>
<td>Non-DMA mode</td>
<td>NDM</td>
<td>The FDC is in the non-DMA mode This bit is set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>only during the execution phase in non-DMA mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Transition to &quot;0&quot; state indicates execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>phase has ended.</td>
</tr>
<tr>
<td>D6</td>
<td>Data Input/Output</td>
<td>DIO</td>
<td>Indicates direction of data transfer between</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FDC and Data Register. If DIO = &quot;1&quot; then</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>transfer is from Data Register to the Processor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If DIO = &quot;0&quot;, then transfer is from the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Processor to Data Register.</td>
</tr>
<tr>
<td>D7</td>
<td>Request for Master</td>
<td>ROM</td>
<td>Indicates Data Register is ready to send or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>receive data to or from the Processor. Both</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bits DIO and ROM should be used to perform the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>handshaking functions of &quot;ready&quot; and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;direction&quot; to the processor.</td>
</tr>
</tbody>
</table>

The DIO and RQM bits in the Status Register indicate when Data is ready and in which direction data will be transferred on the Data Bus.

Note: There is a 12uS or 24uS RQM flag delay when using an 8 or 4 MHz clock respectively.

Figure 5. Status Register Timing

The 8272A is capable of executing 15 different commands. Each command is initiated by a multi-byte transfer from the processor, and the result after execution of the command may also be a multi-byte transfer back to the processor. Because of this multi-byte interchange of information between the 8272A and the processor, it is convenient to consider each command as consisting of three phases:

- **Command Phase:** The FDC receives all information required to perform a particular operation from the processor.
- **Execution Phase:** The FDC performs the operation it was instructed to do.
- **Result Phase:** After completion of the operation, status and other housekeeping information are made available to the processor.

During Command or Result Phases the Main Status Register (described in Table 3) must be read by the processor before each byte of information is written into or read from the Data Register. Bits D6 and D7 in the Main Status Register must be in a 0 and 1 state, respectively, before each byte of the command word may be written into the 8272A. Many of the commands require multiple bytes, and as a result the Main Status Register must be read prior to each byte transfer to the 8272A. On the other hand, during the Result Phase, D6 and D7 in the Main Status Register must both be 1's (D6 = 1 and D7 = 1) before reading each byte from the Data Register. Note, this reading of the Main Status Register before each byte transfer to the 8272A is required in only the Command and Result Phases, and NOT during the Execution Phase.

During the Execution Phase, the Main Status Register need not be read. If the 8272A is in the non-DMA Mode, then the receipt of each data byte (if 8272A is reading data from FDD) is indicated by an Interrupt signal on pin 18 (INT = 1). The generation of a Read signal (RD = 0) will reset the Interrupt as well as output the Data onto
the Data Bus. For example, if the processor cannot handle Interrupts fast enough (every 13 μs for MFM mode) then it may poll the Main Status Register and then bit D7 (RQM) functions just like the Interrupt signal. If a Write Command is in process, then the WR signal performs the reset to the Interrupt signal.

The 8272A always operates in a multi-sector transfer mode. It continues to transfer data until the TC input is active. In Non-DMA Mode, the system must supply the TC input.

If the 8272A is in the DMA Mode, no Interrupts are generated during the Execution Phase. The 8272A generates DRQ's (DMA Requests) when each byte of data is available. The DMA Controller responds to this request with both a DACK = 0 (DMA Acknowledge) and a RD = 0 (Read signal). When the DMA Acknowledge signal goes low (DACK = 0) then the DMA Request is reset (DRQ = 0). If a Write Command has been programmed then a WR signal will appear instead of RD. After the Execution Phase has been completed (Terminal Count has occurred) then an Interrupt will occur (INT = 1). This signifies the beginning of the Result Phase. When the first byte of data is read during the Result Phase, the Interrupt is automatically reset (INT = 0).

It is important to note that during the Result Phase all bytes shown in the Command Table must be read. The Read Data Command, for example, has seven bytes of data in the Result Phase. All seven bytes must be read in order to successfully complete the Read Data Command. The 8272A will not accept a new command until all seven bytes have been read. Other commands may require fewer bytes to be read during the Result Phase.

The 8272A contains five Status Registers. The Main Status Register mentioned above may be read by the processor at any time. The other four Status Registers (ST0, ST1, ST2, and ST3) are only available during the Result Phase, and may be read only after successfully completing a command. The particular command which has been executed determines how many of the Status Registers will be read.

The bytes of data which are sent to the 8272A to form the Command Phase, and are read out of the 8272A in the Result Phase, must occur in the order shown in the Table 4. That is, the Command Code must be sent first and the other bytes sent in the prescribed sequence. No foreshortening of the Command or Result Phases are allowed. After the last byte of data in the Command Phase is sent to the 8272A, the Execution Phase

### Table 4. 8272A Command Set

<table>
<thead>
<tr>
<th>PHASE</th>
<th>RW</th>
<th>DATA BUS</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>READ DATA</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>MT MFM SK 0 0 1 1 0</td>
<td>Command Codes</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>C</td>
<td>Sector ID information prior to Command execution</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>EOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>DTL</td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td>R</td>
<td>ST 0</td>
<td>Data transfer between the FDD and main-system</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST 1</td>
<td>Status information after Command execution</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td><strong>RESULT DATA</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>MT MFM SK 0 0 1 1 0</td>
<td>Command Codes</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>C</td>
<td>Sector ID information prior to Command execution</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>EOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>DTL</td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td>R</td>
<td>ST 0</td>
<td>Data transfer between the FDD and main-system</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST 1</td>
<td>Status information after Command execution</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Note: 1 Symbols used in this table are described at the end of this section
2 A0 = 1 for all operations
3 X = Don't care, usually made to equal binary 0
### Table 4. 8272A Command Set (Continued)

<table>
<thead>
<tr>
<th>PHASE</th>
<th>RW</th>
<th>D7, D6, D5, D4, D3, D2, D1, D0</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>READ A TRACK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>0 MFM SK 0 0 0 1 0</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Execution</td>
<td>W</td>
<td>0 0 0 0 0 0 HDS DS1 DS0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>C</td>
<td>Sector ID information prior to Command execution</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>EOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>GPL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>DTL</td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>R</td>
<td>ST0</td>
<td>Status information about the FDD Phy-</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>C</td>
<td>Sector ID information after Command execution</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td><strong>FORMAT A TRACK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>0 MFM 0 0 0 1 0 1 0</td>
<td>Commands</td>
</tr>
<tr>
<td>Execution</td>
<td>W</td>
<td>0 0 0 0 0 0 HDS DS1 DS0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td>Bytes/Sector</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>SC</td>
<td>Sectors/Cylinder</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>GPL</td>
<td>Gap 3</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>D</td>
<td>Filler Byte</td>
</tr>
<tr>
<td>Result</td>
<td>R</td>
<td>ST0</td>
<td>Status Information after Command execution</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>R</td>
<td>Sector ID information during Execution Phase</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td><strong>SCAN EQUAL</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>MT MFM SK 1 0 0 0 1</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Execution</td>
<td>W</td>
<td>0 0 0 0 0 0 HDS DS1 DS0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>C</td>
<td>Sector ID information prior to Command execution</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>EOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>GPL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>STP</td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>R</td>
<td>ST0</td>
<td>Status information after Command execution</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

**SCAN LOW OR EQUAL**

<table>
<thead>
<tr>
<th>PHASE</th>
<th>RW</th>
<th>D7, D6, D5, D4, D3, D2, D1, D0</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>W</td>
<td>MT MFM SK 1 1 0 0 1</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Execution</td>
<td>W</td>
<td>0 0 0 0 0 0 HDS DS1 DS0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>C</td>
<td>Sector ID information prior to Command execution</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>EOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>GPL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>STP</td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>R</td>
<td>ST0</td>
<td>Status information after Command execution</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td><strong>SCAN HIGH OR EQUAL</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>MT MFM SK 1 1 1 0 1</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Execution</td>
<td>W</td>
<td>0 0 0 0 0 0 HDS DS1 DS0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>C</td>
<td>Sector ID information prior to Command execution</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>EOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>GPL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>STP</td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>R</td>
<td>ST0</td>
<td>Status information after Command execution</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td><strong>RECALIBRATE</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>0 0 0 0 0 0 1 1 1</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Execution</td>
<td>W</td>
<td>0 0 0 0 0 0 0 DS1 DS0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>SRT</td>
<td>Head retracted to Track 0</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>HLT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>FUT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>ND</td>
<td></td>
</tr>
<tr>
<td><strong>SENSE INTERRUPT STATUS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>0 0 0 0 0 0 1 1</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Result</td>
<td>R</td>
<td>ST0</td>
<td>Status information at the end of each seek operation about the FDC</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>PCN</td>
<td></td>
</tr>
<tr>
<td><strong>SPECIFY</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>0 0 0 0 0 0 1 1</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Result</td>
<td>R</td>
<td>ST0</td>
<td>Head is positioned over proper Cylinder on Diskette</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST0</td>
<td></td>
</tr>
<tr>
<td><strong>SENSE DRIVE STATUS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>0 0 0 0 0 1 1 1</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Execution</td>
<td>W</td>
<td>0 0 0 0 0 0 HDS DS1 DS0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>C</td>
<td>Sector ID information about FDD</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td><strong>SEEK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>0 0 0 0 0 1 1 1</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Execution</td>
<td>W</td>
<td>0 0 0 0 0 0 HDS DS1 DS0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>NCN</td>
<td></td>
</tr>
<tr>
<td><strong>INVALID</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>Invalid Codes</td>
<td>Invalid Command Codes (NoOp—FDC goes into Standby State)</td>
</tr>
<tr>
<td>Result</td>
<td>R</td>
<td>ST0</td>
<td>ST 0 = 80 (16)</td>
</tr>
</tbody>
</table>

5-449
Table 5. Command Mneumonics

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>Address Line 0</td>
<td>A0 controls selection of Main Status Register (A0=0) or Data Register (A0=1).</td>
</tr>
<tr>
<td>C</td>
<td>Cylinder Number</td>
<td>C stands for the current selected Cylinder track number 0 through 76 of the medium.</td>
</tr>
<tr>
<td>D</td>
<td>Data</td>
<td>D stands for the data pattern which is going to be written into a Sector.</td>
</tr>
<tr>
<td>D7–D0</td>
<td>Data Bus</td>
<td>8-bit Data Bus where D7 is the most significant bit, and D0 is the least significant bit.</td>
</tr>
<tr>
<td>D50, D51</td>
<td>Drive Select</td>
<td>DS stands for a selected drive number 0 or 1.</td>
</tr>
<tr>
<td>DTL</td>
<td>Data Length</td>
<td>When N is defined as 00, DTL stands for the data length which users are going to read out or write into the Sector.</td>
</tr>
<tr>
<td>EOT</td>
<td>End of Track</td>
<td>EOT stands for the final Sector Number of a Cylinder.</td>
</tr>
<tr>
<td>GPL</td>
<td>Gap Length</td>
<td>GPL stands for the length of Gap 3 (spacing between Sectors excluding VCO Sync Field).</td>
</tr>
<tr>
<td>H</td>
<td>Head Address</td>
<td>H stands for head number 0 or 1, as specified in ID field.</td>
</tr>
<tr>
<td>HDS</td>
<td>Head Select</td>
<td>HDS stands for a selected head number 0 or 1 (H = HDS in all command words).</td>
</tr>
<tr>
<td>HLT</td>
<td>Head Load Time</td>
<td>HLT stands for the head load time in the FDD (2 to 254ms in 2ms increments).</td>
</tr>
<tr>
<td>HUT</td>
<td>Head Unload Time</td>
<td>HUT stands for the head unload time after a read or write operation has occurred (16 to 240ms in 16ms increments).</td>
</tr>
<tr>
<td>MFM</td>
<td>FM or MFM Mode</td>
<td>If MF is low, FM mode is selected and if it is high, MFM mode is selected.</td>
</tr>
<tr>
<td>MT</td>
<td>Multi-Track</td>
<td>If MT is high, a multi-track operation is to be performed (a cylinder under both HDO and HDI will be read or written).</td>
</tr>
<tr>
<td>N</td>
<td>Number</td>
<td>N stands for the number of data bytes written in a Sector.</td>
</tr>
</tbody>
</table>

automatically starts. In a similar fashion, when the last byte of data is read out in the Result Phase, the command is automatically ended and the 8272A is ready for a new command. A command may be aborted by simply sending a Terminal Count signal to pin 18 (TC = 1). This is a convenient means of ensuring that the processor may always get the 8272A’s attention even if the disk system hangs up in an abnormal manner.

**POLLED FEATURE OF THE 8272A**

After power-up RESET, the Drive Select Lines D50 and D51 will automatically go into a polling mode. In between commands (and between step pulses in the SEEK command) the 8272A polls all four FDDs looking for a change in the Ready line from any of the drives. If the Ready line changes state (usually due to a door opening or closing) then the 8272A will generate an interrupt. When Status Register 0 (ST0) is read (after Sense Interrupt Status is issued), Not Ready (NR) will be indicated. The polling of the Ready line by the 8272A occurs continuously between instructions, thus notifying the processor which drives are on or off line. Approximate scan timing is shown in Table 6.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCN</td>
<td>New Cylinder Number</td>
<td>NCN stands for a new Cylinder number, which is going to be reached as a result of the Seek operation. Desired position of Head.</td>
</tr>
<tr>
<td>ND</td>
<td>Non-DMA Mode</td>
<td>ND stands for operation in the Non-DMA Mode.</td>
</tr>
<tr>
<td>PCN</td>
<td>Present Cylinder Number</td>
<td>PCN stands for the Cylinder number at the completion of SENSE INTERRUPT STATUS Command. Position of Head at present time.</td>
</tr>
<tr>
<td>R</td>
<td>Record</td>
<td>R stands for the Sector number, which will be read or written.</td>
</tr>
<tr>
<td>RW</td>
<td>Read/Write</td>
<td>RW stands for either Read (R) or Write (W) signal.</td>
</tr>
<tr>
<td>SC</td>
<td>Sector</td>
<td>SC indicates the number of Sectors per Cylinder.</td>
</tr>
<tr>
<td>SK</td>
<td>Skip</td>
<td>SK stands for Skip Deleted Data Address Mark.</td>
</tr>
<tr>
<td>SRT</td>
<td>Step Rate Time</td>
<td>SRT stands for the Stepping Rate for the FDD (1 to 15ms in 1ms increments). The same Stepping Rate applies to all drives (F=1ms, E=2ms, etc.).</td>
</tr>
<tr>
<td>ST 0</td>
<td>Status 0</td>
<td>ST 0-3 stand for one of four registers which store the status information after a command has been executed. This information is available during the result phase after command execution. These registers should not be confused with the main status register (selected by A0 = 0).</td>
</tr>
<tr>
<td>ST 1</td>
<td>Status 1</td>
<td>ST 1-3 may be read only after a command has been executed and contain information relevant to that particular command.</td>
</tr>
<tr>
<td>ST 2</td>
<td>Status 2</td>
<td></td>
</tr>
<tr>
<td>ST 3</td>
<td>Status 3</td>
<td></td>
</tr>
<tr>
<td>STP</td>
<td>Scan Operation</td>
<td>STP = 1, the data in contiguous sectors is compared byte by byte with data sent from the processor (or DMA), and if STP = 2, then alternate sectors are read and compared.</td>
</tr>
</tbody>
</table>

**COMMAND DESCRIPTIONS**

During the Command Phase, the Main Status Register must be polled by the CPU before each byte is written into the Data Register. The DIO (DB6) and RQM (DB7) bits in the Main Status Register must be in the ‘0’ and ‘1’ states respectively, before each byte of the command may be written into the 8272A. The beginning of the execution phase for any of these commands will cause DIO and RQM to switch to ‘1’ and ‘0’ states respectively.

**READ DATA**

A set of nine (9) byte words are required to place the FDC into the Read Data Mode. After the Read Data command has been issued the FDC loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the Specify Command), and begins reading ID Address Marks and ID fields. When the current sector number ("R") stored in the ID Register (IDR) compares with the sector number read off the diskette, then the FDC outputs data (from the data field) byte-by-byte to the main system via the data bus. After completion of the read operation from the current sector, the Sector Number is incremented by one, and the data from the next sector is read and output on the data bus. This continuous read function is called a "Multi-Sector Read Operation." The Read Data Command must be terminated by the receipt of a Terminal Count signal. Upon receipt of this signal, the FDC stops outputting data to the processor, but will continue to read data from the current sector, check CRC (Cyclic Redundancy Count) bytes, and then at the end of the sector terminate the Read Data Command.

The amount of data which can be handled with a single command to the FDC depends upon MT (multi-track), MFM (MFM/FM), and N (Number of Bytes/ Sector). Table 7 on the next page shows the Transfer Capacity.

Table 6. Scan Timing

<table>
<thead>
<tr>
<th>DS1</th>
<th>DS0</th>
<th>APPROXIMATE SCAN TIMING</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>220μs</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>220μs</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>220μs</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>440μs</td>
</tr>
</tbody>
</table>
The "multi-track" function (MT) allows the FDC to read data from both sides of the diskette. For a particular cylinder, data will be transferred starting at Sector 1, Side 0 and completing at Sector L, Side 1 (Sector L = last sector on the side). Note, this function pertains to only one cylinder (the same track) on each side of the diskette.

When N = 0, then DTL defines the data length which the FDC must treat as a sector. If DTL is smaller than the actual data length in a Sector, the data beyond DTL in the Sector is not sent to the Data Bus. The FDC reads internally the complete Sector performing the CRC check, and depending upon the manner of command terminations, may perform a Multi-Track Read Operation. When N is non-zero, then DTL has no meaning and should be set to 0FFH.

At the completion of the Read Data Command, the head is not unloaded until after Head Unload Time Interval (specified in the Specify Command) has elapsed. If the processor issues another command before the head unloads, then the head settling time may be saved between subsequent reads. This time out is particularly valuable when a diskette is copied from one drive to another.

If the FDC detects the Index Hole twice without finding the right sector, (indicated in "R"), then the FDC sets the ND (No Data) flag in Status Register 1 to a 1 (high), and terminates the Read Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

After reading the ID and Data Fields in each sector, the FDC checks the CRC bytes. If a read error is detected (incorrect CRC in ID field), the FDC sets the DE (Data Error) flag in Status Register 1 to a 1 (high), and if a CRC error occurs in the Data Field the FDC also sets the DD (Data Error in Data Field) flag in Status Register 2 to a 1 (high), and terminates the Read Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

If the FDC reads a Deleted Data Address Mark off the diskette, and the SK bit (bit D5 in the first Command Word) is not set (SK = 0), then the FDC sets the CM (Control Mark) flag in Status Register 2 to a 1 (high), and terminates the Read Data Command, after reading all the data in the Sector. If SK = 1, the FDC skips the sector with the Deleted Data Address Mark and reads the next sector.

During disk data transfers between the FDC and the processor, via the data bus, the FDC must be serviced by the processor every 27 μs in the FM Mode, and every 13 μs in the MFM Mode, or the FDC sets the OR (Over Run) flag in Status Register 1 to a 1 (high), and terminates the Read Data Command.

If the processor terminates a read (or write) operation in the FDC, then the ID Information in the Result Phase is dependent upon the state of the MT bit and EOT byte. Table 5 shows the values for C, H, R, and N, when the processor terminates the Command.

### Table 7. Transfer Capacity

<table>
<thead>
<tr>
<th>Multi-Track MT</th>
<th>MFM/FM</th>
<th>Bytes/sector N</th>
<th>Maximum Transfer Capacity (Bytes/sector) (Number of Sectors)</th>
<th>Final Sector Read from Diskette</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00</td>
<td>(128) (26) = 3,328</td>
<td>26 at Side 0 or 26 at Side 1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>01</td>
<td>(256) (26) = 6,656</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>00</td>
<td>(128) (52) = 6,856</td>
<td>26 at Side 1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>01</td>
<td>(256) (52) = 13,112</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>02</td>
<td>(256) (15) = 3,840</td>
<td>15 at Side 0 or 15 at Side 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>02</td>
<td>(512) (15) = 7,680</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>03</td>
<td>(512) (8) = 4,096</td>
<td>8 at Side 0 or 8 at Side 1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>03</td>
<td>(1024) (8) = 8,192</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>03</td>
<td>(512) (16) = 8,192</td>
<td>8 at Side 1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>03</td>
<td>(1024) (16) = 16,384</td>
<td></td>
</tr>
</tbody>
</table>

### Table 8. ID Information When Processor Terminates Command

<table>
<thead>
<tr>
<th>MT</th>
<th>EOT</th>
<th>Final Sector Transferred to Processor</th>
<th>ID Information at Result Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1A</td>
<td>Sector 1 to 25 at Side 0</td>
<td>C + 1 NC R = 01 NC</td>
</tr>
<tr>
<td></td>
<td>0F</td>
<td>Sector 1 to 14 at Side 0</td>
<td>NC NC R + 1 NC</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>Sector 1 to 7 at Side 0</td>
<td>NC NC R + 1 NC</td>
</tr>
<tr>
<td>1</td>
<td>1A</td>
<td>Sector 1 to 25 at Side 0</td>
<td>C + 1 NC R = 01 NC</td>
</tr>
<tr>
<td></td>
<td>0F</td>
<td>Sector 1 to 14 at Side 0</td>
<td>NC NC R + 1 NC</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>Sector 1 to 7 at Side 0</td>
<td>NC NC R + 1 NC</td>
</tr>
<tr>
<td></td>
<td>1A</td>
<td>Sector 1 to 25 at Side 0</td>
<td>C + 1 NC R = 01 NC</td>
</tr>
<tr>
<td></td>
<td>0F</td>
<td>Sector 1 to 14 at Side 0</td>
<td>NC NC R + 1 NC</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>Sector 1 to 7 at Side 0</td>
<td>NC NC R + 1 NC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Notes: 1 NC (No Change): The same value as the one at the beginning of command execution. 2 LSB (Least Significant Bit): The least significant bit of H is complemented.

### WRITE DATA
A set of nine (9) bytes are required to set the FDC into the Write Data mode. After the Write Data command has been issued the FDC loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the Specify Command), and begins reading ID Fields. When the current sector number ("R"), stored in the ID Register (IDR) compares with the sector
number read off the diskette, then the FDC takes data from the processor byte-by-byte via the data bus, and outputs it to the FDD.

After writing data into the current sector, the Sector Number stored in "R" is incremented by one, and the next data field is written into. The FDC continues this "Multi-Sector Write Operation" until the issuance of a Terminal Count signal. If a Terminal Count signal is sent to the FDC it continues writing into the current sector to complete the data field. If the Terminal Count signal is received while a data field is being written then the remainder of the data field is filled with 00 (zeros).

The FDC reads the ID field of each sector and checks the CRC bytes. If the FDC detects a read error (Incorrect CRC) in one of the ID Fields, it sets the DE (Data Error) flag of Status Register 1 to a 1 (high), and terminates the Write Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

The Write Command operates in much the same manner as the Read Command. The following items are the same; refer to the Read Data Command for details:

- Transfer Capacity
- EN (End of Cylinder) Flag
- ND (No Data) Flag
- Head Unload Time Interval
- ID Information when the processor terminates command (see Table 2)
- Definition of DTL when N = 0 and when N ≠ 0

In the Write Data mode, data transfers between the processor and FDC must occur every 31 μs in the FM mode, and every 15 μs in the MFM mode. If the time interval between data transfers is longer than this then the FDC sets the OR (Over Run) flag in Status Register 1 to a 1 (high), and terminates the Write Data Command.

For mini-floppies, multiple track writes are usually not permitted. This is because of the turn-off time of the erase head coils—the head switches tracks before the erase head turns off. Therefore the system should typically wait 1.3 ms before attempting to step or change sides.

**WRITE DELETED DATA**

This command is the same as the Write Data Command except a Deleted Data Address Mark is written at the beginning of the Data Field instead of the normal Data Address Mark.

**READ DELETED DATA**

This command is the same as the Read Data Command except when the FDC detects a Data Address Mark at the beginning of a Data Field (and SK = 0 (low)), it will read all the data in the sector and set the CM flag in Status Register 2 to a 1 (high), and then terminates the command. If SK = 1, then the FDC skips the sector with the Data Address Mark and reads the next sector.

**READ A TRACK**

This command is similar to READ DATA Command except that the entire data field is read continuously from each of the sectors of a track. Immediately after encountering the INDEX HOLE, the FDC starts reading all data fields on the track as continuous blocks of data. If the FDC finds an error in the ID or DATA CRC check bytes, it continues to read data from the track. The FDC compares the ID information read from each sector with the value stored in the IDR, and sets the ND flag of Status Register 1 to a 1 (high) if there is no comparison. Multi-track or skip operations are not allowed with this command.

This command terminates when EOT number of sectors have been read. If the FDC does not find an ID Address Mark on the diskette after it encounters the INDEX HOLE for the second time, then it sets the MA (Missing Address Mark) flag in Status Register 1 to a 1 (high), and terminates the command. (Status Register 0 has bits 7 and 6 set to 0 and 1 respectively.)

**READ ID**

The READ ID Command is used to give the present position of the recording head. The FDC stores the values from the first ID Field it is able to read. If no proper ID Address Mark is found on the diskette, before the INDEX HOLE is encountered for the second time then the MA (Missing Address Mark) flag in Status Register 1 is set to 1 (high), and if no data is found then the ND (No Data) flag is also set in Status Register 1 to a 1 (high) and the command is terminated.

**FORMAT A TRACK**

The Format Command allows an entire track to be formatted. After the INDEX HOLE is detected, Data is written on the Diskette: Gaps, Address Marks, ID Fields and Data Fields, all per the IBM System 34 (Double Density) or System 3740 (Single Density) Format are recorded. The particular format which will be written is controlled by the values programmed into N (number of bytes/sector), SC (sectors/cylinder), GPL (Gap Length), and D (Data Pattern) which are supplied by the processor during the Command Phase. The Data Field is filled with the Byte of data stored in D. The ID Field for each sector is supplied by the processor; that is, four data requests per sector are made by the FDC for C (Cylinder Number), H (Head Number), R (Sector Number) and N (Number of Bytes/Sec). This allows the diskette to be formatted with nonsequential sector numbers, if desired.

After formatting each sector, the processor must send new values for C, H, R, and N to the 8272A for each sector on the track. The contents of the R Register is incremented by one after each sector is formatted, thus, the R register contains a value of R + 1 when it is read during the Result Phase. This incrementing and formatting continues for the whole track until the FDC encounters the INDEX HOLE for the second time, whereupon it terminates the command.

If a FAULT signal is received from the FDD at the end of a write operation, then the FDC sets the EC flag of Status Register 0 to a 1 (high), and terminates the command after setting bits 7 and 6 of Status Register 0 to 0 and 1 respectively. Also the loss of a READY signal at the beginning of a command execution phase causes command termination.

Table 9 shows the relationship between N, SC, and GPL for various sector sizes:
SCAN COMMANDS

The SCAN Commands allow data which is being read from the diskette to be compared against data which is being supplied from the main system (Processor in NON-DMA mode, and DMA Controller in DMA mode). The FDC compares the data on a byte-by-byte basis, and looks for a sector of data which meets the conditions of

\[
\text{DFDD} = \text{DProcessor} \quad \text{DFDD} < \text{DProcessor} \quad \text{or} \quad \text{DFDD} > \text{DProcessor}
\]

Ones complement arithmetic is used for comparison (FF = largest number, 00 = smallest number). After a whole sector of data is compared, if the conditions are not met, the sector number is incremented (R + 1), and the scan operation is continued. The scan operation continues until one of the following conditions occurs: the conditions for scan are met (equal, low, or high), the last sector on the track is reached (EOT), or the terminal count signal is received.

If the conditions for scan are met then the FDC sets the SH (Scan Hit) flag of Status Register 2 to a 1 (high), and terminates the Scan Command. If the conditions for scan are not met between the starting sector (as specified by R) and the last sector on the cylinder (EOT), then the FDC sets the SN (Scan Not Satisfied) flag of Status Register 2 to a 1 (high), and terminates the Scan Command. The receipt of a TERMINAL COUNT signal from the Processor or DMA Controller during the scan operation will cause the FDC to complete the comparison of the particular byte which is in process, and then to terminate the command. Table 10 shows the status of bits SH and SN under various conditions of SCAN.

Table 10. Scan Status Codes

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>STATUS REGISTER 2</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BIT 2 = SN</td>
<td>BIT 3 = SN</td>
</tr>
<tr>
<td>Scan Equal</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Scan Low or Equal</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Scan High or Equal</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

If the FDC encounters a Deleted Data Address Mark on one of the sectors (and SK = 0), then it regards the sector as the last sector on the cylinder, sets CM (Control Mark) flag of Status Register 2 to a 1 (high) and terminates the command. If SK = 1, the FDC skips the sector with the Deleted Address Mark, and reads the next sector. In the second case (SK = 1), the FDC sets the CM (Control Mark) flag of Status Register 2 to a 1 (high) in order to show that a Deleted Sector had been encountered.

When either the STP (contiguous sectors STP = 01, or alternate sectors STP = 02 sectors are read) or the MT (Multi-Track) are programmed, it is necessary to remember that the last sector on the track must be read. For example, if STP = 02, MT = 0, the sectors are numbered sequentially 1 through 26, and we start the Scan Command at sector 21; the following will happen. Sectors 21, 23, and 25 will be read, then the next sector (26) will be skipped and the Index Hole will be encountered before the EOT value of 26 can be read. This will result in an abnormal termination of the command. If the EOT had been set at 25 or the scanning started at sector 20, then the Scan Command would be completed in a normal manner.

During the Scan Command data is supplied by either the processor or DMA Controller for comparison against the data read from the diskette. In order to avoid having the OR (Over Run) flag set in Status Register 1, it is necessary to have the data available in less than 27 μs (FM Mode) or 13 μs (MFM Mode). If an Overrun occurs the FDC terminates the command.

SEEK

The read/write head within the FDD is moved from cylinder to cylinder under control of the Seek Command. The FDC compares the PCN (Present Cylinder Number) which is the current head position with the NCN (New Cylinder Number), and performs the following operation if there is a difference:

- \( \text{PCN} < \text{NCN} \): Direction signal to FDD set to a 1 (high), and Step Pulses are issued. (Step In.)
- \( \text{PCN} > \text{NCN} \): Direction signal to FDD set to a 0 (low), and Step Pulses are issued. (Step Out.)

The rate at which Step Pulses are issued is controlled by SRT (Stepping Rate Time) in the SPECIFY Command. After each Step Pulse is issued NCN is compared against PCN, and when NCN = PCN, then the SE (Seek End) flag is set in Status Register 0 to a 1 (high), and the command is terminated.
During the Command Phase of the Seek operation the FDC is in the FDC BUSY state, but during the Execution Phase it is in the NON BUSY state. While the FDC is in the NON BUSY state, another Seek Command may be issued, and in this manner parallel seek operations may be done on up to 4 Drives at once.

If an FDD is in a NOT READY state at the beginning of the command execution phase or during the seek operation, then the NR (NOT READY) flag is set in Status Register 0 to a 1 (high), and the command is terminated.

Note that the 8272A Read and Write Commands do not have implied Seeks. Any R/W command should be preceded by: 1) Seek Command; 2) Sense Interrupt Status; and 3) Read ID.

**RECALIBRATE**

This command causes the read/write head within the FDD to retract to the Track 0 position. The FDC clears the contents of the PCN counter, and checks the status of the Track 0 signal from the FDD. As long as the Track 0 signal is low, the Direction signal remains 1 (high) and Step Pulses are issued. When the Track 0 signal goes high, the SE (SEEK END) flag in Status Register 0 is set to a 1 (high) and the command is terminated. If the Track 0 signal is still low after 77 Step Pulses have been issued, the FDC sets the SE (SEEK END) and EC (EQUIPMENT CHECK) flags of Status Register 0 to both 1s (highs), and terminates the command.

The ability to overlap RECALIBRATE Commands to multiple FDDs, and the loss of the READY signal, as described in the SEEK Command, also applies to the RECALIBRATE Command.

**SENSE INTERRUPT STATUS**

An Interrupt signal is generated by the FDC for one of the following reasons:

1. Upon entering the Result Phase of:
   a. Read Data Command
   b. Read a Track Command
   c. Read ID Command
   d. Read Deleted Data Command
   e. Write Data Command
   f. Format a Cylinder Command
   g. Write Deleted Data Command
   h. Scan Commands
2. Ready Line of FDD changes state
3. End of Seek or Recalibrate Command
4. During Execution Phase in the NON-DMA Mode

Interrupts caused by reasons 1 and 4 above occur during normal command operations and are easily discernible by the processor. However, interrupts caused by reasons 2 and 3 above may be uniquely identified with the aid of the Sense Interrupt Status Command. This command when issued resets the interrupt signal and via bits 5, 6, and 7 of Status Register 0 identifies the cause of the interrupt.

Neither the Seek or Recalibrate Command have a Result Phase. Therefore, it is mandatory to use the Sense Interrupt Status Command after these commands to effectively terminate them and to provide verification of the head position (PCN).

### Table 11. Seek, Interrupt Codes

<table>
<thead>
<tr>
<th>SEEK END</th>
<th>INTERRUPT CODE</th>
<th>CAUSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BIT 5</td>
<td>BIT 6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**SPECIFY**

The Specify Command sets the initial values for each of the three internal timers. The HUT (Head Unload Time) defines the time from the end of the Execution Phase of one of the Read/Write Commands to the head unload state. This timer is programmable from 16 to 240 ms in increments of 16 ms (01 = 16 ms, 02 = 32 ms ... 0F = 240 ms). The SRT (Step Rate Time) defines the time interval between adjacent step pulses. This timer is programmable from 1 to 16 ms in increments of 1 ms (F = 1 ms, E = 2 ms, D = 3 ms, etc.). The HLT (Head Load Time) defines the time between when the Head Load signal goes high and when the Read/Write operation starts. This timer is programmable from 2 to 254 ms in increments of 2 ms (01 = 2 ms, 02 = 4 ms, 03 = 6 ms ... FE = 254 ms).

The step rate should be programmed 1 ms longer than the minimum time required by the drive.

The time intervals mentioned above are a direct function of the clock (CLK on pin 19). Times indicated above are for an 8 MHz clock, if the clock was reduced to 4 MHz (mini-floppy application) then all time intervals are increased by a factor of 2.

The choice of DMA or NON-DMA operation is made by the ND (NON-DMA) bit. When this bit is high (ND = 1) the NON-DMA mode is selected, and when ND = 0 the DMA mode is selected.

**SENSE DRIVE STATUS**

This command may be used by the processor whenever it wishes to obtain the status of the FDDs. Status Register 3 contains the Drive Status information.

**INVALID**

If an invalid command is sent to the FDC (a command not defined above), then the FDC will terminate the command. No Interrupt is generated by the 8272A during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both high ("1") indicating to the processor that the 8272A is in the Result Phase and the contents of Status Register 0 (ST0) must be read. When the processor reads Status Register 0 it will find an 80H indicating an invalid command was received.

A Sense Interrupt Status Command must be sent after a Seek or Recalibrate interrupt, otherwise the FDC will consider the next command to be an Invalid Command.

In some applications the user may wish to use this command as a No-Op command, to place the FDC in a standby or no operation state.
### Table 12. Status Registers

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>Interrupt Code</td>
<td>IC</td>
<td>D2 = 0 and D3 = 0 Normal Termination of Command, (NT). If the FDC tries to access a Sector beyond the final Sector of a Cylinder, this flag is set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D2 = 0 and D3 = 1 Abnormal Termination of Command, (AT). Execution of Command was completed and properly executed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D2 = 1 and D3 = 0 Invalid Command issue, (IC) Command which was issued was never started</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D2 = 1 and D3 = 1 Abnormal Termination because during command execution the ready signal from FDD changed state.</td>
</tr>
<tr>
<td>D5</td>
<td>Seek End</td>
<td>SE</td>
<td>When the FDC completes the SEEK Command, this flag is set to 1 (high).</td>
</tr>
<tr>
<td>D4</td>
<td>Equipment Check</td>
<td>EC</td>
<td>If a fault Signal is received from the FDD, or if the Track 0 Signal fails to occur after 77 Step Pulses (Recalibrate Command) then this flag is set.</td>
</tr>
<tr>
<td>D3</td>
<td>Not Ready</td>
<td>NR</td>
<td>When the FDC is in the not-ready state and a read or write command is issued, this flag is set. If a read or write command is issued to Side 1 of a single sided drive, then this flag is set.</td>
</tr>
<tr>
<td>D2</td>
<td>Head Address</td>
<td>HD</td>
<td>This flag is used to indicate the state of the head at Interrupt.</td>
</tr>
<tr>
<td>D1</td>
<td>Unit Select 1</td>
<td>US 1</td>
<td>These flags are used to indicate a Drive Unit Number at Interrupt</td>
</tr>
<tr>
<td>D0</td>
<td>Unit Select 0</td>
<td>US 0</td>
<td></td>
</tr>
</tbody>
</table>

### STATUS REGISTER 1

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>End of Cylinder</td>
<td>EN</td>
<td>When the FDC tries to access a Sector beyond the final Sector of a Cylinder, this flag is set</td>
</tr>
<tr>
<td>D6</td>
<td></td>
<td></td>
<td>Not used. This bit is always 0 (low).</td>
</tr>
<tr>
<td>D5</td>
<td>Data Error</td>
<td>DE</td>
<td>When the FDC detects a CRC error in either the ID field or the data field, this flag is set.</td>
</tr>
<tr>
<td>D4</td>
<td>Over Run</td>
<td>OR</td>
<td>If the FDC is not serviced by the main-systems during data transfers, within a certain time interval, this flag is set</td>
</tr>
<tr>
<td>D3</td>
<td></td>
<td></td>
<td>Not used. This bit always 0 (low)</td>
</tr>
<tr>
<td>D2</td>
<td>No Data</td>
<td>ND</td>
<td>During execution of READ DATA, WRITE DELETED DATA or SCAN Command, if the FDC cannot find the Sector specified in the IDR Register, this flag is set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>During executing the READ ID Command, if the FDC cannot read the ID field without an error, then this flag is set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>During the execution of the READ A Cylinder Command, if the starting sector cannot be found, then this flag is set.</td>
</tr>
</tbody>
</table>

### STATUS REGISTER 2

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td></td>
<td></td>
<td>Not used. This bit is always 0 (low).</td>
</tr>
<tr>
<td>D6</td>
<td>Control Mark</td>
<td>CM</td>
<td>During executing the READ DATA or SCAN Command, if the FDC encounters a Sector which contains a Deleted Data Address Mark, this flag is set.</td>
</tr>
<tr>
<td>D5</td>
<td>Data Error in Data Field</td>
<td>DD</td>
<td>If the FDC detects a CRC error in the data field then this flag is set.</td>
</tr>
<tr>
<td>D4</td>
<td>Wrong Cylinder</td>
<td>WC</td>
<td>This bit is related with the ND bit, and when the contents of C on the medium is different from that stored in the IDR, this flag is set.</td>
</tr>
<tr>
<td>D3</td>
<td>Scan Equal Hit</td>
<td>SH</td>
<td>During execution of the SCAN Command, if the condition of &quot;equal&quot; is satisfied, this flag is set.</td>
</tr>
<tr>
<td>D2</td>
<td>Scan Not Satisfied</td>
<td>SN</td>
<td>During executing the SCAN Command, if the FDC cannot find a Sector on the cylinder which meets the condition, then this flag is set.</td>
</tr>
<tr>
<td>D1</td>
<td>Bad Cylinder</td>
<td>BC</td>
<td>This bit is related with the ND bit, and when the content of C on the medium is different from that stored in the IDR and the content of C is FF, then this flag is set.</td>
</tr>
<tr>
<td>D0</td>
<td>Missing Address Mark in Data Field</td>
<td>MD</td>
<td>When data is read from the medium, if the FDC cannot find a Data Address Mark or Deleted Data Address Mark, then this flag is set.</td>
</tr>
</tbody>
</table>

### STATUS REGISTER 3

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>Fault</td>
<td>FT</td>
<td>This bit is used to indicate the status of the Fault signal from the FDD.</td>
</tr>
<tr>
<td>D6</td>
<td>Write Protected</td>
<td>WP</td>
<td>This bit is used to indicate the status of the Write Protected signal from the FDD</td>
</tr>
<tr>
<td>D5</td>
<td>Ready</td>
<td>RDY</td>
<td>This bit is used to indicate the status of the Ready signal from the FDD.</td>
</tr>
<tr>
<td>D4</td>
<td>Track 0</td>
<td>TO</td>
<td>This bit is used to indicate the status of the Track 0 signal from the FDD.</td>
</tr>
<tr>
<td>D3</td>
<td>Two Side</td>
<td>TS</td>
<td>This bit is used to indicate the status of the Two Side signal from the FDD.</td>
</tr>
<tr>
<td>D2</td>
<td>Head Address</td>
<td>HD</td>
<td>This bit is used to indicate the status of Side Select signal to the FDD.</td>
</tr>
<tr>
<td>D1</td>
<td>Unit Select 1</td>
<td>US 1</td>
<td>This bit is used to indicate the status of the Unit Select 1 signal to the FDD.</td>
</tr>
<tr>
<td>D0</td>
<td>Unit Select 0</td>
<td>US 0</td>
<td>This bit is used to indicate the status of the Unit Select 0 signal to the FDD.</td>
</tr>
</tbody>
</table>
**ABSOLUTE MAXIMUM RATINGS**

- **Operating Temperature**: 0°C to +70°C
- **Storage Temperature**: -40°C to +125°C
- **All Output Voltages**: -0.5 to +7 Volts
- **All Input Voltages**: -0.5 to +7 Volts
- **Supply Voltage $V_{CC}$**: -0.5 to +7 Volts
- **Power Dissipation**: 1 Watt

**NOTICE**: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

### D.C. CHARACTERISTICS ($T_A = 0^\circ C$ to +70°C, $V_{CC} = +5V \pm 10\%$)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IL}$</td>
<td>Input Low Voltage</td>
<td>Min.</td>
<td>Max.</td>
<td></td>
</tr>
<tr>
<td>$V_{IH}$</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>$V_{CC} + 0.5$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}$</td>
<td>Output Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OH}$</td>
<td>Output High Voltage</td>
<td>0.45</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>$I_{CC}$</td>
<td>$V_{CC}$ Supply Current</td>
<td>120</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>Input Load Current (All Input Pins)</td>
<td>10</td>
<td>$V_{IN} = V_{CC}$</td>
<td>µA</td>
</tr>
<tr>
<td>$I_{OH}$</td>
<td>High Level Output Leakage Current</td>
<td>10</td>
<td>$V_{OUT} = V_{CC}$</td>
<td>µA</td>
</tr>
<tr>
<td>$I_{OFL}$</td>
<td>Output Float Leakage Current</td>
<td>±10</td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

### CAPACITANCE ($T_A = 25^\circ C$, $f_c = 1$ MHz, $V_{CC} = 0V$)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{IN(h)}$</td>
<td>Clock Input Capacitance</td>
<td>*</td>
<td>20</td>
<td>pF</td>
</tr>
<tr>
<td>$C_{IN}$</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td></td>
</tr>
<tr>
<td>$C_{IO}$</td>
<td>Input/Output Capacitance</td>
<td>20</td>
<td>pF</td>
<td></td>
</tr>
</tbody>
</table>

### A.C. CHARACTERISTICS ($T_A = 0^\circ C$ to +70°C, $V_{CC} = +5.0V \pm 10\%$)

#### CLOCK TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$tCy$</td>
<td>Clock Period</td>
<td>120</td>
<td>500</td>
</tr>
<tr>
<td>$tCH$</td>
<td>Clock High Period</td>
<td>40</td>
<td>ns</td>
</tr>
<tr>
<td>$tRST$</td>
<td>Reset Width</td>
<td>14</td>
<td>$t_{Cy}$</td>
</tr>
</tbody>
</table>

#### READ CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$tLR$</td>
<td>Select Setup to RDI</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>$tRA$</td>
<td>Select Hold from RDI</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>$tRP$</td>
<td>RD Pulse Width</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>$tRD$</td>
<td>Data Delay from RDI</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>$tDF$</td>
<td>Output Float Delay</td>
<td>20</td>
<td>100</td>
</tr>
</tbody>
</table>
### A.C. CHARACTERISTICS (Continued)  
\((T_A=\text{0}^\circ\text{C to } +70^\circ\text{C}, \text{V}_{\text{CC}}= +5.0\text{V} \pm 10\% )\)

#### WRITE CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Typ.(^1)</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>t\text{AW}</td>
<td>Select Setup to WR(^!)</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t\text{WA}</td>
<td>Select Hold from WR(^!)</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t\text{WW}</td>
<td>WR Pulse Width</td>
<td>250</td>
<td>250</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t\text{DW}</td>
<td>Data Setup to WR(^!)</td>
<td>150</td>
<td>150</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t\text{WD}</td>
<td>Data Hold from WR(^!)</td>
<td>5</td>
<td>5</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### INTERRUPTS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>t\text{RI}</td>
<td>INT Delay from RD(^!)</td>
<td>500 ns</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{WI}</td>
<td>INT Delay from WR(^!)</td>
<td>500 ns</td>
<td>Note 6</td>
</tr>
</tbody>
</table>

#### DMA

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>t\text{RCY}</td>
<td>DRQ Cycle Period</td>
<td>13 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{AKRQ}</td>
<td>DACK(^!) to DRQ(^!)</td>
<td>200 ns</td>
<td></td>
</tr>
<tr>
<td>t\text{RQ}</td>
<td>DRQ(^!) to RD(^!)</td>
<td>800 ns</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{ROW}</td>
<td>DRQ(^!) to WR(^!)</td>
<td>250 ns</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{RQROW}</td>
<td>DRQ(^!) to RD(^!) or WR(^!)</td>
<td>12 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
</tbody>
</table>

#### FDD INTERFACE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>t\text{WCY}</td>
<td>WCK Cycle Time</td>
<td>2 or 4 (\mu\text{s})</td>
<td>MFM = 0, MFM = 1 Note 2</td>
</tr>
<tr>
<td>t\text{WCH}</td>
<td>WCK High Time</td>
<td>250</td>
<td>80</td>
</tr>
<tr>
<td>t\text{CP}</td>
<td>Pre-Shift Delay from WCK(^!)</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>t\text{CD}</td>
<td>WDA Delay from WCK(^!)</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>t\text{WD}</td>
<td>Write Data Width</td>
<td>(t\text{WCH} - 50)</td>
<td>ns</td>
</tr>
<tr>
<td>t\text{WE}</td>
<td>WE(^!) to WCK(^!) or WE(^!) to WCK(^!) Delay</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>t\text{WCY}</td>
<td>Window Cycle Time</td>
<td>2 (\mu\text{s})</td>
<td>MFM = 0, MFM = 1</td>
</tr>
<tr>
<td>t\text{WD}</td>
<td>Window Setup to RDD(^!)</td>
<td>15</td>
<td>ns</td>
</tr>
<tr>
<td>t\text{RDW}</td>
<td>Window Hold from RDD(^!)</td>
<td>15</td>
<td>ns</td>
</tr>
<tr>
<td>t\text{DD}</td>
<td>RDD Active Time (HIGH)</td>
<td>40</td>
<td>ns</td>
</tr>
</tbody>
</table>

#### FDD SEEK/DIRECTION/STEP

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>t\text{US}</td>
<td>US(0,1) Setup to RW\text{SEEK}!</td>
<td>12 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{SU}</td>
<td>US(0,1) Hold after RW\text{SEEK}!</td>
<td>15 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{SD}</td>
<td>RW\text{SEEK} Setup to LCT/DIR</td>
<td>7 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{DS}</td>
<td>RW\text{SEEK} Hold from LCT/DIR</td>
<td>30 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{DST}</td>
<td>LCT/DIR Setup to FR/STEP!</td>
<td>1 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{STD}</td>
<td>LCT/DIR Hold from FR/STEP!</td>
<td>24 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{STU}</td>
<td>DS(0,1) Hold from FR/STEP!</td>
<td>5 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{STPV}</td>
<td>STEP Active Time (High)</td>
<td>5 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{SC}</td>
<td>STEP Cycle Time</td>
<td>33 (\mu\text{s})</td>
<td>Note 3, 6</td>
</tr>
<tr>
<td>t\text{SF}</td>
<td>FAULT RESET Active Time (High)</td>
<td>8 (\mu\text{s})</td>
<td>Note 6</td>
</tr>
<tr>
<td>t\text{IDX}</td>
<td>INDEX Pulse Width</td>
<td>10 (\mu\text{s})</td>
<td>1CY</td>
</tr>
<tr>
<td>t\text{ITC}</td>
<td>Terminal Count Width</td>
<td>1 (\mu\text{s})</td>
<td>1CY</td>
</tr>
</tbody>
</table>

#### NOTES:

1. Typical values for \(T_A = 25\text{C}\) and nominal supply voltage.
2. The former values are used for standard floppy and the latter values are used for mini-floppies.
3. \(t\text{SC} = 33 \mu\text{s}\) min. is for different drive units. In the case of same unit, \(t\text{SC}\) can be ranged from 1 ms to 16 ms with 8 MHz clock period, and 2 ms to 32 ms with 4 MHz clock, under software control.
4. From 2.0V to +2.0V.
5. At 4 MHz, the clock duty cycle may range from 16% to 76%. Using an 8 MHz clock the duty cycle can range from 32% to 52%. Duty cycle is defined as: \(D\text{C} = 100 (t\text{CH} - 1\text{CY})\) with typical rise and fall times of 5 ns.
6. The specified values listed are for an 8 MHz clock period. Multiply timings by 2 when using a 4 MHz clock period.
A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC '1' AND 0.45V FOR A LOGIC '0'. TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC '1' AND 0.8V FOR A LOGIC '0'.

A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

\[ C_L = 100 \text{ pF} \]

\[ C_L \text{ INCLUDES JIG CAPACITANCE} \]

WAVEFORMS

PROCESSOR READ OPERATION

DATA

\[ t_{enu} \]

\[ t_{enu} \]

\[ t_{enu} \]

\[ t_{enu} \]

\[ t_{enu} \]

\[ t_{enu} \]

\[ t_{enu} \]

\[ t_{enu} \]

INT

\[ t_{enu} \]

\[ t_{enu} \]
WAVEFORMS (Continued)

PROCESSOR WRITE OPERATION

A, CS, DACK

WR

DATA

INT

DMA OPERATION

DRQ

DACK

WR or RD
WAVEFORMS (Continued)

CLOCK TIMING

FDD WRITE OPERATION

<table>
<thead>
<tr>
<th>PRESHIFT 0</th>
<th>PRESHIFT 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL</td>
<td>0</td>
</tr>
<tr>
<td>LATE</td>
<td>0</td>
</tr>
<tr>
<td>EARLY</td>
<td>1</td>
</tr>
<tr>
<td>INVALID</td>
<td>1</td>
</tr>
</tbody>
</table>
WAVEFORMS (Continued)

SEEK OPERATION

- DSO: (Direct Seek Operation)
- RW:SEEK (Read/Write Seek Operation)
- LCTI DIRECTION
- STEP

FLT RESET

FMT RESET
FAIL UNSAFE RESET

INDEX

INDEX

faULt RESET
FAIL UNSAFE RESET

faULt RESET
FAIL UNSAFE RESET

INDEX

INDEX
FDD READ OPERATION

TERMINAL COUNT

RESET
An Intelligent Data Base System Using the 8272

Tom Rossi
Peripherals Applications Manager

March 1981
An Intelligent Data Base System Using the 8272

Contents

INTRODUCTION
The Floppy Disk
The Floppy Disk Drive

SUBSYSTEM OVERVIEW
Controller Electronics
Drive Electronics
Controller/Drive Interface
Processor/Memory Interface

DISK FORMAT
Data Recording Techniques
Sectors
Tracks
Sector Interleaving

THE 8272 FLEXIBLE DISKETTE CONTROLLER
Floppy Disk Commands
Interface Registers
Command/Result Phases
Execution Phase
Multi-sector and Multi-track Transfers
Drive Status Polling
Command Details

THE DATA SEPARATOR
Single Density
Double Density
Phase-Locked Loop Design
Initialization
Floppy Disk Data
Startup
PLL Synchronization

AN INTELLIGENT DISKETTE DATA BASE SYSTEM
Processor and Memory
Serial I/O
DMA
Disk Drive Interface

SPECIAL CONSIDERATIONS

APPENDIX
Schematics
Power Distribution
1. INTRODUCTION

Most microcomputer systems in use today require low-cost, high-density removable magnetic media for information storage. In the area of removable media, a designer's choice is limited to magnetic tapes and floppy disks (flexible diskettes), both of which offer non-volatile data storage. The choice between these two technologies is relatively straightforward for a given application. Since disk drives are designed to permit random access to stored information, they are significantly faster than tape units. For example, locating information on a disk requires less than a second, while tape movement (even at the fastest rewind or fast-forward speed) often requires several minutes. This random access ability permits the use of floppy disks in on-line storage applications (where information must be located, read, and modified/updated in real-time under program or operator control). Tapes, on the other hand, are ideally suited to archival or back-up storage due to their large storage capacities (more than 10 million bytes of data can be archived on a cartridge tape).

A sophisticated controller is required to capitalize on the abilities of the disk storage unit. In the past, disk controller designs have required upwards of 150 ICs. Today, the single-chip 8272 Floppy Disk Controller (FDC) plus approximately 30 support devices can handle up to four million bytes of on-line data storage on four floppy disk drives.

The Floppy Disk

A floppy disk is a circular piece of thin plastic material covered with a magnetic coating and enclosed in a protective jacket (Figure 1). The circular piece of plastic revolves at a fixed speed (approximately 360 rpm) within its jacket in much the same manner that a record revolves on a stereo turntable. Disks are manufactured in a variety of configurations for various storage capacities. Two standard physical disk sizes are commonly used. The 8-inch disk (8 inches square) is the larger of the two sizes; the smaller size (5-1/4 inches square) is often referred to as a mini-floppy. Single-sided disks can record information on only one side of the disk, while double-sided disks increase the storage capacity by recording on both sides. In addition, disks are classified as single-density or double-density. Double-density disks use a modified recording method to store twice as much information in the same disk area as can be stored on a single-density disk. Table 1 lists storage capacities for standard floppy disk media.

A magnetic head assembly (in contact with the disk) writes information onto the disk surface and subsequently reads the data back. This head assembly can move from the outside edge of the disk toward the center in fixed increments. Once the head assembly is positioned at one of these fixed positions, the head can read or write information in a circular path as the disk revolves beneath the head assembly. This method divides the surface into a fixed number of cylinders (as shown in Figure 2). There are normally 77 cylinders on a standard disk. Once the head assembly is positioned at a given cylinder, data may be read or written on either side of the disk. The appropriate side of the disk is selected by the read/write head address (zero or one). Of course, a single-sided disk can only use head zero. The combination of cylinder address and head address uniquely specifies a single circular track on the disk. The physical beginning of a track is located by means of a small hole (physical index mark) punched through the plastic near the center of the disk. This hole is optically sensed by the drive on every revolution of the disk.
Each track is subdivided into a number of sectors (see detailed discussion in section 3). Sectors are generally 128, 256, 512, or 1024 data bytes in length. This track sectoring may be accomplished by one of two techniques: hard sectoring or soft sectoring. Hard sectored disks divide each track into a maximum of 32 sectors. The beginning of each sector is indicated by a sector hole punched in the disk plastic. Soft sectoring, the IBM standard method, allows software selection of sector sizes. With this technique, each data sector is preceded by a unique sector identifier that is read/written by the disk controller.

A floppy disk may also contain a write protect notch punched at the edge of the outer jacket of the disk. This notch is detected by the drive and passed to the controller as a write protect signal.

The Floppy Disk Drive

The floppy disk drive is an electromechanical device that records data on, or reads data from, the surface of a floppy disk. The disk drive contains head control electronics that move the head assembly one increment (step) forward (toward the center of the disk) or backward (toward the edge of the disk). Since the recording head must be in contact with the disk material in order to read or write information, the disk drive also contains head-load electronics. Normally the read/write head is unloaded until it is necessary to read or write information on the floppy disk. Once the head assembly has been positioned over the correct track on the disk, the head is loaded (brought into contact with the disk). This sequence prevents excessive disk wear. A small time penalty is paid when the head is loaded. Approximately thirty to fifty milliseconds are needed before data may be reliably read from, or written to, the disk. This time is known as the head load time. If desired, the head may be moved from cylinder to cylinder while loaded. In this manner, only a small time interval (head settling time) is required before data may be read from, or written to, the disk. The drive provides additional signals to the system controller regarding the status of the drive and disk. These signals include:

Drive Ready — Signals the system that the drive door is closed and that a floppy disk is inserted into the drive.

Track Zero — Indicates that the head assembly is located over the outermost track of the disk. This signal may be used for calibration of the disk drive at system initialization and after an error condition.

Write Protect — Indicates that the floppy disk loaded into the drive is write protected.

Dual Sided — Indicates that the floppy disk in the drive is dual-sided.

Write Fault — Indicates that an error occurred during a recording operation.

Index — Inform the system that the physical index mark of the floppy disk (signifying the start of a data track) has been sensed.

Finally, the drive provides additional signals to the system controller regarding the status of the drive and disk. These signals include:

Drive Ready — Signals the system that the drive door is closed and that a floppy disk is inserted into the drive.

Track Zero — Indicates that the head assembly is located over the outermost track of the disk. This signal may be used for calibration of the disk drive at system initialization and after an error condition.

Write Protect — Indicates that the floppy disk loaded into the drive is write protected.

Dual Sided — Indicates that the floppy disk in the drive is dual-sided.

Write Fault — Indicates that an error occurred during a recording operation.

Index — Inform the system that the physical index mark of the floppy disk (signifying the start of a data track) has been sensed.

The Floppy Disk Drive

The floppy disk drive is an electromechanical device that records data on, or reads data from, the surface of a floppy disk. The disk drive contains head control electronics that move the head assembly one increment (step) forward (toward the center of the disk) or backward (toward the edge of the disk). Since the recording head must be in contact with the disk material in order to read or write information, the disk drive also contains head-load electronics. Normally the read/write head is unloaded until it is necessary to read or write information on the floppy disk. Once the head assembly has been positioned over the correct track on the disk, the head is loaded (brought into contact with the disk). This sequence prevents excessive disk wear. A small time penalty is paid when the head is loaded. Approximately thirty to fifty milliseconds are needed before data may be reliably read from, or written to, the disk. This time is known as the head load time. If desired, the head may be moved from cylinder to cylinder while loaded. In this manner, only a small time interval (head settling time) is required before data may be read from, or written to, the disk. The drive provides additional signals to the system controller regarding the status of the drive and disk. These signals include:

Drive Ready — Signals the system that the drive door is closed and that a floppy disk is inserted into the drive.

Track Zero — Indicates that the head assembly is located over the outermost track of the disk. This signal may be used for calibration of the disk drive at system initialization and after an error condition.

Write Protect — Indicates that the floppy disk loaded into the drive is write protected.

Dual Sided — Indicates that the floppy disk in the drive is dual-sided.

Write Fault — Indicates that an error occurred during a recording operation.

Index — Inform the system that the physical index mark of the floppy disk (signifying the start of a data track) has been sensed.

Finally, the drive provides additional signals to the system controller regarding the status of the drive and disk. These signals include:

Drive Ready — Signals the system that the drive door is closed and that a floppy disk is inserted into the drive.

Track Zero — Indicates that the head assembly is located over the outermost track of the disk. This signal may be used for calibration of the disk drive at system initialization and after an error condition.

Write Protect — Indicates that the floppy disk loaded into the drive is write protected.

Dual Sided — Indicates that the floppy disk in the drive is dual-sided.

Write Fault — Indicates that an error occurred during a recording operation.

Index — Inform the system that the physical index mark of the floppy disk (signifying the start of a data track) has been sensed.
2. SUBSYSTEM OVERVIEW

A disk subsystem consists of the following functional electronic units:

1. Disk Controller Electronics
2. Disk Drive Electronics
3. Controller/Disk Interface (cables, drivers, terminators)
4. Controller/Microprocessor System Interface

The operation of these functional units is discussed in the following paragraphs.

Controller Electronics

The disk controller is responsible for converting high-level disk commands (normally issued by software executing on the system processor) into disk drive commands. This function includes:

1. Disk Drive Selection — Disk controllers typically manage the operations of multiple floppy disk drives. This controller function permits the system processor to specify which drive is to be used in a particular operation.

2. Track Selection — The controller issues a timed sequence of step pulses to move the head from its current location to the proper disk cylinder from which data is to be read or to which data is to be written. The controller stores the current cylinder number and computes the stepping distance from the current cylinder to the specified cylinder. The controller also manages the head select signal to select the correct side of the floppy disk.

3. Sector Selection — The controller monitors the data on a track until the requested sector is sensed.

4. Head Loading — The disk controller determines the times at which the head assembly is to be brought in contact with the disk surface in order to read or write data. The controller is also responsible for waiting until the head has settled before reading or writing information. Often the controller maintains the head loaded condition for up to 16 disk revolutions (approximately 2 seconds) after a read or write operation has been completed. This feature eliminates the head load time during periods of heavy disk I/O activity.

5. Data Separation — The actual signal recorded on a floppy disk is a combination of timing information (clock) and data. The serial READ DATA input (from the disk drive) must be converted into two signal streams: clock and data. (The READ DATA input operates at 250K bits/second for single-density disks and 500K bits/second for double-density disks.) The serial data must also be assembled into 8-bit bytes for transfer to system memory. A byte must be assembled and transferred every 32 microseconds for single-density disks and every 16 microseconds for double-density.

6. Error Checking — Information recorded on a floppy disk is subject to both hard and soft errors. Hard (permanent) errors are caused by media defects. Soft errors, on the other hand, are temporary errors caused by electromagnetic noise or mechanical interference. Disk controllers use a standard error checking technique known as a Cyclic Redundancy Check (CRC). As data is written to a disk, a 16-bit CRC character is computed and also stored on the disk. When the data is subsequently read, the CRC character allows the controller to detect data errors. Typically, when CRC errors are detected, the controlling software retries the failed operation (attempting to recover from a soft error). If data cannot reliably be read or written after a number of retries, the system software normally reports the error to the operator. Multiple CRC errors normally indicate unrecoverable media error on the current disk track. Subsequent recovery attempts must be defined by the system designers and tailored to meet system interfacing requirements.

Today, single-chip digital LSI floppy disk controllers such as the 8272 perform all the above functions with the exception of data separation. A data separation circuit (a combination of digital and analog electronics) synchronizes itself to the actual data rate of the disk drive. This data rate varies from drive to drive (due to mechanical factors such as motor tolerances) and varies from disk to disk (due to temperature effects). In order to operate reliably with both single- and double-density storage, the data separation circuit must be based on phase-locked loop (PLL) technology. The phase-locked loop data separation logic is described in section 5. The separation logic, after synchronizing with the data stream, supplies a data window to the LSI disk controller. This window differentiates data information from clock information within the serial stream. The controller uses this window to reconstruct the data previously recorded on the floppy disk.

Drive Electronics

Each floppy disk drive contains digital electronic circuits that translate TTL-compatible command signals into electromechanical operations (such as drive selection and head movement/loading) and that sense and report disk or drive status to the controller (e.g., drive ready, write fault, and write protect). In addition, the drive electronics contain analog components to sense, amplify, and shape data pulses read from, or written to, the floppy disk surface by the read/write head.
Controller/Drive Interface

The controller/drive interface consists of high-current line drivers, Schmitt triggered input gates, and flat or twisted pair cable(s) to connect the disk drive electronics to the controller electronics. Each interface signal line is resistively terminated at the end of the cable farthest from the line drivers. Eight-inch drives may be directly interfaced by means of 50-conductor flat cable. Generally, cable lengths should be less than ten feet in order to maintain noise immunity.

Normally, provisions are made for up to four disk drives to share the same interface cable. The controller may operate as many cable assemblies as practical. LSI floppy disk controllers typically operate one to four drives on a single cable.

Processor/Memory Interface

The disk controller must interface to the system processor and memory for two distinct purposes. First, the processor must specify disk control and command parameters to the controller. These parameters include the selection of the recording density and specification of disk formatting information (discussed in section 3). In addition to disk parameter specification, the processor must also send commands (e.g., read, write, seek, and scan) to the controller. These commands require the specification of the command code, drive number, cylinder address, sector address, and head address. Most LSI controllers receive commands and parameters by means of processor I/O instructions.

In addition to this I/O interface, the controller must also be designed for high-speed data transfer between memory and the disk drive. Two implementation methods may be used to coordinate this data transfer. The lowest-cost method requires direct processor intervention in the transfer. With this method, the controller issues an interrupt to the processor for each data transfer. (An equivalent method allows the processor to poll an interrupt flag in the controller status word.) In the case of a disk write operation, the processor writes a data byte (to be encoded into the serial output stream) to the disk controller following the receipt of each controller interrupt. During a disk read operation, the processor reads a data byte (previously assembled from the input data stream) from the controller after each interrupt. The processor must transfer a data byte from the controller to memory or transfer a data byte from memory to the disk controller within 16 or 32 microseconds after each interrupt (double-density and single-density response times, respectively).

If the system processor must service a variety of other interrupt sources, this interrupt method may not be practical, especially in double-density systems. In this case, the disk controller may be interfaced to a Direct Memory Access (DMA) controller. When the disk controller requires the transfer of a data byte, it simply activates the DMA request line. The DMA controller interfaces to the processor and, in response to the disk controller's request, gains control of the memory interface for a short period of time—long enough to transfer the requested data byte to/from memory. See section 6 for a detailed DMA interface description.

3. DISK FORMAT

New floppy disks must be written with a fixed format by the controller before these disks may be used to store data. Formatting is a method of taking raw media and adding the necessary information to permit the controller to read and write data without error. All formatting is performed by the disk controller on a track-by-track basis under the direction of the system processor. Generally, a track may be formatted at any time. However, since formatting "initializes" a complete disk track, all previously written data is lost (after a format operation). A format operation is normally used only when initializing new floppy disks. Since soft-sectoring in such a predominant formatting technique (due to IBM's influence), the following discussion will limit itself to soft-sectored formats.

Data Recording Techniques

Two standard data recording techniques are used to combine clock and data information for storage on a floppy disk. The single-density technique is referred to as FM encoding. In FM encoding (see Figure 3), a double frequency encoding technique is used that inserts a data bit between two adjacent clock bits. (The presence of a data bit represents a binary "one" while the absence of a data bit represents a binary "zero.") The two adjacent clock bits are referred to as a bit cell, and except for unique field identifiers, all clock bits written on the disk are binary "ones." In FM encoding, each data bit is written at the center of the bit cell and the clock bits are written at the leading edge of the bit cell.

The encoding used for double-density recording is termed MFM encoding (for "Modified FM"). In MFM encoding (Figure 3) the data bits are again written at the center of the bit cell. However, a clock bit is written at the leading edge of the bit cell only if no data bit was written in the previous bit cell and no data bit will be written in the present bit cell.

Sectors

Soft-sectored floppy disks divide each track into a number of data sectors. Typically, sector sizes of 128, 256, 512, or 1024 data bytes are permitted. The sector size is specified when the track is initially formatted by the controller. Table 1 lists the single- and double-
density data storage capacities for each of the four sector sizes. Each sector within a track is composed of the following four fields (illustrated in Figure 4):

1. Sector ID Field — This field, consisting of seven bytes, is written only when the track is formatted. The ID field provides the sector identification that is used by the controller when a sector must be read or written. The first byte of the field is the ID address mark, a unique coding that specifies the beginning of the ID field. The second, third, and fourth bytes are the cylinder, head, and sector addresses, respectively, and the fifth byte is the sector length code. The last two bytes are the 16-bit CRC character for the ID field. During formatting, the controller supplies the address mark. The cylinder, head, and sector addresses and the sector length code are supplied to the controller by the processor software. The CRC character is derived by the controller from the data in the first five bytes.

2. Post ID Field Gap — The post ID field gap (gap 2) is written initially when the track is formatted. During subsequent write operations, the drive's write circuitry is enabled within the gap and the trailing bytes of the gap are rewritten each time the sector is updated (written). During subsequent read operations, the trailing bytes of the gap are used to synchronize the data separator logic with the upcoming data field.

3. Data Field — The length (number of data bytes) of the data field is determined by software when the track is formatted. The first byte of the data field is the data address mark, a unique coding that specifies the beginning of the data field. When a sector is to be deleted, (e.g., a hard error on the disk), a deleted data address mark is written in place of the data address mark. The last two bytes of the data field comprise the CRC character.

4. Post Data Field Gap — The post data field gap (gap 3) is written when the track is formatted and separates the preceding data field from the next physical ID field on the track. Note that a post data field gap is not written following the last physical sector on a track. The gap itself contains a program-selectable number of bytes. Following a sector update (write) operation, the drive's write logic is disabled during the gap. The actual size of gap 3 is determined by the maximum number of data bits that can be recorded on a track, the number of sectors per track and the total sector size (data plus overhead information). The gap size must be adjusted so that it is large enough to contain the discontinuity generated on the floppy disk when the write current is turned on or off (at the start or completion of a disk write operation) and to contain a synchronization field for the upcoming ID field (of the next sector). On the other hand, the gaps must be small enough so that the total number of data bits required on the track (sectors plus gaps) is less than the maximum number of data bits that can be recorded on the track. The gap size must be specified for all read, write, and format operations. The gap size used during disk reads and writes must be smaller than the size used to format the disk to avoid the splice points between contiguous physical sectors. Suggested gap sizes are listed in Table 9.

---

**Figure 3. FM and MFM Encoding**

NOTE THAT THE FM BIT CELL IS TWICE THE SIZE OF THE MFM BIT CELL. THUS, THE FM TIME SCALE IN THIS FIGURE IS 4 \( \mu \text{s} \)/BIT WHILE THE MFM TIME SCALE IS 2 \( \mu \text{s} \)/BIT.
APPLICATIONS

Tracks

The overall format for a track is illustrated in Figure 4. Each track consists of the following fields:

1. Pre-Index Gap — The pre-index gap (gap 5) is written only when the track is formatted.
2. Index Address Mark — The index address mark consists of a unique code that indicates the beginning of a data track. One index mark is written on each track when the track is formatted.
3. Post Index Gap — The post index gap (gap 1) is used during disk read and write operations to synchronize the data separator logic with the data to be read from the ID field (of the first sector). The post index gap is written only when the disk is formatted.
4. Sectors — The sector information (discussed above) is repeated once for each sector on the track.
5. Final Gap — The final gap (gap 4) is written when the track is formatted and extends from the last physical data field on the track to the physical index mark. The length of this gap is dependent on the number of bytes per sector specified, the lengths of the program-selectable gaps specified, and the drive speed.

![Standard Floppy Diskette Track Format](image)

Figure 4. Standard Floppy Diskette Track Format (From SBC 204 Manual)
APPLICATIONS

Sector Interleaving

The initial formatting of a floppy disk determines where sectors are located within a track. It is not necessary to allocate sectors sequentially around the track (i.e., 1, 2, 3, ..., 26). In fact, it is often advantageous to place the sectors on the track in a non-sequential order. Sequential sector ordering optimizes sector access times during multi-sector transfers (e.g., when a program is loaded) by permitting the number of sectors specified (up to an entire track) to be transferred within a single revolution of the disk. A technique known as sector interleaving optimizes access times when, although sectors are accessed sequentially, a small amount of processing must be performed between sector reads/writes. For example, an editing program performing a text search reads sectors sequentially, and after each sector is read, performs a software search. If a match is not found, the software issues a read request for the next sector. Since the floppy disk continues to rotate during the time that the software executes, the next physical sector is already passing under the read/write head when the read request is issued, and the processor must wait for another complete revolution of the disk (approximately 166 milliseconds) before the data may actually be input. With interleaving, the sectors are not stored sequentially on a track; rather, each sector is physically removed from the previous sector by some number (known as the interleave factor) of physical sectors as shown in Figure 5. This method of sector allocation provides the processor additional execution time between sectors on the disk. For example, with a 26 sector/track format, an interleave factor of 2 provides 6.4 milliseconds of processing time between sequential 128 byte sector accesses.

To calculate the correct interleave factor, the maximum processor time between sector operations must be divided by the time required for a complete sector to pass under the disk read/write head. After determining the interleave factor, the correct sector numbers are passed to the disk controller (in the exact order that they are to physically appear on the track) during the execution of a format operation.

4. THE 8272 FLEXIBLE DISKETTE CONTROLLER

The 8272 is a single-chip LSI Floppy Disk Controller (FDC) that contains the circuitry necessary to implement both single-and double-density floppy disk storage subsystems (with up to four dual-sided disk drives per FDC). The 8272 supports the IBM 3740 single-density recording format (FM) and the IBM System 34 double-density recording format (FM). With the 8272, less than 30 ICs are needed to implement a complete disk subsystem. The 8272 accepts and executes high-level disk commands such as format track, seek, read sector, write sector, and read track. All data synchronization and error checking is automatically performed by the FDC to ensure reliable data storage and subsequent retrieval. External logic is required only for the generation of the FDC master clock and write clock (see Section 6) and for data separation (Section 5). The FDC provides signals that control the startup and base frequency selection of the data separator. These signals greatly ease the design of a phase-locked loop data separator.

In addition to the data separator interface signals, the 8272 also provides the necessary signals to interface to microprocessor systems with or without Direct Memory Access (DMA) capabilities. In order to interface to a large number of commercially available floppy disk drives, the FDC permits software specification of the track stepping rate, the head load time, and the head-unload time.

The pin configuration and internal block diagram of the 8272 is shown in Figure 6. Table 2 contains a description for each FDC interface pin.

Floppy Disk Commands

The 8272 executes fifteen high-level disk interface commands:

- Specify
- Sense Drive Status
- Sense Interrupt Status
- Seek
- Recalibrate
- Format Track
- Read Data
- Read Deleted Data
- Write Data
- Write Deleted Data
- Read Track
- Read ID
- Scan Equal
- Scan High or Equal
- Scan Low or Equal

Figure 5. Interleaved Sector Allocation Within a Track
APPLICATIONS

Each command is initiated by a multi-byte transfer from the processor to the FDC (the transferred bytes contain command and parameter information). After complete command specification, the FDC automatically executes the command. The command result data (after execution of the command) may require a multi-byte transfer of status information back to the processor. It is convenient to consider each FDC command as consisting of the following three phases:

**COMMAND PHASE:** The executing program transfers to the FDC all the information required to perform a particular disk operation. The 8272 automatically enters the command phase after RESET and following the completion of the result phase (if any) of a previous command.

**EXECUTION PHASE:** The FDC performs the operation as instructed. The execution phase is entered immediately after the last command parameter is written to the FDC in the preceding command phase. The execution phase normally ends when the last data byte is transferred to/from the disk (signalled by the TC input to the FDC) or when an error occurs.

**RESULT PHASE:**

After completion of the disk operation, status and other housekeeping information are made available to the processor. After the processor reads this information, the FDC reenters the command phase and is ready to accept another command.

---

**Figure 6. 8272 Pin Configuration and Internal Block Diagram**

---

**5-472**

207875-002
### Table 2. 8272 FDC Pin Description

<table>
<thead>
<tr>
<th>Number</th>
<th>Pin Symbol</th>
<th>I/O</th>
<th>To/From</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RST</td>
<td>I</td>
<td>uP</td>
<td>Reset. Active-high signal that places the FDC in the “idle” state and all disk drive output signals are forced inactive (low). This input must be held active during power on reset while the RD and WR inputs are active.</td>
</tr>
<tr>
<td>2</td>
<td>RD</td>
<td>I*</td>
<td>uP</td>
<td>Read. Active-low control signal that enables data transfer from the FDC to the data bus.</td>
</tr>
<tr>
<td>3</td>
<td>WR</td>
<td>I*</td>
<td>uP</td>
<td>Write. Active-low control signal that enables data transfer from the data bus into the FDC.</td>
</tr>
<tr>
<td>4</td>
<td>CS</td>
<td>I</td>
<td>uP</td>
<td>Chip Select. Active-low control signal that selects the FDC. No reading or writing will occur unless the FDC is selected.</td>
</tr>
<tr>
<td>5</td>
<td>A₀</td>
<td>I*</td>
<td>uP</td>
<td>Address. Selects the Data Register or Main Status Register for input/output in conjunction with the RD and WR inputs. (See Table 3.)</td>
</tr>
<tr>
<td>14</td>
<td>DRQ</td>
<td>O</td>
<td>DMA</td>
<td>DMA Request. Active-high output that indicates an FDC request for DMA services.</td>
</tr>
<tr>
<td>15</td>
<td>DACK</td>
<td>I</td>
<td>DMA</td>
<td>DMA Acknowledge. Active-low control signal indicating that the requested DMA transfer is in progress.</td>
</tr>
<tr>
<td>16</td>
<td>TC</td>
<td>I</td>
<td>DMA</td>
<td>Terminal Count. Active-high signal that causes the termination of a command. Normally, the terminal count input is directly connected to the TC/EOP output from the DMA controller, signalling that the DMA transfer has been completed. In a non-DMA environment, the processor must count data transfers and supply a TC signal to the FDC.</td>
</tr>
<tr>
<td>17</td>
<td>IDX</td>
<td>I</td>
<td>Drive</td>
<td>Index. Indicates detection of the physical index mark (the beginning of a track) on the selected disk drive.</td>
</tr>
<tr>
<td>18</td>
<td>INT</td>
<td>O</td>
<td>uP</td>
<td>Interrupt Request. Active-high signal indicating an 8272 interrupt service request.</td>
</tr>
<tr>
<td>19</td>
<td>CLK</td>
<td>I</td>
<td></td>
<td>Clock. Signal phase 8 MHz clock (50% duty cycle).</td>
</tr>
<tr>
<td>20</td>
<td>GND</td>
<td></td>
<td></td>
<td>Ground. DC power return.</td>
</tr>
<tr>
<td>21</td>
<td>WR CLK</td>
<td>I</td>
<td></td>
<td>Write Clock. 500 kHz (FM) or 1 MHz (MFM) write clock with a constant pulse width of 250 ns (for both FM and MFM recording). The write clock must be present at all times.</td>
</tr>
<tr>
<td>22</td>
<td>DW</td>
<td>I</td>
<td>PLL</td>
<td>Data Window. Data sample signal from the phase-locked loop indicating that the FDC should sample input data from the disk drive.</td>
</tr>
<tr>
<td>23</td>
<td>RD DATA</td>
<td>I</td>
<td>Drive</td>
<td>Read Data. FDC input data from the selected disk drive.</td>
</tr>
<tr>
<td>24</td>
<td>VCO</td>
<td>O</td>
<td>PLL</td>
<td>VCO Sync. Active-high output that enables the phase-locked loop to synchronize with the input data from the disk drive.</td>
</tr>
<tr>
<td>25</td>
<td>WE</td>
<td>O</td>
<td>Drive</td>
<td>Write Enable. Active-high output that enables the disk drive write gate.</td>
</tr>
<tr>
<td>26</td>
<td>MFM</td>
<td>O</td>
<td>PLL</td>
<td>MFM Mode. Active-high output used by external logic to enable the MFM double-density recording mode. When the MFM output is low, single-density FM recording is indicated.</td>
</tr>
<tr>
<td>27</td>
<td>HDSEL</td>
<td>O</td>
<td>Drive</td>
<td>Head Select. Selects head 0 or head 1 on a dual-sided disk.</td>
</tr>
<tr>
<td>28,29</td>
<td>DS₁,DS₀</td>
<td>O</td>
<td>Drive</td>
<td>Drive Select. Selects one of four disk drives.</td>
</tr>
<tr>
<td>30</td>
<td>WR DATA</td>
<td>O</td>
<td>Drive</td>
<td>Write Data. Serial data stream (combination of clock and data bits) to be written on the disk.</td>
</tr>
<tr>
<td>31,32</td>
<td>PS₁,PS₀</td>
<td>O</td>
<td>Drive</td>
<td>Precompensation (pre-shift) Control. Write precompensation output control during MFM mode. Specifies early, late, and normal timing signals. See the discussion in Section 5.</td>
</tr>
</tbody>
</table>
Table 2. 8272 FDC Pin Description (continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Pin Symbol</th>
<th>I/O</th>
<th>To/From</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>FLT/TRKO</td>
<td>I</td>
<td>Drive</td>
<td>Fault/Track 0. Senses the disk drive fault condition in the Read/Write mode and the Track 0 condition in the Seek mode.</td>
</tr>
<tr>
<td>34</td>
<td>WP/TS</td>
<td>I</td>
<td>Drive</td>
<td>Write Protect/Two-Sided. Senses the disk write protect status in the Read/Write mode and the dual-sided media status in the Seek mode.</td>
</tr>
<tr>
<td>35</td>
<td>RDY</td>
<td>I</td>
<td>Drive</td>
<td>Ready. Senses the disk drive ready status.</td>
</tr>
<tr>
<td>36</td>
<td>HDL</td>
<td>O</td>
<td>Drive</td>
<td>Head Load. Loads the disk drive read/write head. (The head is placed in contact with the disk.)</td>
</tr>
<tr>
<td>37</td>
<td>FR/STP</td>
<td>O</td>
<td>Drive</td>
<td>Fault Reset/Step. Resets the fault flip-flop in the disk drive when operating in the Read/Write mode. Provides head step pulses (to move the head from one cylinder to another cylinder) in the Seek mode.</td>
</tr>
<tr>
<td>38</td>
<td>LCT/DIR</td>
<td>O</td>
<td>Drive</td>
<td>Low Current/Direction. Signals that the recording head has been positioned over the inner cylinders (44-77) of the floppy disk in the Read/Write mode. (The write current must be lowered when recording on the physically shorter inner cylinders of the disk. Most drives do not track the actual head position and require that the FDC supply this signal.) Determines the head step direction in the Seek mode. In the Seek mode, a high level on this pin steps the read/write head toward the spindle (step-in); a low level steps the head away from the spindle (step-out).</td>
</tr>
<tr>
<td>39</td>
<td>RW/SEEK</td>
<td>O</td>
<td>Drive</td>
<td>Read, Write/Seek Mode Selector. A high level selects the Seek mode; a low level selects the Read/Write mode.</td>
</tr>
<tr>
<td>40</td>
<td>VCC</td>
<td></td>
<td></td>
<td>+5V DC Power.</td>
</tr>
</tbody>
</table>

* Disabled when CS is high.

Interface Registers

To support information transfer between the FDC and the system processor, the 8272 contains two 8-bit registers: the Main Status Register and the Data Register. The Main Status Register (read only) contains FDC status information and may be accessed at any time. The Main Status Register (Table 4) provides the system processor with the status of each disk drive, the status of the FDC, and the status of the processor interface. The Data Register (read/write) stores data, commands, parameters, and disk drive status information. The Data Register is used to program the FDC during the command phase and to obtain result information after completion of FDC operations. Data is read from, or written to, the FDC registers by the combination of the A0, RD, WR, and CS signals, as described in Table 3.

In addition to the Main Status Register, the FDC contains four additional status registers (ST0, ST1, ST2, and ST3). These registers are only available during the result phase of a command.

Table 3. FDC Read/Write Interface

<table>
<thead>
<tr>
<th>CS</th>
<th>A0</th>
<th>RD</th>
<th>WR</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Read Main Status Register</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Illegal</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Illegal</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Illegal</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Read from Data Register</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Write into Data Register</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Data Bus is three-stated</td>
</tr>
</tbody>
</table>
Table 4. Main Status Register Bit Definitions

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>D0B</td>
<td>Disk Drive 0 Busy. Disk Drive 0 is in the Seek mode.</td>
</tr>
<tr>
<td>1</td>
<td>D1B</td>
<td>Disk Drive 1 Busy. Disk Drive 1 is in the Seek mode.</td>
</tr>
<tr>
<td>2</td>
<td>D2B</td>
<td>Disk Drive 2 Busy. Disk Drive 2 is in the Seek mode.</td>
</tr>
<tr>
<td>3</td>
<td>D3B</td>
<td>Disk Drive 3 Busy. Disk Drive 3 is in the Seek mode.</td>
</tr>
<tr>
<td>4</td>
<td>CB</td>
<td>FDC Busy. A read or write command is in process.</td>
</tr>
<tr>
<td>5</td>
<td>NDM</td>
<td>Non-DMA Mode. The FDC is in the non-DMA mode when this bit is high. This bit is set only during the execution phase of commands in the non-DMA mode. Transition to a low level indicates that the execution phase has ended.</td>
</tr>
<tr>
<td>6</td>
<td>DIO</td>
<td>Data Input/Output. Indicates the direction of a data transfer between the FDC and the Data Register. When DIO is high, data is read from the Data Register by the processor; when DIO is low, data is written from the processor to the Data Register.</td>
</tr>
<tr>
<td>7</td>
<td>RQM</td>
<td>Request for Master. Indicates that the Data Register is ready to send data to; or receive data from, the processor.</td>
</tr>
</tbody>
</table>

Command/Result Phases

Table 5 lists the 8272 command set. For each of the fifteen commands, command and result phase data transfers are listed. A list of abbreviations used in the table is given in Table 6, and the contents of the result status registers (ST0-ST3) are illustrated in Table 7.

The bytes of data which are sent to the 8272 during the command phase, and are read out of the 8272 in the result phase, must occur in the order shown in Table 5. That is, the command code must be sent first and the other bytes sent in the prescribed sequence. All bytes of the command and result phases must be read/written as described. After the last byte of data in the command phase is sent to the 8272 the execution phase automatically starts. In a similar fashion, when the last byte of data is read from the 8272 in the result phase, the command is automatically ended and the 8272 is ready for a new command. A command may be aborted by simply raising the terminal count signal (pin 16). This is a convenient means of ensuring that the processor may always gain control of the 8272 (even if the disk system hangs up in an abnormal manner).

It is important to note that during the result phase all bytes shown in Table 5 must be read. The Read Data command, for example, has seven bytes of data in the result phase. All seven bytes must be read in order to successfully complete the Read Data command. The 8272 will not accept a new command until all seven bytes have been read. The number of command and result bytes varies from command-to-command.

In order to read data from, or write data to, the Data Register during the command and result phases, the system processor must examine the Main Status Register to determine if the Data Register is available. The D10 (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command word may be written into the 8272. Many of the commands require multiple bytes, and as a result, the Main Status Register must be read prior to each byte transfer to the 8272. To read status bytes during the result phase, D10 and RQM in the Main Status Register must both be high. Note, checking the Main Status Register in this manner before each byte transfer to/from the 8272 is required only in the command and result phases, and is NOT required during the execution phase.

Execution Phase

All data transfers to (or from) the floppy drive occur during the execution phase. The 8272 has two primary modes of operation for data transfers (selected by the specify command):

1. DMA mode
2. non-DMA mode

In the DMA mode, DRQ (DMA Request) is activated for each transfer request. The DMA controller responds to DRQ with DACK (DMA Acknowledge) and RD (for read commands) or WR (for write commands). DRQ is reset by the FDC during the transfer. INT is activated after the last data transfer, indicating the completion of the execution phase, and the beginning of the result phase. In the DMA mode, the terminal count (TC/EOP) output of the DMA controller should be connected to the 8272 TC input to properly terminate disk data transfer commands.
### Table 5. 8272 Command Set

<table>
<thead>
<tr>
<th>PHASE</th>
<th>RW</th>
<th>DATA BUS</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>MT MFM SK</td>
<td>0 0 0 1 0 0</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>0 0 0 0 0 0</td>
<td>HDS DS1 DS0</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>EOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>GPL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>DTL</td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>R</td>
<td>ST0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>ST2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

| READ DELETED DATA | | | |
| Command | W | MT MFM SK | 0 0 0 1 0 0 | Command Codes |
| | W | 0 0 0 0 0 0 | HDS DS1 DS0 | Sector ID information prior to Command execution |
| | W | C | | |
| | W | H | | |
| | W | R | | |
| | W | N | | |
| | W | EC1 | | |
| | W | GPL | | |
| | W | DTL | | |
| Execution | | | |
| Result | R | ST0 | | Data transfer between the FDD and the main-system |
| | R | ST1 | | Status information after Command execution |
| | R | ST2 | | Status information after Command execution |
| | R | C | | |
| | R | H | | |
| | R | R | | |

| WRITE DATA | | | |
| Command | W | MT MFM 0 0 0 0 1 0 | Command Codes |
| | W | 0 0 0 0 0 0 | HDS DS1 DS0 | Sector ID information prior to Command execution |
| | W | C | | |
| | W | H | | |
| | W | R | | |
| | W | N | | |
| | W | EOT | | |
| | W | GPL | | |
| | W | DTL | | |
| Execution | | | |
| Result | R | ST0 | | Data transfer between the main-system and the FDD |
| | R | ST1 | | Status information after Command execution |
| | R | ST2 | | Status information after Command execution |
| | R | C | | |
| | R | H | | |
| | R | R | | |

| WRITE DELETED DATA | | | |
| Command | W | MT MFM 0 0 0 0 1 0 | Command Codes |
| | W | 0 0 0 0 0 0 | HDS DS1 DS0 | Sector ID information prior to Command execution |
| | W | C | | |
| | W | H | | |
| | W | R | | |
| | W | N | | |
| | W | EOT | | |
| | W | GPL | | |
| | W | DTL | | |
| Execution | | | |
| Result | R | ST0 | | Data transfer between the FDD and the main-system |
| | R | ST1 | | Status information after Command execution |
| | R | ST2 | | Status information after Command execution |
| | R | C | | |
| | R | H | | |
| | R | R | | |

| READ A TRACK | | | |
| Command | W | 0 MFM 0 0 0 0 1 0 | Command Codes |
| | W | 0 0 0 0 0 0 | HDS DS1 DS0 | Sector ID information prior to Command execution |
| | W | C | | |
| | W | H | | |
| | W | R | | |
| | W | N | | |
| | W | EOT | | |
| | W | GPL | | |
| | W | DTL | | |
| Execution | | | |
| Result | R | ST0 | | Data transfer between the FDD and the main-system |
| | R | ST1 | | Status information after Command execution |
| | R | ST2 | | Status information after Command execution |
| | R | C | | |
| | R | H | | |
| | R | R | | |

| FORMAT A TRACK | | | |
| Command | W | 0 MFM 0 0 0 0 1 0 | Command Codes |
| | W | 0 0 0 0 0 0 | HDS DS1 DS0 | Sector ID information during Execution phase |
| | W | C | | |
| | W | H | | |
| | W | R | | |
| | W | N | | |
| | W | EOT | | |
| | W | GPL | | |
| | W | D | | |
| Execution | | | |
| Result | R | ST0 | | Status information after Command execution |
| | R | ST1 | | Status information after Command execution |
| | R | ST2 | | Status information after Command execution |
| | R | C | | |
| | R | H | | |
| | R | R | | |

| SCAN EQUAL | | | |
| Command | W | MT MFM SK 1 0 0 0 1 | Command Codes |
| | W | 0 0 0 0 0 0 | HDS DS1 DS0 | Sector ID information prior to Command execution |
| | W | C | | |
| | W | H | | |
| | W | R | | |
| | W | N | | |
| | W | EOT | | |
| | W | GPL | | |
| | W | STP | | |
| Execution | | | |
| Result | R | ST0 | | Status information after Command execution |
| | R | ST1 | | Status information after Command execution |
| | R | ST2 | | Status information after Command execution |
| | R | C | | |
| | R | H | | |
| | R | R | | |

Note 1: $A_0 = 1$ for all operations.
### Table 5. Command Set (Continued)

<table>
<thead>
<tr>
<th>PHASE</th>
<th>RW</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAN LOW OR EQUAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Command Codes</td>
</tr>
<tr>
<td>Command</td>
<td>W</td>
<td>MT</td>
<td>MFM</td>
<td>SK</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Execution</td>
<td>W</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>HDS</td>
<td>DSI</td>
<td>DSO</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Output</td>
<td>W</td>
<td>C</td>
<td>H</td>
<td>R</td>
<td>N</td>
<td>EOT</td>
<td>GPL</td>
<td>STP</td>
<td></td>
<td>Command Codes</td>
</tr>
<tr>
<td>Result</td>
<td>R</td>
<td>ST</td>
<td>0</td>
<td>ST</td>
<td>1</td>
<td>ST</td>
<td>2</td>
<td>C</td>
<td>H</td>
<td>N</td>
</tr>
</tbody>
</table>

| SCAN HIGH OR EQUAL              |    |    |    |    |    |    |    |    |    | Command Codes |
| Command | W | MT | MFM | SK | 1 | 1 | 0 | 1 | 0 | Command Codes |
| Execution | W | 0 | 0 | 0 | 0 | 0 | HDS | DSI | DSO | Command Codes |
| Output | W | C | H | R | N | EOT | GPL | STP |               | Command Codes |
| Result | R | ST | 0 | ST | 1 | ST | 2 | C | H | N | Command Codes |

### Table 6. Command/Result Parameter Abbreviations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Cylinder Address. The currently selected cylinder address (0 to 76) on the disk.</td>
</tr>
<tr>
<td>D</td>
<td>Data Pattern. The pattern to be written in each sector data field during formatting.</td>
</tr>
<tr>
<td>DS0,DS1</td>
<td>Disk Drive Select.</td>
</tr>
<tr>
<td>DTL</td>
<td>Special Sector Size. During the execution of disk read/write commands, this parameter is used to temporarily alter the effective disk sector size. By setting N to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the diskette) is larger than DTL specifies, the remainder of the actual sector is not passed to the system during read commands; during write commands, the remainder of the actual sector is written with all-zeroes bytes. DTL should be set to FF hexadecimal when N is not zero.</td>
</tr>
<tr>
<td>EOT</td>
<td>End of Track. The final sector number of the current track.</td>
</tr>
<tr>
<td>GPL</td>
<td>Gap Length. The gap 3 size. (Gap 3 is the space between sectors excluding the VCO synchronization field as defined in section 3.)</td>
</tr>
<tr>
<td>H</td>
<td>Head Address. Selected head: 0 or 1 (disk side 0 or 1, respectively) as encoded in the sector ID field.</td>
</tr>
<tr>
<td>HLT</td>
<td>Head Load Time. Defines the time interval that the FDC waits before initiating a read or write operation. Programmable from 2 to 254 milliseconds (in increments of 2 ms).</td>
</tr>
<tr>
<td>HUT</td>
<td>Head Unload Time. Defines the time interval from the end of the execution phase (of a read or write command) until the head is unloaded. Programmable from 16 to 240 milliseconds (in increments of 16 ms).</td>
</tr>
<tr>
<td>MFM</td>
<td>MFM/FM Mode Selector. Selects MFM double-density recording mode when high, FM single-density mode when low.</td>
</tr>
</tbody>
</table>
Table 6. Command/Result Parameter Abbreviations (continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT</td>
<td>Multi-Track Selector. When set, this flag selects the multi-track operating mode. In this mode (used only with dual-sided disks), the FDC treats a complete cylinder (under both read/write head 0 and read/write head 1) as a single track. The FDC operates as if this expanded track started at the first sector under head 0 and ended at the last sector under head 1. With this flag set (high), a multi-sector read operation will automatically continue to the first sector under head 1 when the FDC finishes operating on the last sector under head 0.</td>
<td>SK</td>
<td>Skip Flag. When this flag is set, sectors containing deleted data address marks will automatically be skipped during the execution of multi-sector Read Data or Scan commands. In the same manner, a sector containing a data address mark will automatically be skipped during the execution of a multi-sector Read Deleted Data command.</td>
</tr>
<tr>
<td>N</td>
<td>Sector Size. The number of data bytes within a sector. (See Table 9.)</td>
<td>SRT</td>
<td>Step Rate Interval. Defines the time interval between step pulses issued by the FDC (track-to-track access time). Programmable from 1 to 16 milliseconds (in increments of 1 ms).</td>
</tr>
<tr>
<td>ND</td>
<td>Non-DMA Mode Flag. When set (high), this flag indicates that the FDC is to operate in the non-DMA mode. In this mode, the processor is interrupted for each data transfer. When low, the FDC interfaces to a DMA controller by means of the DRQ and DACK signals.</td>
<td>ST0</td>
<td>Status Register 0-3. Registers within the FDC that store status information after a command has been executed. This status information is available to the processor during the Result Phase after command execution. These registers may only be read after a command has been executed (in the exact order shown in Table 5 for each command). These registers should not be confused with the Main Status Register.</td>
</tr>
<tr>
<td>R</td>
<td>Sector Address. Specifies the sector number to be read or written. In multi-sector transfers, this parameter specifies the sector number of the first sector to be read or written.</td>
<td>ST1</td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>Number of Sectors per Track. Specifies the number of sectors per track to be initialized by the Format Track command.</td>
<td>ST2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ST3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STP</td>
<td>Scan Sector Increment. During Scan operations, this parameter is added to the current sector number in order to determine the next sector to be scanned.</td>
</tr>
</tbody>
</table>

Table 7. Status Register Definitions

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Register 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7,6</td>
<td>IC</td>
<td>Interrupt Code. 00 — Normal termination of command. The specified command was properly executed and completed without error. 01 — Abnormal termination of command. Command execution was started but could not be successfully completed. 10 — Invalid command. The requested command could not be executed. 11 — Abnormal termination. During command execution, the disk drive ready signal changed state.</td>
</tr>
<tr>
<td>5</td>
<td>SE</td>
<td>Seek End. This flag is set (high) when the FDC has completed the Seek command and the head address is positioned over the correct cylinder.</td>
</tr>
<tr>
<td>4</td>
<td>EC</td>
<td>Equipment Check Error. This flag is set (high) if a fault signal is received from the disk drive or if the track 0 signal fails to become active after 77 step pulses (Recalibrate command).</td>
</tr>
<tr>
<td>3</td>
<td>NR</td>
<td>Not Ready Error. This flag is set if a read or write command is issued and either the drive is not ready or the command specifies side 1 (head 1) of a single-sided disk.</td>
</tr>
<tr>
<td>2</td>
<td>H</td>
<td>Head Address. The head address at the time of the interrupt.</td>
</tr>
<tr>
<td>1,0</td>
<td>DS1,DS0</td>
<td>Drive Select. The number of the drive selected at the time of the interrupt.</td>
</tr>
</tbody>
</table>
## APPLICATIONS

### Table 7. Status Register Definitions (continued)

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status Register 1</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>EN</td>
<td>End of Track Error. This flag is set if the FDC attempts to access a sector beyond the final sector of the track.</td>
</tr>
<tr>
<td>6</td>
<td>DE</td>
<td>Data Error. Set when the FDC detects a CRC error in either the ID field or the data field of a sector.</td>
</tr>
<tr>
<td>5</td>
<td>OR</td>
<td>Overrun Error. Set (during data transfers) if the FDC does not receive DMA or processor service within the specified time interval.</td>
</tr>
<tr>
<td>3</td>
<td>ND</td>
<td>Sector Not Found Error. This flag is set by any of the following conditions: a) The FDC cannot locate the sector specified in the Read Data, Read Deleted Data, or Scan command. b) The FDC cannot locate the starting sector specified in the Read Track command. c) The FDC cannot read the ID field without error during a Read ID command.</td>
</tr>
<tr>
<td>1</td>
<td>NW</td>
<td>Write Protect Error. This flag is set if the FDC detects a write protect signal from the disk drive during the execution of a Write Data, Write Deleted Data, or Format Track command.</td>
</tr>
<tr>
<td>0</td>
<td>MA</td>
<td>Missing Address Mark Error. This flag is set by either of the following conditions: a) The FDC cannot detect the ID address mark on the specified track (after two occurrences of the physical index mark). b) The FDC cannot detect the data address mark or deleted data address mark on the specified track. (See also the MD bit of Status Register 2.)</td>
</tr>
<tr>
<td><strong>Status Register 2</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CM</td>
<td>Not used. This bit is always low.</td>
</tr>
<tr>
<td>6</td>
<td>DD</td>
<td>Control Mark. This flag is set when the FDC encounters one of the following conditions: a) A deleted data address mark during the execution of a Read Data or Scan command. b) A data address mark during the execution of a Read Deleted Data command.</td>
</tr>
<tr>
<td>5</td>
<td>WC</td>
<td>Data Error. Set (high) when the FDC detects a CRC error in a sector data field. This flag is not set when a CRC error is detected in the ID field.</td>
</tr>
<tr>
<td>4</td>
<td>SH</td>
<td>Cylinder Address Error. Set when the cylinder address from the disk sector ID field is different from the current cylinder address maintained within the FDC.</td>
</tr>
<tr>
<td>3</td>
<td>SN</td>
<td>Scan Hit. Set during the execution of the Scan command if the scan condition is satisfied.</td>
</tr>
<tr>
<td>2</td>
<td>BC</td>
<td>Scan Not Satisfied. Set during execution of the Scan command if the FDC cannot locate a sector on the specified cylinder that satisfies the scan condition.</td>
</tr>
<tr>
<td>1</td>
<td>MD</td>
<td>Bad Track Error. Set when the cylinder address from the disk sector ID field is FF hexadecimal and this cylinder address is different from the current cylinder address maintained within the FDC. This all “ones” cylinder number indicates a bad track (one containing hard errors) according to the IBM soft-sectored format specifications.</td>
</tr>
<tr>
<td>0</td>
<td>Missing Data Address Mark Error. Set if the FDC cannot detect a data address mark or deleted data address mark on the specified track.</td>
<td></td>
</tr>
</tbody>
</table>
APPLICATIONS

Table 7. Status Register Definitions (continued)

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>FT</td>
<td>Fault. This flag indicates the status of the fault signal from the selected disk drive.</td>
</tr>
<tr>
<td>6</td>
<td>WP</td>
<td>Write Protected. This flag indicates the status of the write protect signal from the selected disk drive.</td>
</tr>
<tr>
<td>5</td>
<td>RDY</td>
<td>Ready. This flag indicates the status of the ready signal from the selected disk drive.</td>
</tr>
<tr>
<td>4</td>
<td>TO</td>
<td>Track 0. This flag indicates the status of the track 0 signal from the selected disk drive.</td>
</tr>
<tr>
<td>3</td>
<td>TS</td>
<td>Two-Sided. This flag indicates the status of the two-sided signal from the selected disk drive.</td>
</tr>
<tr>
<td>2</td>
<td>H</td>
<td>Head Address. This flag indicates the status of the side select signal for the currently selected disk drive.</td>
</tr>
<tr>
<td>1,0</td>
<td>DS1,DS0</td>
<td>Drive Select. Indicates the currently selected disk drive number.</td>
</tr>
</tbody>
</table>

In the non-DMA mode, transfer requests are indicated by activation of both the INT output signal and the RQM flag (bit 7) in the Main Status Register. INT can be used for interrupt-driven systems and RQM can be used for polled systems. The system processor must respond to the transfer request by reading data from (activating RD), or writing data to (activating WR), the FDC. This response removes the transfer request (INT and RQM are set inactive). After completing the last transfer, the 8272 activates the INT output to indicate the beginning of the result phase. In the non-DMA mode, the processor must activate the TC signal to the FDC (normally by means of an I/O port) after the transfer request for the last data byte has been received (by the processor) and before the appropriate data byte has been read from (or written to) the FDC.

In either mode of operation (DMA or non-DMA), the execution phase ends when a terminal count signal is sensed or when the last sector on a track (the EOT parameter—Table 5) has been read or written. In addition, if the disk drive is in a "not ready" state at the beginning of the execution phase, the "not ready" flag (bit 3 in Status Register 0) is set (high) and the command is terminated.

If a fault signal is received from the disk drive at the end of a write operation (Write Data, Write Deleted Data, or Format), the FDC sets the "equipment check" flag (bit 4 in Status Register 0), and terminates the command after setting the interrupt code (bits 7 and 6 of Status Register 0) to "01" (bit 7 low, bit 6 high).

**Multi-sector and Multi-track Transfers**

During disk read/write transfers (Read Data, Write Data, Read Deleted Data, and Write Deleted Data), the FDC will continue to transfer data from sequential sectors until the TC input is sensed. In the DMA mode, the TC input is normally connected to the TC/EOP (terminal count) output of the DMA controller. In the non-DMA mode, the processor directly controls the FDC TC input as previously described. Once the TC input is received, the FDC stops requesting data transfers (from the system processor or DMA controller). The FDC, however, continues to read data from, or write data to, the floppy disk until the end of the current disk sector. During a disk read operation, the data read from the disk (after reception of the TC input) is discarded, but the data CRC is checked for errors; during a disk write operation, the remainder of the sector is filled with all-zero bytes.

If the TC signal is not received before the last byte of the current sector has been transferred to/from the system, the FDC increments the sector number by one and initiates a read or write command for this new disk sector.

The FDC is also designed to operate in a multi-track mode for dual-sided disks. In the multi-track mode (specified by means of the MT flag in the command byte—Table 5) the FDC will automatically increment the head address (from 0 to 1) when the last sector (on the track under head 0) has been read or written. Reading or writing is then continued on the first sector (sector 1) of head 1.

**Drive Status Polling**

After the power-on reset, the 8272 automatically enters a drive status polling mode. If a change in drive status is detected (all drives are assumed to be "not ready" at power-on), an interrupt is generated. The 8272 continues this status polling between command executions (and between step pulses in the Seek command). In this manner, the 8272 automatically notifies the system processor when a floppy disk is inserted, removed, or changed by the operator.
Command Details

During the command phase, the Main Status Register must be polled by the CPU before each byte is written into the Data Register. The D10 (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command may be written into the 8272. The beginning of the execution phase for any of these commands will cause D10 to be set high and RQM to be set low.

The following paragraphs describe the fifteen FDC commands in detail.

Specify

The Specify command is used prior to performing any disk operations (including the formatting of a new disk) to define drive/FDC operating characteristics. The Specify command parameters set the values for three internal timers:

1. Head Load Time (HLT) — This seven-bit value defines the time interval that the FDC waits after loading the head before initiating a read or write operation. This timer is programmable from 2 to 254 milliseconds in increments of 2 ms.

2. Head Unload Time (HUT) — This four-bit value defines the time from the end of the execution phase (of a read or write command) until the head is unloaded. This timer is programmable from 16 to 240 milliseconds in increments of 16 ms. If the processor issues another command before the head unloads, the head will remain loaded and the head load wait will be eliminated.

3. Step Rate Time (SRT) — This four-bit value defines the time interval between step pulses issued by the FDC (track-to-track access time). This timer is programmable from 1 to 16 milliseconds in increments of 1 ms.

The time intervals mentioned above are a direct function of the FDC clock (CLK on pin 19). Times indicated above are for an 8 MHz clock.

The Specify command also indicates the choice of DMA or non-DMA operation (by means of the ND bit). When this bit is high the non-DMA mode is selected; when ND is low, the DMA mode is selected.

Sense Drive Status

This command may be used by the processor whenever it wishes to obtain the status of the disk drives. Status Register 3 (returned during the result phase) contains the drive status information as described in Table 7.

Sense Interrupt Status

An interrupt signal is generated by the FDC when one or more of the following events occurs:

1. The FDC enters the result phase for:
   a. Read Data command
   b. Read Track command
   c. Read ID command
   d. Read Deleted Data command
   e. Write Data command
   f. Format Track command
   g. Write Deleted Data command
   h. Scan commands

2. The ready signal from one of the disk drives changes state.

3. A Seek or Recalibrate command completes operation.

4. The FDC requires a data transfer during the execution phase of a command in the non-DMA mode.

Interrupts caused by reasons (1) and (4) above occur during normal command operations and are easily discernible by the processor. However, interrupts caused by reasons (2) and (3) above are uniquely identified with the aid of the Sense Interrupt Status command. This command, when issued, resets the interrupt signal and by means of bits 5, 6, and 7 of Status Register 0 (returned during the result phase) identifies the cause of the interrupt (see Table 8).

<table>
<thead>
<tr>
<th>Table 8. Interrupt Codes</th>
<th>Seek End Bit 5</th>
<th>Interrupt Code Bit 6</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>Ready Line changed state, either polarity</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>Normal Termination of Seek or Recalibrate Command</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>Abnormal Termination of Seek or Recalibrate Command</td>
</tr>
</tbody>
</table>

Neither the Seek nor the Recalibrate command has a result phase. Therefore, it is mandatory to use the Sense Interrupt Status Command after these commands to effectively terminate them and to provide verification of the disk head position.
When an interrupt is received by the processor, the FDC busy flag (bit 4) and the non-DMA flag (bit 5) may be used to distinguish the above interrupt causes:

<table>
<thead>
<tr>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Asynchronous event-(2) or (3) above</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Result phase-(1) above</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Data transfer required-(4) above</td>
</tr>
</tbody>
</table>

A single interrupt request to the processor may, in fact, be caused by more than one of the above events. The processor should continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are serviced.

Seek

The Seek command causes the drive's read/write head to be positioned over the specified cylinder. The FDC determines the difference between the current cylinder address and the desired (specified) address, and issues the appropriate number of step pulses. If the desired cylinder address is larger than the current address, the direction signal (LCT/DIR, pin 38) is set high (step-in); the direction signal is set low (step-out) if the desired cylinder address is less than the current address. No head movement occurs (no step pulses are issued) if the desired cylinder is the same as the current cylinder.

The rate at which step pulses are issued is controlled by the step rate time (SRT) in the Specify command. After each step pulse is issued, the desired cylinder address is compared against the current cylinder address. When the cylinder addresses are equal, the "seek end" flag (bit 5 in Status Register 0) is set (high) and the command is terminated. If the disk drive becomes "not ready" during the seek operation, the "not ready" flag (in Status Register 0) is set (high) and the command is terminated.

During the command phase of the Seek operation the FDC is in the FDC busy state, but during the execution phase it is in the non-busy state. While the FDC is in the non-busy state, another Seek command may be issued. In this manner parallel seek operations may be in operation on up to four floppy disk drives at once. The Main Status Register contains a flag for each drive (Table 4) that indicates whether the associated drive is currently operating in the seek mode. When a drive has completed a seek operation, the FDC generates an interrupt. In response to this interrupt, the system software must issue a Sense Interrupt Status command. During the result phase of this command, Status Register 0 (containing the drive number in bits 0 and 1) is read by the processor.

Recalibrate

This command causes the read/write head of the disk drive to retract to the track 0 position. The FDC clears the contents of its internal cylinder counter, and checks the status of the track 0 signal from the disk drive. As long as the track 0 signal is low, the direction signal remains high and step pulses are issued. When the track 0 signal goes high, the seek end flag (in Status Register 0) is set (high) and the command is terminated. If the track 0 signal is still low after 77 step pulses have been issued, the seek end and equipment check flags (in Status Register 0) are both set and the Recalibrate command is terminated.

Recalibrate commands for multiple drives can be overlapped in the same manner that Seek commands are overlapped.

Format Track

The Format Track command formats or "initializes" a track on a floppy disk by writing the ID field, gaps, and address marks for each sector. Before issuing the Format command, the Seek command must be used to position the read/write head over the correct cylinder. In addition, a table of ID field values (cylinder, head, and sector addresses and sector length code) must be prepared before the command is executed. During command execution, the FDC accesses the table and, using the values supplied, writes each sector on the track. The ID field address mark originates from the FDC and is written automatically as the first byte of each sector's ID field. The cylinder, head, and sector addresses are taken, in order, from the table. The ID field CRC character (derived from the data written in the first five bytes) is written as the last two bytes of the ID field. Gaps are written automatically by the FDC, with the length of the variable gap determined by one of the Format command parameters.

The data field address mark is generated by the FDC and is written automatically as the first byte of the data field. The data pattern specified in the command phase is written into each data byte of each sector. A CRC character is derived from the data written in the sector's data field. The two CRC bytes are appended to the last data byte.

The formatting of a track begins at the physical index mark. As previously mentioned, the order of sector assignment is taken directly from the formatting table. Four entries are required for each sector: a cylinder address, a head address, a sector address, and a sector length code. The cylinder address in the ID field should be equal to the cylinder address of the track currently being formatted.
The sector addresses must be unique (no two equal). The order of the sector entries in the table is the sequence in which sector numbers appear on the track when it is formatted. The number of entry sets (cylinder, head, and sector address and sector length code) must equal the number of sectors allocated to the track (specified in the command phase).

Since the sector address is supplied, in order, for each sector, tracks can be formatted sequentially (the first sector following the index mark is assigned sector address 1, the adjacent sector is assigned sector address 2, and so on) or sector numbers can be interleaved (see section 3) on a track.

Table 9 lists recommended gap sizes and sectors/track for various sector sizes.

**Read Data**

Nine (9) bytes are required to complete the command phase specification for the Read Data command. During the execution phase, the FDC loads the head (if it is in the unloaded state), waits the specified head load time (defined in the Specify command), and begins reading ID address marks and ID fields. When the requested sector address compares with the sector address read from the disk, the FDC outputs data (from the data field) byte-by-byte to the system. The Read Data command automatically operates in the multi-sector mode described earlier. In addition, multi-track operation may be specified by means of the MT command flag (Table 5). The amount of data that can be transferred with a single command to the FDC depends on the multi-track flag, the recording density flag, and the number of bytes per sector.

During the execution of read and write commands, the special sector size parameter (DTL) is used to temporarily alter the effective disk sector size. By setting the sector size code (N) to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the disk) is larger than DTL specifies, only the number of bytes specified by the DTL parameter are passed to the system; the remainder of the actual disk sector is not transferred (although the data is checked for CRC errors). Multi-sector read operations are performed in the same manner as they are when the sector size code is non-zero. (The N and DTL parameters are always present in the command sequence. DTL should be set to FF hexadecimal when N is not zero.)

If the FDC detects the physical index mark twice without finding the requested sector, the FDC sets the "sector not found error" flag (bit 2 in Status Register 1) and terminates the Read Data command. The interrupt code (bits 7 and 6 of Status Register 0) is set to "01." Note that the FDC searches for each sector in a multi-sector operation. Therefore, a "sector not found" error may occur after successful transfer of one or more preceding sectors. This error could occur if a particular sector number was not included when the track was first formatted or if a hard error on the disk has invalidated a sector ID field.

After reading the ID field and data field in each sector, the FDC checks the CRC bytes. If a read error is detected (incorrect CRC in the ID field), the FDC sets the "data error" flag in Status Register 1; if a CRC error occurs in the data field, the FDC sets the "data error" flag in Status Register 2. In either error condition, the FDC terminates the Read Data command. The interrupt code (bits 7 and 6 in Status Register 0) is set to "01."

If the FDC reads a deleted data address mark from the disk, and the skip flag (specified during the command phase) is not set, the FDC sets the "control mark" flag (bit 6 in Status Register 2) and terminates the Read Data command (after reading all the data in the sector). If the skip flag is set, the FDC skips the sector with the deleted data address mark and reads the next sector. Thus, the skip flag may be used to cause the FDC to ignore deleted data sectors during a multi-sector read operation.

During disk data transfers between the FDC and the system, the FDC must be serviced by the system (processor or DMA controller) every 27 μs in the FM mode, and every 13 μs in the MFM mode. If the FDC is not

---

**Table 9. Sector Size Relationships**

<table>
<thead>
<tr>
<th>Format</th>
<th>Sector Size</th>
<th>N Sector Size Code</th>
<th>SC Sectors/Track</th>
<th>GPL¹ Gap 3 Length</th>
<th>GPL² Gap 3 Length</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM Mode</td>
<td>128 bytes/Sector</td>
<td>00</td>
<td>1A(16)</td>
<td>07(16)</td>
<td>1B(16)</td>
<td>IBM Diskette 1</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>01</td>
<td>0F(16)</td>
<td>0E(16)</td>
<td>2A(16)</td>
<td>IBM Diskette 2</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>02</td>
<td>08</td>
<td>1B(16)</td>
<td>3A(16)</td>
<td></td>
</tr>
<tr>
<td>MFM Mode</td>
<td>256</td>
<td>01</td>
<td>1A(16)</td>
<td>0E(16)</td>
<td>36(16)</td>
<td>IBM Diskette 2D</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>02</td>
<td>0F(16)</td>
<td>1B(16)</td>
<td>54(16)</td>
<td>IBM Diskette 2D</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>03</td>
<td>08</td>
<td>35(16)</td>
<td>74(16)</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
1. Suggested values of GPL in Read or Write commands to avoid splice point between data field and ID field of contiguous sectors.
2. Suggested values of GPL in Format command.
APPLICATIONS

serviced within this interval, the “overrun error” flag (bit 4 in Status Register 1) is set and the Read Data command is terminated.

If the processor terminates a read (or write) operation in the FDC, the ID information in the result phase is dependent upon the state of the multi-track flag and end of track byte. Table 11 shows the values for C, H, R, and N, when the processor terminates the command.

Write Data

Nine (9) bytes are required to complete the command phase specification for the Write Data command. During the execution phase the FDC loads the head (if it is in the unloaded state), waits the specified head load time (defined by the Specify command), and begins reading sector ID fields. When the requested sector address compares with the sector address read from the disk, the FDC reads data from the processor one byte at a time via the data bus and outputs the data to the data field of that sector. The CRC is computed on this data and two CRC bytes are written at the end of the data field.

The FDC reads the ID field of each sector and checks the CRC bytes. If the FDC detects a read error (incorrect CRC in one of the ID fields, it sets the “data error” flag (bit 5 in Status Register 1) and terminates the Write Data command. The interrupt code (bits 7 and 6 in Status Register 0) is set to “01.”

The Write Data command operates in much the same manner as the Read Data command. The following items are the same; refer to the Read Data command for details:

- Multi-sector and Multi-track operation
- Data transfer capacity
- “End of track error” flag
- “Sector not found error” flag
- “Data error” flag
- Head unload time interval
- ID information when the processor terminates the command (see Table 11)
- Definition of DTL when N = 0 and when N ≠ 0

During the Write Data execution phase, data transfers between the processor and FDC must occur every 31 μs in the FM mode, and every 15 μs in the MFM mode. If the time interval between data transfers is longer than this, the FDC sets the “overrun error” flag (bit 4 in Status Register 1) and terminates the Write Data command.

Read Deleted Data

This command operates in almost the same manner as the Read Data command operates. The only difference involves the treatment of the data address mark and the skip flag. When the FDC detects a data address mark at the beginning of a data field (and the skip flag is not set), the FDC reads all the data in the sector, sets the “control mark” flag (bit 6 in Status Register 2), and terminates the command. If the skip flag is set, the FDC skips the sector with the data address mark and continues reading at the next sector. Thus, the skip flag may be used to cause the FDC to read only deleted data sectors during a multi-sector read operation.

Write Deleted Data

This command operates in the same manner as the Write Data command operates except that a deleted data address mark is written at the beginning of the data field instead of the normal data address mark. This command is used to mark a bad sector (containing a hard error) on the floppy disk.

Read Track

The Read Track command is similar to the Read Data command except that the entire data field is read continuously from each of the sectors of a track. Immediately after encountering the physical index mark, the FDC starts reading all data fields on the track as continuous blocks of data. If the FDC finds an error in the ID field or data field CRC check bytes, it continues to read data from the track. The FDC compares the ID information read from each sector with the values specified during the command phase. If the specified ID field information is not found on the track, the “sector not found error” flag (in Status Register 1) is set. Multi-track and skip operations are not allowed with this command.

This command terminates when the last sector on the track has been read. (The number of sectors on the track is specified by the end of track parameter byte during the command phase.) If the FDC does not find an ID address mark on the disk after it encounters the physical index mark for the second time, it sets the “missing address mark error” flag (bit 0 in Status Register 1) and terminates the command. The interrupt code (bits 7 and 6 of Status Register 0) is set to “01.”

Read ID

The Read ID command transfers (reads) the first correct ID field from the current disk track (following the physical index mark) to the processor. If no correct ID address mark is found on the track, the “missing address mark error” flag is set (bit 0 in Status Register 1). If no data mark is found on the track, the “sector not found error” flag is also set (bit 2 in Status Register 1). Either error condition causes the command to be terminated.
Scan Commands

The Scan commands allow the data being read from the disk to be compared against data supplied by the system (by the processor in non-DMA mode, and by the DMA controller in DMA mode). The FDC compares the data on a byte-by-byte basis, and searches for a sector of data that meets the conditions of “disk data equal to system data”, “disk data less than or equal to system data”, or “disk data greater than or equal to system data”. Simple binary (ones complement) arithmetic is used for comparison (FF = largest number, 00 = smallest number). If, after a complete sector of data is compared, the conditions are not met, the sector number is incremented by the scan sector increment (specified in the command phase), and the scan operation is continued. The scan operation continues until one of the following conditions occurs; the conditions for scan are met (equal, low, or high), the last sector on the track is reached, or the terminal count signal is received.

If the conditions for scan are met, the FDC sets the “scan hit” flag (bit 3 in Status Register 2) and terminates the Scan command. If the conditions for scan are not met between the starting sector and the last sector on the track (specified in the command phase), the FDC sets the “scan not satisfied” flag (bit 2 in Status Register 2) and terminates the Scan command. The receipt of a terminal count signal from the processor or DMA controller during the scan operation will cause the FDC to complete the comparison of the particular byte which is in process, and to terminate the command.

Table 10 shows the status of the “scan hit” and “scan

Table 10. Scan Status Codes

<table>
<thead>
<tr>
<th>Command</th>
<th>Status Register 2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 2 = SN Bit 3 = SH</td>
<td>( D_{\text{FD}} = D_{\text{Processor}} )</td>
<td></td>
</tr>
<tr>
<td>Scan Equal</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Scan Low or Equal</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Scan High or Equal</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 11. ID Information When Processor Terminates Command

<table>
<thead>
<tr>
<th>MT</th>
<th>EOT</th>
<th>Final Sector Transferred to Processor</th>
<th>ID Information at Result Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>1A</td>
<td>Sector 1 to 25 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>0F</td>
<td>Sector 1 to 14 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>Sector 1 to 7 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>1A</td>
<td>Sector 26 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>0F</td>
<td>Sector 15 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>Sector 8 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>1A</td>
<td>Sector 1 to 25 at Side 1</td>
<td>C+1</td>
</tr>
<tr>
<td></td>
<td>0F</td>
<td>Sector 1 to 14 at Side 1</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>Sector 1 to 7 at Side 1</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>1A</td>
<td>Sector 26 at Side 1</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>0F</td>
<td>Sector 15 at Side 1</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>Sector 8 at Side 1</td>
<td>NC</td>
</tr>
<tr>
<td>1</td>
<td>1A</td>
<td>Sector 1 to 25 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>0F</td>
<td>Sector 1 to 14 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>Sector 1 to 7 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>1A</td>
<td>Sector 26 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>0F</td>
<td>Sector 15 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>Sector 8 at Side 0</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>1A</td>
<td>Sector 26 at Side 1</td>
<td>C+1</td>
</tr>
<tr>
<td></td>
<td>0F</td>
<td>Sector 15 at Side 1</td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>Sector 8 at Side 1</td>
<td>NC</td>
</tr>
</tbody>
</table>

Notes: 1. NC (No Change): The same value as the one at the beginning of command execution.
2. LSB (Least Significant Bit): The least significant bit of H is complemented.
not satisfied” flags under various scan termination conditions.

If the FDC encounters a deleted data address mark in one of the sectors and the skip flag is low, it regards the sector as the last sector on the cylinder, sets the “control mark” flag (bit 6 in Status Register 2) and terminates the command. If the skip flag is high, the FDC skips the sector with the deleted address mark, and reads the next sector. In this case, the FDC also sets the “control mark” flag (bit 6 in Status Register 2) in order to show that a deleted sector had been encountered.

NOTE: During scan command execution, the last sector on the track must be read for the command to terminate properly. For example, if the scan sector increment is set to 2, the end of track parameter is set to 26, and the scan begins at sector 21, sectors 21, 23, and 25 will be scanned. The next sector, 27 will not be found on the track and an abnormal command termination will occur. The command would be completed in a normal manner if either a) the scan had started at sector 20 or b) the end of track parameter had been set to 25.

During the Scan command, data is supplied by the processor or DMA controller for comparison against the data read from the disk. In order to avoid having the “overflow error” flag set (bit 4 in Status Register 1), it is necessary to have the data available in less than 27 \( \mu s \) (FM Mode) or 13 \( \mu s \) (MFM Mode). If an overrun error occurs, the FDC terminates the command.

**Invalid Commands**

If an invalid (undefined) command is sent to the FDC, the FDC will terminate the command. No interrupt is generated by the 8272 during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both set indicating to the processor that the 8272 is in the result phase and the contents of Status Register 0 must be read. When the processor reads Status Register 0 it will find an 80H code indicating that an invalid command was received.

A Sense Interrupt Status command must be sent after a Seek or Recalibrate interrupt; otherwise the FDC will consider the next command to be an invalid command. Also, when the last “hidden” interrupt has been serviced, further Sense Interrupt Status commands will result in invalid command codes.

In some applications the user may wish to use this command as a No-Op command to place the FDC in a stand-by or no operation state.

5. **THE DATA SEPARATOR**

As briefly discussed in section 2, LSI disk controllers such as the 8272 require external circuitry to generate a data window signal. This signal is used within the FDC to isolate the data bits contained within the READ DATA input signal from the disk drive. (The disk READ DATA signal is a composite signal constructed from both clock and data information.) After isolating the data bits from this input signal, the FDC assembles the data bits into 8-bit bytes for transfer to the system processor or memory.

**Single Density**

In single-density (FM) recording (Figure 3), the bit cell is 4 microseconds wide. Each bit cell contains a clock bit at the leading edge of the cell. The data bit (if present) is always located at the center of the cell. The job of data separation is relatively straightforward for single-density; simply generate a data window 2 \( \mu s \) wide starting 1 \( \mu s \) after each clock bit. Since every cell has a clock bit, a fixed window reference is available for every data bit, and because the window is 2 \( \mu s \) wide, a slightly shifted data bit will still remain within the data window.

A single-density data separator with these specifications may be easily generated using a digital or analog one-shot triggered by the clock bit.

**Double-Density**

Double-density (MFM) bit cells are reduced to 2 \( \mu s \) (in order to double the disk data storage capacity). Clock bits are inserted into the data stream only if data bits are not present in both the current and preceding bit cells (Figure 3). The data bit (if present) still occurs at the center of the bit cell and the clock bit (if present) still occurs at the leading edge of the bit cell.

MFM data separation has two problems. First, only some bit cells contain a clock bit. In this manner, MFM encoding loses the fixed bit cell reference pulse present in FM encoding. Second, the bit cell for MFM is one-half the size of the bit cell for FM. This shorter bit cell means that MFM cannot tolerate as large a playback data-shift (as FM can tolerate) without errors.

Since most playback data-shift is predictable, the FDC can precompensate the write data stream so that data/clock pulses will be correctly positioned for subsequent playback. This function is completely controlled by the FDC and is only required for MFM recording. During write operations, the FDC specifies an early, normal, or late bit positioning. This timing information is specified with respect to the FDC write clock. Early and late timing is typically 125 ns to 250 ns before or after the write clock transition (depending on disk drive requirements).
The data separator circuitry for double-density recording must continuously analyze the total READ DATA stream, synchronizing its operation (window generation) with the actual clock/data bits of the data stream. The data separation circuit must track the disk input data frequency very closely—unpredictable bit shifts leave less than 50 ns margin to the window edges.

**Phase-Locked Loop**

Only an analog phase-locked loop (PLL) can provide the reliability required for a double-density data separation circuit. (A phase-locked loop is an electronic circuit that constantly analyzes the frequency of an input signal and locks another oscillator to that frequency.) Using analog PLL techniques, a data separator can be designed with ±1 ns resolution (this would require a 100 MHz clock in a digital phase-locked loop). The analog PLL determines the clock and data bit positions by sampling each bit in the serial data stream. The phase relationship between a data bit and the PLL generated data window is constantly fed back to adjust the position of the data window, enabling the PLL to track input data frequency changes, and thereby reliably read previously recorded data from a floppy disk.

**PLL Design**

A block diagram of the phase-locked loop described in this application note is shown in Figure 7. Basically, the phase-locked loop operates by comparing the frequency of the input data (from the disk drive) against the frequency of a local oscillator. The difference of these frequencies is used to increase or decrease the frequency of the local oscillator in order to bring its frequency closer to that of the input. The PLL synchronizes the local oscillator to the frequency of the input during the all "zeroes" synchronization field on the floppy disk (immediately preceding both the ID field and the data field).

The PLL consists of nine ICs and is located on page 3 of the schematics in the Appendix. The 8272 VCO output essentially turns the PLL circuitry on and off. When the PLL is off, it "idles" at its center frequency. The VCO output turns the PLL on only when valid data is being received from the disk drive. The VCO turns the PLL on after the read/write head has been loaded and the head load time has elapsed. The PLL is turned off in the gap between the ID field and the data field and in the gap after the data field (before the next sector ID field). The GPL parameter in the FDC read and write commands specifies the elapsed time (number of data bytes) that the PLL is turned off in order to blank out discontinuities that appear in the gaps when the write current is turned on and off. The PLL operates with either MFM or FM input data. The MFM output from the FDC controls the PLL operation frequency.

The PLL consists of six functional blocks as follows:

1. **Pulse Shaping** — A 96LS02 senses a READ DATA pulse and provides a clean output signal to the FDC and to the PLL Phase Comparator and Frequency Discriminator circuitry.

2. **Phase Comparator** — The phase difference between the PLL oscillator and the READ DATA input is compared. Pump up (PU) and pump down (PD) error signals are derived from this phase difference and output to the filter. If there is no phase difference between the PLL oscillator and the READ DATA input, the PU and PD pulse widths are equal. If the READ DATA pulse occurs early, the PU duration is shorter than the PD duration. If the data pulse occurs late, the PU duration is longer than the PD duration.

3. **Filter** — This analog circuit filters the PU and PD pulses into an error voltage. This error voltage is buffered by an LM358 operational amplifier.

---

**Figure 7. Phase-Locked Loop Data Separator**
4. PLL Oscillator — This oscillator is composed of a 74LS393, 74LS74, and 96LS02. The oscillator frequency is controlled by the error voltage output by the filter. This oscillator also generates the data window signal to the FDC.

5. Frequency Discriminator — This logic tracks the READ DATA input from the disk drive and discriminates between the synchronization gap for FM recording (250 KHz) and the gap for MFM recording (500 KHz). Synchronization gaps immediately precede address marks.

6. Start Logic — The function of this logic is to clamp the PLL oscillator to its center frequency (2 MHz) until the FDC YCO signal is enabled and a valid data pattern is sensed by the frequency discriminator. The start logic (consisting of a 74LS393 and 74LS74) ensures that the PLL oscillator is started with zero phase error.

PLL Adjustments

The PLL must be initially adjusted to operate at its center frequency with the VCO output off and the adjustment jumper removed. The 5K trimpot should be adjusted until the frequency at the test point (Q output of the 96LS02) is 2 MHz. The jumper should then be replaced for normal operation.

PLL Design Details

The following paragraphs describe the operational and design details of the phase-locked loop data separator illustrated in the appendix. Note that the analog section is operated from a separately filtered +5V supply.

Initialization

As long as the 8272 maintains a low VCO signal, the data separator logic is “turned off”. In this state, the PLL oscillator (96LS02) is not oscillating and therefore the 2XBR signal is constantly low. In addition, the pump up (PU) and pump down (PD) signals are inactive (PU low and PD high), the CNT8 signal is inactive (low), and the filter input voltage is held at 2.5 volts by two 1Mohm resistors between ground and +5 volts.

Floppy Disk Data

The data separator frequency discriminator, the input pulse shaping circuitry, and the start logic are enabled and respond to rising edges of the READ DATA signal. The rising edge of every data bit from the disk drive triggers two pulse shaping one-shots. The first pulse shaper generates a stable and well-defined 200 ns read data pulse for input to the 8272 and other portions of the data separator logic. The second one-shot generates a 2.5 μs data pulse that is used for input data frequency discrimination.

The frequency discriminator operates as illustrated in Figure 8. The 2F output signal is active (high) during reception of valid MFM (double-density) sync fields on the disk while the 1F signal is active (high) during reception of valid FM (single-density) sync fields. A multiplexer (controlled by the 8272 MFM signal) selects the appropriate 1F or 2F signal depending on the programmed mode.
Startup

The data separator is designed to require reception of eight valid sync bits (one sync byte) before enabling the PLL oscillator and attempting to synchronize with the input data stream (see Figure 9). This delay ensures that the PLL will not erroneously synchronize outside a valid sync field in the data stream if the VCO signal is enabled slightly early. The sync bit counter is asynchronously reset by the CNTEN signal when valid sync data is not being received by the drive.

Once the VCO signal is active and eight sync bits have been counted, the CNT8 signal is enabled. This signal turns on the PLL oscillator. Note that this oscillator starts synchronously with the rising edge of the disk input data (because CNT8 is synchronous with the data rising edge) and the oscillator also starts at its center frequency of 2 MHz (because the LM348 filter input is held at its center voltage of approximately 2.5 volts). This frequency is divided by two and four to generate the 2XBR signal (1 MHz for MFM and 500 KHz for FM).
PLL Synchronization

At this point, the PLL is enabled and begins to synchronize with the input data stream. This synchronization is accomplished very simply in the following manner. The pump up (PU) signal is enabled on the rising edge of the READ DATA from the disk drive. (When the PLL is synchronized with the data stream, this point will occur at the same time as the falling edge of the 2XBR signal as shown in Figure 9). The PU signal is turned off and the PD signal is activated on the next rising edge of the 2XBR clock. With this scheme, the difference between PU active time and the PD active time is equal to the difference between the input bit rate and the PLL clock rate. Thus, if PU is turned on longer than PD is on, the input bit rate is faster than the PLL clock.

As long as PU and PD are both inactive, no charge is transferred to or from the LM358 input holding capacitor, and the PLL output frequency is maintained (the LM358 operational amplifier has a very high input impedance). Whenever PU is turned on, current flows from the +5 volt supply through a 20K resistor into the holding capacitor. When the PD signal is turned on, current flows from the holding capacitor to ground through a 20K resistor. In this manner, both the pump up and pump down charging rates are balanced.

The change in capacitor charge (and therefore voltage) after a complete PU/PD cycle is proportional to the difference between the PU and PD pulse widths and is also proportional to the frequency difference between the incoming data stream and the PLL oscillator. As the capacitor voltage is raised (PU active longer than PD), the PLL oscillator time constant (RC of the 96LS02) is modified by the filter output (LM358) to raise the oscillator frequency. As the capacitor voltage is lowered (PD active longer than PU), the oscillator frequency is lowered. If both frequencies are equal, the voltage on the holding capacitor does not change, and the PLL oscillator frequency remains constant.

6. AN INTELLIGENT DISKETTE DATA BASE SYSTEM

The system described in this application note is designed to function as an intelligent data base controller. The schematics for this data base unit are presented in Appendix A; a block diagram of the unit is illustrated in Figure 10. As designed, the unit can access over four million bytes of mass storage on four floppy disk drives (using a single 8272 FDC); the system can easily be expanded to four FDC devices (and 16 megabytes of on-line disk storage). Three serial data links are also included. These data links may be used by CRT terminals or other microprocessor systems to access the data base.

Processor and Memory

A high-performance 8088 eight-bit microprocessor (operating at 5 MHz with no wait states) controls system operation. The 8088 was selected because of its memory addressing capabilities and its sophisticated string handling instructions. These features improve the speed of data base search operations. In addition, these capabilities allow the system to be easily upgraded with additional memory, disk drives, and if required, a bubble memory or Winchester disk unit.

The schematics for the basic design provide 8K bytes of 2732A high-speed EPROM program storage and 8K bytes of disk directory and file buffer RAM. This memory can easily be expanded to 1 megabyte for performance upgrades.

An 8259A Programmable Interrupt Controller (PIC) is also included in the design to field interrupts from both the serial port and the FDC. This interrupt controller provides a large degree of programming flexibility for the implementation of data base functions in an asynchronous, demand driven environment. The PIC allows the system to accumulate asynchronous data base requests from all serial I/O ports while previously specified data base operations are currently in progress. This feature is made possible by the ability of the 8251A RXRDY signal to cause a processor interrupt. After receiving this interrupt, the processor can temporarily halt work on existing requests and enter the incoming information into a data base request buffer. Once the information has been entered into the buffer, the system can resume its previous processing.

In addition, the PIC permits some portions of data base requests to be processed in parallel. For example, once a disk record has been loaded into a memory buffer, a memory search can proceed in parallel with the loading of the next record. After the FDC completes the record transfer, the memory search will be interrupted and the processor can begin another disk transfer before resuming the memory search.

The bus structure of the system is split into three functional buffered units. A 20-bit address from the processor is latched by three-state transparent 74LS373 devices. When the processor is in control of the address and data busses, these devices are output enabled to the system buffered address bus. All I/O devices are placed directly on the local data bus. Finally, the memory data bus is isolated from the local data bus by an 8286 octal transceiver. The direction of this transceiver is determined by the Memory Read signal, while its output enable is activated by a Memory Read or Memory Write command.
Figure 10. Intelligent Data Base Block Diagram
APPLICATIONS

Serial I/O

The three RS-232-C compatible serial I/O ports operate at software-programmable baud rates to 19.2K. Each I/O port is controlled by an 8251A USART (Universal Synchronous/Asynchronous Receiver/Transmitter). Each USART is individually programmable for operation in many synchronous and asynchronous serial data transmission formats (including IBM Bi-sync). In operation, USART error detection circuits can check for parity, data overrun, and framing errors. An 8253 Programmable Interval Timer is employed to generate the baud rates for the serial I/O ports.

The Transmitter Ready and Receiver Ready output signals of the 8251A are routed to the interrupt inputs of the 8259A interrupt controller. These signals interrupt processor execution when a data byte is received by a USART and also when the USART is ready to accept another data byte for transmission.

DMA

The 8272 FDC interfaces to system memory by means of an 8237-2 high-speed DMA controller. Transfers between the disk controller and memory also operate with no wait states when 2114-3 (150 ns) or faster static RAM is used. In operation, the 8272 presents a DMA request to the 8237 for every byte of data to be transferred. This request causes the 8273 to present a HOLD request to the 8088. As soon as the 8088 is able to relinquish data/address bus control, the processor signals a HOLD acknowledge to the 8237. The 8237 then assumes control over the data and address busses. After latching the address for the DMA transfer, the 8237 generates simultaneous I/O Read and Memory Write commands (for a disk read) or simultaneous I/O Write and Memory Read commands (for a disk write). At the same time, the 8272 is selected as the I/O device by means of the DMA acknowledge signal from the 8237. After this single byte has been transferred between the FDC and memory, the DMA controller releases the data/address busses to the 8088 by deactivating the HOLD request. In a short period of time (13 µs for double-density and 27 µs for single-density) the FDC requests a subsequent data transfer. This transfer occurs in exactly the same manner as the previous transfer. After all data transfers have been completed (specified by the word count programmed into the 8237 before the FDC operation was initiated), the 8237 signals a terminal count (EOP pin). This terminal count signal informs the 8272 that the data transfer is complete. Upon reception of this terminal count signal, the 8272 halts DMA requests and initiates an "operation complete" interrupt.

Since the system is designed for 20-bit addressing, a four-bit DMA-address latch is included as a processor addressable I/O port. The processor writes the upper four DMA address bits before a data transfer. When the DMA controller assumes bus control, the contents of this latch are output enabled on the upper four bits of the address bus. The only restriction in the use of this address latch is that a single disk read or write transfer cannot cross a 64K memory boundary.

Disk Drive Interface

The 8272 FDC may be interfaced to a maximum of four eight-inch floppy disk drives. Both single- and double-density drives are accommodated using the data separation circuit described in section 5. In addition, single- or dual-sided disk drives may be used. The 8272 is driven by an 8 MHz crystal clock generator.

Drive select signals are decoded by means of a 74LS139 from the DSO, DS1 outputs of the FDC. The fault reset, step, low current, and direction outputs to the disk drives are generated from the FR/STEP, LCT/DIR, and RW/SEEK FDC output signals by means of a 74LS240. The other half of the 74LS240 functions as an input multiplexer for the disk write protect, two-sided, fault, and track zero status signals. These signals are multiplexed into the WP/TS and FLT/TRK0 inputs to the 8272.

The 8272 write clock (WR CLK) is generated by a ring counter/multiplexer combination. The write clock frequency is 1 MHz for MFM recording and 500 KHz for FM recording (selected by the MFM output of the 8272). The pulse width is a constant 250 ns. The write clock is constantly generated and input to the FDC (during both read and write operations). The FDC write enable output (WE) is transmitted directly to the write gate disk drive input.

Write data to the disk drive is preshifted (according to the PSO, PS1 FDC outputs) by the combination of a 74LS175 four-bit latch and a 74LS153 multiplexer. The amount of preshift is completely controlled within the 8272 FDC. Three cases are possible: the data may be written one clock cycle early, one clock cycle late, or with no preshift. The data preshift circuit is activated by the FDC only in the double-density mode. The preshift is required to cancel predictable playback data shifts when recorded data is later read from the floppy disk.

A single 50-conductor flat cable connects the board to the floppy disk drives. FDC outputs are driven by 7438 open collector high-current line-drivers. These drivers are resistively terminated on the last disk drive by means of a 150 ohm resistor to +5V. The line receivers are 7414 Schmitt triggered inverters with 150 ohm pull-up resistors on board.
7. SPECIAL CONSIDERATIONS
This section contains a quick review of key features and issues, most of which have been mentioned in other sections of this application note. Before designing with the 8272 FDC, it is advisable that the information in this section be completely understood.

1. Multi-Sector Transfers
The 8272 always operates in a multi-sector transfer mode. The 8272 continues to transfer data until the TC input is activated. In a DMA configuration, the TC input of the 8272 must always be connected to the EOP/TC output of the DMA controller. When multiple DMA channels are used on a single DMA controller, EOP must be gated with the select signal for the proper FDC. If the TC signal is not gated, a terminal count on another channel will abort FDC operation.

In a processor driven configuration with no DMA controller, the system must count the transfers and supply a TC signal to the FDC. In a DMA environment, ORing a programmable TC with the TC from the DMA controller is a convenient means of ensuring that the processor may always gain control of the FDC (even if the diskette system hangs up in an abnormal manner).

2. Processor Command/Result Phase Interface
In the command phase, the processor must write the exact number of parameters in the exact order shown in Table 5. During the result phase, the processor must read the complete result status. For example, the Format Track command requires six command bytes and presents seven result bytes. The 8272 will not accept a new command until all result bytes are read. Note that the number of command and result bytes varies from command-to-command. Command and result phases cannot be shortened.

During both the command and result phases, the Main Status Register must be read by the processor before each byte of information is read from, or written to, the FDC Data Register. Before each command byte is written, DIO (bit 6) must be low (indicating a data transfer from the processor) and RQM (bit 7) must be high (indicating that the FDC is ready for data). During the result phase, DIO must be high (indicating a data transfer to the processor) and RQM must also be high (indicating that data is ready for the processor).

NOTE: After the 8272 receives a command byte, the RQM flag may remain set for 12 microseconds (with an 8 MHz clock). Software should not attempt to read the Main Status Register before this time interval has elapsed; otherwise, the software will erroneously assume that the FDC is ready to accept the next byte.

3. Sector Sizes
The 8272 does not support 128 byte sectors in the MFM (double-density) mode.

4. Write Clock
The FDC Write Clock input (WR CLK) must be present at all times.

5. Reset
The FDC Reset input (RST) must be held active during power-on reset while the RD and WR inputs are active. If the reset input becomes inactive while RD and WR are still active, the 8272 enters the test mode. Once activated, the test mode can only be deactivated by a power-down condition.

6. Drive Status
The 8272 constantly polls (starting after the power-on reset) all drives for changes in the drive ready status. At power-on, the FDC assumes that all drives are not ready. If a drive application requires that the ready line be strapped active, the FDC will generate an interrupt immediately after power is applied.

7. Gap Length
Only the gap 3 size is software programmable. All other gap sizes are fixed. In addition, different gap 3 sizes must be specified in format, read, write, and scan commands. Refer to Section 3 and Table 9 for gap size recommendations.

8. Seek Command
The drive busy flag in the Main Status Register remains set after a Seek command is issued until the Sense Interrupt Status command is issued (following reception of the seek complete interrupt).

The FDC does not perform implied seeks. Before issuing data read or write commands, the read/write head must be positioned over the correct cylinder. If the head is not positioned correctly, a cylinder address error is generated.

After issuing a step pulse, the 8272 resumes drive status polling. For correct stepper operation in this mode, the stepper motor must be constantly enabled. (Most drives provide a jumper to permit the stepper motor to be constantly enabled.)

9. Step Rate
The 8272 can emit a step pulse that is one millisecond faster than the rate programmed by the SRT parameter in the Specify command. This action may cause subsequent sector not found errors. The step rate time should be programmed to be 1 ms longer than the step rate time required by the drive.

10. Cable Length
A cable length of less than 10 feet is recommended for drive interfacing.
11. Scan Commands
The current 8272 has several problems when using the scan commands. These commands should not be used at this time.

12. Interrupts
When the processor receives an interrupt from the FDC, the FDC may be reporting one of two distinct events:

   a) The beginning of the result phase of a previously requested read, write, or scan command.

   b) An asynchronous event such as a seek/recalibrate completion, an attention, an abnormal command termination, or an invalid command.

These two cases are distinguished by the FDC busy flag (bit 4) in the Main Status Register. If the FDC busy flag is high, the interrupt is of type (a). If the FDC busy flag is low, the interrupt was caused by an asynchronous event (b).

A single interrupt from the FDC may signal more than one of the above events. After receiving an interrupt, the processor must continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all “hidden” interrupts are ferreted out and serviced.

13. Skip Flag (SK)
The skip flag is used during the execution of Read Data, Read Deleted Data, Read Track, and various Scan commands. This flag permits the FDC to skip unwanted sectors on a disk track.

When performing a Read Data, Read Track, or Scan command, a high SK flag indicates that the FDC is to skip over (not transfer) any sector containing a deleted data address mark. A low SK flag indicates that the FDC is to terminate the command (after reading all the data in the sector) when a deleted data address mark is encountered.

When performing a Read Deleted Data command, a high SK flag indicates that sectors containing normal data address marks are to be skipped. Note that this is just the opposite situation from that described in the last paragraph. When a data address mark is encountered during a Read Deleted Data command (and the SK flag is low), the FDC terminates the command after reading all the data in the sector.

14. Bad Track Maintenance
The 8272 does not internally maintain bad track information. The maintenance of this information must be performed by system software. As an example of typical bad track operation, assume that a media test determines that track 31 and track 66 of a given floppy disk are bad. When the disk is formatted for use, the system software formats physical track 0 as logical cylinder 0 (C=0 in the command phase parameters), physical track 1 as logical track 1 (C=1), and so on, until physical track 30 is formatted as logical cylinder 30 (C=30). Physical track 31 is bad and should be formatted as logical cylinder FF (indicating a bad track). Next, physical track 32 is formatted as logical cylinder 31, and so on, until physical track 67 is formatted as logical cylinder 64. Next, bad physical track 66 is formatted as logical cylinder FF (another bad track marker), and physical track 67 is formatted as logical cylinder 65. This formatting continues until the last physical track (77) is formatted as logical cylinder 75. Normally, after this formatting is complete, the bad track information is stored in a prespecified area on the floppy disk (typically in a sector on track 0) so that the system will be able to recreate the bad track information when the disk is removed from the drive and reinserted at some later time.

To illustrate how the system software performs a transfer operation disk with bad tracks, assume that the disk drive head is positioned at track 0 and the disk described above is loaded into the drive. If a command to read track 36 is issued by an application program, the system software translates this read command into a seek to physical track 37 (since there is one bad track between 0 and 36, namely 31) followed by a read of logical cylinder 36. Thus, the cylinder parameter C is set to 37 for the Seek command and 36 for the Read Sector command.

15. Head Load versus Head Settle Times
The 8272 does not permit separate specification of the head load time and the head settle time. When the Specify command is issued for a given disk drive, the proper value for the HLT parameter is the maximum of the head load time and the head settle time.
<table>
<thead>
<tr>
<th>Part</th>
<th>Ref Desig</th>
<th>+5</th>
<th>GND</th>
<th>+12</th>
<th>-12</th>
</tr>
</thead>
<tbody>
<tr>
<td>8088</td>
<td>A2</td>
<td>40</td>
<td>1,20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8224</td>
<td>I6</td>
<td>9,16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8237-2</td>
<td>A6</td>
<td>31</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8251A</td>
<td>A9,B9,C9</td>
<td>26</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8253-5</td>
<td>A10</td>
<td>24</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8259A</td>
<td>B10</td>
<td>28</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8272</td>
<td>D10</td>
<td>40</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8284</td>
<td>A1</td>
<td>18</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8286</td>
<td>B6,F4</td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2114</td>
<td>F1,F2,G1,G2,H1,H2,I1,I2</td>
<td>18</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2732A</td>
<td>D1,D2</td>
<td>24</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS00</td>
<td>E1</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS04</td>
<td>B2,E6,E8,F8</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS27</td>
<td>E2,E5</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS32</td>
<td>B1</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS74</td>
<td>A4,G5,H6</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS138</td>
<td>F3</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS139</td>
<td>E10</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS153</td>
<td>I3</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS157</td>
<td>F6</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS164</td>
<td>F5</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS173</td>
<td>G3</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS175</td>
<td>G4</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS240</td>
<td>G10</td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS257</td>
<td>D3</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS367</td>
<td>C3,E9</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS373</td>
<td>B4,C4,D4,C6</td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74LS393</td>
<td>I5,F7</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74S08</td>
<td>E4</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74S138</td>
<td>D6,E3</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7414</td>
<td>H7</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7438</td>
<td>H8,H9,H10</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1488</td>
<td>H3</td>
<td>7</td>
<td></td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>1489</td>
<td>H4</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96LS02</td>
<td>G7</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96LS02</td>
<td>G6</td>
<td></td>
<td></td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>LM358</td>
<td>H5</td>
<td></td>
<td></td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>
REFERENCES

Implementation of Design and Software Disk Subsystems Floppy Disk
Software Design and Implementation of Floppy Disk Subsystems

Contents

1. INTRODUCTION
   The Physical Interface Level
   The Logical Interface Level
   The File System Interface Level
   Scope of this Note

2. DISK I/O TECHNIQUES
   FDC Data Transfer Interface
   Overlapped Operations
   Buffers

3. THE 8272 FLOPPY DISK CONTROLLER
   Floppy Disk Commands
   Interface Registers
   Command/Result Phases
   Execution Phase
   Multi-sector and Multi-track Transfers
   Drive Status Polling
   Command Details
   Invalid Commands

4. 8272 PHYSICAL INTERFACE SOFTWARE
   INITIALIZE$DRIVERS
   EXECUTE$DOCB
   FDCINT
   OUTPUT$CONTROLS$TO$DMA
   OUTPUT$COMMAND$TO$FDC
   INPUT$RESULT$FROM$FDC
   OUTPUT$BYTE$TO$FDC
   INPUT$BYTE$FROM$FDC
   FDC$READY$FOR$COMMAND
   FDC$READY$FOR$RESULT
   OPERATION$CLEANS$UP
   Modifications for Polling Operation

5. 8272 LOGICAL INTERFACE SOFTWARE
   SPECIFY
   RECALIBRATE
   SEEK
   FORMAT
   WRITE
   READ
   Coping With Errors
Contents (Continued)

6. FILE SYSTEMS
   File Allocation
   The Intel File System
   Disk File System Functions

7. KEY 8272 SOFTWARE
   INTERFACING CONSIDERATIONS

REFERENCES

APPENDIX A—8272 FDC
   DEVICE DRIVER SOFTWARE

APPENDIX B—8272 FDC
   EXERCISER PROGRAM

APPENDIX C—8272 DRIVER FLOWCHARTS
1. Introduction

Disk interface software is a major contributor to the efficient and reliable operation of a floppy disk subsystem. This software must be a well-designed compromise between the needs of the application software modules and the capabilities of the floppy disk controller (FDC). In an effort to meet these requirements, the implementation of disk interface software is often divided into several levels of abstraction. The purpose of this application note is to define these software interface levels and describe the design and implementation of a modular and flexible software driver for the 8272 FDC. This note is a companion to AP-116, "An Intelligent Data Base System Using the 8272."

The Physical Interface Level

The software interface level closest to the FDC hardware is referred to as the physical interface level. At this level, interface modules (often called disk drivers or disk handlers) communicate directly with the FDC device. Disk drivers accept floppy disk commands from other software modules, control and monitor the FDC execution of the commands, and finally return operational status information (at command termination) to the requesting modules.

In order to perform these functions, the drivers must support the bit/byte level FDC interface for status and data transfers. In addition, the drivers must field, classify, and service a variety of FDC interrupts.

The Logical Interface Level

System and application software modules often specify disk operation parameters that are not directly compatible with the FDC device. This software incompatibility is typically caused by one of the following:

1. The change from an existing FDC to a functionally equivalent design. Replacing a TTL based controller with an LSI device is an example of a change that may result in software incompatibilities.

2. The upgrade of an existing FDC subsystem to a higher capability design. An expansion from a single-sided, single-density system to a dual-sided, double-density system to increase data storage capacity is an example of such a system change.

3. The abstraction of the disk software interface to avoid redundancy. Many FDC parameters (in particular the density, gap size, number of sectors per track and number of bytes per sector) are fixed for a floppy disk (after formatting). In fact, in many systems these parameters are never changed during the life of the system.
4. The requirement to support a software interface that is independent of the type of disk attached to the system. In this case, a system generated ("logical") disk address (drive, head, cylinder, and sector numbers) must be mapped into a physical floppy disk address. For example, to switch between single- and dual-sided disks, it may be easier and more cost-effective for the software to treat the dual-sided disk as containing twice as many sectors per track (52) rather than as having two sides. With this technique, accesses to sectors 1 through 26 are mapped onto head 0 while accesses to sectors 27 through 52 are mapped onto head 1.

5. The necessity of supporting a bad track map. Since bad tracks depend on the disk media, the bad track mapping varies from disk to disk. In general, the system and application software should not be concerned with calculating bad track parameters. Instead, these software modules should refer to cylinders logically (0 through 76). The logical interface level procedures must map these cylinders into physical cylinder positions in order to avoid the bad tracks.

The key to logical interface software design is the mapping of the "logical disk interface" (as seen by the application software) into the "physical disk interface" (as implemented by the floppy disk drivers). This logical to physical mapping is tightly coupled to system software design and the mapping serves to isolate both applications and system software from the peculiarities of the FDC device. Typical logical interface procedures are described in Table 1.

The File System Interface Level

The file system typically comprises the highest level of disk interface software used by application programs. The file system is designed to treat the disk as a collection of named data areas (known as files). These files are cataloged in the disk directory. File system interface software permits the creation of new files and the deletion of existing files under software control. When a file is created, its name and disk address are entered into the directory; when a file is deleted, its name is removed from the directory. Application software requests the use of a file by executing an OPEN function. Once opened, a file is normally reserved for use by the requesting program or task and the file cannot be reopened by other tasks. When a task no longer needs to use an open file, the task closes the file, releasing it for use by other tasks.

Most file systems also support a set of file attributes that can be specified for each file. File attributes may be used to protect files (e.g., the WRITE PROTECT attribute ensures that an existing file cannot accidentally be overwritten) and to supply system configuration information (e.g., a FORMAT attribute may specify that a file should automatically be created on a new disk when the disk is formatted).

At the file system interface level, application programs need not be explicitly aware of disk storage allocation techniques, block sizes, or file coding strategies. Only a "file name" must be presented in order to open, read or write, and subsequently close a file. Typical file system functions are listed in Table 2.
### APPLICATIONS

#### Table 1: Examples of Logical Interface Procedures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT DISK</td>
<td>Controls physical disk formatting for all tracks on a disk. Formatting adds FDC recognized cylinder, head, and sector addresses as well as address marks and data synchronization fields (gaps) to the floppy disk media.</td>
</tr>
<tr>
<td>RECALIBRATE</td>
<td>Moves the disk read/write head to track 0 (at the outside edge of the disk).</td>
</tr>
<tr>
<td>SEEK</td>
<td>Moves the disk read/write head to a specified logical cylinder. The logical and physical cylinder numbers may be different if bad track mapping is used.</td>
</tr>
<tr>
<td>READ STATUS</td>
<td>Indicates the status of the floppy disk drive and media. One important use of this procedure is to determine whether a floppy disk is dual-sided.</td>
</tr>
<tr>
<td>READ SECTOR</td>
<td>Reads one or more complete sectors starting at a specified disk address (drive, head, cylinder, and sector).</td>
</tr>
<tr>
<td>WRITE SECTOR</td>
<td>Writes one or more complete sectors starting at a specified disk address (drive, head, cylinder, and sector).</td>
</tr>
</tbody>
</table>
### Table 2: Disk File System Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN</td>
<td>Prepare a file for processing. If the file is to be opened for input and the file name is not found in the directory, an error is generated. If the file is opened for output and the file name is not found in the directory, the file is automatically created.</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Terminate processing of an open file.</td>
</tr>
<tr>
<td>READ</td>
<td>Transfer data from an open file to memory. The READ function is often designed to buffer one or more sectors of data from the disk drive and supply this data to the requesting program, as required.</td>
</tr>
<tr>
<td>WRITE</td>
<td>Transfer data from memory to an open file. The WRITE function is often designed to buffer data from the application program until enough data is available to fill a disk sector.</td>
</tr>
<tr>
<td>CREATE</td>
<td>Initialize a file and enter its name and attributes into the file directory.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove a file from the directory and release its storage space.</td>
</tr>
<tr>
<td>RENAME</td>
<td>Change the name of a file in the directory.</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>Change the attributes of a file.</td>
</tr>
<tr>
<td>LOAD</td>
<td>Read a file of executable code into memory.</td>
</tr>
<tr>
<td>INITDISK</td>
<td>Initialize a disk by formatting the media and establishing the directory file, the bit map file, and other system files.</td>
</tr>
</tbody>
</table>
Scope of this Note

This application note directly addresses the logical and physical interface levels. A complete 8272 driver (including interrupt service software) is listed in Appendix A. In addition, examples of recalibrate, seek, format, read, and write logical interface level procedures are included as part of the exerciser program found in Appendix B. Wherever possible, specific hardware configuration dependencies are parametized to provide maximum flexibility without requiring major software changes.
2. Disk I/O Techniques

One of the most important software aspects of disk interfacing is the fixed sector size. (Sector sizes are fixed when the disk is formatted.) Individual bytes of disk storage cannot be read/written; instead, complete sectors must be transferred between the floppy disk and system memory.

Selection of the appropriate sector size involves a tradeoff between memory size, disk storage efficiency, and disk transfer efficiency. Basically, the following factors must be weighed:

1. Memory size. The larger the sector size, the larger the memory area that must be reserved for use during disk I/O transfers. For example, a 1K byte disk sector size requires that at least one 1K memory block be reserved for disk I/O.

2. Disk Storage efficiency. Both very large and very small sectors can waste disk storage space as follows. In disk file systems, space must be allocated somewhere on the disk to link the sectors of each file together. If most files are composed of many small sectors, a large amount of linkage overhead information is required. At the other extreme, when most files are smaller than a single disk sector, a large amount of space is wasted at the end of each sector.

3. Disk transfer efficiency. A file composed of a few large sectors can be transferred to/from memory more efficiently (faster and with less overhead) than a file composed of many small sectors.

Balancing these considerations requires knowledge of the intended system applications. Typically, for general purpose systems, sector sizes from 128 bytes to 1K bytes are used. For compatibility between single-density and double-density recording with the 8272 floppy disk controller, 256 byte sectors or 512 byte sectors are most useful.

FDC Data Transfer Interface

Three distinct software interface techniques may be used to interface system memory to the FDC device during sector data transfers:

1. DMA - In a DMA implementation, the software is only required to set up theDMA controller memory address and transfer count, and to initiate the data transfer. The DMA controller hardware handshakes with the processor/system bus in order to perform each data transfer.

2. Interrupt Driven - The FDC generates an interrupt when a data byte is ready to be transferred to memory, or when a data byte is needed from memory. It is the software's responsibility to perform appropriate memory reads/writes in order to transfer data from/to the FDC upon receipt of the interrupt.

3. Polling - Software responsibilities in the polling mode are identical to the responsibilities in the interrupt driven mode. The polling mode, however, is used when interrupt service overhead (context switching) is too large to support the disk data
rate. In this mode, the software determines when to transfer data by continually polling a data request status flag in the FDC status register.

The DMA mode has the advantage of permitting the processor to continue executing instructions while a disk transfer is in progress. (This capability is especially useful in multiprogramming environments when the operating system is designed to permit other tasks to execute while a program is waiting for I/O.) Modes 2 and 3 are often combined and described as non-DMA operating modes. Non-DMA modes have the advantage of significantly lower system cost, but are often performance limited for double-density systems (where data bytes must be transferred to/from the FDC every 16 microseconds).

Overlapped Operations

Some FDC devices support simultaneous disk operations on more than one disk drive. Normally seek and recalibrate operations can be overlapped in this manner. Since seek operations on most floppy drives are extremely slow, this mode of operation can often be used by the system software to reduce overall disk access times.

Buffers

The buffer concept is an extremely important element in advanced disk I/O strategies. A buffer is nothing more than a memory area containing the same amount of data as a disk sector contains. Generally, when an application program requests data from a disk, the system software allocates a buffer (memory area) and transfers the data from the appropriate disk sector into the buffer. The address of the buffer is then returned to the application software. In the same manner, after the application program has filled a buffer for output, the buffer address is passed to the system software, which writes data from the buffer into a disk sector. In multitasking systems, multiple buffers may be allocated from a buffer pool. In these systems, the disk controller is often requested to read ahead and fill additional data buffers while the application software is processing a previous buffer. Using this technique, system software attempts to fill buffers before they are needed by the application programs, thereby eliminating program waits during I/O transfers. Figure 1 illustrates the use of multiple buffers in a ring configuration.
a) The first disk read request by the application software causes the disk subsystem to begin filling the first empty buffer. The application software must wait until the buffer is filled before it may continue execution.

Figure 1. Using Multiple Memory Buffers for Disk I/O
b) After the first buffer is filled, the disk system continues to transfer disk data into the next buffer while the application software begins operating on the first full buffer.

Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)
c) When all empty buffers have been filled, disk activity is stopped until the application software releases one or more buffers for reuse.

Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)
d) When the application software releases a buffer (for reuse), the disk subsystem begins a disk sector read to refill the buffer. This strategy attempts to anticipate application software needs by maintaining a sufficient number of full data buffers in order to minimize data transfer delays. If disk data is already in memory when the application software requests it, no disk transfer delays are incurred.

Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)
3. **THE 8272 FLOPPY DISK CONTROLLER**

The 8272 is a single-chip LSI Floppy Disk Controller (FDC) that implements both single- and double-density floppy disk storage subsystems (with up to four dual-sided disk drives per FDC). The 8272 supports the IBM 3740 single-density recording format (FM) and the IBM System 34 double-density recording format (MFM). The 8272 accepts and executes high-level disk commands such as format track, seek, read sector, and write sector. All data synchronization and error checking is automatically performed by the FDC to ensure reliable data storage and subsequent retrieval. The 8272 interfaces to microprocessor systems with or without Direct Memory Access (DMA) capabilities and also interfaces to a large number of commercially available floppy disk drives.

**Floppy Disk Commands**

The 8272 executes fifteen high-level disk interface commands:

- Specify
- Sense Drive Status
- Sense Interrupt Status
- Seek
- Recalibrate
- Format Track
- Read Data
- Read Deleted Data
- Write Data
- Write Deleted Data
- Read Track
- Read ID
- Scan Equal
- Scan High or Equal
- Scan Low or Equal

Each command is initiated by a multi-byte transfer from the driver software to the FDC (the transferred bytes contain command and parameter information). After complete command specification, the FDC automatically executes the command. The command result data (after execution of the command) may require a multi-byte transfer of status information back to the driver. It is convenient to consider each FDC command as consisting of the following three phases:

**Command Phase:** The driver transfers to the FDC all the information required to perform a particular disk operation. The 8272 automatically enters the command phase after RESET and following the completion of the result phase (if any) of a previous command.

**Execution Phase:** The FDC performs the operation as instructed. The execution phase is entered immediately after the last command parameter is written to the FDC in the preceding command phase. The execution phase normally ends when the last data byte is transferred to/from the disk or when an error occurs.

**Result Phase:** After completion of the disk operation, status and other housekeeping information are made available to the driver software. After this information is read, the FDC reenters the command phase and is ready to accept another command.
APPLICATIONS

Interface Registers

To support information transfer between the FDC and the system software, the 8272 contains two 8-bit registers: the Main Status Register and the Data Register. The Main Status Register (read only) contains FDC status information and may be accessed at any time. The Main Status Register (Table 3) provides the system processor with the status of each disk drive, the status of the FDC, and the status of the processor interface. The Data Register (read/write) stores data, commands, parameters, and disk drive status information. The Data Register is used to program the FDC during the command phase and to obtain result information after completion of FDC operations.

In addition to the Main Status Register, the FDC contains four additional status registers (ST0, ST1, ST2, and ST3). These registers are only available during the result phase of a command.

Command/Result Phases

Table 4 lists the 8272 command set. For each of the fifteen commands, command and result phase data transfers are listed. A list of abbreviations used in the table is given in Table 5, and the contents of the result status registers (ST0–ST3) are illustrated in Table 6.

The bytes of data which are sent to the 8272 by the drivers during the command phase, and are read out of the 8272 in the result phase, must occur in the order shown in Table 4. That is, the command code must be sent first and the other bytes sent in the prescribed sequence. All bytes of the command and result phases must be read/written as described. After the last byte of data in the command phase is sent to the 8272 the execution phase automatically starts. In a similar fashion, when the last byte of data is read from the 8272 in the result phase, the result phase is automatically ended and the 8272 reenters the command phase.

It is important to note that during the result phase all bytes shown in Table 4 must be read. The Read Data command, for example, has seven bytes of data in the result phase. All seven bytes must be read in order to successfully complete the Read Data command. The 8272 will not accept a new command until all seven bytes have been read. The number of command and result bytes varies from command-to-command.

In order to read data from, or write data to, the Data Register during the command and result phases, the software driver must examine the Main Status Register to determine if the Data Register is available. The DIO (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command word may be written into the 8272. Many of the commands require multiple bytes, and as a result, the Main Status Register must be read prior to each byte transfer to the 8272. To read status bytes during the result phase, DIO and RQM in the Main Status Register must both be high. Note, checking the Main Status Register in this manner before each byte transfer to/from the 8272 is required only in the command and result phases, and is NOT required during the execution phase.
## APPLICATIONS

### Table 3: Main Status Register Bit Definitions

<table>
<thead>
<tr>
<th>BIT NUMBER</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>D_{0}B</td>
<td>Disk Drive 0 Busy. Disk Drive 0 is seeking.</td>
</tr>
<tr>
<td>1</td>
<td>D_{1}B</td>
<td>Disk Drive 1 Busy. Disk Drive 1 is seeking.</td>
</tr>
<tr>
<td>2</td>
<td>D_{2}B</td>
<td>Disk Drive 2 Busy. Disk Drive 2 is seeking.</td>
</tr>
<tr>
<td>3</td>
<td>D_{3}B</td>
<td>Disk Drive 3 Busy. Disk Drive 3 is seeking.</td>
</tr>
<tr>
<td>4</td>
<td>CB</td>
<td>FDC Busy. A read or write command is in progress.</td>
</tr>
<tr>
<td>5</td>
<td>NDM</td>
<td>Non-DMA Mode. The FDC is in the non-DMA mode when this flag is set (1). This flag is set only during the execution phase of commands in the non-DMA mode. Transition of this flag to a zero (0) indicates that the execution phase has ended.</td>
</tr>
<tr>
<td>6</td>
<td>DIO</td>
<td>Data Input/Output. Indicates the direction of a data transfer between the FDC and the Data Register. When DIO is set (1), data is read from the Data Register by the processor; when DIO is reset (0), data is written from the processor to the Data Register.</td>
</tr>
<tr>
<td>7</td>
<td>RQM</td>
<td>Request for Master. When set (1), this flag indicates that the Data Register is ready to send data to, or receive data from, the processor.</td>
</tr>
</tbody>
</table>
### Applications

#### Table 4: 8272 Command Set

<table>
<thead>
<tr>
<th>PHASE</th>
<th>Read Data</th>
<th>DATA BUS</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>MT MFM SK</td>
<td>0 0 0 1 1 0</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Execution</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>ST 0</td>
<td></td>
<td>Sector ID information after Command execution</td>
</tr>
<tr>
<td></td>
<td>ST 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ST 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td></td>
<td>Sector ID information after command execution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>READ DATA</th>
<th>READ A TRACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>MT MFM SK</td>
</tr>
<tr>
<td>Execution</td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>ST 0</td>
</tr>
<tr>
<td></td>
<td>ST 1</td>
</tr>
<tr>
<td></td>
<td>ST 2</td>
</tr>
<tr>
<td></td>
<td>H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WRITE DATA</th>
<th>FORMAT A TRACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>MT MFM 0 0 0 1 0 1</td>
</tr>
<tr>
<td>Execution</td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>ST 0</td>
</tr>
<tr>
<td></td>
<td>ST 1</td>
</tr>
<tr>
<td></td>
<td>ST 2</td>
</tr>
<tr>
<td></td>
<td>H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WRITE DELETED DATA</th>
<th>SCAN EQUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>MT MFM SK</td>
</tr>
<tr>
<td>Execution</td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>ST 0</td>
</tr>
<tr>
<td></td>
<td>ST 1</td>
</tr>
<tr>
<td></td>
<td>ST 2</td>
</tr>
<tr>
<td></td>
<td>H</td>
</tr>
</tbody>
</table>

Note 1: A0 = 1 for all operations

| Command | MT MFM SK | 0 0 0 1 0 0 |
| Execution | | |
| Result | ST 0 | Status information after Command execution |
| | ST 1 | |
| | ST 2 | |
| | H | |

Command Codes:
- MT MFM SK: Mode 1, Mode 2, Mode 3
- 000000: No Change
- 000001: Sector ID information after Command execution
- 000011: Sector ID information after command execution
- 001011: Sector ID information after Command execution

Data transfer between the FDD and the main-system.

The first correct ID information on the track is stored in Data Register.

FDC formats an entire track.

In this case, the ID information has no meaning.
### APPLICATIONS

#### DATA BUS

<table>
<thead>
<tr>
<th>PHASE</th>
<th>R/W</th>
<th>D7 D6 D5 D4 D3 D2 D1 D0</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAN LOW OR EQUAL</td>
<td>Command</td>
<td>W MT MFM SK 1 1 0 0 1 1 0 0</td>
<td>Command Codes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W 0 0 0 0 0 0 0 0 HDS D51 D50</td>
<td>Sector ID information prior Command execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W C H R R R R R R</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>W EOT GPL STP</td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td>Result</td>
<td>R ST 0 C ST 2</td>
<td>Status information after Command execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R H R R R</td>
<td>Sector ID information after Command execution</td>
</tr>
<tr>
<td>SCAN HIGH OR EQUAL</td>
<td>Command</td>
<td>W MT MFM SK 1 1 1 1 0 1 0 0</td>
<td>Command Codes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W 0 0 0 0 0 0 0 0 HDS D51 D50</td>
<td>Sector ID information prior Command execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W C H R R R R R R</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>W EOT GPL STP</td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td>Result</td>
<td>R ST 0 C ST 2</td>
<td>Status information after Command execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R H R R R</td>
<td>Sector ID information after Command execution</td>
</tr>
</tbody>
</table>

#### RECALIBRATE

<table>
<thead>
<tr>
<th>PHASE</th>
<th>R/W</th>
<th>D7 D6 D5 D4 D3 D2 D1 D0</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>W</td>
<td>0 0 0 0 0 0 0 0 0 D51 D50</td>
<td>Command Codes</td>
</tr>
<tr>
<td>Execution</td>
<td>W</td>
<td>0 0 0 0 0 0 0 0 0 D51 D50</td>
<td>Head retracted to Track 0</td>
</tr>
</tbody>
</table>

#### SENSE INTERRUPT STATUS

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
<th>R ST 0 C</th>
<th>Status information at the end of each seek operation about the FDC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W SPT</td>
<td>HLT HLT</td>
<td>Timer Settings</td>
</tr>
</tbody>
</table>

#### SENSE DRIVE STATUS

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
<th>R ST 3</th>
<th>Status information about the FDD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W SPT</td>
<td>HLT HLT</td>
<td>Timer Settings</td>
</tr>
</tbody>
</table>

#### SEEK

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
<th>R C</th>
<th>Head is positioned over proper Cylinder on Diskette</th>
</tr>
</thead>
</table>

#### INVALID

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
<th>R ST 0</th>
<th>Invalid Command Codes (NoOp—FDC goes into Standby State)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W</td>
<td></td>
<td>ST 0 = 80 (18)</td>
</tr>
</tbody>
</table>

---

5-522

207885-001
## APPLICATIONS

### Table 5: Command/Result Parameter Abbreviations

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Cylinder Address. The currently selected cylinder address (0 to 76) on the disk.</td>
</tr>
<tr>
<td>D</td>
<td>Data Pattern. The pattern to be written in each sector data field during formatting.</td>
</tr>
<tr>
<td>DS0,DS1</td>
<td>Disk Drive Select.</td>
</tr>
<tr>
<td></td>
<td><img src="#" alt="Table" /></td>
</tr>
<tr>
<td>DS1 DS0</td>
<td>Drive</td>
</tr>
<tr>
<td>0 0</td>
<td>Drive 0</td>
</tr>
<tr>
<td>0 1</td>
<td>Drive 1</td>
</tr>
<tr>
<td>1 0</td>
<td>Drive 2</td>
</tr>
<tr>
<td>1 1</td>
<td>Drive 3</td>
</tr>
<tr>
<td>DTL</td>
<td>Special Sector Size. During the execution of disk read/write commands, this parameter is used to temporarily alter the effective disk sector size. By setting N to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the disk) is larger than DTL specifies, the remainder of the actual sector is not passed to the system during read commands; during write commands, the remainder of the actual sector is written with all-zeroes bytes. DTL should be set to FF hexadecimal when N is not zero.</td>
</tr>
<tr>
<td>EOT</td>
<td>End of Track. The final sector number of the current track.</td>
</tr>
<tr>
<td>GPL</td>
<td>Gap Length. The gap 3 size. (Gap 3 is the space between sectors.)</td>
</tr>
<tr>
<td>H</td>
<td>Head Address. Selected head: 0 or 1 (disk side 0 or 1, respectively) as encoded in the sector ID field.</td>
</tr>
<tr>
<td>HLT</td>
<td>Head Load Time. Defines the time interval that the FDC waits after loading the head before initiating a read or write operation. Programmable from 2 to 254 milliseconds (in increments of 2 ms).</td>
</tr>
<tr>
<td>HUT</td>
<td>Head Unload Time. Defines the time interval from the end of the execution phase (of a read or write command) until the head is unloaded. Programmable from 16 to 240 milliseconds (in increments of 16 ms).</td>
</tr>
<tr>
<td>MFM</td>
<td>MFM/FM Mode Selector. Selects MFM double-density recording mode when high, FM single-density mode when low.</td>
</tr>
<tr>
<td>MT</td>
<td>Multi-Track Selector. When set, this flag selects the multi-track operating mode. In this mode (used only with dual-sided disks), the FDC treats a complete cylinder (under both read/write head 0 and read/write head 1) as a single track. The FDC operates as if this expanded track started at the first sector under head 0 and ended at the last sector under head 1. With this flag set (high), a multi-sector read operation will automatically continue to the first sector under head 1 when the FDC finishes operating on the last sector under head 0.</td>
</tr>
<tr>
<td>N</td>
<td>Sector Size Code. The number of data bytes within a sector.</td>
</tr>
</tbody>
</table>
### APPLICATIONS

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND</td>
<td>Non-DMA Mode Flag. When set (1), this flag indicates that the FDC is to operate in the non-DMA mode. In this mode, the processor participates in each data transfer (by means of an interrupt or by polling the RQM flag in the Main Status Register). When reset (0), the FDC interfaces to a DMA controller.</td>
</tr>
<tr>
<td>R</td>
<td>Sector Address. Specifies the sector number to be read or written. In multi-sector transfers, this parameter specifies the sector number of the first sector to be read or written.</td>
</tr>
<tr>
<td>SC</td>
<td>Number of Sectors per Track. Specifies the number of sectors per track to be initialized by the Format Track command.</td>
</tr>
<tr>
<td>SK</td>
<td>Skip Flag. When this flag is set, sectors containing deleted data address marks will automatically be skipped during the execution of multi-sector Read Data or Scan commands. In the same manner, a sector containing a data address mark will automatically be skipped during the execution of a multi-sector Read Deleted Data command.</td>
</tr>
<tr>
<td>SRT</td>
<td>Step Rate Interval. Defines the time interval between step pulses issued by the FDC (track-to-track access time). Programmable from 1 to 16 milliseconds (in increments of 1 ms).</td>
</tr>
<tr>
<td>ST0</td>
<td>Status Register 0-3. Registers within the FDC that store status information after a command has been executed. This status information is available to the processor during the Result Phase after command execution. These registers may only be read after a command has been executed (in the exact order shown in Table 4 for each command). These registers should not be confused with the Main Status Register.</td>
</tr>
<tr>
<td>STP</td>
<td>Scan Sector Increment. During Scan operations, this parameter is added to the current sector number in order to determine the next sector to be scanned.</td>
</tr>
</tbody>
</table>
### Table 6: Status Register Definitions

#### Status Register 0

<table>
<thead>
<tr>
<th>BIT NUMBER</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>7,6</td>
<td>IC</td>
<td>Interrupt Code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 - Normal termination of command. The specified command was properly executed and completed without error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 - Abnormal termination of command. Command execution was started but could not be successfully completed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 - Invalid command. The requested command could not be executed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 - Abnormal termination. During command execution, the disk drive ready signal changed state.</td>
</tr>
<tr>
<td>5</td>
<td>SE</td>
<td>Seek End. This flag is set (1) when the FDC has completed the Seek command and the read/write head is positioned over the correct cylinder.</td>
</tr>
<tr>
<td>4</td>
<td>EC</td>
<td>Equipment Check Error. This flag is set (1) if a fault signal is received from the disk drive or if the track 0 signal is not received from the disk drive after 77 step pulses (Recalibrate command).</td>
</tr>
<tr>
<td>3</td>
<td>NR</td>
<td>Not Ready Error. This flag is set if a read or write command is issued and either the drive is not ready or the command specifies side 1 (head 1) of a single-sided disk.</td>
</tr>
<tr>
<td>2</td>
<td>H</td>
<td>Head Address. The head address at the time of the interrupt.</td>
</tr>
<tr>
<td>1,0</td>
<td>DSL,DSO</td>
<td>Drive Select. The number of the drive selected at the time of the interrupt.</td>
</tr>
</tbody>
</table>

#### Status Register 1

<table>
<thead>
<tr>
<th>BIT NUMBER</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>EN</td>
<td>End of Track Error. This flag is set if the FDC attempts to access a sector beyond the final sector of the track.</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Undefined</td>
</tr>
<tr>
<td>5</td>
<td>DE</td>
<td>Data Error. Set when the FDC detects a CRC error in either the ID field or the data field of a sector.</td>
</tr>
<tr>
<td>4</td>
<td>OR</td>
<td>Overrun Error. Set (during data transfers) if the FDC does not receive DMA or processor service within the specified time interval.</td>
</tr>
</tbody>
</table>
Applications

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND</td>
<td>Undefined</td>
</tr>
<tr>
<td>NW</td>
<td>Write Protect Error. This flag is set if the FDC detects a write protect signal from the disk drive during the execution of a Write Data, Write Deleted Data, or Format Track command.</td>
</tr>
<tr>
<td>MA</td>
<td>Missing Address Mark Error. This flag is set by either of the following conditions:</td>
</tr>
<tr>
<td></td>
<td>a) The FDC cannot detect the ID address mark on the specified track (after two rotations of the disk).</td>
</tr>
<tr>
<td></td>
<td>b) The FDC cannot detect the data address mark or deleted data address mark on the specified track. (See also the MD bit of Status Register 2.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Register 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT NUMBER</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
Bad Track Error. Set when the cylinder address from the disk sector ID field is FF hexadecimal and this cylinder address is different from the current cylinder address maintained within the FDC. This all "ones" cylinder number indicates a bad track (one containing hard errors) according to the IBM soft-sectored format specifications.

Missing Data Address Mark Error. Set if the FDC cannot detect a data address mark or deleted data address mark on the specified track.

### Status Register 3

<table>
<thead>
<tr>
<th>BIT NUMBER</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>FT</td>
<td>Fault. This flag indicates the status of the fault signal from the selected disk drive.</td>
</tr>
<tr>
<td>6</td>
<td>WP</td>
<td>Write Protected. This flag indicates the status of the write protect signal from the selected disk drive.</td>
</tr>
<tr>
<td>5</td>
<td>RDY</td>
<td>Ready. This flag indicates the status of the ready signal from the selected disk drive.</td>
</tr>
<tr>
<td>4</td>
<td>TO</td>
<td>Track 0. This flag indicates the status of the track 0 signal from the selected disk drive.</td>
</tr>
<tr>
<td>3</td>
<td>TS</td>
<td>Two-Sided. This flag indicates the status of the two-sided signal from the selected disk drive.</td>
</tr>
<tr>
<td>2</td>
<td>H</td>
<td>Head Address. This flag indicates the status of the side select signal for the currently selected disk drive.</td>
</tr>
<tr>
<td>1,0</td>
<td>DSL,DSO</td>
<td>Drive Select. Indicates the currently selected disk drive number.</td>
</tr>
</tbody>
</table>
Execution Phase

All data transfers to (or from) the floppy drive occur during the execution phase. The 8272 has two primary modes of operation for data transfers (selected by the specify command):

1) DMA mode
2) non-DMA mode

In the DMA mode, execution phase data transfers are handled by the DMA controller hardware (invisible to the driver software). The driver software, however, must set all appropriate DMA controller registers prior to the beginning of the disk operation. An interrupt is generated by the 8272 after the last data transfer, indicating the completion of the execution phase, and the beginning of the result phase.

In the non-DMA mode, transfer requests are indicated by generation of an interrupt and by activation of the RQM flag (bit 7 in the Main Status Register). The interrupt signal can be used for interrupt-driven systems and RQM can be used for polled systems. The driver software must respond to the transfer request by reading data from, or writing data to, the FDC. After completing the last transfer, the 8272 generates an interrupt to indicate the beginning of the result phase. In the non-DMA mode, the processor must activate the "terminal count" (TC) signal to the FDC (normally by means of an I/O port) after the transfer request for the last data byte has been received (by the driver) and before the appropriate data byte has been read from (or written to) the FDC.

In either mode of operation (DMA or non-DMA), the execution phase ends when a "terminal count" signal is sensed by the FDC, when the last sector on a track (the EOT parameter - Table 4) has been read or written, or when an error occurs.

Multi-sector and Multi-track Transfers

During disk read/write transfers (Read Data, Write Data, Read Deleted Data, and Write Deleted Data), the FDC will continue to transfer data from sequential sectors until the TC input is sensed. In the DMA mode, the TC input is normally set by the DMA controller. In the non-DMA mode, the processor directly controls the FDC TC input as previously described. Once the TC input is received, the FDC stops requesting data transfers (from the system software or DMA controller). The FDC, however, continues to read data from, or write data to, the floppy disk until the end of the current disk sector. During a disk read operation, the data read from the disk (after reception of the TC input) is discarded, but the data CRC is checked for errors; during a disk write operation, the remainder of the sector is filled with all-zero bytes.

If the TC signal is not received before the last byte of the current sector has been transferred to/from the system, the FDC increments the sector number by one and initiates a read or write command for this new disk sector.
The FDC is also designed to operate in a multi-track mode for dual-sided disks. In the multi-track mode (specified by means of the MT flag in the command byte - Table 4) the FDC will automatically increment the head address (from 0 to 1) when the last sector (on the track under head 0) has been read or written. Reading or writing is then continued on the first sector (sector 1) of head 1.

Drive Status Polling

After the power-on reset, the 8272 automatically enters a drive status polling mode. If a change in drive status is detected (all drives are assumed to be "not ready" at power-on), an interrupt is generated. The 8272 continues this status polling between command executions (and between step pulses in the Seek command). In this manner, the 8272 automatically notifies the system software whenever a floppy disk is inserted, removed, or changed by the operator.

Command Details

During the command phase, the Main Status Register must be polled by the driver software before each byte is written into the Data Register. The DIO (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command may be written into the 8272. The beginning of the execution phase for any of these commands will cause DIO to be set high and RQM to be set low.

Operation of the FDC commands is described in detail in Application Note AP-116, "An Intelligent Data Base System Using the 8272."

Invalid Commands

If an invalid (undefined) command is sent to the FDC, the FDC will terminate the command. No interrupt is generated by the 8272 during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both set indicating to the processor that the 8272 is in the result phase and the contents of Status Register 0 must be read. When the processor reads Status Register 0 it will find an 80H code indicating that an invalid command was received. The driver software in Appendix B checks each requested command and will not issue an invalid command to the 8272.

A Sense Interrupt Status command must be sent after a Seek or Recalibrate interrupt; otherwise the FDC will consider the next command to be an invalid command. Also, when the last "hidden" interrupt has been serviced, further Sense Interrupt Status commands will result in invalid command codes.
4. 8272 Physical Interface Software

PL/M software driver listings for the 8272 FDC are contained in Appendix A. These drivers have been designed to operate in a DMA environment (as described in Application Note AP-116, "An Intelligent Data Base System Using the 8272"). In the following paragraphs, each driver procedure is described. (A description of the driver data base variables is given in Table 7.) In addition, the modifications necessary to reconfigure the drivers for operation in a polled environment are discussed.

INITIALIZE$DRIVERS

This initialization procedure must be called before any FDC operations are attempted. This module initializes the DRIVE$READY, DRIVE$STATUS$CHANGE, OPERATION$IN$PROGRESS, and OPERATION$COMPLETE arrays as well as the GLOBAL$DRIVE$NO variable.

EXECUTE$DOCB

This procedure contains the main 8272 driver control software and handles the execution of a complete FDC command. EXECUTE$DOCB is called with two parameters: a) a pointer to a disk operation control block and b) a pointer to a result status byte. The format of the disk operation control block is illustrated in Figure 2 and the result status codes are described in Table 8.

Before starting the command phase for the specified disk operation, the command is checked for validity and to determine whether the FDC is busy. (For an overlapped operation, if the FDC BUSY flag is set — in the Main Status Register — the command cannot be started; non-overlapped operations cannot be started if the FDC BUSY flag is set, if any drive is in the process of seeking/recalibrating, or if an operation is currently in progress on the specified drive.)

After these checks are made, interrupts are disabled in order to set the OPERATION$IN$PROGRESS flag, reset the OPERATION$COMPLETE flag, load a pointer to the current operation control block into the OPERATION$DOCB$PTR array and set GLOBAL$DRIVE$NO (if a non-overlapped operation is to be started).

At this point, parameters from the operation control block are output to the DMA controller and the FDC command phase is initiated. After completion of the command phase, a test is made to determine the type of result phase required for the current operation. If no result phase is needed, control is immediately returned to the calling program. If an immediate result phase is required, the result bytes are input from the FDC. Otherwise, the CPU waits until the OPERATION$COMPLETE flag is set (by the interrupt service procedure).

Finally, if an error is detected in the result status code (from the FDC), an FDC operation error is reported to the calling program.
### APPLICATIONS

#### Table 7: Driver Data Base

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRIVE$READY</td>
<td>A public array containing the current &quot;ready&quot; status of each drive.</td>
</tr>
<tr>
<td>DRIVE$STATUS$CHANGE</td>
<td>A public array containing a flag for each drive. The appropriate flag is set whenever the ready status of a drive changes.</td>
</tr>
<tr>
<td>OPERATION$DOCB$PTR</td>
<td>An internal array of pointers to the operation control block currently in progress for each drive.</td>
</tr>
<tr>
<td>OPERATION$IN$PROGRESS</td>
<td>An internal array used by the driver procedures to determine if a disk operation is in progress on a given drive.</td>
</tr>
<tr>
<td>OPERATION$COMPLETE</td>
<td>An internal array used by the driver procedures to determine when the execution phase of a disk operation is complete.</td>
</tr>
<tr>
<td>GLOBAL$DRIVE$NO</td>
<td>A data byte that records the current drive number for non-overlapped disk operations.</td>
</tr>
<tr>
<td>VALID$COMMAND</td>
<td>A constant flag array that indicates whether a specified FDC command code is valid.</td>
</tr>
<tr>
<td>COMMAND$LENGTH</td>
<td>A constant byte array specifying the number of command/parameter bytes to be transferred to the FDC during the command phase.</td>
</tr>
<tr>
<td>DRIVE$NO$PRESENT</td>
<td>A constant flag array that indicates whether a drive number is encoded into an FDC command.</td>
</tr>
<tr>
<td>OVERLAP$OPERATION</td>
<td>A constant flag array that indicates whether an FDC command can be overlapped with other commands.</td>
</tr>
<tr>
<td>NO$RESULT</td>
<td>A constant flag array that is used to determine when an FDC operation does not have a result phase.</td>
</tr>
<tr>
<td>IMMED$RESULT</td>
<td>A constant flag array that indicates that an FDC operation has a result phase beginning immediately after the command phase is complete.</td>
</tr>
<tr>
<td>POSSIBLE$ERROR</td>
<td>A constant flag array that indicates if an FDC operation should be checked for an error status indication during the result phase.</td>
</tr>
<tr>
<td>Address Offset</td>
<td>Disk Operation Control Block (DOCB)</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>0</td>
<td>DMA$OP</td>
</tr>
<tr>
<td>1</td>
<td>DMA$ADDR</td>
</tr>
<tr>
<td>3</td>
<td>DMA$ADDR$EXT</td>
</tr>
<tr>
<td>4</td>
<td>DMA$COUNT</td>
</tr>
<tr>
<td>6</td>
<td>DISK$COMMAND(0)</td>
</tr>
<tr>
<td>7</td>
<td>DISK$COMMAND(1)</td>
</tr>
<tr>
<td>8</td>
<td>DISK$COMMAND(2)</td>
</tr>
<tr>
<td>9</td>
<td>DISK$COMMAND(3)</td>
</tr>
<tr>
<td>10</td>
<td>DISK$COMMAND(4)</td>
</tr>
<tr>
<td>11</td>
<td>DISK$COMMAND(5)</td>
</tr>
<tr>
<td>12</td>
<td>DISK$COMMAND(6)</td>
</tr>
<tr>
<td>13</td>
<td>DISK$COMMAND(7)</td>
</tr>
<tr>
<td>14</td>
<td>DISK$COMMAND(8)</td>
</tr>
<tr>
<td>15</td>
<td>DISK$RESULT(0)</td>
</tr>
<tr>
<td>16</td>
<td>DISK$RESULT(1)</td>
</tr>
<tr>
<td>17</td>
<td>DISK$RESULT(2)</td>
</tr>
<tr>
<td>18</td>
<td>DISK$RESULT(3)</td>
</tr>
<tr>
<td>19</td>
<td>DISK$RESULT(4)</td>
</tr>
<tr>
<td>20</td>
<td>DISK$RESULT(5)</td>
</tr>
<tr>
<td>21</td>
<td>DISK$RESULT(6)</td>
</tr>
<tr>
<td>22</td>
<td>MISC</td>
</tr>
</tbody>
</table>

Figure 2. Disk Operation Control Block (DOCB) Format
### Table 8: EXECUTE$DOCB Return Status Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No errors. The specified operation was completed without error.</td>
</tr>
<tr>
<td>1</td>
<td>FDC busy. The requested operation cannot be started. This error occurs if an attempt is made to start an operation before the previous operation is completed.</td>
</tr>
<tr>
<td>2</td>
<td>FDC error. An error was detected by the FDC during the execution phase of a disk operation. Additional error information is contained in the result data portion of the disk operation control block (DOCB.DISK$RESULT) as described in the 8272 data sheet. This error occurs whenever the 8272 reports an execution phase error (e.g., missing address mark).</td>
</tr>
<tr>
<td>3</td>
<td>8272 command interface error. An 8272 interfacing error was detected during the command phase. This error occurs when the command phase of a disk operation cannot be successfully completed (e.g., incorrect setting of the DIO flag in the Main Status Register).</td>
</tr>
<tr>
<td>4</td>
<td>8272 result interface error. An 8272 interfacing error was detected during the result phase. This error occurs when the result phase of a disk operation cannot be successfully completed (e.g., incorrect setting of the DIO flag in the Main Status Register).</td>
</tr>
<tr>
<td>5</td>
<td>Invalid FDC Command.</td>
</tr>
</tbody>
</table>
FDCINT

This procedure performs all interrupt processing for the 8272 interface drivers. Basically, two types of interrupts are generated by the 8272: (a) an interrupt that signals the end of a command execution phase and the beginning of the result phase and (b) an interrupt that signals the completion of an overlapped operation or the occurrence of an unexpected event (e.g., change in the drive "ready" status).

An interrupt of type (a) is indicated when the FDC BUSY flag is set (in the Main Status Register). When a type (a) interrupt is sensed, the result bytes are read from the 8272 and placed in the result portion of the disk operation control block, the appropriate OPERATION$COMPLETE flag is set, and the OPERATION$IN$PROGRESS flag is reset.

When an interrupt of type (b) is indicated (FDC not busy), a sense interrupt status command is issued (to the FDC). The upper two bits of the result status register (Status Register Zero - ST0) are used to determine the cause of the interrupt. The following four cases are possible:

1) Operation Complete. An overlapped operation is complete. The drive number is found in the lower two bits of ST0. The ST0 data is transferred to the active operation control block, the OPERATION$COMPLETE flag is set, and the OPERATION$IN$PROGRESS flag is reset.

2) Abnormal Termination. A disk operation has abnormally terminated. The drive number is found in the lower two bits of ST0. The ST0 data is transferred to the active control block, the OPERATION$COMPLETE flag is set, and the OPERATION$IN$PROGRESS flag is reset.

3) Invalid Command. The execution of an invalid command (i.e., a sense interrupt command with no interrupt pending) has been attempted. This interrupt signals the successful completion of all interrupt processing.

4) Drive Status Change. A change has occurred in the "ready" status of a disk drive. The drive number is found in the lower two bits of ST0. The DRIVE$READY flag for this disk drive is set to the new drive "ready" status and the DRIVE$STATUS$CHANGE flag for the drive is also set. In addition, if a command is currently in progress, the ST0 data is transferred to the active control block, the OPERATION$COMPLETE flag is set, and the OPERATION$IN$PROGRESS flag is reset.

After processing a type (b) interrupt, additional sense interrupt status commands must be issued and processed until an "invalid command" result is returned from the FDC. This action guarantees that all "hidden" interrupts are serviced.

In addition to the major driver procedures described above, a number of support procedures are required. These support routines are briefly described in the following paragraphs.
APPLICATIONS

OUTPUT$CONTROLS$TO$DMA

This procedure outputs the DMA mode, the DMA address, and the DMA word count to the 8237 DMA controller. In addition, the upper four bits of the 20-bit DMA address are output to the address extension latch. Finally, the disk DMA channel is started.

OUTPUT$COMMAND$TO$FDC

This software module outputs a complete disk command to the 8272 FDC. The number of required command/parameter bytes is found in the COMMAND$LENGTH table. The appropriate bytes are output one at a time (by calls to OUTPUT$BYTE$TO$FDC) from the command portion of the disk operation control block.

INPUT$RESULT$FROM$FDC

This procedure is used to read result phase status information from the disk controller. At most, seven bytes are read. In order to read each byte, a call is made to INPUT$BYTE$FROM$FDC. When the last byte has been read, a check is made to insure that the FDC is no longer busy.

OUTPUT$BYTE$TO$FDC

This software is used to output a single command/parameter byte to the FDC. This procedure waits until the FDC is ready for a command byte and then outputs the byte to the FDC data port.

INPUT$BYTE$FROM$FDC

This procedure inputs a single result byte from the FDC. The software waits until the FDC is ready to transfer a result byte and then reads the byte from the FDC data port.

FDC$READY$FOR$COMMAND

This procedure assures that the FDC is ready to accept a command/parameter byte by performing the following three steps. First, a small time interval (more than 20 microseconds) is inserted to assure that the RQM flag has time to become valid (after the last byte transfer). Second, the master request flag (RQM) is polled until it is activated by the FDC. Finally, the DIO flag is checked to ensure that it is properly set for FDC input (from the processor).

FDC$READY$FOR$RESULT

The operation of this procedure is similar to the FDC$READY$FOR$COMMAND with the following exception. If the FDC BUSY flag (in the Main Status Register) is not set, the result phase is complete and no more data is available from the FDC. Otherwise, the procedure waits for the RQM flag and checks the DIO flag for FDC output (to the processor).
OPERATION$CLEAN$UP

This procedure is called after the execution of a disk operation that has no result phase. OPERATION$CLEAN$UP resets the OPERATION$IN$PROGRESS flag and the GLOBAL$DRIVE$NO variable if appropriate. This procedure is also called to clean up after some disk operation errors.

Modifications for Polling Operation

To operate in the polling mode, the following modifications should be made to the previous routines:

1. The OUTPUT$CONTROLS$TO$DMA routine should be deleted.

2. In EXECUTE$DOCB, immediately prior to WAIT$FOR$OP$COMPLETE, a polling loop should be inserted into the code. The loop should test the RQM flag (in the Main Status Register). When RQM is set, a data byte should be written to, or read from, the 8272. The buffer address may be computed from the base address contained in DOCB.DMA$ADDR and DOCB.DMA$ADDR$EXT. After the correct number of bytes have been transferred, an operation complete interrupt will be issued by the FDC. During data transfer in the non-DMA mode, the NON-DMA MODE flag (bit 5 of the Main Status Register) will be set. This flag will remain set for the complete execution phase. When the transfer is finished, the NON-DMA MODE flag is reset and the result phase interrupt is issued by the FDC.
5. 8272 Logical Interface Software

Appendix B of this Application Note contains a PL/M listing of an exerciser program for the 8272 drivers. This program illustrates the design of logical interface level procedures to specify disk parameters, recalibrate a drive, seek to a cylinder, format a disk, read data, and write data.

The exerciser program is written to operate a standard single-sided 8" floppy disk drive in either the single- or double-density recording mode. Only the eight parameters listed in Table 9 must be specified. All other parameters are derived from these 8 basic variables.

Each of these logical interface procedures is described in the following paragraphs (refer to the listing in Appendix B).

SPECIFY

This procedure sets the FDC signal timing so that the FDC will interface correctly to the attached disk drive. The SPECIFY procedure requires four parameters, the step rate (SRT), head load time (HLT), head unload time (HUT), and the non-DMA mode flag (ND). This procedure builds a disk operation control block (SPECIFY$DOCB) and passes the control block to the FDC driver module (EXECUTE$DOCB) for execution. (Note carefully the computation required to transform the step rate (SRT) into the correct 8272 parameter byte.)

RECALIBRATE

This procedure causes the floppy disk read/write head to retract to track 0. The RECALIBRATE procedure requires only one parameter – the drive number on which the recalibrate operation is to be performed. This procedure builds a disk operation control block (RECALIBRATES$DOCB) and passes the control block to the FDC driver for execution.

SEEK

This procedure causes the disk read/write head (on the selected drive) to move to the desired cylinder position. The SEEK procedure is called with three parameters: drive number (DRV), head/side number (HD), and cylinder number (CYL). This software module builds a disk operation control block (SEEK$DOCB) that is executed by the FDC driver.

FORMAT

The FORMAT procedure is designed to initialize a complete floppy disk so that sectors can subsequently be read and written by system and application programs. Three parameters must be supplied to this procedure: the drive number (DRV), the recording density (DENS), and the interleave factor (INTLVE). The FORMAT procedure generates a data block (FMTBLK) and a disk operation control block (FORMAT$DOCB) for each track on the floppy disk (normally 77).
### Table 9: Basic Disk Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DENSITY</td>
<td>The recording mode (FM or MFM).</td>
</tr>
<tr>
<td>FILLER$BYTE</td>
<td>The data byte to be written in all sectors during formatting.</td>
</tr>
<tr>
<td>TRACKS$PER$DISK</td>
<td>The number of cylinders on the floppy disk.</td>
</tr>
<tr>
<td>BYTES$PER$SECTOR</td>
<td>The number of bytes in each disk sector. The exerciser accepts 128, 256, and 512 in FM mode, and 256, 512, and 1024 in MFM mode.</td>
</tr>
<tr>
<td>INTERLEAVE</td>
<td>The sector interleave factor for each disk track.</td>
</tr>
<tr>
<td>STEP$RATE</td>
<td>The disk drive step rate (1-16 milliseconds).</td>
</tr>
<tr>
<td>HEAD$LOAD$TIME</td>
<td>The disk drive head load time (2-254 milliseconds).</td>
</tr>
<tr>
<td>HEAD$UNLOAD$TIME</td>
<td>The head unload time (16-240 milliseconds).</td>
</tr>
</tbody>
</table>
The format data block specifies the four sector ID field parameters (cylinder, head, sector, and bytes per sector) for each sector on the track. The sector numbers need not be sequential; the interleave factor (INTLVE parameter) is used to compute the logical to physical sector mapping.

After both the format data block and the operation control block are generated for a given cylinder, control is passed to the 8272 drivers for execution. After the format operation is complete, a SEEK to the next cylinder is performed, a new format table is generated, and another track formatting operation is executed by the drivers. This track formatting continues until all tracks on the diskette are formatted.

In some systems, bad tracks must also be specified when a disk is formatted. For these systems, the existing FORMAT procedure should be modified to format bad tracks with a cylinder number of OFFH.

**WRITE**

The WRITE procedure transfers a complete sector of data to the disk drive. Five parameters must be supplied to this software module: the drive number (DRV), the cylinder number (CYL), the head/side number (HD), the sector number (SEC) and the recording density (DENS). This procedure generates a disk operation control block (WRITE$DOCB) from these parameters and passes the control block to the 8272 driver for execution. When control returns to the calling program, the data has been transferred to disk.

**READ**

This procedure is identical to the WRITE procedure except the direction of data transfer is reversed. The READ procedure transfers a sector of data from the floppy disk to system memory.

**Coping With Errors**

In actual practice all logical disk interface routines would contain error processing mechanisms. (Errors have been ignored for the sake of simplicity in the exerciser programs listed in Appendix B.) A typical error recovery technique consists of a two-stage procedure. First, when an error is detected, a recalibrate operation is performed followed by a retry of the failed operation. This procedure forces the drive to seek directly to the requested cylinder (lowering the probability of a seek error) and attempts to perform the requested operation an additional time. Soft (temporary) errors caused by mechanical or electrical interference do not normally recur during the retry operation; hard errors (caused by media or drive failures), on the other hand, will continue to occur during retry operations. If, after a number of retries (approximately 10), the operation continues to fail, an error message is displayed to the system operator. This error message lists the drive number, type of operation, and failure status (from the FDC). It is the operator's responsibility to take additional action as required.
6. File Systems

The file system provides the disk I/O interface level most familiar to users of interactive microcomputer and minicomputer systems. In a file system, all data is stored in named disk areas called files. The user and applications programs need not be concerned with the exact location of a file on the disk - the disk file system automatically determines the file location from the file name. Files may be created, read, written, modified, and finally deleted (destroyed) when they are no longer needed. Each floppy disk typically contains a directory that lists all the files existing on the disk. A directory entry for a file contains information such as file name, file size, and the disk address (track and sector) of the beginning of the file.

File Allocation

File storage is actually allocated on the disk (by the file system) in fixed size areas called blocks. Normally a block is the same size as a disk sector. Files are created by finding and reserving enough unused blocks to contain the data in the file. Two file allocation methods are currently in widespread use. The first method allocates blocks (for a file) from a sequential pool of unused blocks. Thus, a file is always contained in a set of sequential blocks on the disk. Unfortunately, as files are created, updated, and deleted, these free-block pools become fragmented (separated from one another). When this fragmentation occurs, it often becomes impossible for the file system to create a file even though there is a sufficient number of free blocks on the disk. At this point, special programs must be run to "squeeze" or compact the disk, in order to re-create a single contiguous free-block pool.

The second file allocation method uses a more flexible technique in which individual data blocks may be located anywhere on the disk (with no restrictions). With this technique, a file directory entry contains the disk address of a file pointer block rather than the disk address of the first data block of the file. This file pointer block contains pointers (disk addresses) for each data block in the file. For example, the first pointer in the file pointer block contains the track and sector address of the first data block in the file, the second pointer contains the disk address of the second data block, etc.

In practice, pointer blocks are usually the same size as data blocks. Therefore, some files will require multiple pointer blocks. To accommodate this requirement without loss of flexibility, pointer blocks are linked together, that is, each pointer block contains the disk address of the following pointer block. The last pointer block of the file is signalled by an illegal disk address (e.g., track 0, sector 0 or track OFFH, sector OFFH).
The Intel File System

The Intel file system (described in detail in the RMX-80 Users Guide) uses the second disk file allocation method (previously discussed). In order to lower the system overhead involved in finding free data blocks, the Intel file system incorporates a free space management data structure known as a bit map. Each disk sector is represented by a single bit in the bit map. If a bit in the bit map is set to 1, the corresponding disk sector has been allocated. A zero in the bit map indicates that the corresponding sector is free. With this technique, the process of allocating or freeing a sector is accomplished by simply altering the bit map.

File names consist of a basic file name (up to six characters) and a file extension (up to three characters). The basic file name and the file extension are separated by a period (.). Examples of valid file names are: DRIV72.OBJ, XX.TMP, and FILE.CS. In addition, four file attributes are supported (see Figure 3 for attribute definitions).

The bit map and the file directory are placed on prespecified disk tracks (reserved for system use) beginning at track zero.

Disk File System Functions

Table 2 illustrates the typical functions implemented by a disk file system. As an example, the disk directory function (DIR) lists disk file information on the console display terminal. Figure 3 details the contents of a display entry in the Intel file system. The PL/M procedure outlined in Figure 4 illustrates a disk directory algorithm that displays the file name, the file attributes, and the file size (in blocks) for each file in the directory.
Directory Entry

Presence is a flag that can contain one of three values:

- **000H** - The file associated with this entry is present on the disk.
- **07FH** - No file is associated with this entry; the content of the rest of the entry is undefined. The first entry with its flag set to 07FH marks the current logical end of the directory and directory searches stop at this entry.
- **0FFH** - The file named in this entry once existed on the disk but is currently deleted. The next file added to the directory will be placed in the first entry marked OFFH. This flag cannot, therefore, be used to (reliably) find a file that has been deleted. A value of OFFH should be thought of as simply marking an open directory entry.

FileName is a string of up to 6 non-blank ASCII characters specifying the name of the file associated with the directory entry. If the file name is shorter than six characters, the remaining bytes contain binary zeros. For example, the name ALPHA would be stored as: 414C50484100H.

Extension is a string of up to 3 non-blank ASCII characters that specifies an extension to the file name. Extensions often identify the type of data in the file such as OBJ (object module), or PLM (PL/M source module). As with the file name, unused positions in the extension field are filled with binary zeros.

Figure 3. Intel Directory Entry Format
**APPLICATIONS**

Attributes are bits that identify certain characteristics of the file. A 1 bit indicates that the file has the attribute, while a 0 bit means that the file does not have the attribute. The bit positions and their corresponding attributes are listed below (bit 0 is the low-order or rightmost bit, bit 7 is the leftmost bit):

0: Invisible. Files with this attribute are not listed by the ISIS-II DIR command unless the I switch is used. All system files are invisible.

1: System. Files with this attribute are copied to the disk in drive 1 when the S switch is specified with the ISIS-II FORMAT command.

2: Write-protect. Files with this attribute cannot be opened for output or update, nor can they be deleted or renamed.

3-6: These positions are reserved for future use.

7: Format. Files with this attribute are treated as though they are write-protected. In addition, these files are created on a new diskette when the ISIS-II FORMAT command is issued. The system files all have the FORMAT attribute and it should not be given to any other files.

**EOF Count** contains the number of the last byte in the last data block of the file. If the value of this field is 080H, for example, the last byte in the file is byte number 128 in the last data block (the last block is full).

**Number of Data Blocks** is an address variable that indicates the number of data blocks currently used by the file. ISIS-II and the RMX/80 Disk File system both maintain a counter called LENGTH that is the current number of bytes in the file. This is calculated as:

\[
\text{LENGTH} = (\text{NUMBER OF DATA BLOCKS} - 1) \times 128 + \text{EOF COUNT}. 
\]

**Header Block Pointer** is the address of the file's header block. The high byte of the field is the sector number and the low byte is the track number. The system "finds" a disk file by searching the directory for the name and then using the header block pointer to seek to the beginning of the file.

Figure 3. Intel Directory Entry Format (Continued)
dir: procedure(drv,dens) public;
declare drv
byte,
dens
byte,
sector
byte,
i
byte,
dir$ptr
byte,
dir$entry based rdbptr structure (presence byte,
file$name(6) byte,extension(3) byte,
attribute byte,eof$count byte,
data$blocks address,header$ptr address),
size (5)
byte,
invisible$flag literally '1',
system$flag literally '2',
protected$flag literally '4',
format$flag literally '80H';
/* The disk directory starts at cylinder 1, sector 2 */
call seek(drv,l,0);
do sector=2 to 26;
call read(drv,l,0,sector,dens);
do dir$ptr=0 to 112 by 4;
   if dir$entry.presence=7FH then return;
   if dir$entry.presence=0 then do;
      do i=0 to 5; call co(dir$entry.file$name(i)); end;
call co(period);
      do i=0 to 2; call co(dir$entry.extension(i)); end;
      do i=0 to 4; call co(space); end;
      call convert$to$decimal(@size,dir$entry.data$blocks);
      do i=0 to 4; call co(size(i)); end;
      if (dir$entry.attribute and invisible$flag) <> 0 then call co('I');
      if (dir$entry.attribute and system$flag) <> 0 then call co('S');
      if (dir$entry.attribute and protected$flag) <> 0 then call co('W');
      if (dir$entry.attribute and format$flag) <> 0 then call co('F');
      end;
end;
end dir;

Figure 4. Sample PL/M Directory Procedure
7. Key 8272 Software Interfacing Considerations

This section contains a quick review of Key 8272 Software design features and issues. (Most items have been mentioned in other sections of this application note.) Before designing 8272 software drivers, it is advisable that the information in this section be thoroughly understood.

1. Non-DMA Data Transfers

In systems that operate without a DMA controller (in the polled or interrupt driven mode), the system software is responsible for counting data transfers to/from the 8272 and generating a TC signal to the FDC when the transfer is complete.

2. Processor Command/Result Phase Interface

In the command phase, the driver software must write the exact number of parameters in the exact order shown in Table 5. During the result phase, the driver must read the complete result status. For example, the Format Track command requires six command bytes and presents seven result bytes. The 8272 will not accept a new command until all result bytes are read. Note that the number of command and result bytes varies from command-to-command. Command and result phases cannot be shortened.

During both the command and result phases, the Main Status Register must be read by the driver before each byte of information is read from, or written to, the FDC Data Register. Before each command byte is written, DIO (bit 6) must be low (indicating a data transfer from the processor) and RQM (bit 7) must be high (indicating that the FDC is ready for data). During the result phase, DIO must be high (indicating a data transfer to the processor) and RQM must also be high (indicating that data is ready for the processor).

Note: After the 8272 receives a command byte, the RQM flag may remain set for approximately 16 microseconds (with an 8 MHz clock). The driver should not attempt to read the Main Status Register before this time interval has elapsed; otherwise, the driver may erroneously assume that the FDC is ready to accept the next byte.

3. Sector Sizes

The 8272 does not support 128 byte sectors in the MFM (double-density) mode.

4. Drive Status Changes

The 8272 constantly polls all drives for changes in the drive ready status. This polling begins immediately following RESET. An interrupt is generated every time the FDC senses a change in the drive ready status. After reset, the FDC assumes that all drives are "not ready". If a drive is ready immediately after reset, the 8272 generates a drive status change interrupt.
5. Seek Commands

The 8272 FDC does not perform implied seeks. Before issuing a data read or write command, the read/write head must be positioned over the correct cylinder by means of an explicit seek command. If the head is not positioned correctly, a cylinder address error is generated.

6. Interrupt Processing

When the processor receives an interrupt from the FDC, the FDC may be reporting one of two distinct events:

a) The beginning of the result phase of a previously requested read, write, or scan command.

b) An asynchronous event such as a seek/recalibrate completion, an attention, an abnormal command termination, or an invalid command.

These two cases are distinguished by the FDC BUSY flag (bit 4) in the Main Status Register. If the FDC BUSY flag is high, the interrupt is of type (a). If the FDC BUSY flag is low, the interrupt was caused by an asynchronous event (b).

A single interrupt from the FDC may signal more than one of the above events. After receiving an interrupt, the processor must continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are ferreted out and serviced.

7. Skip Flag (SK)

The skip flag is used during the execution of Read Data, Read Deleted Data, Read Track, and various Scan commands. This flag permits the FDC to skip unwanted sectors on a disk track.

When performing a Read Data, Read Track, or Scan command, a high SK flag indicates that the FDC is to skip over (not transfer) any sector containing a deleted data address mark. A low SK flag indicates that the FDC is to terminate the command (after reading all the data in the sector) when a deleted data address mark is encountered.

When performing a Read Deleted Data command, a high SK flag indicates that sectors containing normal data address marks are to be skipped. Note that this is just the opposite situation from that described in the last paragraph. When a data address mark is encountered during a Read Deleted Data command (and the SK flag is low), the FDC terminates the command after reading all the data in the sector.
8. Bad Track Maintenance

The 8272 does not internally maintain bad track information. The maintenance of this information must be performed by system software. As an example of typical bad track operation, assume that a media test determines that track 31 and track 66 of a given floppy disk are bad. When the disk is formatted for use, the system software formats physical track 0 as logical cylinder 0 (C=0 in the command phase parameters), physical track 1 as logical track 1 (C=1), and so on, until physical track 30 is formatted as logical cylinder 30 (C=30). Physical track 31 is bad and should be formatted as logical cylinder FF (indicating a bad track). Next, physical track 32 is formatted as logical cylinder 31, and so on, until physical track 65 is formatted as logical cylinder 64. Next, bad physical track 66 is formatted as logical cylinder FF (another bad track marker), and physical track 67 is formatted as logical cylinder 65. This formatting continues until the last physical track (77) is formatted as logical cylinder 75. Normally, after this formatting is complete, the bad track information is stored in a prespecified area on the floppy disk (typically in a sector on track 0) so that the system will be able to recreate the bad track information when the disk is removed from the drive and reinserted at some later time.

To illustrate how the system software performs a transfer operation on a disk with bad tracks, assume that the disk drive head is positioned at track 0 and the disk described above is loaded into the drive. If a command to read track 36 is issued by an application program, the system software translates this read command into a seek to physical track 37 (since there is one bad track between 0 and 36, namely 31) followed by a read of logical cylinder 36. Thus, the cylinder parameter C is set to 37 for the Seek command and 36 for the Read Sector command.
REFERENCES


APPENDIX A
8272 FDC DEVICE DRIVER SOFTWARE
APPLICATIONS

PL/M-86 COMPILER 8272 FLOPPY DISK CONTROLLER DEVICE DRIVERS

ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE DRIVERS
OBJECT MODULE PLACED IN :Pl:driv72.obj
COMPILER INVOKED BY: plm86 :Pl:driv72.p86 DEBUG

$TITLE('8272 floppy disk controller device drivers')
$nointvector
$optimize(2)
$large

1

drivers: do;

2 1 declare
   /* floppy disk port definitions */
   fdc$status$port literally '10H', /* 8272 status port */
   fdc$data$port literally '31H'; /* 8272 data port */

3 1 declare
   /* floppy disk commands */
   sense$int$status literally '08H';

4 1 declare
   /* interrupt definitions */
   fdc$int$level literally '33'; /* fdc interrupt level */

5 1 declare
   /* return status and error codes */
   error literally '0',
   ok literally '1',
   complete literally '3',
   false literally '0',
   true literally '1',
   error$in propagate$error literally 'not',
   propagate$error literally 'return error',
   stat$ok literally '0', /* fdc operation completed without errors */
   stat$busy literally '1', /* fdc is busy, operation cannot be started */
   stat$error literally '2', /* fdc operation error */
   stat$result$error literally '3', /* fdc not ready for result phase */
   stat$invalid literally '5', /* invalid fdc command */

6 1 declare
   /* masks */
   busy$mask literally '10H',
   DIO$mask literally '40H',
   RQM$mask literally '80H',
   seek$mask literally 'OFH',
   result$error$mask literally 'OCOH',
   result$drive$mask literally 'O3H',
   result$ready$mask literally '08H';

7 1 declare
   /* drive numbers */
   max$no$drives literally '3',
   fdc$general literally '4';

8 1 declare
   /* miscellaneous control */
   any$drive$seeking literally '((input(fdc$status$port) and seek$mask) <> 0)',
   command$code literally '((docb.disk$command(0) and 1FH)',
   DIO$set$for$input literally '((input(fdc$status$port) and DIO$mask) = 0)',
   DIO$set$for$output literally '((input(fdc$status$port) and DIO$mask) <> 0)',
   extract$drive$no literally '((docb.disk$command(1) and 03H)',
   fdc$busy literally '((input(fdc$status$port) and busy$mask) <> 0)',
   no$fdc$error literally 'possible$error(command$code) and ((docb.disk$result(0)
   and result$error$mask) = 0)',
   wait$for$op$complete literally 'do while not operation$complete(drive$no); end',
   wait$for$RQM literally 'do while (input(fdc$status$port) and RQM$mask) = 0; end';

9 1 declare
   /* structures */
   docb$type literally /* disk operation control block */
   'dma$op byte,dma$addr word, dma$addr$ext byte,dma$account word,
   disk$command(9) byte,disk$result(7) byte,misc byte');

$seect

10 1 declare
    drive$status$change(4) byte public,
    drive$ready(4) byte public;
    /* when set - indicates that drive status changed */
    /* current status of drives */
11 1 declare
  operation$in$progress(5) byte,
  operation$complete(5) byte,
  operation$docb$ptr(5) pointer,
  interrupt$docb$structure docb$stype,
  global$drive$no byte;
/* internal flags for operation with multiple drives */
/* fdc execution phase completed */
/* pointers for operations in progress */
/* temporary docb for interrupt processing */
/* drive number of non-overlapped operation in progress - if any */

12 1 declare
/* internal vectors that contain command operational information */
  no$result(32) byte /* no result phase to command */
  data(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
  immed$result(32) byte /* immediate result phase for command */
  data(0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
  overlap$operation(32) byte /* command permits overlapped operation of drives */
  data(0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
  drive$no$present(32) byte /* drive number present in command information */
  data(0,0,0,0,0,0,1,0,1,1,1,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0)
  possible$error(32) byte /* determines if command can return with an error */
  data(0,0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
  command$length(32) byte /* contains number of command bytes for each command */
  data(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
  valid$command(32) byte /* flags invalid command codes */
  data(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
$eject

13 1 /***/
14 1 initialize$drivers: procedure public;
15 2 /* initialize 8272 drivers */
16 2 declare drv$no byte;
18 2 do drv$no=0 to max$no$drives:
19 3 drive$ready(drv$no)=false;
20 3 drive$status$change(drv$no)=false;
21 3 operation$in$progress(drv$no)=false;
22 3 operation$complete(drv$no)=false;
23 3 end;
24 2 end initialize$drivers;

25 1 fdc$ready$for$command: procedure byte;
26 2 /* wait for valid flag settings in status register */
27 2 call time(1);
28 2 /* wait for "master request" flag */
29 2 wait$for$ROM;
30 2 /* check data direction flag */
31 2 if DIO$set$for$input then return ok;
32 2 else return error;
33 2 end fdc$ready$for$command;

34 1 fdc$ready$for$result: procedure byte;
35 2 /* wait for valid settings in status register */
36 2 call time(1);
37 2 /* result phase has ended when the 8272 busy flag is reset */
38 2 if not fdc$busy then return complete;
APPLICATIONS

/* wait for "master request" flag */
wait$for$RQM;

/* check data direction flag */
if DIO$set$for$output
then return ok;
else return error;
end fdc$ready$for$result;

=====

output$byte$to$fdc: procedure (data$byte)

declare data$byte

/* check to see if fdc is ready for command */
if not fdc$readv$for$command
then propagate$error;
output (fdc$data$port) =data$byte;
return ok;
end output$byte$to$fdc;

=====

input$byte$from$fdc: procedure (data$byte$ptr)

declare data$byte$ptr
declare

/* check to see if fdc is ready */
status=fdc$readySfor$result:
if error$in status
then propagate$error,
/* check for result phase complete */
if status=complete
then return
data$byte=input(fdc$data$port);
return ok;
end
input$byte$from$fdc;

$eject

=====

output$contro1s$to$dma: procedure (docb$ptr)

declare docb$ptr
declare docb

/* set dma mode and clear first/last flip-flop */
output (dma$mode$port) =sh1 (docb.dma$op, 2) or
output (dma$c1ear$ff$port)=0;

if docb.dma$op < 3
then do;
/* set dma mode and clear first/last flip-flop */
output (dma$mode$port)=shl(docb.dma$op, 2) or 40H;
output (dma$clear$ff$port)=0;
APPLICATIONS

/* set dma address */
output(dma$disk$addr$port)=low(docb.dma$addr);
output(dma$disk$addr$port)=high(docb.dma$addr);
output(dma$upper$addr$port)=docb.dma$addr$ext;

/* output disk transfer word count to dma controller */
output(dma$disk$word$count)=low(docb.dma$count);
output(dma$disk$word$count)=high(docb.dma$count);

/* start dma channel 0 for fdc */
output(dma$mask$sr$port)=dma$disk$chan$start;
end;

end output$controls$to$dma;

/**** output a high-level disk command to the 8272 fdc. The number of bytes
required for each command is contained in the "command$length" table.
The "docb$ptr" parameter is a pointer to the appropriate disk operation
control block. ****/
output$command$to$fdc: procedure(docb$ptr) byte;
declare docb$ptr pointer;
declare docb based docb$ptr structure docb$type,
   cmd$byte$no byte;
disable;
/* output all command bytes to the fdc */
do cmd$byte$no=0 to command$length(command$code)-l:
   if error$in output$byte$to$fdc(docb.disk$command(cmd$byte$no))
      then do; enable; propagate$error; end;
end;
enable;
return ok;
end output$command$to$fdc;

/**** input the result data from the 8272 fdc during the result phase (after
command execution). The "docb$ptr" parameter is a pointer to the
appropriate disk operation control block. ****/
input$result$from$fdc: procedure(docb$ptr) byte;
declare docb$ptr pointer;
declare docb based docb$ptr structure docb$type,
   result$byte$no byte,
   temp byte,
   status byte;
disable:
do result$byte$no=0 to 7:
   status=input$byte$from$fdc(@temp);
   if error$in status
      then do; enable; propagate$error; end;
   if status=complete
      then do; enable; return ok; end;
   docb.disk$result(result$byte$no)=temp;
end;
enable;
if fdc$busy
   then return error;
else return ok;
end input$result$from$fdc;

/**** cleans up after the execution of a disk operation that has no result
phase. The procedure is also used after some disk operation errors.
"drv" is the drive number, and "cc" is the command code for the
disk operation. ****/
operation$clean$up: procedure(drv,cc);
declare (drv,cc) byte;
disable;
operation$in$progress(drv)=false;
/* execute the disk operation control block specified by the pointer
   parameter "docb$ptr". The "status$ptr" parameter is a pointer to
   a byte variable that is to contain the status of the requested
   operation when it has been completed. Six status conditions are
   possible on return:
   0 The specified operation was completed without error.
   1 The fdc is busy and the requested operation cannot be started.
   2 Fdc error (further information is contained in the result
      storage portion of the disk operation control block - as
      described in the 8272 data sheet).
   3 Transfer error during output of the command bytes to the fdc.
   4 Transfer error during input of the result bytes from the fdc.
   5 Invalid fdc command. */

execute$docb: procedure(docb$ptr, status$ptr) public;
/* execute a disk operation control block */
declare docb$ptr pointer, status$ptr pointer;
declare docb based docb$ptr, structure docb$type,
drive$no byte,
status based status$ptr byte,
/* check command validity */
if not valid$command(command$code)
   then do; status=stat$invalid; return; end;
/* determine if command has a drive number field - if not, set the drive
   number for a general fdc command */
if drive$no$present(command$code)
   then drive$no=extract$drive$no:
   else drive$no=fdc$general;
/* an overlapped operation can not be performed if the fdc is busy */
if overlap$operation(command$code) and fdc$busy
   then do; status=stat$busy; return; end;
/* for a non-overlapped operation, check fdc busy or any drive seeking */
if not overlap$operation(command$code) and (fdc$busy or any$drive$seeking)
   then do; status=stat$busy; return; end;
/* check for drive operation in progress - if none, set flag and start operation */
disable;
if operation$in$progress(drive$no)
   then do; enable; status=stat$busy; return; end;
else operation$in$progress(drive$no)=true;
/* at this point, an fdc operation is about to begin, so:
   1. reset the operation complete flag
   2. set the docb pointer for the current operation
   3. if this is not an overlapped operation, set the global drive
      number for the subsequent result phase interrupt. */
operation$complete(drive$no)=0;
operation$docb$ptr(drive$no)=docb$ptr;
if not overlap$operation(command$code)
   then global$drive$no=drive$no+1;
enable;
call output$controls$to$dma(docb$ptr);
if error$in output$command$to$fdc(docb$ptr)
   then do;
      call operation$clean$up(drive$no, command$code);
      status=stat$command$error;
      return;
end;
/* return immediately if the command has no result phase or completion interrupt - specify */
if no$result(command$code)
   then do;
call operation$clean$up(drive$no, command$code);
   status=stat$ok;
   return;
end;
APPLICATIONS

171 2 if immed$result(command$code)
then do;
173 3 if error$in input$result$from$fdc(docb$ptr)
then do;
175 4 call operation$clean$up(drive$no,command$code);
176 4 status=stat$result$error;
177 4 return;
178 4 end;
179 3 end;
180 2 else do;
181 3 wait$for$op$complete;
183 3 if docb.misc = error
then do; status=stat$result$error; return; end;
188 3 end;
189 2 if no$fdc$error
then status=stat$ok;
191 2 else status=stat$error;
192 2 end execute$docb;

\$eject

/**** copy disk command results from the interrupt control block to the 
currently active disk operation control block if a disk operation is 
in progress. ****/

193 1 copy$int$result: procedure(drv);
194 2 declare drv byte;
195 2 declare 
i byte,
   docb$ptr pointer,
   docb based docb$ptr structure docb$type;
196 2 if operation$in$progress(drv)
then do;
198 3 docb$ptr=operation$docb$ptr(drv);
199 3 do i=1 to 6; docb.disk$result(i)=interrupt$docb.disk$result(i); end;
202 3 docb.misc=ok;
203 3 operation$in$progress(drv)=false;
204 3 operation$complete(drv)=true;
205 3 end;
206 2 end copy$int$result;

/**** interrupt processing for 8272 fdc drivers. Basically, two types of 
interrupts are generated by the 8272: (a) when the execution phase of 
an operation has been completed, an interrupt is generated to signal 
the beginning of the result phase (the fdc busy flag is set 
when this interrupt is received), and (b) when an overlapped operation 
is completed or an unexpected interrupt is received (the fdc busy flag 
is not set when this interrupt is received).

When interrupt type (a) is received, the result bytes from the operation 
are read from the 8272 and the operation complete flag is set.

When an interrupt of type (b) is received, the interrupt result code is 
examined to determine which of the following four actions are indicated:

1. An overlapped option (recalibrate or seek) has been completed. The 
   result data is read from the 8272 and placed in the currently active 
disk operation control block.
2. An abnormal termination of an operation has occurred. The result 
data is read and placed in the currently active disk operation 
control block.
3. The execution of an invalid command has been attempted. This 
signals the successful completion of all interrupt processing.
4. The ready status of a drive has changed. The "drive$ready" and 
   "drive$ready$status" change tables are updated. If an operation 
is currently in progress on the affected drive, the result data 
is placed in the currently active disk operation control block.

After an interrupt is processed, additional sense interrupt status commands 
must be issued and processed until an invalid command result is returned 
from the fdc. This action guarantees that all "hidden" interrupts 
are serviced. ****/
APPLICATIONS

207 1 fdcint: procedure public interrupt fdc$int$level;
208 2 declare
209 2 invalid byte,
210 2 drive$no byte,
211 2 docb$ptr pointer,
212 2 docb based docb$ptr structure docb$type;
213 2 declare
214 2 /* interrupt port definitions */
215 2 ocw2 literally '70H',
216 2 nseoi literally 'shl(1,5)';
217 2 declare
218 2 /* miscellaneous flags */
219 2 result$code literally 'shr(interrupt$docb.disk$result(0) and result$error$mask,6)',
220 2 result$drive$ready literally '((interrupt$docb.disk$result(0) and result$ready$mask) = 0)',
221 2 extract$result$drive$no literally '(interrupt$docb.disk$result(0) and result$drive$mask)',
222 2 end$of$interrupt literally 'output(ocw2)=nseoi';
223 2 /* if the fdc is busy when an interrupt is received, then the result
224 2 phase of the previous non-overlapped operation has begun */
225 2 if fdc$busy
226 2 then do;
227 2 /* process interrupt if operation in progress */
228 2 if global$drive$no <> 0
229 2 then do;
230 2 docb$ptr=operation$docb$ptr(global$drive$no-l);
231 2 if error$in input$result$fromfdc(docb$ptr)
232 2 then docb.misc=error;
233 2 else docb.misc=ok;
234 2 operation$in$progress(global$drive$no-l)=false;
235 2 operation$complete (global$drive$no-l) =true;
236 2 global$drive$no=O;
237 2 end;
238 2 /* if the fdc is not busy, then either an overlapped operation has been
239 2 completed or an unexpected interrupt has occurred (e.g., drive status
240 2 change) */
241 2 else do;
242 2 invalid=false;
243 2 do while not invalid;
244 2 /* perform a sense interrupt status operation - if errors are detected,
245 2 in the actual fdc interface, interrupt processing is discontinued */
246 2 if error$in output$byte$tofdc(sense$int$status) then go to ignore;
247 2 if error$in input$result$fromfdc(@interrupt$docb) then go to ignore;
248 2 do case result$code:
249 2 /* case 0 - operation complete */
250 2 do;
251 2 drive$no=extract$result$drive$no;
252 2 call copy$int$result(drive$no);
253 2 end;
254 2 /* case 1 - abnormal termination */
255 2 do;
256 2 drive$no=extract$result$drive$no;
257 2 call copy$int$result(drive$no);
258 2 end;
259 2 /* case 2 - invalid command */
260 2 invalid=true;
261 2 /* case 3 - drive ready change */
262 2 do;
263 2 drive$no=extract$result$drive$no;
264 2 call copy$int$result(drive$no);
265 2 drive$status$change(drive$no)=true;
266 2 if result$drive$ready
267 2 then drive$ready(drive$no)=true;
268 2 else drive$ready(drive$no)=false;
269 2 end;
270 2 end;
271 2 end:
272 2 ignore: end$of$interrupt;
273 2 end fdcint;
274 2 end drivers;
APPLICATIONS

MODULE INFORMATION:

<table>
<thead>
<tr>
<th>CODE AREA SIZE</th>
<th>0615H</th>
<th>1557D</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT AREA SIZE</td>
<td>0000H</td>
<td>0D</td>
</tr>
<tr>
<td>VARIABLE AREA SIZE</td>
<td>0050H</td>
<td>80D</td>
</tr>
<tr>
<td>MAXIMUM STACK SIZE</td>
<td>0032H</td>
<td>50D</td>
</tr>
</tbody>
</table>

564 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
APPENDIX B
8272 FDC EXERCISER PROGRAM
 PL/M-86 COMPILER 8272 FLOPPY DISK DRIVER EXERCISE PROGRAM

ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE RUN72
OBJECT MODULE PLACED IN :Fl:run72.obj
COMPILER INVOKED BY: plm86 :Fl:run72.p86 DEBUG

$title ("8272 floppy disk driver exercise program")
$noinvector
$optimize(2)
$large
1 run72: do;
2 1 declare
docb$type literally /* disk operation control block */
'(dma$op byte,dma$addr word,dma$addr$ext byte,dma$cnt word,
disk$cmd(9) byte,disk$result(7) byte,misc byte)';
3 1 declare
'/* 8272 FDC commands */
fmode literally '0',
mfmode literally '1',
dma$mode literally '0',
non$dma$mode literally '1',
recalibrate$command literally '2',
specify$command literally '3',
read$command literally '4',
write$command literally '5',
format$command literally '0DH',
seek$command literally '0FH';
4 1 declare
dma$address literally '0',
dma$read literally '1',
dma$write literally '2',
dma$noop literally '3';
5 1 declare
'/* disk operation control blocks */
format$docb structure docb$type,
seek$docb structure docb$type,
recalibrate$docb structure docb$type,
specify$docb structure docb$type,
read$docb structure docb$type,
write$docb structure docb$type;
6 1 declare
step$rate byte,
head$load$time byte,
head$unload$time byte,
filler$byte byte,
operation$status byte,
interleave byte,
format$gap byte,
read$write$gap byte,
index byte,
drive byte,
density byte,
multitrack byte,
sector byte,
cylinder byte,
head byte,
tracks$per$disk byte,
sectors$per$track byte,
bytes$per$sector$code byte,
bytes$per$sector word; /* disk drive head */
7 1 declare
'/* read and write buffers */
fmbk(104) byte public,
wrbuf(1024) byte public,
rdbuf(1024) byte public;
8 1 declare
'/* disk format initialization tables */
sectr$k$tab(3) byte data(26,15,8),
dg$gap$tab(8) byte (18H,2AH,3AH,0,0,36H,54H,74H),
r$wr$gap$tab(8) byte data(07H,0EH,1BH,0,0,0EH,1BH,35H);
9 1 declare
    /* external pointer tables and interrupt vector */
    rdbptr(2) word external,
    wrbptr(2) word external,
    fbptr(2) word external,
    intptr(2) word external,
    intvec(00H) word external;

10 1 execute$docb: procedure(docb$ptr, status$ptr) external;
11 2 declare docb$ptr pointer, status$ptr pointer;
12 2 end execute$docb;
13 1 initialize$drivers: procedure external;
14 2 end initialize$drivers;

15 1 /* external pointer tables and interrupt vector */
16 2 execute$docb: procedure(docb$ptr, status$ptr) external;
17 2 declare docb$ptr pointer, status$ptr pointer;
18 2 end execute$docb;
19 1 initialize$drivers: procedure external;
20 2 end initialize$drivers;

23 1 /* external pointer tables and interrupt vector */
24 2 execute$docb: procedure(docb$ptr, status$ptr) external;
25 2 declare docb$ptr pointer, status$ptr pointer;
26 2 end execute$docb;
27 1 initialize$drivers: procedure external;
28 2 end initialize$drivers;

23 1 execute$docb: procedure(docb$ptr, status$ptr) external;
24 2 declare docb$ptr pointer, status$ptr pointer;
25 2 end execute$docb;
26 1 initialize$drivers: procedure external;
27 2 end initialize$drivers;

23 1 execute$docb: procedure(docb$ptr, status$ptr) external;
24 2 declare docb$ptr pointer, status$ptr pointer;
25 2 end execute$docb;
26 1 initialize$drivers: procedure external;
27 2 end initialize$drivers;

30 1 seek: procedure(drv, cyl, hd);
31 2 declare (drv, cyl, hd) byte;
32 2 seek$docb.dma$op=dma$noop;
33 2 seek$docb.disk$command(0)=seek$command;
34 2 seek$docb.disk$command(1)=drv or shl(hd,2);
35 2 seek$docb.disk$command(2)=cyl;
36 2 call execute$docb(@seek$docb, @operation$status);
37 2 end seek;

38 1 format: procedure(drv, dens, intlve);
39 2 /* format disk */
40 2 declare (drv, dens, intlve) byte;
41 2 call recalibrate(drv);
42 2 do cylinder=0 to tracks$per$disk-1;
43 3 /* set sector numbers in format block to zero before computing interleave */
44 3 do physical$sector=0 to sectors$per$track;
45 4 fmtblk((physical$sector-1)*4+2)=0; end;
46 3 physical$sector=1;
47 3 /* assign interleaved sectors */
48 4 do sector=1 to sectors$per$track;
49 5 index=(physical$sector-1)*4;
/* change sector and index if sector has already been assigned */
do while fmtblk(index+2) <> 0; index=index+4; physical$sector=physical$sector+l; end;

/* set cylinder, head, sector, and size code for current sector into table */
fmbblk(index)=cylinder;
fmbblk(index+l)=head;
fmbblk(index+2)=sector;
fmbblk(index+3)=bytes$per$sector$code;

/* update physical sector number by interleave */
if physical$sector > sectors$per$track then physical$sector=physical$sector-sectors$per$track;

end;

/* seek to next cylinder */
call seek(drv,cylinder,head);

/* set up format control block */
format$docb.dma$op=dma$write;
format$docb.dma$addr=wrptr (O)+shl(wrptr (1),4);
format$docb.dma$addr$ext=O;
format$docb.dma$count=bytes$per$sector-l;
format$docb.disk$command(O)=write$command or shl(dens,6) or shl(multitrack,7);
format$docb.disk$command(l)=drv or shl(head,2);
format$docb.disk$command(2)=cyl;
format$docb.disk$command(3)=hd;
format$docb.disk$command(4)=sec;
format$docb.disk$command(5)=bytes$per$sector$code;
format$docb.disk$command(6)=sectors$per$track;
format$docb.disk$command(7)=read$write$gap;
format$docb.disk$command(8)=bytes$per$sector;

if bytes$per$sector$code = 0 then format$docb.disk$command(8)=bytes$per$sector;
else write$docb.disk$command(8)=OFFH;
call execute$docb(@write$docb,@operation$status);
end write;

write: procedure(drv,cyl,hd,sec,dens);
declare (drv,cyl,hd,sec,dens) byte;
write$docb.dma$op=dma$write;
write$docb.dma$addr=wrptr (O)+shl(wrptr (1),4);
write$docb.dma$addr$ext=O;
write$docb.dma$count=bytes$per$sector-l;
write$docb.disk$command(O)=write$command or shl(dens,6) or shl(multitrack,7);
write$docb.disk$command(l)=drv or shl(head,2);
write$docb.disk$command(2)=cyl;
write$docb.disk$command(3)=hd;
write$docb.disk$command(4)=sec;
write$docb.disk$command(5)=bytes$per$sector$code;
write$docb.disk$command(6)=sectors$per$track;
write$docb.disk$command(7)=read$write$gap;
if bytes$per$sector$code = 0 then write$docb.disk$command(8)=bytes$per$sector;
else write$docb.disk$command(8)=OFFH;
call execute$docb(@write$docb,@operation$status);
end write;

read: procedure(drv,cyl,hd,sec,dens);
declare (drv,cyl,hd,sec,dens) byte;
read$docb.dma$op=dma$read;
read$docb.dma$addr=rdptr (O)+shl(rdptr (1),4);
read$docb.dma$addr$ext=O;
read$docb.dma$count=bytes$per$sector-l;
read$docb.disk$command(O)=read$command or shl(dens,6) or shl(multitrack,7);
read$docb.disk$command(l)=drv or shl(head,2);
read$docb.disk$command(2)=cyl;
read$docb.disk$command(3)=hd;
read$docb.disk$command(4)=sec;
read$docb.disk$command(5)=bytes$per$sector$code;
read$docb.disk$command(6)=sectors$per$track;
read$docb.disk$command(7)=read$write$gap;
if bytes$per$sector$code = 0 then read$docb.disk$command(8)=bytes$per$sector;
else read$docb.disk$command(8)=OFFH;
call execute$docb(@read$docb,@operation$status);
end read;

\$eject

/**** initialize system by setting up 8237 dma controller and 8259A interrupt controller. ****/

initialize$system: procedure;
declare
/* I/O ports */
dma$disk$addr$port literally '00H', /* current address port */
dma$disk$word$count$port literally '01H', /* word count port */
dma$command$port literally '02H', /* command port */
dma$mode$port literally '03H', /* mode port */
dma$mask$addr$port literally '04H', /* mask set/reset port */
dma$clear$ff$port literally '0CH', /* clear first/last flip-flop port */
dma$master$clear$port literally '0DH', /* dma master clear port */
dma$mask$port literally '0FH', /* parallel mask set port*/
dma$cl$addr$port literally '02H',
dma$cl$word$count$port literally '03H',
dma$cl$mode literally '04H',
dma$cl$mask port'/
dma$mask$cl$addr$port literally '05H',
dma$c2$word$count$port literally '06H',
dma$c2$mode literally '07H',
dma$c2$addr$port literally '08H',
dma$c2$mode literally '09H',
dma$c2$mask port' /
dma$cl$clear$ff$port literally '0AH',
dma$c3$word$count$port literally '0BH',
dma$c3$mode literally '0CH',
dma$c3$addr$port literally '0DH',
dma$c3$mode literally '0EH',
dma$c3$mask port' /

/* mise masks and literals */
dma$extended$write literally '0FH', /* extended write flag */
dma$single$transfer literally '10H', /* single transfer flag */
dma$disk$mode literally '11H',
dma$c1$mode literally '12H',
dma$c2$mode literally '13H',
dma$c3$mode literally '14H',
mode$8088 literally '15H',
interrupt$base literally '16H',
single$controller literally '17H',
level$Sensitive literally '18H',
control$word$}$required literally '19H',
basic$icw1 literally '1AH',
mask$all literally 'OFFH',
disk$interrupt$mask literally '1BH',

output (dma$master$clear$port)=0;
output (dma$mode$port)=dma$extended$write;
output (dma$mask$port)=mask$all;

/* set all dma registers to valid values */
output (dma$cl$clear$ff$port)=0;
output (dma$cl$addr$port)=0;
output (dma$cl$word$count$port)=0;
output (dma$c2$clear$ff$port)=0;
output (dma$c2$addr$port)=0;
output (dma$c2$word$count$port)=0;
output (dma$c3$clear$ff$port)=0;
output (dma$c3$addr$port)=0;
output (dma$c3$word$count$port)=0;

/* set all addresses to zero */
output (dma$cl$clear$ff$port)=0;
output (dma$cl$addr$port)=0;
output (dma$cl$word$count$port)=0;
output (dma$c2$clear$ff$port)=0;
output (dma$c2$addr$port)=0;
output (dma$c2$word$count$port)=0;
output (dma$c3$clear$ff$port)=0;
output (dma$c3$addr$port)=0;
output (dma$c3$word$count$port)=0;

/* set all word counts to valid values */
output (dma$cl$clear$ff$port)=0;
output (dma$cl$word$count$port)=1;
output (dma$c2$clear$ff$port)=0;
output (dma$c2$word$count$port)=1;
output (dma$c3$clear$ff$port)=0;
output (dma$c3$word$count$port)=1;
/* initialize all dma channel modes */
output(dma$mode$port)=dma$disk$mode;
output(dma$mode$port)=dma$c1$mode;
output(dma$mode$port)=dma$c2$mode;
output(dma$mode$port)=dma$c3$mode;

/* initialize 8259A interrupt controller */
output(icw1)=single$controller or level$sensitive or control$word4$required or base$icw1;
output(icw2)=interrupt$base;
output(icw4)=mode$8088;
output(ocw1)=not disk$interrupt$mask;

/* initialize interrupt vector for fdc */
intvec(40H)= intptr(0);
intvec(41H)= intptr(1);

end initialize$systern;

APPLICATIONS

$eject

**** main program: first format disk (all tracks on side (head) 0. Then read each sector on every track of the disk forever. ****/

137 2 declare drive$ready(4) byte external;
138 1 /* disable until interrupt vector setup and initialization complete */
139 1 disable;
140 1
141 1 /* set initial floppy disk parameters */
142 1 density=mfm;
143 1 head=0;
144 1 multitrack=0;
145 1 filler$byte=55H;
146 1 tracks$per$disk=77;
147 1 bytes$per$sector=1024;
148 1 interleave=6;
149 1 step$rate=11;
150 1 head$load$time=401;
151 1 head$unload$time=2401;
152 1
153 1 /* derive dependent parameters from those above */
154 1 bytes$per$sector$code=shr(bytes$per$sector,7);
155 2 do index=0 to 3 1
156 2 if (bytes$per$sector$code and 1) <> 0
157 3 then do; bytes$per$sector$code=index; go to donebc; end;
158 3 else bytes$per$sector$code=shr(bytes$per$sector$code,1);
159 2 end;
160 1 donebc:
161 1 sectors$per$track=sectr$tab1e(bytes$per$sector$code$density);
162 1 format$gap=fmt$gap$tab1e(shl(density,2)+bytes$per$sector$code);
163 1 read$write$gap=rwd$gap$tab1e(shl(density,2)+bytes$per$sector$code);
164 1
165 1 /* initialize system and drivers */
166 1 call initialize$systern,
167 1 call initialize$drivers:
168 1 /* reenable interrupts and give 8272 a chance to report on drive status before proceeding */
169 1 enable;
170 1 call time(10);
171 1
172 1 /* specify disk drive parameters */
173 1 call specify(step$rate,head$load$time,head$unload$time,dma$mode);
174 1
175 1 /* run single disk drive #0 */
176 1 drive=0;
177 1
178 1 /* wait until drive ready */
179 1 do while 1;
180 2 if drive$ready(drive)
181 3 then go to start;
182 3 end;
183 2 start;
184 1 call format(drive,density,interleave);
185 1 do while 1;
186 2 do cylinder=0 to tracks$per$disk-1;
187 3 call seek(drive,cylinder,head);
188 3 do sector=1 to sectors$per$track;
189 4 /* set up write buffer */
190 4 do index=0 to bytes$per$sector-1; wrbuf(index)=index$sector$role$ cylinder; end;
190 4 call write(drive, cylinder, head, sector, density);
191 4 call read(drive, cylinder, head, sector, density);
192 4 /* check read buffer against write buffer */
193 4 if cmpw(@wrbuf, @rdbuf, shr(bytes$per$sector, 1)) <> 0FFFFH
194 4 then halt;
195 3 end;
196 2 end;
197 1 end run72;

MODULE INFORMATION:

CODE AREA SIZE = 0570H 1392D
CONSTANT AREA SIZE = 0000H 0D
VARIABLE AREA SIZE = 0907H 2311D
MAXIMUM STACK SIZE = 0022H 34D
412 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
APPENDIX C
8272 DRIVER FLOWCHARTS
APPLICATIONS

OUTPUT$CONTROL$STO$DMA

SET DMA MODE
CLEAR FIRST/LAST
FLIP-FLOP

WRITE DMA ADDRESS
TO 8237 AND
EXTENDED ADDRESS
LATCH

WRITE DATA TRANSFER
BYTE COUNT
TO 8237

START DMA
CHANNEL

RETURN

OUTPUT$COMMAND$STO$FDC

DISABLE
INTERRUPTS

DO
0 TO COMMAND
LENGTH

DONE

ENABLE
INTERRUPTS

RETURN

OUTPUT A
COMMAND
BYTE TO THE
8272

ERROR
REPORTED

YES

ENABLE
INTERRUPTS

RETURN

ERROR

ERROR
APPLICATIONS

EXECUTESDOCB

VALID COMMAND? NO
RETURN INVALID STATUS

YES

OVERLAPPED OPERATION AND 8272 BUSY? YES
RETURN BUSY STATUS

NO

NON-OVERLAPPED OPERATION AND 8272 BUSY OR DRIVE SEEKING? YES
RETURN BUSY STATUS

NO

DISABLE INTERRUPTS

OPERATION IN PROGRESS FLAG SET? YES
ENABLE INTERRUPTS

RETURN BUSY STATUS

NO

SET OPERATIONSINPROGRESS
RESET OPERATIONSCOMPLETE
SAVE DOCB POINTER
SET GLOBALS/DRIVESNO

ENABLE INTERRUPTS

CALL OUTPUTSCONTROLSSTOSDMA TO SET UP 8237

CALL OUTPUTSCOMMANDSTOSFDOC TO PERFORM 8272 COMMAND PHASE

ERROR REPORTED IN COMMAND PHASE? YES
CALL OPERATIONSCLEANUP BEFORE RETURN

NO

RETURN COMMAND ERROR STATUS
APPLICATIONS

RESULT PHASE OF PREVIOUS COMMAND

YES

8272 BUzy

ASYNCHRONOUS INTERRUPT

NO

RESTORE PREVIOUSLY SAVED DOCB POINTER

CALL INPUTSRESULT FROM FDC TO PERFORM RESULT PHASE

ERROR DETECTED

CASE RESULT CODE OF

OPERATION COMPLETE 0

ABNORMAL TERMINATION 1

INVALID COMMAND 2

DRIVE READY CHANGE 3

CALL COPYRESULT TO STORE RESULT INFORMATION IN DOCB (IF OPERATION IN PROGRESS)

SET DRIVES STATUS CHANGE FLAG

SET DRIVES READY FLAG FOR DRIVE BASED ON STATUS FLAG IN STO

RESET OPERATIONS IN PROGRESS
SET OPERATION COMPLETE
RESET GLOBAL DRIVES NO

SEND END OF INTERRUPT TO 8259A

RETURN

APPLICATIONS

RESULT PHASE OF PREVIOUS COMMAND

YES

8272 BUSY

ASYNCHRONOUS INTERRUPT

NO

RESTORE PREVIOUSLY SAVED DOCB POINTER

CALL INPUTSRESULT FROM FDC TO PERFORM RESULT PHASE

ERROR DETECTED

CASE RESULT CODE OF

OPERATION COMPLETE 0

ABNORMAL TERMINATION 1

INVALID COMMAND 2

DRIVE READY CHANGE 3

CALL COPYRESULT TO STORE RESULT INFORMATION IN DOCB (IF OPERATION IN PROGRESS)

SET DRIVES STATUS CHANGE FLAG

SET DRIVES READY FLAG FOR DRIVE BASED ON STATUS FLAG IN STO

RESET OPERATIONS IN PROGRESS
SET OPERATION COMPLETE
RESET GLOBAL DRIVES NO

SEND END OF INTERRUPT TO 8259A

RETURN
APPLICATIONS

COPYBINTRESULT

OPERATION IN PROGRESS

NO

YES

RETRIEVE SAVED DOCB POINTER

COPY RESULT PHASE DATA FROM THE INTERRUPT DOCB TO CALLING DOCB

RESET OPERATIONS IN PROGRESS FLAG SET OPERATIONS COMPLETE FLAG

RETURN

5-572
82062 WINCHESTER DISK CONTROLLER

- Controls ST506/ST412 Interface Winchester Drives
- 5 MBit/Sec Transfer Rate
- 128, 256, 512, and 1024 Byte Sector Lengths
- Six High-Level Commands: Restore, Seek, Read Sector, Write Sector, Scan ID, and Write Format
- Multiple Sector Transfer Capability
- Implied Seek With Read/Write Commands
- 7 Byte Sector Length Extension For External Error Correction Code
- Single +5 Volt Power Supply

The 82062 Winchester Disk Controller (WDC) device interfaces microprocessor systems to Winchester Disks that use the Seagate Technology ST506/ST412 interface. Examples include the Seagate ST506 and ST412, Shugart SA604 and SA606, Tandon 600, and Computer Memories CM5206 and CM5412. The device translates parallel data from the microprocessor to a 5 mbit/sec, MFM-encoded serial bit stream. It provides all of the drive control logic and, in addition, control signals which simplify the design of an external phase locked loop and write precompensation circuitry. The 82062 is designed to interface to the host controller through an external sector buffer.

Figure 1. 82062 Block Diagram

Figure 2. Pin Configuration
### Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCS</td>
<td>1</td>
<td>O</td>
<td><strong>Buffer Chip Select:</strong> Output used to enable reading or writing of the external sector buffer by the 82062. When low, the host should not able to drive the 82062 data bus, RD, or WR lines.</td>
</tr>
<tr>
<td>BCR</td>
<td>2</td>
<td>O</td>
<td><strong>Buffer Counter Reset:</strong> Output that is strobed by the 82062 prior to read/write operation. This pin is strobed whenever BCS changes state. Used to reset the address counter of the buffer memory.</td>
</tr>
<tr>
<td>INTRQ</td>
<td>3</td>
<td>O</td>
<td><strong>Interrupt Request:</strong> Interrupt generated by the 82062 upon command termination. It is reset when any register is read. Optionally signifies when a data transfer is required on Read Sector commands.</td>
</tr>
<tr>
<td>N/C</td>
<td>4</td>
<td></td>
<td><strong>No connection. Reserved for future use.</strong></td>
</tr>
<tr>
<td>RESET</td>
<td>5</td>
<td>I</td>
<td><strong>Reset:</strong> Initializes the controller and clears all status flags. Does not clear the Task Registers.</td>
</tr>
<tr>
<td>RD</td>
<td>6</td>
<td>I/O</td>
<td><strong>Read:</strong> As an input, RD controls the transfer of information from the 82062 registers to the host. RD is an output when the 82062 is reading data from the sector buffer (BCS low).</td>
</tr>
<tr>
<td>WR</td>
<td>7</td>
<td>I/O</td>
<td><strong>Write:</strong> As an input, WR controls the transfer of command or task information into the 82062 registers. WR is an output when the 82062 is writing data to the sector buffer (BCS low).</td>
</tr>
<tr>
<td>CS</td>
<td>8</td>
<td>I</td>
<td><strong>Chip Select:</strong> Enables RD and WR as inputs for access to the Task Registers. It has no effect once a disk command starts.</td>
</tr>
<tr>
<td>A0-A2</td>
<td>9-11</td>
<td>I</td>
<td><strong>Address:</strong> Used to select a register from the task register file.</td>
</tr>
<tr>
<td>DB0-DB7</td>
<td>12-19</td>
<td>I/O</td>
<td><strong>Data Bus:</strong> Bidirectional 8-bit Data Bus with control determined by BCS. When BCS is high the microprocessor has full control of the data bus for reading and writing the Task Registers. When BCS is low the 82062 controls the data bus to transfer data to or from the buffer.</td>
</tr>
<tr>
<td>GND</td>
<td>20</td>
<td></td>
<td><strong>Ground</strong></td>
</tr>
<tr>
<td>WR DATA</td>
<td>21</td>
<td>O</td>
<td><strong>Write Data:</strong> Open drain output that shifts out MFM data at a rate determined by Write Clock. Requires an external flip-flop clocked at 10 MHz. See note 1.</td>
</tr>
<tr>
<td>LATE</td>
<td>22</td>
<td>O</td>
<td><strong>Late:</strong> Open drain output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders. See note 1.</td>
</tr>
<tr>
<td>EARLY</td>
<td>23</td>
<td>O</td>
<td><strong>Early:</strong> Open drain output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders. See note 1.</td>
</tr>
<tr>
<td>WR GATE</td>
<td>24</td>
<td>O</td>
<td><strong>Write Gate:</strong> High when write data is valid. WR GATE goes low if the WR FAULT input is active. This output is used by the drive to enable head write current.</td>
</tr>
<tr>
<td>WR CLOCK</td>
<td>25</td>
<td>I</td>
<td><strong>Write Clock:</strong> Clock input used to derive the write data rate. Frequency - 5MHz for the ST506 interface, 4.34MHz for the SA 1000 interface. See note 2.</td>
</tr>
<tr>
<td>DIR</td>
<td>26</td>
<td>O</td>
<td><strong>Direction:</strong> High level on this output tells the drive to move the head inward (increasing cylinder number). The state of this signal is determined by the 82062's internal comparison of actual cylinder location vs desired cylinder.</td>
</tr>
<tr>
<td>STEP</td>
<td>27</td>
<td>O</td>
<td><strong>Step:</strong> Provides 8.4 microsecond pulses to move the drive head to another cylinder at a programmable frequency.</td>
</tr>
<tr>
<td>DRDY</td>
<td>28</td>
<td>I</td>
<td><strong>Drive Ready:</strong> If DRDY from the drive goes low, the command will be terminated.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Pin No.</td>
<td>Type</td>
<td>Name and Function</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INDEX</td>
<td>29</td>
<td>I</td>
<td><strong>Index</strong>: Signal from the drive indicating the beginning of a track. It is used by the 82062 during formatting, and for counting retries. Index is edge triggered. Only the rising edge is valid.</td>
</tr>
<tr>
<td>WR FAULT</td>
<td>30</td>
<td>I</td>
<td><strong>Write Fault</strong>: An error input to the 82062 which indicates a fault condition at the drive. If WR FAULT from the drive goes high, the command will be terminated.</td>
</tr>
<tr>
<td>TRACK 000</td>
<td>31</td>
<td>I</td>
<td><strong>Track Zero</strong>: Signal from the drive which indicates that the head is at the outermost cylinder. Used by the Restore command.</td>
</tr>
<tr>
<td>SC</td>
<td>32</td>
<td>I</td>
<td><strong>Seek Complete</strong>: Signal from the drive indicating to the 82062 that the drive head has settled and that reads or writes can be made. SC is edge triggered. Only the rising edge is valid.</td>
</tr>
<tr>
<td>RWC</td>
<td>33</td>
<td>O</td>
<td><strong>Reduced Write Current</strong>: Signal goes high for all cylinder numbers above the value programmed in the Write Precomp Cylinder register. It is used by the precompensation logic and by the drive to reduce the effects of bit shifting.</td>
</tr>
<tr>
<td>DRUN</td>
<td>34</td>
<td>I</td>
<td><strong>Data Run</strong>: This signal informs the 82062 when a field of ones or zeroes has been detected in the read data stream by an external one-shot. This indicates the beginning of an ID field. RD GATE is brought high when DRUN is sampled high for 16 clock periods. See note 2.</td>
</tr>
<tr>
<td>BRDY</td>
<td>35</td>
<td>I</td>
<td><strong>Buffer Ready</strong>: Input used to signal the controller that the buffer is ready for reading (full), or writing (empty), by the host uP. Only the rising edge indicates the condition.</td>
</tr>
<tr>
<td>BDRQ</td>
<td>36</td>
<td>O</td>
<td><strong>Buffer Data Request</strong>: Activated during Read or Write commands when a data transfer between the host and the 82062's sector buffer is required. Typically used as a DMA request line, or to generate an interrupt.</td>
</tr>
<tr>
<td>RD DATA</td>
<td>37</td>
<td>I</td>
<td><strong>Read Data</strong>: Single ended input that accepts MFM data from the drive. See note 2.</td>
</tr>
<tr>
<td>RD GATE</td>
<td>38</td>
<td>O</td>
<td><strong>Read Gate</strong>: Output that is high for data and ID fields. Goes active when DRUN has been high for 16 WR CLOCK periods to permit the external phase lock loop to lock onto the incoming disk data stream.</td>
</tr>
<tr>
<td>RD CLOCK</td>
<td>39</td>
<td>I</td>
<td><strong>Read Clock</strong>: Clock input derived from the external data recovery circuits. See note 2.</td>
</tr>
<tr>
<td>Vcc</td>
<td>40</td>
<td>I</td>
<td><strong>D.C. Power</strong>: +5V</td>
</tr>
</tbody>
</table>

**Note 1:** This pin requires a pull-up resistor to function properly. A value of 1000 ohms will work satisfactorily.

**Note 2:** This pin requires input levels that are not TTL compatible. These lines can be interfaced to TTL with a pull-up resistor. Too small of a resistor will produce a VIL level that is too high. Too large of a resistor will degrade the signal's rise time. A minimum value for the resistor is determined as follows:

\[(V_{cc\ max}) - (82062\ V_{IL\ max})\]

\[(TTL\ I_{OL\ min}) - (82062\ I_{IL\ max})\]

This would typically be:

\[5.25V - 0.5V\]

\[\frac{1.6\ mA - 10\ \mu A}{3k \Omega}\]
FUNCTIONAL DESCRIPTION

The Intel 82062 Winchester Disk Controller (WDC) integrates much of the logic needed to implement Winchester Disk controller subsystems. It provides MFM-encoded data and all the control lines required by hard disks using the Seagate Technology ST506 or Shugart Associates SA1000 interface standard. Currently, most 5¼ inch and many 8 inch Winchester Drives use this interface.

Due to the higher data rates required by these drives—1 byte every 1.6 usec—the 82062 is designed to interface with the host CPU or I/O controller through an external buffer RAM. The 82062 WDC has four pins that minimize the logic required to design a buffer interface.

Figure 3 shows a block diagram of an 82062 subsystem. The WDC is controlled by the host CPU through six commands:

- Restore
- Seek
- Read Sector
- Write Sector
- Scan ID
- Write Format

These commands use information stored by six task registers. Command execution starts immediately after the command register is loaded—therefore commands require only one byte from the CPU after the WDC has been initialized.

The 82062 adds all the required track formatting to the data field, including two bytes of CRC. Optionally, these two bytes can be replaced by seven bytes of EGC information for external error correction.

INTERNAL ARCHITECTURE

The internal architecture of the 82062 WDC is shown in more detail in Figure 4. The major functional blocks are:

PLA Controller

The PLA interprets commands and provides all control functions. It is synchronized with WR CLOCK.

Magnitude Comparator

A 10-bit magnitude comparator is used to calculate the direction and number of step pulses needed to move the head from the present to the desired cylinder.

CRC Logic

Generates and checks the cyclic redundancy check characters appended to the ID and data fields. The polynomial used is:

\[ x^{16} + x^{12} + x^5 + 1. \]

MFM Encode/Decode

Encodes and decodes MFM data to be written/read from the drive. The MFM encoder operates from WR CLOCK, a clock having a frequency equivalent to the bit rate. The MFM decoder operates from RD CLOCK, a bit rate clock generated from the external data separator. RD CLOCK and WR CLOCK need not be synchronized.

---

Figure 3. System Block Diagram
AM Detect

The address mark detector checks the incoming data stream for a unique missing clock pattern (Data = A1H, Clock = 0AH) used in each ID and data field.

Host/Buffer Interface Control

The Host/Buffer IFC logic contains all of the necessary circuitry to communicate with the 8-bit bus from the host processor.

Drive Interface Control

The Drive IFC logic controls and monitors all lines from the drive, with the exception of read and write data.

DRIVE INTERFACE

The drive side of the 82062 WDC requires three sections of external logic. These are buffer/receivers, data separator, and write precompensation. Figure 5 illustrates a drive side interface.

The buffer/receivers condition the control lines to be driven down the cable to the drive. The control lines are typically single-ended, resistor terminated TTL levels. The data lines to and from the drive also require buffering, but are differential RS-422 levels. The interface specification to the drive can be found in the manufacturers' OEM manual. The WDC supplies TTL compatible signals, and will interface to most buffer/driver devices.

The data recovery circuits consist of a phase-lock loop data separator and associated components. The 82062 WDC interacts with the data separator thru the DATA RUN (DRUN) and RD GATE signals. A block diagram of a typical data separator circuit is shown in Figure 6. Read data from the drive is presented to the RD DATA input of the WDC, the reference multiplexor, and a retriggerable one-shot. The RD GATE (Pin 38) output will be low when the WDC is not inspecting data. The PLL at this time should remain locked to the reference clock.
Figure 5. Drive Interface

Figure 6. Data Recovery Circuit
When any Read/Write command is initiated and a search for address mark begins, the DRUN input is examined. The DRUN one-shot is set for slightly greater than one bit time, allowing it to retrigger constantly on a field of ones and zeros. An internal counter times out to see that DRUN is high for 2 byte times. RD GATE is set by the WDC, switching the data separator to lock onto the incoming data stream. If DRUN falls prior to an additional 7 byte times, RD GATE is lowered and the process is repeated. RD GATE will remain active high until a non-zero, non-address mark byte is detected. It will then lower RD GATE for two byte times (to allow the PLL to lock back on to the reference clock), and start the DRUN search again. If an address mark is detected, RD GATE will be held high and the command will continue searching for the proper ID field. This sequence is shown in the flow chart in Figure 7.

The write precompensation logic is controlled by the signals REDUCE WRITE CURRENT (RWC), EARLY and LATE. The cylinder in which the RWC line becomes active is controlled by the REDUCE WRITE CURRENT register in the Task Register File. It can be used to turn on the precomp circuitry on a predetermined cylinder. If the REDUCE WRITE CURRENT register contents are FFH, then RWC will always be low.

The signals EARLY and LATE are used to tell the precomp circuitry how much delay is required on the WR DATA pulse about to be sent. The amount of delay is determined externally through a digital delay line or equivalent circuitry. Since the EARLY signal occurs after the fact, WR DATA should be delayed by one interval when both EARLY and LATE are deasserted, two intervals when LATE is asserted, and no delay when EARLY is asserted. An interval is, for example, 12-15 ns. for the ST506 interface. EARLY or LATE will be active slightly ahead of the WR DATA pulse. EARLY and LATE will never be asserted at the same time. EARLY and LATE are always active, and should be gated externally by the RWC signal.

**HOST PROCESSOR INTERFACE**

The primary interface between the host processor and the 82062 WDC is through an 8-bit bi-directional data bus. This bus is used to transmit/receive data to both the WDC and a sector buffer. The sector buffer is constructed with either FIFO memory, or static RAM and a counter. Since the WDC will use the data bus when accessing the sector buffer, a transceiver must be used to isolate the host during this time. Figure 8 shows a typical connection to a sector buffer implemented with RAM memory. Whenever the WDC is not using the sector buffer, the BUFFER CHIP SELECT (BCS) is high (disabled). This allows the host to access the WDC's Task Register File, and

![Figure 7. PLL Control Sequence](image)
to set up parameters prior to issuing a command. It also allows the host to access the RAM buffer. A decoder is used to generate a chip select when Ao-2 is '000', an unused address in the WDC. A binary counter is enabled whenever RD or WR go active and is incremented on the trailing edge of the chip select. This allows the host to access sequential bytes within the RAM. The decoder also generates another chip select when Ao-2 does not equal '000', allowing access to the WDC's internal registers while keeping the RAM tri-stated.

During a WRITE SECTOR command, the host processor sets up data in the Task Register File and then issues the command. It then generates a status to inform the host that it may load the buffer with the data to be written. When the counter reaches its maximum count, the BUFFER READY (BRDY) signal is made active (by the "carry" out of the counter), informing the WDC that the buffer is full. (BRDY is a rising edge triggered signal which will be ignored if activated before the WDC issues BCR). BCS is then made active, disconnecting the host through the transceivers, and the RD and WR lines become outputs from the WDC to allow it to access the buffer. When the WDC is done using the buffer, it disables BCS which again allows the host to access the local bus. The READ SECTOR command operates in a similar manner, except the buffer is loaded by the WDC instead of the host processor.

Another control signal called BUFFER DATA REQUEST (BDRQ, not used in Figure 8) is a DMA signal that can inform a DMA controller when the 82062 WDC is requesting data. For further explanation, refer to the individual command descriptions and the A.C. Characteristics. In a READ SECTOR command, interrupts are generated at the termination of the command. An interrupt may be specified to occur either at the end of the command, or when BDRQ is activated. The INTERRUPT line (INTRQ) is cleared either by reading the STATUS register, or by writing a new command in the COMMAND register.

Figure 8. CPU Buffer Interface
TASK REGISTER FILE

The Task Register File is a bank of registers used to hold parameter information pertaining to each command. These registers and their addresses are:

<table>
<thead>
<tr>
<th>A2 A1 A0</th>
<th>READ</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>(Bus Tri-Stated)</td>
<td>(Bus Tri-Stated)</td>
</tr>
<tr>
<td>0 0 1</td>
<td>Error Flags</td>
<td>Reduce Write Current</td>
</tr>
<tr>
<td>0 1 0</td>
<td>Sector Count</td>
<td>Sector Count</td>
</tr>
<tr>
<td>0 1 1</td>
<td>Sector Number</td>
<td>Sector Number</td>
</tr>
<tr>
<td>1 0 0</td>
<td>Cylinder Low</td>
<td>Cylinder Low</td>
</tr>
<tr>
<td>1 0 1</td>
<td>Cylinder High</td>
<td>Cylinder High</td>
</tr>
<tr>
<td>1 1 0</td>
<td>SDH</td>
<td>SDH</td>
</tr>
<tr>
<td>1 1 1</td>
<td>Status Register</td>
<td>Command Register</td>
</tr>
</tbody>
</table>

NOTE: Registers are not cleared by RESET.

ERROR REGISTER

This read-only register contains specific error status after the completion of a command. The bits are defined as follows:

Bit 7 - Bad Block Detect
This bit is set when an ID field has been encountered that contains a bad block mark. It is used for bad sector mapping.

Bit 6 - CRC Data Field
This bit is set when a data field CRC error has occurred. The sector buffer may still be read but will contain errors.

Bit 5 - Reserved Not used.
Forced to zero.

Bit 4 - ID Not Found
This bit is set when the desired cylinder, head, sector, or size parameter cannot be found after 8 revolutions of the disk, or if an ID field CRC error has occurred.

Bit 3 - Reserved Not used.
Forced to zero.

Bit 2 - Aborted Command
This bit is set if a command was issued while DRDY (Pin 28) is deasserted or WR FAULT (Pin 30) is asserted. The Aborted Command bit will also be set if an undefined command is written into the COMMAND register, but an implied seek will be executed.

Bit 1 - TRACK 000
This bit is set only by the RESTORE command. It indicates that TRACK 000 (Pin 31) has not gone active after the issuance of 1024 stepping pulses.

Bit 0 - Data Address Mark
This bit is set during a READ SECTOR command if the Data Address Mark is not found after the proper Sector ID is read.

REDUCE WRITE CURRENT REGISTER

This register is used to define the cylinder number where RWC (Pin 33) is asserted:

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBD  CRC  ID  AC  TK000  DM</td>
</tr>
</tbody>
</table>

The value (0-255) loaded into this register is internally multiplied by 4 to specify the actual cylinder where RWC is asserted. Thus a value of 01H will cause RWC to activate on cylinder 4, 02H on cylinder 8, and so on. RWC switching points are then 0,4,8,...,1020. RWC will be asserted when the present cylinder is greater than or equal to the cylinder indicated by this register. For example, the ST506 interface requires precomp on cylinder 128 (80H) and above. Therefore, the REDUCE WRITE CURRENT register should be loaded with 32 (20H). A value of FFH will make RWC stay low, regardless of the actual cylinder number.
SECTOR COUNT REGISTER

This register is used to define the number of sectors that need to be transferred to the buffer during a READ MULTIPLE SECTOR or WRITE MULTIPLE SECTOR command:

```
7 6 5 4 3 2 1 0
```

The value contained in the register is decremented after each sector is transferred to/from the sector buffer. A zero represents a 256 sector transfer, a one a 1 sector transfer, etc. This register is a "don't care" when single sector commands are specified.

SECTOR NUMBER

This register holds the sector number of the desired sector:

```
7 6 5 4 3 2 1 0
```

For a multiple sector command, it specifies the first sector to transferred. It is incremented after each sector is transferred to/from the sector buffer. The SECTOR NUMBER register may contain any value from 0 to 255.

The SECTOR NUMBER register is also used to program the Gap 1 and Gap 3 lengths to be used when formatting a disk. See the WRITE FORMAT command description for further explanation.

CYLINDER NUMBER LOW REGISTER

This register holds the lower byte of the desired cylinder number:

```
7 6 5 4 3 2 1 0
```

It is used in conjunction with the CYLINDER NUMBER HIGH register to specify a range of 0 to 1023.

CYLINDER NUMBER HIGH REGISTER

This register holds the two most significant bits of the desired cylinder number:

```
7 6 5 4 3 2 1 0
```

Internal to the 82062 WDC is another pair of registers that hold the actual position where the R/W heads are located. The CYLINDER NUMBER HIGH and LOW registers can be considered the cylinder destination for seeks and other commands. After these commands are executed, the internal cylinder position registers' contents are equal to the cylinder high/low registers. If a drive number change is detected on a new command, the WDC automatically reads an ID field to update its internal cylinder position registers. This affects all commands except a RESTORE.

SECTOR/DRIVE/HEAD REGISTER

The SDH register contains the desired sector size, drive number, and head number parameters. The format is diagramed below.

```
7 6 5 4 3 2 1 0
```

```
       EXT   SIZE   DRIVE   HEAD
```

```
6 5  SECTOR SIZE
0 0  256
0 1  512
1 0  1024
1 1  128
```

```
4 3  DRIVE #
0 0  DSEL1
0 1  DSEL2
1 0  DSEL3
1 1  DSEL4
```

```
2 1 0  HEAD #
0 0 0  HSEL0
0 0 1  HSEL1
0 1 0  HSEL2
0 1 1  HSEL3
1 0 0  HSEL4
1 0 1  HSEL5
1 1 0  HSEL6
1 1 1  HSEL7
```

ORDER NUMBER: 210446-004
Both head number and sector size are compared against the disks' ID field. Head select and drive select lines are not available as outputs from the 82062 WDC and must be generated externally. Figure 9 shows a possible logic implementation of these select lines.

Bit 7, the extension bit (EXT), is used to extend the data field by seven bytes when using ECC codes. When EXT = 1, the CRC is not appended to the end of the data field, the data field becomes “sector size + 7” bytes long. The CRC is checked on the ID field regardless of the state of EXT. Note that the sector size bits (SIZE) are written to the ID field during a formatting command. The SDH byte written into the ID field is different than the SDH Register contents. The recorded SDH byte does not have the drive number (DRIVE) written but does have the BAD BLOCK mark written. The format is:

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>READY</td>
<td>WF</td>
<td>SC</td>
<td>DRQ</td>
<td>CIP</td>
<td>ERROR</td>
<td></td>
</tr>
</tbody>
</table>
```

**Bit 7 - Busy**

This bit is set whenever the 82062 WDC is accessing the disk. Commands should not be loaded into the COMMAND register while Busy is set. Busy is set when a command is written into the WDC and is cleared at the end of all commands except READ SECTOR. While executing a READ SECTOR command, Busy is cleared after the sector buffer has been filled. When the Busy bit is set, no other bits in either the STATUS or any other registers are valid.

**Bit 6 - Ready**

This bit reflects the state of the DRDY (Pin 28) line.

**Bit 5 - Write Fault**

This bit reflects the state of the WR FAULT (Pin 30) line. Whenever WR FAULT goes high, an interrupt will be generated.

**Bit 4 - Seek Complete**

This bit reflects the state of the SC (Pin 32) line. Commands which initiate a seek will pause until Seek Complete is set.

---

STATUS REGISTER

The status register is a read-only register which informs the host of certain events performed by the 82062 WDC as well as reporting status from the drive control lines. The INTRQ line will be reset when the status register is read. The format is:

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD</td>
<td>BLOCK</td>
<td>SIZE</td>
<td>0</td>
<td>0</td>
<td>HEAD</td>
<td>#</td>
<td></td>
</tr>
</tbody>
</table>
```

Note that use of the extension bit requires the gap lengths to be modified as described in the WRITE FORMAT command description.
Bit 3 - Data Request

The Data request bit (DRO) reflects the state of the BDRO (Pin 36) line. It is set when the sector buffer should be loaded with data or read by the host processor, depending upon the command. The DRQ bit and the BDRO line remain high until BRDY is sensed, indicating the operation is completed. BDRO can be used in DMA interfacing, while DRQ can be used for programmed I/O transfers.

Bit 2 - Reserved

Not Used. Forced to zero.

Bit 1 - Command in Progress

When this bit is set, a command is being executed and a new command should not be loaded until it is cleared. Although a command may be executing, the sector buffer is still available for access by the host processor. Only the STATUS register may be read. If other registers are read, the STATUS register contents will be returned.

Bit 0 - Error

This bit is set whenever any bits in the ERROR register are set. It is the logical 'or' of the bits in the error register and may be used by the host processor to quickly check for successful completion of a command. This bit is reset when a new command is written into the COMMAND register.

COMMAND REGISTER

This write-only register is loaded with the desired command:

```
  7 6 5 4 3 2 1 0
  COMMAND
```

The command begins to execute immediately upon loading. This register should not be loaded while the Busy or Command in Progress bits are set in the STATUS register. The INTRQ line (Pin 3), if set, will be cleared by a write to the COMMAND register.

INSTRUCTION SET

The 82062 WDC instruction set contains six commands. Prior to loading the command register, the host processor must first set up the Task Register File with the information needed for the command. Except for the COMMAND register, the registers may be loaded in any order. If a command is in progress, a subsequent write to the COMMAND register will be ignored until execution of the current command is completed as indicated by the command in progress bit in the STATUS register being cleared.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTORE</td>
<td>0 0 0 1 R3 R2 R1 R0</td>
</tr>
<tr>
<td>SEEK</td>
<td>0 1 1 1 R3 R2 R1 R0</td>
</tr>
<tr>
<td>READ SECTOR</td>
<td>0 0 1 0 1 M 0 T</td>
</tr>
<tr>
<td>WRITE SECTOR</td>
<td>0 0 1 1 0 M 0 T</td>
</tr>
<tr>
<td>SCAN ID</td>
<td>0 1 0 0 0 0 0 T</td>
</tr>
<tr>
<td>WRITE FORMAT</td>
<td>0 1 0 1 0 0 0 0</td>
</tr>
</tbody>
</table>

\[ R_{3-0} = \text{Rate Field} \]

For 5 MHz WR CLOCK:

\[ R_{3-0} = 0000 = 35 \text{us} \]
\[ 0001 = 0.5 \text{ms} \]
\[ 0010 = 1.0 \text{ms} \]
\[ 0011 = 1.5 \text{ms} \]
\[ 0100 = 2.0 \text{ms} \]
\[ 0101 = 2.5 \text{ms} \]
\[ 0110 = 3.0 \text{ms} \]
\[ 0111 = 3.5 \text{ms} \]
\[ 1000 = 4.0 \text{ms} \]
\[ 1001 = 4.5 \text{ms} \]
\[ 1010 = 5.0 \text{ms} \]
\[ 1011 = 5.5 \text{ms} \]
\[ 1100 = 6.0 \text{ms} \]
\[ 1101 = 6.5 \text{ms} \]
\[ 1110 = 7.0 \text{ms} \]
\[ 1111 = 7.5 \text{ms} \]

\[ T = \text{Retry Enable} \]

\[ T = 0 \text{ Enable Retries} \]
\[ T = 1 \text{ Disable Retries} \]

\[ M = \text{Multiple Sector Flag} \]

\[ M = 0 \text{ Transfer 1 Sector} \]
\[ M = 1 \text{ Transfer Multiple Sectors} \]

\[ I = \text{Interrupt Enable} \]

\[ I = 0 \text{ Interrupt at BDRQ time} \]
\[ I = 1 \text{ Interrupt at end of command} \]

ORDER NUMBER: 210446-004
RESTORE COMMAND

The RESTORE command is usually used on a power-up condition. The actual stepping rate used for the RESTORE is determined by the Seek Complete time. A step pulse is issued and the 82062 WDC waits for a rising edge on the Seek Complete (SC) line before issuing the next pulse. If 10 index pulses are received without a rising edge of SC, the 82062 will switch to sensing the level of the SC line. If after 1,024 stepping pulses the TRACK 000 line does not go active, the WDC will set the TRACK 000 bit in the ERROR register and terminate with an INTRQ. An interrupt will also occur if WR FAULT goes active or DRDY goes inactive at any time during execution.

The rate field specified R_3-6 is stored in an internal register for future use in commands with implied seeks.

A flowchart of the RESTORE command is shown in Figure 10.

SEEK COMMAND

Since all commands except the SCAN ID command feature an implied seek, the SEEK command can be used for overlap seek operations on multiple drives. The actual stepping rate used is taken from the Rate Field of the command, and is stored in an internal register for future use. If DRDY goes inactive or WR FAULT goes active at any time during the seek, the command is terminated and an INTRQ is generated.

The direction and number of step pulses needed is calculated by comparing the contents of the CYLINDER NUMBER LOW/HIGH register pair to the internal cylinder position register. After all steps have been issued, the internal cylinder position register is updated and the command is terminated. The Seek Complete (SC) line is not checked at the beginning or end of the command.

If an implied seek was performed, the 82062 will search until a rising edge of SC is received. If 10 index pulses are received without a rising edge of SC, the 82062 will switch to sensing the level of the SC line.

A flowchart of the SEEK command is shown in Figure 11.

READ SECTOR

The READ SECTOR command is used to transfer one or more sectors of data from the disk to the sector buffer. Upon receipt of the READ SECTOR command, the 82062 WDC checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation is performed and a seek takes place. If an implied seek was performed, the WDC will search until a rising edge of SC is received. The WR FAULT and DRDY lines are monitored throughout the command.

![Figure 10. Restore-Command Flow](image-url)
When the Seek Complete (SC) line is high (with or without an implied seek having occurred), the search for an ID field begins. If \( T = 0 \) (retries enabled), the 82062 WDC must find an ID with the correct cylinder number, head, sector size and CRC within 10 revolutions, or an automatic scan ID will be performed to obtain cylinder position information, and then a seek performed (if necessary). The search for the proper ID will be retried for up to 10 revolutions. If the correct sector is still not found, the appropriate error bits will be set and the command terminated. Data CRC errors will also be retried for up to 10 revolutions (if \( T = 0 \)).

If \( T = 1 \) (retries disabled), the ID search must find the correct sector within 2 revolutions or the appropriate error bits will be set and the command terminated.

Both the READ SECTOR and WRITE SECTOR commands feature a "simulated completion" to ease programming. DRQ/BDRQ will be generated upon detecting an error condition. This allows the same program flow for successful or unsuccessful completion of a command.

When the data address mark is found, the WDC is ready to transfer data to the sector buffer. After the data has been transferred, the I bit is checked. If \( I = 0 \), INTRQ is made active coincident with BDRQ, indicating that a transfer of data from the buffer to the host processor is required. If \( I = 1 \), INTRQ will occur at the end of the command, i.e. after the buffer is unloaded by the host.

An optional M bit may be set for multiple sector transfers. When \( M = 0 \), one sector is transferred and the SECTOR COUNT register is ignored. When \( M = 1 \), multiple sectors are transferred. After each sector is transferred the 82062 decrements the SECTOR COUNT register and increments the SECTOR NUMBER register. The next logical sector will be transferred regardless of any interleave. Sectors are numbered at format time by a byte in the ID field.

For the 82062 to make multiple sector transfers to the buffer, the BRDY line must be toggled low to high for each sector. Transfers will continue until the SECTOR COUNT register equals zero, or the BRDY line goes active. If the SECTOR COUNT register is non-zero (indicating more sectors are to be transferred but the buffer is full), BDRQ will be made active and the host must unload the buffer. After this occurs, the buffer will again be free to accept the remaining sectors from the WDC. This scheme enables the user to transfer more sectors than the buffer memory has capacity for.

In summary then, READ SECTOR operation is as follows:

---

**Figure 11. Seek Command Flow**
When \( M = 0 \) (READ SECTOR)

1. Host: Sets up parameters; issues READ SECTOR command.
2. \( 82062 \): Strobes BCR; sets BCS = 0.
3. \( 82062 \): Finds sector specified; transfers data to buffer.
4. \( 82062 \): Strobes BCR; sets BCS = 1.
5. \( 82062 \): Sets BDRQ = 1, DRQ = 1.
6. \( 82062 \): If bit = 1 then go to (9).
8. \( 82062 \): Waits for BRDY, then sets INTRQ = 1; END.
9. \( 82062 \): Sets INTRQ = 1.
10. Host: Reads out contents of buffer; END.

When \( M = 1 \) (READ MULTIPLE SECTOR)

1. Host: Sets up parameters; issues READ SECTOR command.
2. \( 82062 \): Strobes BCR; sets BCS = 0.
3. \( 82062 \): Finds sector specified; transfers data to buffer.
4. \( 82062 \): Decrements SECTOR COUNT register; increments SECTOR NUMBER register.
5. \( 82062 \): Strobes BCR; sets BCS = 1.
6. \( 82062 \): Sets BDRQ = 1, DRQ = 1.
7. Host: Reads out contents of buffer;
8. Buffer: Indicates data has been transferred by activating BRDY.
9. \( 82062 \): When BRDY = 1, if Sector Count = 0, then go to (11).
10. \( 82062 \): Go to (2).
11. \( 82062 \): Set INTRQ = 1; END.

A flowchart of the READ SECTOR command is shown in Figure 12.

WRITE SECTOR

The WRITE SECTOR command is used to write one or more sectors of data to the disk from the sector buffer. Upon receipt of WRITE SECTOR command, the 82062 WDC checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation is performed and a seek takes place. The WR FAULT and DRDY lines are checked throughout the command.

When the Seek Complete (SC) line is found to be true (with or without an implied seek having occurred), the BDRQ signal is made active and the host proceeds to load the buffer. When the 82062 senses BRDY going high, the ID field with the specified cylinder number, head, and sector size is searched for. Once found, WR GATE is made active and the data is written to the disk. It is necessary to resynchronize the write data since a bit cell can extend from 295 nS to 315 nS during a write cycle. If retries are enabled (T = 0), and if the ID field cannot be found within 10 revolutions, automatic scan ID and seek commands are performed. The ID Not Found error bit is set and the command is terminated if the correct ID field is not found within 10 additional revolutions. If retries are disabled, (T = 1), and if the ID field cannot be found within 2 revolutions, the ID Not Found error bit is set and the command is terminated.

During a WRITE MULTIPLE SECTOR command (M = 1), the SECTOR NUMBER register is incremented and the SECTOR COUNT register is decremented. If the BRDY line is asserted after the first sector is transferred from the buffer, the 82062 will transfer the next sector. If BRDY is deasserted, the 82062 will set BDRQ and wait for the host processor to place more data in the buffer. In summary then, the WRITE SECTOR operation is as follows:

When \( M = 0,1 \) (WRITE SECTOR)

1. Host: Sets up parameters; issues WRITE SECTOR command.
2. \( 82062 \): Sets BDRQ = 1, DRQ = 1.
3. Host: Loads sector buffer with data.
4. \( 82062 \): Waits for BRDY = low to high.
5. \( 82062 \): Finds specified ID field; writes sector to disk.
6. \( 82062 \): If M = 0, then set INTRQ = 1; END.
7. \( 82062 \): Increment SECTOR NUMBER register; decrement SECTOR COUNT register.
8. \( 82062 \): If SECTOR = 0, then set INTRQ = 1; END.
9. \( 82062 \): Go to (2).

A flowchart of the WRITE SECTOR command is shown in Figure 13.

SCAN ID

The SCAN ID command is used to update the SECTOR/DRIVE/HEAD, SECTOR NUMBER, and CYLINDER NUMBER LOW/HIGH registers.

After the command is loaded, the Seek Complete (SC) line is sampled until it is valid. The DRDY and WR FAULT lines are also monitored throughout execution of the command. When the first ID field is
**Figure 12A. Read Sector Command Flow**

*If T bit of command = 1 then dashed path is taken after 2 index pulses.*
Figure 12B. Read Sector Command Flow

*If T bit of command = 1 then dashed path is taken.
**If T bit of command = 1 then least is for 2 index pulses.
Figure 13. Write Sector Command Flow

*If returns disabled, then dashed path is taken after 2 index pulses.
found, the ID information is loaded into the SDH, SECTOR NUMBER, and CYLINDER NUMBER registers. The internal cylinder position register is also updated. If a bad block is detected, the BAD BLOCK bit will also be set. The CRC is checked and if an error is found, the 82062 will retry up to 10 revolutions to find an error-free ID field. There is no implied seek with this command and the sector buffer is not disturbed.

A flowchart of the SCAN ID command is shown in Figure 14.

**WRITE FORMAT**

The WRITE FORMAT command is used to format one track using the Task Register File and the sector buffer. During execution of this command, the sector buffer is used for additional parameter information instead of sector data. Shown in Figure 15 is the contents of the sector buffer for a 32 sector track format with an interleave factor of two. Each sector requires a two byte sequence. The first byte designates whether a bad block mark is to be recorded in the sector's ID field. An 00H is normal; an 80H indicates a bad block mark for that sector. In the example of Figure 15, sector 04 will get a bad block mark recorded.

The second byte indicates the logical sector number to be recorded. This allows sectors to be recorded with any interleave factor desired. The remaining memory in the sector buffer may be filled with any value; its only purpose is to generate a BRDY to tell the 82062 to begin formatting the track.

An implied seek is in effect on this command. As for other commands, if the drive number has been changed, an ID field will be scanned for cylinder position information before the implied seek is performed. If no ID field can be read (because the track had been erased or because an incomplete format had been used), an ID Not Found error will result and the WRITE FORMAT command will be aborted. This can be avoided by issuing a RESTORE command before formatting.

The SECTOR COUNT register is used to hold the total number of sectors to be formatted (FFH = 255 sectors), while the SECTOR NUMBER register holds the number of bytes minus three to be used for Gap 1 and Gap 3; for instance, if the SECTOR COUNT register value is 02H and the SECTOR NUMBER register value is 00H, then 2 sectors are written and 3 bytes of 4EH are written for Gap 1 and Gap 3. The data fields are filled with FFH and the CRC is automatically generated and appended. The sector extension bit in the SDH register should not be set. After the last sector is written the track is filled with 4EH.

![Figure 14. Scan ID Command Flow](image-url)
The Gap 3 value is determined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 interleave is used. The formula for determining the minimum Gap 3 length value is:

\[ \text{Gap 3} = (2 \times M \times S) + K + E \]

- \( M \) = motor speed variation (e.g., 0.03 for \( \pm 3\% \))
- \( S \) = sector length in bytes
- \( K \) = 25 for interleave factor of 1
- \( K \) = 0 for any other interleave factor
- \( E \) = 7 if the sector is to be extended

Like all commands, a WR FAULT or drive not ready condition will terminate execution of the WRITE FORMAT command. Figure 16 shows the format that the 82062 will write on the disk.

A flowchart of the WRITE FORMAT command is shown in Figure 17.
Figure 17. Write Format Command Flow
ELECTRICAL CHARACTERISTICS

**ABSOLUTE MAXIMUM RATINGS**

- Ambient Temperature Under Bias: 0°C to 70°C
- Storage Temperature: -65°C to +150°C
- Voltage on any pin with respect to GND: -0.5V to +7V
- Power Dissipation: 1.5 Watt

*NORE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (TA = 0°C to 70°C; VCC = +5V ± 10%; GND = 0V)

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>MIN</th>
<th>MAX</th>
<th>UNIT</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIL</td>
<td>Input Leakage Current</td>
<td>±10</td>
<td>μA</td>
<td></td>
<td>VIN = VCC to 0V</td>
</tr>
<tr>
<td>IOFL</td>
<td>Output Leakage Current</td>
<td>±10</td>
<td>μA</td>
<td></td>
<td>VOUT = VCC to 0.45V</td>
</tr>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>0.8</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIL</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td>V</td>
<td></td>
<td>IOH = -100μA</td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td>V</td>
<td></td>
<td>IOL = 1.6mA</td>
</tr>
<tr>
<td>ICC</td>
<td>Supply Current</td>
<td>250</td>
<td>mA</td>
<td></td>
<td>All Outputs Open</td>
</tr>
<tr>
<td>CIN</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td>fc = 1 MHz Open</td>
<td></td>
</tr>
<tr>
<td>CV/O</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
<td>Unmeasured pins returned to GND</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For Pins 25,34,37,39</td>
<td></td>
</tr>
<tr>
<td>VIL</td>
<td>Input High Voltage</td>
<td>4.6</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>0.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRS</td>
<td>Rise Time</td>
<td>30</td>
<td>ns</td>
<td>10% to 90% points</td>
<td></td>
</tr>
</tbody>
</table>
A.C. CHARACTERISTICS \((T_A = 0^\circ \text{C} \text{ to } 70^\circ \text{C}; \ V_{CC} = +5\text{V } \pm 10\% \text{; } \text{GND} = 0\text{V})\)

HOST READ TIMING

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>MIN</th>
<th>MAX</th>
<th>UNIT</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address Stable Before RD^i</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Data Delay From RD^i</td>
<td></td>
<td>375</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RD Pulse Width</td>
<td>0.4</td>
<td>10</td>
<td>(\mu\text{s})</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>RD to Data Floating</td>
<td>20</td>
<td>200</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Address Hold Time after RD^i</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Read Recovery Time</td>
<td>300</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CS Stable before RD^i</td>
<td>0</td>
<td></td>
<td>ns</td>
<td>See Note 6</td>
</tr>
</tbody>
</table>

HOST WRITE TIMING

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>MIN</th>
<th>MAX</th>
<th>UNIT</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Address Stable Before WR^i</td>
<td>0</td>
<td>10</td>
<td>(\mu\text{s})</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>CS Stable Before WR^i</td>
<td>0</td>
<td>10</td>
<td>(\mu\text{s})</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Data Setup Time Before WR^i</td>
<td>0.2</td>
<td>10</td>
<td>(\mu\text{s})</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>WR Pulse Width</td>
<td>0.2</td>
<td>10</td>
<td>(\mu\text{s})</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Data Hold Time After WR^i</td>
<td>0</td>
<td>10</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Address Hold Time After WR^i</td>
<td>30</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>CS Hold Time After WR^i</td>
<td>0</td>
<td></td>
<td>ns</td>
<td>See Note 7</td>
</tr>
<tr>
<td>15</td>
<td>Write Recovery Time</td>
<td>1.0</td>
<td></td>
<td>(\mu\text{s})</td>
<td></td>
</tr>
</tbody>
</table>
### BUFFER READ TIMING (WRITE SECTOR COMMAND)

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>UNIT</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>BCS! to RD Valid</td>
<td>15</td>
<td></td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>RD Output Pulse Width</td>
<td>300</td>
<td>400</td>
<td>500</td>
<td>ns</td>
<td>See Note 3</td>
</tr>
<tr>
<td>18</td>
<td>Data Setup to RD!</td>
<td>140</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Data Hold from RD!</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>RD Repetition Rate</td>
<td>1.2</td>
<td>1.6</td>
<td>2.0</td>
<td>µs</td>
<td>See Note 1</td>
</tr>
<tr>
<td>21</td>
<td>RD Float from BCS!</td>
<td>15</td>
<td></td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of BUFFER READ TIMING](image)

### BUFFER WRITE TIMING (READ SECTOR COMMAND)

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>UNIT</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>BCS! to WR Valid</td>
<td>15</td>
<td></td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>WR Output Pulse Width</td>
<td>300</td>
<td>400</td>
<td>500</td>
<td>ns</td>
<td>See Note 3</td>
</tr>
<tr>
<td>24</td>
<td>Data Valid from WR!</td>
<td>150</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Data Hold from WR!</td>
<td>60</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>WR Repetition Rate</td>
<td>1.2</td>
<td>1.6</td>
<td>2.0</td>
<td>µs</td>
<td>See Note 1</td>
</tr>
<tr>
<td>27</td>
<td>WR Float from BCS!</td>
<td>15</td>
<td></td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of BUFFER WRITE TIMING](image)
### MISCELLANEOUS TIMING

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>UNIT</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>BDRQ Reset from BRDY</td>
<td>40</td>
<td></td>
<td>200</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>BRDY Pulse Width</td>
<td>800</td>
<td></td>
<td></td>
<td>ns</td>
<td>See Note 4</td>
</tr>
<tr>
<td>30</td>
<td>BCR Pulse Width</td>
<td>1.4</td>
<td>1.6</td>
<td>1.8</td>
<td>μs</td>
<td>See Note 1</td>
</tr>
<tr>
<td>31</td>
<td>STEP Pulse Width</td>
<td>8.3</td>
<td>8.4</td>
<td>8.7</td>
<td>μs</td>
<td>See Note 1</td>
</tr>
<tr>
<td>32</td>
<td>INDEX Pulse Width</td>
<td>500</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>RESET Pulse Width</td>
<td>24</td>
<td></td>
<td></td>
<td>ns</td>
<td>See Note 2</td>
</tr>
<tr>
<td>34</td>
<td>RESET(\uparrow) to BCR</td>
<td>1.6</td>
<td>3.2</td>
<td>6.4</td>
<td>μs</td>
<td>See Note 1</td>
</tr>
<tr>
<td>35</td>
<td>RESET(\downarrow) to WR, CSI</td>
<td>6.4</td>
<td></td>
<td></td>
<td>μs</td>
<td>See Note 1</td>
</tr>
<tr>
<td>36</td>
<td>WR CLOCK Frequency</td>
<td>0.25</td>
<td>5.0</td>
<td>5.25</td>
<td>MHz</td>
<td>50% Duty Cycle</td>
</tr>
<tr>
<td>37</td>
<td>RD CLOCK Frequency</td>
<td>0.25</td>
<td>5.0</td>
<td>5.25</td>
<td>MHz</td>
<td>50% Duty Cycle</td>
</tr>
</tbody>
</table>

---

[Diagrams are included showing signal waveforms for BRDY, BDRQ, BCR, STEP, INDEX, CS, WR, RESET, BCR, WR CLOCK, and RD CLOCK.]

**NOTE:**
- See Note 1
- See Note 2
- See Note 4
- See Note 5
### READ DATA TIMING

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER description</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>UNIT</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>RD CLOCK Pulse Width</td>
<td>95</td>
<td></td>
<td>2000</td>
<td>ns</td>
<td>50% Duty Cycle</td>
</tr>
<tr>
<td>39</td>
<td>RD DATA after RD CLOCK↑</td>
<td>0</td>
<td></td>
<td>T38</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>RD DATA before RD CLOCK↓</td>
<td>20</td>
<td></td>
<td>T38</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>RD DATA Pulse Width</td>
<td>40</td>
<td></td>
<td>T38</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>DRUN Pulse Width</td>
<td>30</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### WRITE DATA TIMING

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER description</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>UNIT</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>WR CLOCK Pulse Width</td>
<td>95</td>
<td></td>
<td>2000</td>
<td>ns</td>
<td>Propagation Delay</td>
</tr>
<tr>
<td>44A</td>
<td>WR CLOCK↑ to WR DATA↑</td>
<td>10</td>
<td></td>
<td>65</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>44B</td>
<td>WR CLOCK↓ to WR DATA↓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44C</td>
<td>WR CLOCK↑ to WR DATA↓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44D</td>
<td>WR CLOCK↓ to WR DATA↑</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45A</td>
<td>WR CLOCK↑ to EARLY/LATE↓</td>
<td>10</td>
<td></td>
<td>65</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>45B</td>
<td>WR CLOCK↓ to EARLY/LATE↑</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46A</td>
<td>WR CLOCK↑ to EARLY/LATE↑</td>
<td>10</td>
<td></td>
<td>65</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>46B</td>
<td>WR CLOCK↓ to EARLY/LATE↓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AC TESTING INPUT, OUTPUT WAVEFORM

INPUT OUTPUT

AC TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC 1, AND 0.45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC 1, AND 0.8V FOR A LOGIC 0

A.C. TESTING LOAD CIRCUIT

C, INCLUDES JIG CAPACITANCE

NOTES:

1. Based on WR CLOCK = 5.0 MHz.
2. 24 WR CLOCK periods = 4.8 µs at 5.0 MHz.
3. 2 WR CLOCK periods ± 100 ns.
4. When used with a DMA controller BRDY must be > 4 µs or a spurious BDRQ pulse may exist for up to 4 µs after the rising edge of BRDY.
5. WR CLOCK Frequency = RD CLOCK Frequency ± 15%.
6. RD may be asserted before CS as long as it remains active for at least the minimum T3 pulse width after CS is asserted.
7. WR may be asserted before CS as long as it remains active for at least the minimum T11 pulse width after CS is asserted.
The 82064 Winchester Disk Controller (WDC) with on-chip error detection and correction circuitry interfaces microprocessor systems to 5½" Winchester disk drives. It is socket and software compatible with the 82062 Winchester Disk Controller, and additionally includes on-chip ECC, support for drives with up to 2k tracks, and has an additional control signal which eliminates an external decoder.

The 82064 is fabricated on Intel’s advanced HMOS III technology and is available in 40-pin CERDIP and plastic packages.
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCS</td>
<td>1</td>
<td>O</td>
<td>BUFFER CHIP SELECT: Output used to enable reading or writing of the external sector buffer by the 82064. When low, the host should not be able to drive the 82064 data bus, RD, or WR lines.</td>
</tr>
<tr>
<td>BCR</td>
<td>2</td>
<td>O</td>
<td>BUFFER COUNTER RESET: Output that is asserted by the 82064 prior to read/write operation. This pin is asserted whenever BCS changes state. Used to reset the address counter of the buffer memory.</td>
</tr>
<tr>
<td>INTRQ</td>
<td>3</td>
<td>O</td>
<td>INTERRUPT REQUEST: Interrupt generated by the 82064 upon command termination. It is reset when the STATUS register is read, or a new command is written to the COMMAND register. Optionally signifies when a data transfer is required on Read Sector commands.</td>
</tr>
<tr>
<td>SDHLE</td>
<td>4</td>
<td>O</td>
<td>SDHLE is asserted when the SDH register is written by the host.</td>
</tr>
<tr>
<td>RESET</td>
<td>5</td>
<td>I</td>
<td>RESET: Initializes the controller and clears all status flags. Does not clear the Task Register File.</td>
</tr>
<tr>
<td>RD</td>
<td>6</td>
<td>I/O</td>
<td>READ: Tri-state, bi-directional signal. As an input, RD controls the transfer of information from the 82064 registers to the host. RD is an output when the 82064 is reading data from the sector buffer (BCS low).</td>
</tr>
<tr>
<td>WR</td>
<td>7</td>
<td>I/O</td>
<td>WRITE: Tri-state, bi-directional signal. As an input, WR controls the transfer of command or task information into the 82064 registers. WR is an output when the 82064 is writing data to the sector buffer (BCS low).</td>
</tr>
<tr>
<td>CS</td>
<td>8</td>
<td>I</td>
<td>CHIP SELECT: Enables RD and WR as inputs for access to the Task Registers. It has no effect once a disk command starts.</td>
</tr>
<tr>
<td>A0-2</td>
<td>9-11</td>
<td>I</td>
<td>ADDRESS: Used to select a register from the task register file.</td>
</tr>
<tr>
<td>DB0-7</td>
<td>12-19</td>
<td>I/O</td>
<td>DATA BUS: Tri-state, bi-directional 8-bit Data Bus with control determined by BCS. When BCS is high the microprocessor has full control of the data bus for reading and writing the Task Register File. When BCS is low the 82064 controls the data bus to transfer to or from the buffer.</td>
</tr>
<tr>
<td>Vss</td>
<td>20</td>
<td></td>
<td>Ground</td>
</tr>
<tr>
<td>WR DATA</td>
<td>21</td>
<td>O</td>
<td>WRITE DATA: Output that shifts out MFM data at a rate determined by Write Clock. Requires an external D flip-flop clocked at 10 MHz. The output has an active pullup and pulldown that can sink 4.8 mA.</td>
</tr>
<tr>
<td>LATE</td>
<td>22</td>
<td>O</td>
<td>LATE: Output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders.</td>
</tr>
<tr>
<td>EARLY</td>
<td>23</td>
<td>O</td>
<td>EARLY: Output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders.</td>
</tr>
</tbody>
</table>
Table 1. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>WR GATE</td>
<td>24</td>
<td>O</td>
<td>WRITE GATE: High when write data is valid. WR GATE goes low if the WR FAULT input is active. This output is used by the drive to enable head write current.</td>
</tr>
<tr>
<td>WR CLOCK</td>
<td>25</td>
<td>I</td>
<td>WRITE CLOCK: Clock input used to derive the write data rate. Frequency = 5 MHz for the ST506 interface. See Note 1.</td>
</tr>
<tr>
<td>DIR</td>
<td>26</td>
<td>O</td>
<td>DIRECTION: High level on this output tells the drive to move the head inward (increasing cylinder number). The state of this signal is determined by the 82064's internal comparison of actual cylinder location vs. desired cylinder.</td>
</tr>
<tr>
<td>STEP</td>
<td>27</td>
<td>O</td>
<td>STEP: This signal is used to move the drive head to another cylinder at a programmable frequency. Pulse width = 1.6 μs for a step rate of 3.2 μs/step, and 8.4 μs for all other step rates.</td>
</tr>
<tr>
<td>DRDY</td>
<td>28</td>
<td>I</td>
<td>DRIVE READY: If DRDY from the drive goes low, the command will be terminated.</td>
</tr>
<tr>
<td>INDEX</td>
<td>29</td>
<td>I</td>
<td>INDEX: Signal from the drive indicating the beginning of a track. It is used by the 82064 during formatting, and for counting retries. Index is edge triggered. Only the rising edge is valid.</td>
</tr>
<tr>
<td>WR FAULT</td>
<td>30</td>
<td>I</td>
<td>WRITE FAULT: An error input to the 82064 which indicates a fault condition at the drive. If WR FAULT from the drive goes high, the command will be terminated.</td>
</tr>
<tr>
<td>TRACK 000</td>
<td>31</td>
<td>I</td>
<td>TRACK ZERO: Signal from the drive which indicates that the head is at the outermost cylinder. Used to verify proper completion of a RESTORE command.</td>
</tr>
<tr>
<td>SC</td>
<td>32</td>
<td>I</td>
<td>SEEK COMPLETE: Signal from the drive indicating to the 82064 that the drive head has settled and that reads or writes can be made. SC is edge triggered. Only the rising edge is valid.</td>
</tr>
<tr>
<td>RWC</td>
<td>33</td>
<td>O</td>
<td>REDUCED WRITE CURRENT: Signal goes high for all cylinder numbers above the value programmed in the Write Precomp Cylinder register. It is used by the precompensation logic and by the drive to reduce the effects of bit shifting.</td>
</tr>
<tr>
<td>DRUN</td>
<td>34</td>
<td>I</td>
<td>DATA RUN: This signal informs the 82064 when a field of all ones or all zeroes has been detected in the read data stream by an external one-shot. This indicates the beginning of an ID field. RD GATE is brought high when DRUN is sampled high for 16 clock periods. See Note 1.</td>
</tr>
<tr>
<td>BRDY</td>
<td>35</td>
<td>I</td>
<td>BUFFER READY: Input used to signal the controller that the buffer is ready for reading (full), or writing (empty), by the host μP. Only the rising edge indicates the condition.</td>
</tr>
</tbody>
</table>
Table 1. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDRQ</td>
<td>36</td>
<td>O</td>
<td>BUFFER DATA REQUEST: Activated during Read or Write commands when a data transfer between the host and the 82064's sector buffer is required. Typically used as a DMA request line, or to generate an interrupt.</td>
</tr>
<tr>
<td>RD DATA</td>
<td>37</td>
<td>I</td>
<td>READ DATA: Single ended input that accepts MFM data from the drive. See note 1.</td>
</tr>
<tr>
<td>RD GATE</td>
<td>38</td>
<td>O</td>
<td>READ GATE: Output that is asserted when a search for an address mark is initiated. It remains asserted until the end of the ID or data field.</td>
</tr>
<tr>
<td>RD CLOCK</td>
<td>39</td>
<td>I</td>
<td>READ CLOCK: Clock input derived from the external data recovery circuits. See note 1.</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td>I</td>
<td>D.C. POWER: +5V.</td>
</tr>
</tbody>
</table>

Note 1: This pin requires input levels that are not TTL compatible. These lines can be interfaced to TTL with a pull-up resistor. Too small a resistor will produce a VIL level that is too high. Too large a resistor will degrade the signal's rise time. A minimum value for the resistor is determined as follows:

\[
\frac{(V_{CC\ max}) - (82064\ V_{IL\ max})}{(TTL\ I_{OL\ min}) - (82064\ I_{LL\ max})}
\]

This would typically be:

\[
\frac{5.5V - 0.5V}{1.6\ mA - 10\ \mu A} = 3\ \text{k}\Omega
\]

FUNCTIONAL DESCRIPTION

The Intel 82064 Winchester Disk Controller (WDC) interfaces microprocessor systems to Winchester disk drives that use the Seagate Technology ST506/ST412 interface. The device translates parallel data from the microprocessor to a 5 Mbit/sec, MFM-encoded serial bit stream. It provides all of the drive control logic and control signals which simplify the design of external data separation and write pre-compensation circuitry. The 82064 is designed to interface to the host processor through an external sector buffer.

On-chip error detection algorithms include the CRC/CCITT and a 32-bit computer generated ECC polynomial. If the ECC code is selected, the 82064 provides three possible error handling techniques if an error is detected during a read operation:
1. Automatically correct the data in the sector buffer, providing the host with good information.
2. Provide the host with the error location and pattern, allowing the host to correct the error.
3. Take no action other than setting the error flag.

The 82064 is software compatible with the 82062.

INTERNAL ARCHITECTURE

The internal architecture of the 82064 is shown in more detail in Figure 3. It is made up of seven major blocks as described below.

PLA Controller

The PLA interprets commands and provides all control functions. It is synchronized with WR CLOCK.

Magnitude Comparator

An 11-bit magnitude comparator is used to calculate the direction and number of steps needed to move the heads from the present to the desired cylinder position. It compares the cylinder number in the task file to the internal "present position" cylinder number.

A separate high-speed equivalence comparator is used to compare ID field bytes when searching for a sector ID field.
**CRC and ECC Generator and Checker**

The 82064 provides two options for protecting the integrity of the data field. The data field may have either a CRC (SDH register, bit 7 = 0), or a 32-bit ECC (SDH register, bit 7 = 1) appended to it. The ID field is always protected by a CRC.

The CRC mode provides a means of verifying the accuracy of the data read from the disk, but does not attempt to correct it. The CRC generator computes and checks cyclic redundancy check characters that are written and read from the disk after ID and data fields. The polynomial used is:

\[ x^{16} + x^{12} + x^5 + 1 \]

The CRC register is preset to all one's before computation starts.

If the CRC character generated while reading the data does not equal the one previously written an error exists. If an ID field CRC error occurs the “ID not found” bit in the error register will be set. If a data field CRC error occurs the “ECC/CRC” bit in the error register will be set.

The ECC mode is only applicable to the data field. It provides the user with the ability to detect and correct errors in the data field automatically. The commands and registers which must be considered when ECC is used are:

1. SDH Register, bit 7 (CRC/ECC)
2. READ SECTOR Command, bit 0 (T)
3. READ SECTOR and WRITE SECTOR Commands, bit 1 (L)
4. COMPUTE CORRECTION Command
5. SET PARAMETER Command
6. STATUS Register, bit 2 - error correction successful
7. STATUS Register, bit 0 - error occurred
8. ERROR Register, bit 6 - uncorrectable error

To enable the ECC mode, bit 7 of the SDH register must be set to one.
Bit 0 (T) of the READ Command controls whether or not error correction is attempted. When T = 0 and an error is detected, the 82064 tries up to 10 times to correct the error. If the error is successfully corrected, bit 2 of the STATUS Register is set. The host can interrogate the status register and detect that an error occurred and was corrected. If the error was not correctable, bit 6 of the ERROR Register is set. If the correction span was set to 5 bits, the host may now execute the SET PARAMETER Command to change the correction span to 11 bits, and attempt the read again. If the error persists, the host can read the data, but it will contain errors.

When T = 1 and an error is detected, no attempt is made to correct it. Bit 0 of the STATUS Register and bit 6 of the ERROR Register are set. The user now has two choices:

1. Ignore the error and make no attempt to correct it.
2. Use the COMPUTE CORRECTION Command to determine the location and pattern of the error, and correct it within the user’s program.

When the COMPUTE CORRECTION Command is implemented, it must be done before executing any command which can alter the contents of the ECC Register. The READ SECTOR, WRITE SECTOR, SCAN ID, and FORMAT Commands will alter this register and correction will be impossible. The COMPUTE CORRECTION Command may determine that the error is uncorrectable, at which point the error bits in the STATUS and ERROR Registers are set.

Although ECC generation starts with the first bit of the F8H byte in the data field, the actual ECC bytes written will be the same as if the A1H byte was included. The ECC polynomial used is:

\[ x^{32} + x^{28} + x^{26} + x^{19} + x^{17} + x^{10} + x^6 + x^2 + 1 \]

For automatic error correction, the external sector buffer must be implemented with a static RAM and counter, not with a FIFO.

The SET PARAMETER Command is used to select a 5-bit or 11-bit correction span.

When the L Bit (bit 1) of the READ SECTOR and WRITE SECTOR commands is set to one, they are referred to as READ LONG and WRITE LONG commands. For these commands, no CRC or ECC characters are generated or checked by the 82064. In effect, the data field is extended by 4 bytes which are passed to/from the sector buffer.

With proper use of the WRITE SECTOR, READ LONG, WRITE LONG, and READ SECTOR Commands, a diagnostic routine may be developed to test the accuracy of the error correction process.

MF M ENCODER/DECODER

Encodes and decodes MFM data to be written/read from the drive. The MFM encoder operates from WR CLOCK, a clock having a frequency equal to the bit rate. The MFM decoder operates from RD CLOCK, a bit rate clock generated by the external data separator. RD CLOCK and WR CLOCK need not be synchronous.

The MFM encoder also generates the write precompensation control signals. Depending on the bit pattern of the data, EARLY or LATE may be asserted. External circuitry uses these signals to compensate for drift caused by the influence one bit has over another. More information on the use of the EARLY and LATE control signals can be found in the section which describes the drive interface.

Address Mark (AM) Detection

An address mark is comprised of two unique bytes preceding both the ID field and the data field. The first byte is used for resynchronization. The second byte indicates whether it is an ID field or a data field.

The first byte, A1H, normally has a clock pattern of 0EH; however, one clock pulse has been suppressed, making it 0AH. With this pattern, the AM detector knows it is looking at an address mark. It now examines the next byte to determine if it is an ID or data field. If this byte is 111101XX or 111111XX it is an ID field. Bits 3, 1, and 0 are the high order cylinder number bits. If the second byte is F8H, it is a data field.

Host/Buffer Interface Control

The primary interface between the host processor and the 82064 is an 8-bit bi-directional bus. This bus is used to transmit and receive data for both the 82064 and the sector buffer. The sector buffer consists of a static RAM and counter. Since the 82064 makes the bus active when accessing the sector buffer, a transceiver must be used to isolate the host during this time. Figure 4 illustrates a typical interface with a sector buffer. Whenever the 82064 is not using the sector buffer, the BUFFER CHIP SELECT (BCS) is high (disabled). This allows the host access to the 82064’s Task Register File and to the sector buffer. A decoder is used to generate BCS when A0–2 is ‘000’, an unused address in the 82064. A binary counter is enabled whenever RD or WR go active. The location within the sector buffer which is addressed by the counter will be accessed. The counter will be incremented by the trailing edge of the RD or WR. This allows the host to access se-
sequential bytes within the sector buffer. The decoder also generates a CS for the 82064 whenever \( A_{0-2} \) does not equal '000', allowing access to the 82064's internal Task Register File while keeping the sector buffer tri-stated.

During a WRITE SECTOR Command, the host processor sets up data in the Task Register File and then issues the command. The 82064 asserts BUFFER COUNTER RESET (BCR) to reset the counter. It then generates a status to inform the host that it can load the sector buffer with data to be written. When the counter reaches its maximum count, the BUFFER READY (BRDY) signal is asserted by the carry out of the counter, informing the 82064 that the sector buffer is full. (BRDY is a rising edge triggered signal which will be ignored if asserted before the 82064 asserts BCR.) BCS is then asserted, disconnecting the host through the transceivers, and the RD and WR lines become outputs from the 82064 to allow access to the sector buffer. When the 82064 is done using the buffer, it deasserts BCS which again allows the host to access the local bus. The READ SECTOR command operates in a similar manner, except the buffer is loaded by the 82064 instead of the host.

Another control signal, BUFFER DATA REQUEST (BDRQ), can be used with a DMA controller or in a programmed I/O environment. In either case, it indicates that the 82064 is ready to send or receive data. For further explanation, refer to the individual command descriptions and the A.C. Characteristics.
When INTRQ is asserted, the host is signaled that execution of a command has terminated (either a normal termination or an aborted command). For the READ SECTOR command, interrupts may be programmed to be asserted either at the termination of the command, or when BDRQ is asserted. INTRQ will remain active until the host reads the STATUS register to determine the cause of the termination, or writes a new command into the COMMAND register.

The 82064 asserts SDHLE whenever the SDH register is being written. This signal can be used to latch the drive and head select information in an external register for decoding. Figure 5 illustrates one method.

**Drive Interface**

The drive side of the 82064 WDC requires three sections of external logic. These are the control line buffer/receivers, data separator, and write precompensation. Figure 5 illustrates a drive interface.

The buffer/receivers condition the control lines to be driven down the cable to the drive. The control lines are typically single-ended, resistor terminated, TTL levels. The data lines to and from the drive also require buffering. This is typically done with differential RS-422 drivers. The interface specification for the drive will be found in the drive manufacturer's OEM
The 82064 supplies TTL compatible signals, and will interface to most buffer/driver devices.

The data recovery circuits consist of a phase locked loop, data separator, and associated components. The 82064 interacts with the data separator through the DATA RUN (DRUN) and RD GATE signals. A block diagram of a typical data separator circuit is shown in Figure 6. Read data from the drive is presented to the RD DATA input of the 82064, the reference multiplexor, and a retriggerable one shot. The RD GATE output will be deasserted when the 82064 is not inspecting data. The PLL should remain locked to the reference clock.

When any READ or WRITE command is initiated and a search for an address mark begins, the DRUN input is examined. The DRUN one-shot is set for slightly longer than one bit time, allowing it to retrigger constantly on a field of all ones or all zeroes. An internal counter times out to see that DRUN is asserted for two byte times. RD GATE is asserted by the 82064, switching the data separator to lock on to the incoming data stream. If DRUN is deasserted prior to an additional seven byte times, RD GATE is deasserted and the process is repeated. RD GATE will remain asserted until a non-zero, non-address mark byte is detected. The 82064 will then deassert RD GATE for two byte times to allow the PLL to lock back on the reference clock, and start the DRUN search again. If an address mark is detected, RD GATE remains asserted and the command will continue searching for the proper ID field. This sequence is shown in the flow chart in Figure 7.

Drift may also be caused by the bit pattern. With certain combinations of ones and zeroes some of the bits can drift far enough apart to be difficult to read without error. This phenomenon can be minimized by using EARLY and LATE as described below. The 82064 examines three bits, the last one written, the one being written, and the next one to be written. From this, it determines whether to assert EARLY or LATE. Since the bit leaving the 82064 has already been written, it is too late to make it early. Therefore, the external delay circuit must be as follows:

- EARLY asserted and LATE deasserted = no delay
- EARLY deasserted and LATE deasserted = one unit delay (typically 12–15 ns)
- EARLY deasserted and LATE asserted = two units delay (typically 24–30 ns)

EARLY and LATE are always active, and should be gated externally by the RWC signal. Figure 8 illustrates one method of using these signals.

The write precompensation circuitry is designed to reduce the drift in the data caused by interaction between bits. It is divided into two parts, REDUCED WRITE CURRENT (RWC) and EARLY/LATE writing of bits. A block diagram of a typical write precompensation circuit is shown in Figure 8.
Figure 7a. PLL Control Sequence For ID Field
Figure 7b. PLL Control Sequence For Data Field
**TASK REGISTER FILE**

The Task Register File is a bank of registers used to hold parameter information pertaining to each command, status information, and the command itself. These registers and their addresses are:

<table>
<thead>
<tr>
<th>A2</th>
<th>A1</th>
<th>A0</th>
<th>READ</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>BUS TRISTATED</td>
<td>BUS TRISTATED</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>ERROR REGISTER</td>
<td>REDUCE WRITE CURRENT</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>SECTOR COUNT</td>
<td>SECTOR COUNT</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>SECTOR NUMBER</td>
<td>SECTOR NUMBER</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CYLINDER LOW</td>
<td>CYLINDER LOW</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>CYLINDER HIGH</td>
<td>CYLINDER HIGH</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>SDH</td>
<td>SDH</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>STATUS</td>
<td>COMMAND</td>
</tr>
</tbody>
</table>

**NOTE:**
These registers are not cleared by RESET being asserted.

**ERROR REGISTER**

This read only register contains specific error information after the termination of a command. The bits are defined as follows:

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB</td>
<td>CRC/ECC</td>
<td>0</td>
<td>ID</td>
<td>0</td>
<td>AC</td>
<td>TK000</td>
<td>DAM</td>
</tr>
</tbody>
</table>

**Bit 7 - Bad Block Detect (BB)**

This bit is set when an ID field has been encountered that contains a bad block mark. It is used for bad sector mapping.

**Bit 6 - CRC/ECC Data Field Error (CRC/ECC)**

When in the CRC mode (SDH register, bit 7 = 0), this bit is set when a CRC error occurs in the data field. When retries are enabled, ten more attempts are made to read the sector correctly. If none of these attempts are successful bit 0 in the STATUS register is also set. If one of the attempts is successful, the CRC/ECC error bit remains set to inform the host that a marginal condition exists; however, bit 0 in the STATUS register is not set.

When in the ECC mode (SDH register, bit 7 = 1), this bit is set when the first non-zero syndrome is detected. When retries are enabled, up to ten attempts are made to correct the error. If the error is successfully corrected, this bit remains set; however, bit 2 of the STATUS register is also set to inform the host that the error has been corrected. If the error is not correctable, the CRC/ECC error bit remains set and bit 0 of the STATUS register is also set.

The data may be read even if uncorrectable errors exist.

**NOTE:** If the long mode (L) bit is set in the READ or WRITE command, no error checking is performed.

**Bit 5 - Reserved**

Not used. Forced to zero.
Bit 4 - ID Not Found (ID)

This bit is set to indicate that the correct cylinder, head, sector, or size parameter could not be found, or that a CRC error occurred in the ID field. This bit is set on the first failure and remains set even if the error is recovered on a retry. When recovery is unsuccessful, the Error bit (bit 0) of the STATUS register is also set.

For a SCAN ID command with retries enabled (T = 0), the Error bit in the STATUS register is set after ten unsuccessful attempts have been made to find the correct ID. With retries disabled (T = 1), only two attempts are made before setting the Error bit.

For a READ or WRITE command with retries enabled (T = 0), ten attempts are made to find the correct ID field. If there is still an error on the tenth try, an auto-scan and auto-seek are performed. Then ten more retries are made before setting the Error bit. When retries are disabled (T = 1), only two tries are made. No auto-scan or auto-seek operations are performed.

Bit 3 - Reserved

Not used. Forced to zero.

Bit 2 - Aborted Command (AC)

Command execution is aborted and this bit is set if a command was issued while DRDY is deasserted or WR FAULT is asserted. This bit will also be set if an undefined command is written to the COMMAND register; however, an implied seek will be executed.

Bit 1 - Track 000 Error (TK000)

This bit is set during the execution of a RESTORE command if the TRACK 000 pin has not gone active after the issuance of 2047 step pulses.

Bit 0 - Data Address Mark (DAM) Not Found

This bit is set during the execution of a READ SECTOR command if the DAM is not found following the proper sector ID.

REDUCE WRITE CURRENT REGISTER

This register is used to define the cylinder number where the RWC output (Pin 33) is asserted.

The value (00-FFH) loaded into this cylinder is internally multiplied by four to specify the actual cylinder where RWC is asserted. Thus a value of 01H will cause RWC to be asserted on cylinder 04H, 02H on cylinder 08H, . . . , 9CH on cylinder 270H, 9DH on cylinder 274H, and so on. RWC will be asserted when the present cylinder is greater than or equal to the cylinder indicated by this register. For example, the ST506 interface requires precomp on cylinder 80H and above. Therefore, the REDUCE WRITE CURRENT register should be loaded with 20H.

A value of FFH causes RWC to remain deasserted, regardless of the actual cylinder number.

SECTOR COUNT REGISTER

This register is used to define the number of sectors that need to be transferred to the buffer during a READ MULTIPLE SECTOR or WRITE MULTIPLE SECTOR command.

The value contained in the register is decremented after each sector is transferred to/from the sector buffer. A zero represents a 256 sector transfer, a one a one sector transfer, etc. This register is a "don't care" when single sector commands are specified.

SECTOR NUMBER REGISTER

This register holds the sector number of the desired sector.

For a multiple sector command, it specifies the first sector to be transferred. It is incremented after each sector is transferred to/from the sector buffer. The SECTOR NUMBER register may contain any value from 0 to 255.

The SECTOR NUMBER register is also used to program the Gap 1 and Gap 3 lengths to be used when formatting a disk. See the WRITE FORMAT command description for further explanation.
CYLINDER NUMBER LOW REGISTER

This register holds the lower byte of the desired cylinder number.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LS BYT E OF CYL. NUMBER</td>
</tr>
</tbody>
</table>

It is used with the CYLINDER NUMBER HIGH register to specify the desired cylinder number over a range of 0 to 2047.

CYLINDER NUMBER HIGH REGISTER

This register holds the three most significant bits of the desired cylinder number.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
</tbody>
</table>

The CYLINDER NUMBER LOW/HIGH register pair determine where the R/W heads are to be positioned. The host writes the desired cylinder number into these registers. Internal to the 82064 is another pair of registers that hold the present head location. When any command other than a RESTORE is executed, the internal head location registers are compared to the CYLINDER NUMBER registers to determine how many cylinders to move the heads and in what direction.

The internal head location registers are updated to equal the CYLINDER NUMBER registers after the completion of the seek.

When a RESTORE command is executed, the internal head location registers are reset to zero while DIR and STEP move the heads to track zero.

SECTOR/DRI VE/HEAD (SDH) REGISTER

The SDH register contains the desired sector size, drive number, and head parameters. The format is shown in Figure 9. The EXT bit (bit 7) is used to select between the CRC or ECC mode. When bit 7 = 0 the ECC mode is selected for the data field. When bit 7 = 1 the CRC mode is selected.

The SDH byte written in the ID field of the disk by the FORMAT command is different than the SDH register contents. The recorded SDH byte does not have the drive number recorded, but does have the bad block mark written. The format of the SDH byte written on the disk is:

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD B.</td>
<td>SIZE</td>
<td>0</td>
<td>0</td>
<td>HEAD</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

STATUS REGISTER

The status register is used to inform the host of certain events performed by the 82064, as well as reporting status from the drive control lines. Reading the STATUS register deasserts INTRQ. The format is:

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>READY</td>
<td>WF</td>
<td>SC</td>
<td>DRQ</td>
<td>DWC</td>
<td>CIP</td>
<td>ERR</td>
</tr>
</tbody>
</table>

Bit 7 - Busy

This bit is asserted when a command is written into the COMMAND register and, except for the READ command, is deasserted at the end of the command. When executing a READ command, Busy will be deasserted when the sector buffer is full. Commands should not be loaded into the COMMAND register when Busy is set. When the Busy bit is set, no other bits in the STATUS or ERROR registers are valid.

Bit 6 - Ready

This bit reflects the status of DRDY (pin 28). When this bit equals zero, the command is aborted and the status of this bit is latched.

Bit 5 - Write Fault (WF)

This bit reflects the status of WR FAULT (pin 30). When this bit equals one the command is aborted, INTRQ is asserted, and the status of this bit is latched.

Bit 4 - Seek Complete (SC)

This bit reflects the status of SC (pin 32). When a seek or implied seek has been initiated by a command, execution of the command pauses until the seek is complete. This bit is latched after an aborted command error.
NOTE:
Drive select and head select lines must be generated externally. Figure 3 represents one method of achieving this.

Figure 9. SDH Register Format

Bit 3 - Data Request (DRQ)

The DRQ bit reflects the status of BDRQ (pin 36). It is asserted when the sector buffer must be written into or read from. DRQ and BDRQ remain asserted until BRDY indicates that the sector buffer has been filled or emptied, depending upon the command. BDRQ can be used for DMA interfacing, while DRQ is used in a programmed I/O environment.

Bit 2 - Data Was Corrected (DWC)

When set, this bit indicates that an ECC error has been detected during a read operation, and that the data in the sector buffer has been corrected. This provides the user with an indication that there may be a marginal condition within the drive before the errors become uncorrectable. This bit is forced to zero when not in the ECC mode.

Bit 1 - Command In Progress (CIP)

When this bit is set a command is being executed and a new command should not be loaded. Although a command is being executed, the sector buffer is still available for access by the host. When the 82064 is no longer Busy (bit 7 = 0) the STATUS register can be read. If other registers are read while CIP is set the contents of the STATUS register will be returned.

Bit 0 - Error

This bit is set whenever any bits in the ERROR register are set. It is the logical 'or' of the bits in the ERROR register and may be used by the host processor to quickly check for nonrecoverable errors. The host must read the ERROR register to determine what type of error occurred. This bit is reset when a new command is written into the COMMAND register.

COMMAND REGISTER

The command to be executed is written into this write-only register:

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The command sets Busy and CIP, and begins to execute as soon as it is written into this register. Therefore, all necessary information should be loaded into the Task Register File prior to entering the command. Any attempt to write a register will be ignored until command execution has terminated, as indicated by the CIP bit being cleared. INTRQ is deasserted when the COMMAND register is written.
### COMMAND 7 6 5 4 3 2 1 0

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTORE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
</tr>
<tr>
<td>SEEK</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
</tr>
<tr>
<td>READ SECTOR</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>M</td>
<td>L</td>
<td>T</td>
</tr>
<tr>
<td>WRITE SECTOR</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>M</td>
<td>L</td>
<td>T</td>
</tr>
<tr>
<td>SCAN ID</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>WRITE FORMAT</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>COMPUTE CORRECTION</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>SET PARAMETER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
</tr>
</tbody>
</table>

### RESTORE COMMAND

The RESTORE command is used to position the R/W heads over track zero. It is usually issued by the host when a drive has just been turned on. The 82064 forces an auto-restore when a FORMAT command has been issued following a drive number change.

The actual step rate used for the RESTORE command is determined by the seek complete time. A step pulse is issued and the 82064 waits for a rising edge on the SC line before issuing the next pulse. If the rising edge of SC has not occurred within ten revolutions (INDEX pulses) the 82064 switches to sensing the level of SC. If after 2047 step pulses the TRACK 000 line does not go active the 82064 will set the TRACK 000 bit in the ERROR register, assert INTRQ, and terminate execution of the command. An interrupt will also occur if WR FAULT is asserted on DRDY is deasserted at any time during execution.

The rate field specified (R3...0) is stored in an internal register for future use in commands with implied seeks.

A flowchart of the RESTORE command is shown in Figure 10.

### SEEK COMMAND

The SEEK command can be used for overlapping seeks on multiple drives. The step rate used is taken from the Rate Field of the command, and is stored in an internal register for future use by those commands with implied seek capability.

The direction and number of step pulses needed are calculated by comparing the contents of the CYLINDER NUMBER registers in the Task Register File to the present cylinder position stored internally. After all the step pulses have been issued the present cylinder position is updated, INTRQ is asserted, and the command terminated.

### Parameters

- **R3...0** = Stepping Rate Field
  - **For 5 MHz WR CLOCK:**
    - R3...0 = 0000 35 μs
    - 0001 0.5 ms
    - 0010 1.0 ms
    - 0011 1.5 ms
    - 0100 2.0 ms
    - 0101 2.5 ms
    - 0110 3.0 ms
    - 0111 3.5 ms
    - 1000 4.0 ms
    - 1001 4.5 ms
    - 1010 5.0 ms
    - 1011 5.5 ms
    - 1100 6.0 ms
    - 1101 6.5 ms
    - 1110 3.2 μs
    - 1111 16 μs
  - **I** = Interrupt Control
    - I = 0 INTRQ occurs with BDRQ/DRQ indicating the sector buffer is full. Valid only when M = 0.
    - I = 1 INTRQ occurs when the command is completed and the host has read the sector buffer.
  - **M** = Multiple Sector Flag
    - M = 0 Transfer one sector. Ignore the SECTOR COUNT register.
    - M = 1 Transfer multiple sectors.
  - **L** = Long Mode
    - L = 0 Normal mode. Normal CRC or ECC functions are performed.
    - L = 1 Long mode. No CRC or ECC bytes are developed or error checking performed on the data field. The 82064 appends the four additional bytes supplied by the host or disk to the data field.
  - **T** = Retry Enable
    - T = 0 Enable retries.
If DRDY is deasserted or WR FAULT is asserted during the execution of the command, INTRQ is asserted and the command aborts setting the AC bit in the ERROR register.

If an implied seek is performed, the step rate indicated by the rate field is used for all but the last step pulse. On the last pulse, the command execution continues until the rising edge of SC is detected. If 10 INDEX pulses are received without a rising edge of SC, the 82064 will switch to sensing the level of SC.

A flowchart of the SEEK command flow is shown in Figure 11.

READ SECTOR

The READ SECTOR command is used to transfer one or more sectors of data from the disk to the sector buffer. Upon receipt of the command, the 82064 checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation takes place, and a seek is initiated. As stated in the description of the SEEK command, if an implied seek occurs, the step rate specified by the rate field is used for all but the last step pulse. On the last step pulse the seek continues until the rising edge of SC is detected.

If the 82064 detects a change in the drive number since the last command, an auto-scan ID is performed. This updates the internal cylinder position register to reflect the current drive before the seek begins.

After the 82064 senses SC (with or without an implied seek) it must find an ID field with the correct cylinder number, head, sector size, and CRC. If retries are enabled (T = 0), ten attempts are made to find the correct ID field. If there is still an error on the tenth try, an auto-scan ID and auto-seek are performed. Then ten more retries are attempted before setting the ID Not Found error bit. When retries are disabled (T = 1) only two tries are made. No auto-scan or auto-seek operations are performed.

When the data address mark (DAM) is found, the 82064 is ready to transfer data into the sector buffer. When the disk has filled the sector buffer, the 82064 asserts BDRQ and DRQ and then checks the I flag. If I = 0, INTRQ is asserted, signaling the host to read the contents of the sector buffer. If I = 1, INTRQ occurs after the host has read the sector buffer and the command has terminated. If after successfully reading the ID field, the DAM is not found the DAM Not Found bit in the ERROR register is set.
Figure 11. Seek Command Flow
An optional M flag can be set for multiple sector transfers. When \( M = 0 \), one sector is transferred and the SECTOR COUNT register is ignored. When \( M = 1 \), multiple sectors are transferred. After each sector is transferred, the 82064 decrements the SECTOR COUNT register and increments the SECTOR NUMBER register. The next logical sector is transferred regardless of any interleave. Sectors are numbered during the FORMAT command by a byte in the ID field.

For the 82064 to make multiple sector transfers to the sector buffer, the BRDY signal must be toggled from low to high for each sector. The transfers continue until the SECTOR COUNT register equal zero. If the SECTOR COUNT is not zero (indicating more sectors remain to be read), and the sector buffer is full, BDRQ will be asserted and the host must unload the sector buffer. Once this occurs, the sector buffer is free to accept the next sector.

WR FAULT and DRDY are monitored throughout the command execution. If WR FAULT is asserted or DRDY is deasserted, the command will terminate and the Aborted Command bit in the ERROR register will be set. For a description of the error checking procedure on the data field see the explanation in the section entitled "CRC and ECC Generator and Checker."

Both the READ and WRITE commands feature a "simulated completion" to ease programming. BDRQ, DRQ, and INTRA are generated in a normal manner upon detection of an error condition. This allows the same program flow for successful or unsuccessful completion of a command.

In summary then, the READ SECTOR operation is as follows:

When \( M = 0 \) (Single Sector Read)
1. HOST: Sets up parameters. Issues READ SECTOR command.
2. 82064: Asserts BCR.
3. 82064: Finds sector specified. Asserts BCR and BCS. Transfers data to sector buffer.
4. 82064: Asserts BCR. Deasserts BCS.
5. 82064: Asserts BDRQ and DRQ.
6. 82064: If \( I = 1 \) then go to 9.
7. HOST: Read contents of sector buffer.
8. 82064: Wait for BRDY, then assert INTRQ. End.
9. 82064: Assert INTRQ.
10. HOST: Read contents of sector buffer. End.

When \( M = 1 \) (Multiple Sector Read)
1. HOST: Sets up parameters. Issues READ SECTOR command.
2. 82064: Asserts BCR.
3. 82064: Finds sector specified. Asserts BCR and BCS. Transfers data to sector buffer.
4. 82064: Asserts BCR. Deasserts BCS.
5. 82064: Asserts BDRQ and DRQ.
7. SECTOR BUFFER: Indicates data has been transferred by asserting BRDY.
8. 82064: When BRDY is asserted, decrement SECTOR COUNT, increment SECTOR NUMBER. If SECTOR COUNT = 0, go to 11.
9. 82064: Go to 2.
10. 82064: Assert INTRQ.

A flowchart of the READ SECTOR command is shown in Figure 12.

WRITE SECTOR
The WRITE SECTOR command is used to write one or more sectors of data from the sector buffer to the disk. Upon receipt of the command, the 82064 checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation takes place, and a seek is initiated. As stated in the description of the SEEK command, if an implied seek occurs, the step rate specified by the rate field is used for all but the last step pulse. On the last step pulse the seek continues until the rising edge of SC is detected.

If the 82064 detects a change in the drive number since the last command, an auto-scan ID is performed. This updates the internal cylinder position register to reflect the current drive before the seek begins.

After the 82064 senses SC (with or without an implied seek) BDRQ and DRQ are asserted and the host begins filling the sector buffer with data. When BRDY is asserted, a search for the ID field with the correct cylinder number, head, sector size, and CRC is initiated. If retries are enabled (\( T = 0 \)), ten attempts are made to find the correct ID field. If there is still an error on the tenth try, an auto-scan ID and auto-seek are performed. Then ten more retries are attempted before setting the ID Not Found error bit. When retries are disabled (\( T = 1 \)) only two tries are made. No auto-scan or auto-seek operations are performed.
If T bit of command = 1 then dashed path is taken after 2 index pulses.

Figure 12a. Read Sector Command Flow
*If T bit of command = 1 then dashed path is taken.
**If T bit of command = 1 then test is for 2 index pulses.
When the correct ID is found, WR GATE is asserted and data is written to the disk. When the CRC/ECC bit (SDH Register, bit 7) is zero, the 82064 generates a two byte CRC character to be appended to the data. When the CRC/ECC bit is one, four ECC bytes replace the CRC character. When L = 1, the polynomial generator is inhibited and neither CRC or ECC bytes are generated. Instead four bytes of data supplied by the host are written.

During a WRITE MULTIPLE SECTOR command (M = 1), the SECTOR NUMBER register is increment-ed and the SECTOR COUNT register is decremen-ted. If BRDY is asserted after the first sector is read from the sector buffer, the 82064 continues to read data from the sector buffer for the next sector. If BRDY is deasserted, the 82064 asserts BDRQ and waits for the host to place more data in the sector buffer.

In summary then, the WRITE SECTOR operation is as follows:

When M = 0,1
1. HOST: Sets up parameters. Issues READ SECTOR command.
2. 82064: Asserts BDRQ and DRQ.
3. HOST: Loads sector buffer with data.
4. 82064: Waits for rising edge of BRDY.
5. 82064: Finds specified ID field. Writes sector to disk.
6. 82064: If M = 0, asserts INTRQ. End.
7. 82064: Increments SECTOR NUMBER. Decrements SECTOR COUNT.
8. 82064: IF SECTOR COUNT = 0, assert INTRQ. End.
9. 82064: Go to 2.

A flowchart of the WRITE SECTOR command is shown in Figure 13.

SCAN ID

The SCAN ID command is used to update the SDH, SECTOR NUMBER, and CYLINDER NUMBER LOW/HIGH registers.

After the command is loaded, the SC line is sampled until it is valid. The DRDY and WR FAULT lines are also monitored throughout execution of the command. If a fault occurs the command is aborted and the appropriate error bits are set. When the first ID field is found, the ID information is loaded into the SDH, SECTOR NUMBER, and CYLINDER NUMBER registers. The internal cylinder position register is also updated. If this is an auto-scan caused by a change in drive numbers, only the internal position register is updated. If a bad block is detected, the BAD BLOCK bit will also be set.

If an ID field is not found, or if a CRC error occurs, and if retries are enabled (T = 0), ten attempts are made to read it. If retries are disabled (T = 1), only two tries are made. There is no auto-seek in this command and the sector buffer is not disturbed.

A flowchart of the SCAN ID command is shown in Figure 13.

WRITE FORMAT

The WRITE FORMAT command is used to format one track using information in the Task Register File and the sector buffer. During execution of this command, the sector buffer is used for additional parameter information instead of data. Shown in Figure 15 is the contents of a sector buffer for a 32 sector track with an interleave factor of two.

Each sector requires a two byte sequence. The first byte designates whether a bad block mark is to be recorded in the sector’s ID field. An 00H is normal; an 80H indicates a bad block mark for that sector. In the example of Figure 15, sector 04 will get a bad block mark recorded. The second byte indicates the logical sector number to be recorded. This allows sectors to be recorded with any interleave factor desired. The remaining memory in the sector buffer may be filled with any value; its only purpose is to generate a BRDY to tell the 82064 to begin formatting the track.

If the drive number has been changed since the last command, an auto-restore is initiated, positioning the heads to track 000. The internal cylinder position register is set to zero and the heads seek to the track specified in the Task Register File CYLINDER NUMBER register. This prevents an ID Not Found error from occurring due to an incompatible format, or the track having been erased. A normal implied seek is also in effect for this command.

The SECTOR COUNT register is used to hold the total number of sectors to be formatted (FFH = 255 sectors), while the SECTOR NUMBER register holds the number of bytes, minus three, to be used for Gap 1 and Gap 3. If, for example, the SECTOR COUNT register value is 02H and the SECTOR NUMBER register value is 00H, then 2 sectors are formatted and 3 bytes of 4EH are written for Gap 1 and Gap 3. The data fields are filled with FFH and the CRC or ECC is automatically generated and appended. After the last sector is written the track is filled with 4EH.
WRITE SECTOR

DE-ASSERT INTRQ, ERRORS, DRO
ASSERT BUSY, CIP

YES

DRIVE # CHANGED

SCAN ID GET CYL #

NO

VALID ID FOUND

YES

CYL. REGISTERS & INTERNAL CYL. REGISTER SAME

NO

PERFORM SEEK COMMAND

SEEK COMPLETE

YES

ASSERT BDRQ, DRQ

NO

BRDY ACTIVE

YES

DE-ASSERT BDRQ

NO

SEARCH FOR ID FIELD

1

CORRECT ID FOUND

10 INDEXES

YES

NOTE*

ASSIGN BCS PULSE, BCR
ASSERT WR GATE

NO

SCAN ID GET CYL #

2

ASSIGN BCR;
ASSIGN ID NOT FOUND,
INTRQ; DE-ASSERT BUSY,
CIP; DE-ASSERT BCS

WR FAULT

YES

WRITE DATA TO SECTOR
DE-ASSERT WR GATE

NO

Rewind to CYL #

2

ASSERT BCR;
ASSERT ID NOT FOUND,
INTRQ; DE-ASSERT BUSY,
CIP; DE-ASSERT BCS

INCR SECTOR
NUMBER; DECREMENT
SECTOR COUNT

YES

W = 0

NO

DE-ASSERT BCS
ASSERT BCR
ASSERT BDRQ

NO

DE-ASSERT BCS
ASSERT BCR
ASSERT INTRQ
DE-ASSERT BUSY, CIP

YES

DE-ASSERT BDRQ

NO

BRDY ACTIVE

2

ASSERT ABORTED COMMAND BIT


dash path is taken after 2 index pulses.

Figure 13. Write Sector Command Flow

*If retries disabled then dashed path is taken after 2 index pulses.
Figure 14. Scan ID Command Flow

*If retries are disabled, path is taken after 2 index pulses.*
The Gap 3 value is determined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 interleave is used. The formula for determining the minimum Gap 3 length is:

\[ \text{Gap 3} = (2 \cdot M \cdot S) + K + E \]

where:

- \( M \) = motor speed variation (e.g., 0.03 for \(+3\%\))
- \( S \) = sector length in bytes
- \( K \) = 18 for an interleave factor of 1
- \( 0 \) for any other interleave factor
- \( E = 2 \) if ECC is enabled (SDH register, bit 7 = 1)

As for all commands, if WR FAULT is asserted or DRDY is deasserted during execution of the command, the command terminates and the Aborted Command bit in the ERROR register is set.

Figure 16 shows the format which the 82064 will write on the disk.

A flowchart of the WRITE FORMAT command is shown in Figure 17.

**COMPUTE CORRECTION**

The COMPUTE CORRECTION command determines the location and pattern of a single burst error, but does not correct it. The host, using the data provided by the 82064, must perform the actual correction. The COMPUTE CORRECTION command is used following a data field ECC error. The command initiating the read must specify no retries (\( T = 1 \)).

The COMPUTE CORRECTION command first writes the four syndrome bytes from the internal ECC register to the sector buffer. Then the ECC register is clocked. With each clock, a counter is incremented and the pattern examined. If the pattern is correctable, the procedure is stopped and the count and pattern are written to the sector buffer, following the syndrome. The process is also stopped if the count exceeds the sector size before a correctable pattern is found.

When the command terminates the sector buffer contains the following data:

- Syndrome MSB
- Syndrome
- Syndrome LSB
- Error Pattern Offset
- Error Pattern Offset
- Error Pattern MSB
- Error Pattern
- Error Pattern LSB

As an example, when the Error Pattern Offset is zero the following procedure may correct the error. The first data byte of the sector is exclusive OR'd with the MSB of the Error Pattern, the second data byte with the second byte of the Error Pattern, and the third data byte with the LSB of the Error Pattern.

If the sector buffer count exceeds the sector size, or if the error burst length is greater than that selected by the Set Parameter command, the ECC/CRC error in the ERROR register and the Error bit in the STATUS register is set.

**SET PARAMETER**

This command selects the correction span to be used for the error correction process. A 5-bit span is selected when bit zero of the command equals 0, and an 11-bit span when bit zero equals 1. The 82064 defaults to a 5-bit span after a RESET.
ID FIELD

A1 = A1H with 0AH Clock

IDENT = Bits 3, 1, 0 = Cylinder High
        FE = 0-255 Cylinders
        FF = 265-511 Cylinders
        FC = 512-767 Cylinders
        FD = 768-1023 Cylinders
        F6 = 1024-1279 Cylinders
        F7 = 1280-1535 Cylinders
        F4 = 1536-1791 Cylinders
        F5 = 1792-2047 Cylinders

HEAD = Bits 0, 1, 2 = Head Number
       Bits 3, 4 = 0
       Bits 5, 6 = Sector Size
       Bit 7 = Bad Block Mark

Sec # = Logical Sector Number

DATA FIELD

A1 = A1H with 0AH clock
F8 = Data Address Mark; Normal Clock
USER = Data Field 128 to 1024 Bytes

NOTES:
1. GAP 1 and 3 length determined by Sector Number Register contents during formatting.
2. RD GATE asserted 2 bytes after the start of DRUN.
3. RD GATE de-asserted:
   • If DRUN does not last until A1
   • When any part of ID does not match the one expected.
   • After CRC if correct ID has been read.
4. Write splice recorded on disk by asserting WR GATE
5. RD GATE is suppressed until after write splice.
6. Not a proper A1 or F8, set DAM error.
7. Sector size as stated in ID field, plus two for CRC or 4 for ECC.

Figure 16. Track Format
Figure 17. Write Format Command Flow
ELECTRICAL CHARACTERISTICS
ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias .......... 0°C to 70°C
Storage Temperature .................. -65°C to +150°C
Voltage on any pin with
respect to GND ...................... -0.5V to +7V
Power Dissipation ..................... 1.5 Watt

*Notice: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent dam-
age to the device. This is a stress rating only and
functional operation of the device at these or any
other conditions above those indicated in the opera-
tional sections of this specification is not implied.
Exposure to absolute maximum rating conditions for
extended periods may affect device reliability.

NOTICE: Specifications contained within the
following tables are subject to change.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_CC = +5V ± 10%; GND = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_{IL}</td>
<td>Input Leakage Current</td>
<td>±10</td>
<td>μA</td>
<td>V_{IN} = V_CC to 0V</td>
</tr>
<tr>
<td>I_{OFL}</td>
<td>Output Leakage Current</td>
<td>±10</td>
<td>μA</td>
<td>V_{OUTPUT} = V_CC to 0.45V</td>
</tr>
<tr>
<td>V_{IH}</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{IL}</td>
<td>Input Low Voltage</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{OH}</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td>V</td>
<td>I_{OH} = -100 μA</td>
</tr>
<tr>
<td>V_{OL}</td>
<td>Output Low Voltage</td>
<td>0.40</td>
<td>V</td>
<td>I_{OL} = 1.6 mA</td>
</tr>
<tr>
<td>0.45</td>
<td>6.0 mA P21, 22, 23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_{CC}</td>
<td>Supply Current</td>
<td>250</td>
<td>mA</td>
<td>All Outputs Open</td>
</tr>
<tr>
<td>C_{IN}</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td>f_c = 1 MHz</td>
</tr>
<tr>
<td>C_{I/O}</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
<td>Unmeasured pins returned to GND</td>
</tr>
<tr>
<td>For Pins 25, 34, 37, 39</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_{IH}</td>
<td>Input High Voltage</td>
<td>4.6</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{IL}</td>
<td>Input Low Voltage</td>
<td>0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>TRS</td>
<td>Rise Time</td>
<td>30</td>
<td>ns</td>
<td>0.9V to 4.2V</td>
</tr>
</tbody>
</table>

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_CC = +5V ± 10%; GND = 0V)

HOST READ TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address Stable Before RD ↓</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Data Delay from RD ↓</td>
<td>70</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RD Pulse Width</td>
<td>0.2</td>
<td>10</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>RD to Data Floating</td>
<td>10</td>
<td>200</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Address Hold Time after RD ↑</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Read Recovery Time</td>
<td>300</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CS Stable before RD ↓</td>
<td>0</td>
<td></td>
<td>ns</td>
<td>See Note 6</td>
</tr>
</tbody>
</table>
### HOST WRITE TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Address Stable Before WR ↓</td>
<td>0</td>
<td>10</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>CS Stable Before WR ↓</td>
<td>0</td>
<td>10</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Data Setup Time Before WR ↑</td>
<td>0.16</td>
<td>10</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>WR Pulse Width</td>
<td>0.2</td>
<td>10</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Data Hold Time After WR ↑</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Address Hold Time After WR ↑</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>CS Hold Time After WR ↑</td>
<td>0</td>
<td></td>
<td>ns</td>
<td>See Note 7</td>
</tr>
<tr>
<td>15</td>
<td>Write Recovery Time</td>
<td>300</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>SDHLE Propagation Delay</td>
<td>20</td>
<td>150</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>
### BUFFER READ TIMING (WRITE SECTOR COMMAND)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>BCS ( \downarrow ) to RD Valid</td>
<td>15</td>
<td></td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>RD Output Pulse Width</td>
<td>300</td>
<td>400</td>
<td>500</td>
<td>ns</td>
<td>See Note 3</td>
</tr>
<tr>
<td>18</td>
<td>Data Setup to RD ( \uparrow )</td>
<td>140</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Data Hold from RD ( \uparrow )</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>RD Repetition Rate</td>
<td>1.2</td>
<td>1.6</td>
<td>2.0</td>
<td>( \mu )s</td>
<td>See Note 1</td>
</tr>
<tr>
<td>21</td>
<td>RD Float from BCS ( \uparrow )</td>
<td>15</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### BUFFER WRITE TIMING (READ SECTOR COMMAND)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>BCS ( \downarrow ) to WR Valid</td>
<td>15</td>
<td></td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>WR Output Pulse Width</td>
<td>300</td>
<td>400</td>
<td>500</td>
<td>ns</td>
<td>See Note 3</td>
</tr>
<tr>
<td>24</td>
<td>Data Valid from WR ( \downarrow )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Data Hold from WR ( \uparrow )</td>
<td>20</td>
<td>200</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>WR Repetition Rate</td>
<td>1.2</td>
<td>1.6</td>
<td>2.0</td>
<td>( \mu )s</td>
<td>See Note 1</td>
</tr>
<tr>
<td>27</td>
<td>WR Float from BCS ( \uparrow )</td>
<td>15</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>
### MISCELLANEOUS TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>BDRQ Reset from BRDY</td>
<td>20</td>
<td></td>
<td>200</td>
<td>ns</td>
<td>Test Conditions</td>
</tr>
<tr>
<td>29</td>
<td>BRDY Pulse Width</td>
<td>800</td>
<td></td>
<td></td>
<td>ns</td>
<td>See Note 4</td>
</tr>
<tr>
<td>30</td>
<td>BCR Pulse Width</td>
<td>1.4</td>
<td>1.6</td>
<td>1.8</td>
<td>μs</td>
<td>See Note 1</td>
</tr>
<tr>
<td>31</td>
<td>STEP Pulse Width</td>
<td>1.5</td>
<td>1.6</td>
<td>1.7</td>
<td>μs</td>
<td>Step Rate = 3.2 μs/step</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.9</td>
<td>8.4</td>
<td>8.7</td>
<td>μs</td>
<td>All other step rates</td>
</tr>
<tr>
<td>32</td>
<td>INDEX Pulse Width</td>
<td>500</td>
<td></td>
<td></td>
<td>ns</td>
<td>Test Conditions</td>
</tr>
<tr>
<td>33</td>
<td>RESET Pulse Width</td>
<td>24</td>
<td></td>
<td></td>
<td>ns</td>
<td>See Note 2</td>
</tr>
<tr>
<td>34</td>
<td>RESET ↑ to BCR</td>
<td>1.6</td>
<td>3.2</td>
<td>6.4</td>
<td>μs</td>
<td>See Note 1</td>
</tr>
<tr>
<td>35</td>
<td>RESET ↑ to WR, CS ↓</td>
<td>6.4</td>
<td></td>
<td></td>
<td>μs</td>
<td>See Note 1</td>
</tr>
<tr>
<td>36</td>
<td>WR CLOCK Frequency</td>
<td>0.25</td>
<td>5.0</td>
<td>5.25</td>
<td>MHz</td>
<td>50% Duty Cycle</td>
</tr>
<tr>
<td>37</td>
<td>RD CLOCK Frequency</td>
<td>0.25</td>
<td>5.0</td>
<td>5.25</td>
<td>MHz</td>
<td>50% Duty Cycle</td>
</tr>
</tbody>
</table>

### READ DATA TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>RD CLOCK Pulse Width</td>
<td>95</td>
<td></td>
<td>2000</td>
<td>ns</td>
<td>50% Duty Cycle</td>
</tr>
<tr>
<td>39</td>
<td>RD DATA after RD CLOCK ↓</td>
<td>0</td>
<td></td>
<td>T38</td>
<td>ns</td>
<td>Test Conditions</td>
</tr>
<tr>
<td>40</td>
<td>RD DATA before RD CLOCK ↑</td>
<td>20</td>
<td></td>
<td>T38</td>
<td>ns</td>
<td>Test Conditions</td>
</tr>
<tr>
<td>41</td>
<td>RD DATA Pulse Width</td>
<td>40</td>
<td></td>
<td>T38</td>
<td>ns</td>
<td>Test Conditions</td>
</tr>
<tr>
<td>42</td>
<td>DRUN Pulse Width</td>
<td>30</td>
<td></td>
<td></td>
<td>ns</td>
<td>Test Conditions</td>
</tr>
</tbody>
</table>
WRITE DATA TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>WR CLOCK Pulse Width</td>
<td>95</td>
<td>2000</td>
<td>65</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Propagation Delay</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44A</td>
<td>WR CLOCK ↑ to WR DATA ↑</td>
<td>10</td>
<td></td>
<td>65</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>44B</td>
<td>WR CLOCK ↓ to WR DATA ↓</td>
<td></td>
<td></td>
<td>85</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>44C</td>
<td>WR CLOCK ↑ to WR DATA ↓</td>
<td></td>
<td></td>
<td>85</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>44D</td>
<td>WR CLOCK ↓ to WR DATA ↑</td>
<td></td>
<td></td>
<td>85</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>45A</td>
<td>WR CLOCK ↑ to EARLY/LATE ↓</td>
<td>10</td>
<td></td>
<td>65</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>45B</td>
<td>WR CLOCK ↓ to EARLY/LATE ↓</td>
<td></td>
<td></td>
<td>55</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>46A</td>
<td>WR CLOCK ↑ to EARLY/LATE ↑</td>
<td>10</td>
<td></td>
<td>65</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>46B</td>
<td>WR CLOCK ↓ to EARLY/LATE ↑</td>
<td></td>
<td></td>
<td>55</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>
**A.C. TESTING INPUT, OUTPUT WAVEFORM**

**A.C. TESTING LOAD CIRCUIT**

---

**NOTES**

1. Based on WR CLOCK = 5.0 MHz.
2. 24 WR CLOCK periods = 4.8 μs at 5.0 MHz.
3. 2 WR CLOCK periods ± 100 ns.
4. When used with a DMA controller BRDY must be > 4 μs or a spurious BDRQ pulse may exist for up to 4 μs after the rising edge of BDRY.
5. WR CLOCK Frequency = RD CLOCK Frequency ± 15%.
6. RD may be asserted before CS as long as it remains active for at least the minimum. T3 pulse width after CS is asserted.
7. WR may be asserted before CS as long as it remains active for at least the minimum T11 pulse width after CS is asserted.
CHAPTER 1
INTRODUCTION

Accompanying the introduction of microprocessors such as the 8080, 8085, 8088, and 8086 there has been a rapid proliferation of intelligent peripheral devices. These special purpose peripherals extend CPU performance and flexibility in a number of important ways.

Table 1-1. Intelligent Peripheral Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8255 (GPIO)</td>
<td>Programmable Peripheral Interface</td>
</tr>
<tr>
<td>8251A (USART)</td>
<td>Programmable Communication Interface</td>
</tr>
<tr>
<td>8253 (TIMER)</td>
<td>Programmable Interval Timer</td>
</tr>
<tr>
<td>8257 (DMA)</td>
<td>Programmable DMA Controller</td>
</tr>
<tr>
<td>8259</td>
<td>Programmable Interrupt Controller</td>
</tr>
<tr>
<td>8271 (SDFDC), 8272 (DDFDC)</td>
<td>Programmable Floppy Disk Controllers</td>
</tr>
<tr>
<td>8273 (SDLC)</td>
<td>Programmable Synchronous Data Link Controller</td>
</tr>
<tr>
<td>8274</td>
<td>Programmable Multiprotocol Serial Communications Controller</td>
</tr>
<tr>
<td>8275/8276 (CRT)</td>
<td>Programmable CRT Controllers</td>
</tr>
<tr>
<td>8279 (PKD)</td>
<td>Programmable Keyboard/Display Controller</td>
</tr>
<tr>
<td>8291A, 8292, 8293</td>
<td>Programmable GPIB System Talker, Listener, Controller</td>
</tr>
</tbody>
</table>

Intelligent devices like the 8272 floppy disk controller and 8273 synchronous data link controller (see Table 1-1) can preprocess serial data and perform control tasks which off-load the main system processor. Higher overall system throughput is achieved and software complexity is greatly reduced. The intelligent peripheral chips simplify master processor control tasks by performing many functions externally in peripheral hardware rather than internally in main processor software.

Intelligent peripherals also provide system flexibility. They contain on-chip mode registers which are programmed by the master processor during system initialization. These control registers allow the peripheral to be configured into many different operation modes. The user-defined program for the peripheral is stored in main system memory and is transferred to the peripheral's registers whenever a mode change is required. Of course, this type of flexibility requires software overhead in the master system which tends to limit the benefit derived from the peripheral chip.

In the past, intelligent peripherals were designed to handle very specialized tasks. Separate chips were designed for communication disciplines, parallel I/O, keyboard encoding, interval timing, CRT control, etc. Yet, in spite of the large number of devices available and the increased flexibility built into these chips, there is still a large number of microcomputer peripheral control tasks which are not satisfied.

With the introduction of the Universal Peripheral Interface (UPI) microcomputer, Intel has taken the intelligent peripheral concept a step further by providing an intelligent controller that is fully user programmable. It is a complete single-chip microcomputer which can connect directly to a master processor data bus. It has the same advantages of intelligence and flexibility which previous peripheral chips offered. In addition, the UPI is user-programmable: it has 1K bytes of ROM or EPROM memory for program storage plus 64 bytes of RAM memory for data storage or initialization from the master processor. The UPI device allows a designer to fully specify his control algorithm in the peripheral chip without relying on the master processor. Devices like printer controllers and keyboard scanners can be completely self-contained, relying on the master processor. The UPI family currently consists of five components:

- 8741A microcomputer with 1K EPROM memory
- 8041AH microcomputer with 1K ROM memory
- 8042 microcomputer with 2K ROM memory
- 8243 I/O expander device
- 8742 microcomputer with 2K EPROM memory

The 8741A, 8041AH, 8742 and 8042 single chip microcomputers are functionally equivalent except for the type and amount of program memory available with each. These devices have the following main features:

- 8-bit CPU
- 8-bit data bus interface registers
- 1K by 8 bit ROM or EPROM memory (2K for 8042/8742)
- 64 by 8 bit RAM memory (128 bytes for 8042/8742)
- Interval timer/event counter
- Two 8-bit TTL compatible I/O ports
- Resident clock oscillator
- 12 MHZ operation, 1.25 µsec instruction cycle for 8041AH, 8742, 8042
INTRODUCTION

HMOS processing has been applied to the UPI family to allow for additional performance and memory capability while reducing costs. The 8041AH, 8741A, 8042, 8742 are all pin and software compatible. This allows growth in present designs to incorporate new features and add additional performance. For new designs, the additional memory and performance of the 8042/8742 extends the UPI ‘grow your own solution’ concept to more complex motor control tasks, 80-column printers and process control applications as examples.

The 8243 device is an I/O multiplexer which allows expansion of I/O to over 100 lines (if seven devices are used). All three parts are fabricated with N-channel MOS technology and require a single, 5V supply for operation.

INTERFACE REGISTERS FOR MULTI-PROCESSOR CONFIGURATIONS

In the normal configuration, the 8041AH/8741A, 8042/8742 interfaces to the system bus, just like any intelligent peripheral device (see Figure 1-1). The host processor and the 8041AH/8741A, 8042/8742 form a loosely coupled multi-processor system, that is, communications between the two processors are direct. Common resources are three addressable registers located physically on the 8041AH/8741A, 8042/8742. These registers are the Data Bus Buffer Input (DBBIN), Data Bus Buffer Output (DBBOUT), and Status (STATUS) registers. The host processor may read data from DBBOUT or write commands and data into DBBIN. The status of DBBOUT and DBBIN plus user-defined status is supplied in STATUS. The host may read STATUS at any time. An interrupt to the UPI processor is automatically generated (if enabled) when DBBIN is loaded.

Because the UPI contains a complete microcomputer with program memory, data memory, and CPU it can function as a “Universal” controller. A designer can program the UPI to control printers, tape transports, or multiple serial communication channels. The UPI can also handle off-line arithmetic processing, or any number of other low speed control tasks.
**POWERFUL 8-BIT PROCESSOR**

The UPI contains a powerful, 8-bit CPU with as fast as 1.25 μsec cycle time and two single-level interrupts. Its instruction set includes over 90 instructions for easy software development. Most instructions are single byte and single cycle and none are more than two bytes long. The instruction set is optimized for bit manipulation and I/O operations. Special instructions are included to allow binary or BCD arithmetic operations, table lookup routines, loop counters, and N-way branch routines.

**SPECIAL INSTRUCTION SET FEATURES**

- For Loop Counters:
  - Decrement Register and Jump if not zero.
- For Bit Manipulation:
  - AND to A (immediate data or Register)
  - OR to A (immediate data or Register)
  - XOR to A (immediate data or Register)
  - AND to Output Ports (Accumulator)
  - OR to Output Ports (Accumulator)
  - Jump Conditionally on any bit in A
- For BCD Arithmetic:
  - Decimal Adjust A
  - Swap 4-bit Nibbles of A
  - Exchange lower nibbles of A and Register
  - Rotate A left or right with or without Carry
- For Lookup Tables:
  - Load A from Page of ROM (Address in A)
  - Load A from Current Page of ROM (Address in A)

**Features for Peripheral Control**

The UPI 8-bit interval timer/event counter can be used to generate complex timing sequences for control applications or it can count external events such as switch closures and position encoder pulses. Software timing loops can be simplified or eliminated by the interval timer. If enabled, an interrupt to the CPU will occur when the timer overflows.

The UPI I/O complement contains two TTL-compatible 8-bit bidirectional I/O ports and two general-purpose test inputs. Each of the 16 port lines can individually function as either input or output under software control. Four of the port lines can also function as an interface for the 8243 I/O expander which provides four additional 4-bit ports that are directly addressable by UPI software. The 8243 expander allows low cost I/O expansion for large control applications while maintaining easy and efficient software port addressing.

![Figure 1-3. Interfaces And Protocols For Multiprocessor Systems](image)

![Figure 1-4. 8243 I/O Expander Interface](image)
INTRODUCTION

On-Chip Memory
The UPI's 64 (128) bytes of data memory include dual working register banks and an 8-level program counter stack. Switching between the register banks allows fast response to interrupts. The stack is used to store return addresses and processor status upon entering a subroutine.

The UPI program memory is available in two types to allow flexibility in moving from design to prototype to production with the same PC layout. The 8741A, 8742 device with EPROM memory is very economical for initial system design and development. Its program memory can be electrically programmed using the Intel Universal PROM Programmer. When changes are needed, the entire program can be erased using UV lamp and reprogrammed in about 20 minutes. This means the 8741A/8742 can be used as a single chip "breadboard" for very complex interface and control problems. After the 8741A/8742 is programmed it can be tested in the actual production level PC board and the actual functional environment. Changes required during system debugging can be made in the 8741A/8742 program much more easily than they could be made in a random logic design. The system configuration and PC layout can remain fixed during the development process and the turn around time between changes can be reduced to a minimum.

At any point during the development cycle, the 8741A/8742 EPROM part can be replaced with the low cost 8041AH, 8042 respectively with factory mask programmed memory. The transition from system development to mass production is made smoothly because the 8741A and 8041AH, 8742 and 8042 parts are completely pin compatible. 8742s or 8042s can be used in an 8041AH/8741 socket. This feature allows extensive testing with the EPROM part, even into initial shipments to customers. Yet, the transition to low-cost ROM is simplified to the point of being merely a package substitution.

PREPROGRAMMED UPI's
The 8292, 8294, and 8295 are 8041A's that are programmed by Intel and sold as standard peripherals. The 8292 is a GPIB controller, part of a three chip GPIB system. The 8294 is a Data Encryption Unit that implements the National Bureau of Standards data encryption algorithm. The 8295 is a dot matrix printer controller designed especially for the LRC 7040 series dot matrix impact printers. These parts illustrate the great flexibility offered by the UPI family.

DEVELOPMENT SUPPORT
The UPI microcomputer is fully supported by Intel with development tools like the UPP PROM programmer already mentioned. An ICE-41A in-circuit emulator is also available to allow UPI software and hardware to be developed easily and quickly. The combination of device features and Intel development support make the UPI an ideal component for low-speed peripheral control applications.

UPI DEVELOPMENT SUPPORT
- 8048/8041AH/8042 Assembler
- Universal PROM Programmer UPP Series
- ICE-41A Module
- MULTI-ICE
- Insite User's Library
- Application Engineers
- Training Courses
CHAPTER 2
FUNCTIONAL DESCRIPTION

The UPI-41AH, 42 microcomputer is an intelligent peripheral controller designed to operate in iAPX-86, 88, MCS-85, MCS-80, MCS-51 and MCS-48 systems. The UPI's architecture, illustrated in Figure 2-1, is based on a low cost, single-chip microcomputer with program memory, data memory, CPU, I/O, event timer and clock oscillator in a single 40-pin package. Special interface registers are included which enable the UPI to function as a peripheral to an 8-bit master processor.

This chapter provides a basic description of the UPI microcomputer and its system interface registers. Unless otherwise noted the descriptions in this section apply to both the 8741A, 8742 (with UV erasable program memory) and the 8041AH, 8042 (with factory mask programmed memory). These two devices are so similar that they can be considered identical under most circumstances. All functions described in this chapter apply to the 8041AH, 8042, and 8741A, 8742.

PIN DESCRIPTION
The 8041AH/8741A, 8042/8742 are packaged in 40-pin Dual In-Line (DIP) packages. The pin configuration for both devices is shown in Figure 2-2. Figure 2-3 illustrates the UPI Logic Symbol.
The following section summarizes the functions of each UPI-41A pin. NOTE that several pins have two or more functions which are described in separate paragraphs.

### Table 2-1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0-D7 (BUS)</td>
<td>12-19</td>
<td>I/O</td>
<td>Data Bus: Three-state, bidirectional DATA BUS BUFFER lines used to interface the UPI-41AH, 42 microcomputer to an 8-bit master system data bus.</td>
</tr>
<tr>
<td>P10-P17</td>
<td>27-34</td>
<td>I/O</td>
<td>Port 1: 8-bit, PORT 1 quasi-bidirectional I/O lines.</td>
</tr>
<tr>
<td>P20-P27</td>
<td>21-24, 35-38</td>
<td>I/O</td>
<td>Port 2: 8-bit, PORT 2 quasi-bidirectional I/O lines. The lower 4 bits (P20-P23) interface directly to the 8243 I/O expander device and contain address and data information during PORT 4-7 access. The upper 4 bits (P24-P27) can be programmed to provide interrupt Request and DMA Handshake capability. Software control can configure P24 as Output Buffer Full (OBF) interrupt, P25 as Input Buffer Full (IBF) interrupt, P26 as DMA Request (DRQ), and P27 as DMA ACKnowledge (DACK).</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>I</td>
<td>Write: I/O write input which enables the master CPU to write data and command words to the UPI-41A INPUT DATA BUS BUFFER.</td>
</tr>
<tr>
<td>RD</td>
<td>8</td>
<td>I</td>
<td>Read: I/O read input which enables the master CPU to read data and status words from the OUTPUT DATA BUS BUFFER or status register.</td>
</tr>
<tr>
<td>CS</td>
<td>6</td>
<td>I</td>
<td>Chip Select: Chip select input used to select one UPI-41AH, 42 microcomputer out of several connected to a common data bus.</td>
</tr>
<tr>
<td>A0</td>
<td>9</td>
<td>I</td>
<td>Command/Data Select: Address input used by the master processor to indicate whether byte transfer is data (A0=0) or command (A0=1).</td>
</tr>
<tr>
<td>TEST 0, TEST 1</td>
<td>1, 39</td>
<td>I</td>
<td>Test Inputs: Input pins which can be directly tested using conditional branch instructions. Frequency Reference: TEST 1 (T1) also functions as the event timer input (under software control). TEST 0 (T0) is used during PROM programming and verification in the 8741A, 8742.</td>
</tr>
</tbody>
</table>

![Figure 2-2. Pin Configuration](image-url)

![Figure 2-3. Logic Symbol](image-url)
FUNCTIONAL DESCRIPTION

Table 2-1. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAL 1, XTAL 2</td>
<td>2, 3</td>
<td>I</td>
<td>Inputs: Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.</td>
</tr>
<tr>
<td>SYNC</td>
<td>11</td>
<td>O</td>
<td>Output Clock: Output signal which occurs once per UPI-41A instruction cycle. SYNC can be used as a strobe for external circuitry; it is also used to synchronize single step operation.</td>
</tr>
<tr>
<td>EA</td>
<td>7</td>
<td>I</td>
<td>External Access: External access input which allows emulation, testing and PROM/ROM verification.</td>
</tr>
<tr>
<td>PROG</td>
<td>25</td>
<td>I/O</td>
<td>Program: Multifunction pin used as the program pulse input during PROM programming. During I/O expander access the PROG pin acts as an address/data strobe to the 8243.</td>
</tr>
<tr>
<td>RESET</td>
<td>4</td>
<td>I</td>
<td>Reset: Input used to reset status flip-flops and to set the program counter to zero. RESET is also used during PROM programming and verification.</td>
</tr>
<tr>
<td>SS</td>
<td>5</td>
<td>I</td>
<td>Single Step: Single step input used in conjunction with the SYNC output to step the program through each instruction.</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td>I</td>
<td>Power: +5V main power supply pin.</td>
</tr>
<tr>
<td>VDD</td>
<td>26</td>
<td>Power: +5V during normal operation. +25V during programming operation, +21V for programming 8742. Low power standby pin in ROM version.</td>
<td></td>
</tr>
<tr>
<td>VSS</td>
<td>20</td>
<td>Ground: Circuit ground potential.</td>
<td></td>
</tr>
</tbody>
</table>

The following sections provide a detailed functional description of the UPI microcomputer. Figure 2-4 illustrates the functional blocks within the UPI device.

Figure 2-4. UPI-41AH, 42™ Block Diagram
FUNCTIONAL DESCRIPTION

CPU SECTION
The CPU section of the UPI-41AH, 42 microcomputer performs basic data manipulations and controls data flow throughout the single chip computer via the internal 8-bit data bus. The CPU section includes the following functional blocks shown in Figure 2-4:

• Arithmetic Logic Unit (ALU)
• Instruction Decoder
• Accumulator
• Flags

Arithmetic Logic Units (ALU)
The ALU is capable of performing the following operations:

• ADD with or without carry
• AND, OR, and EXCLUSIVE OR
• Increment, Decrement
• Bit complement
• Rotate left or right
• Swap
• BCD decimal adjust

In a typical operation data from the accumulator is combined in the ALU with data from some other source on the UPI-41AH, 42 internal bus (such as a register or an I/O port). The result of an ALU operation can be transferred to the internal bus or back to the accumulator.

If an operation such as an ADD or ROTATE requires more than 8 bits, the CARRY flag is used as an indicator. Likewise, during decimal adjust and other BCD operations the AUXILIARY CARRY flag can be set and acted upon. These flags are part of the Program Status Word (PSW).

Instruction Decoder
During an instruction fetch, the operation code (opcode) portion of each program instruction is stored and decoded by the instruction decoder. The decoder generates outputs used along with various timing signals to control the functions performed in the ALU. Also, the instruction decoder controls the source and destination of ALU data.

Accumulator
The accumulator is the single most important register in the processor. It is the primary source of data to the ALU and is often the destination for results as well. Data to and from the I/O ports and memory normally passes through the accumulator.

PROGRAM MEMORY
The UPI-41AH, 42 microcomputer has 1024, 2048-8-bit words of resident, read-only memory for program storage. Each of these memory locations is directly addressable by a 10-bit program counter. Depending on the type of application and the number of program changes anticipated, two types of program memory are available:

- 8041AH, 8042 with mask programmed ROM Memory
- 8741A, 8742 with electrically programmable EPROM Memory

The 8041AH and 8741A, 8042 and 8742 are functionally identical parts and are completely pin compatible. The 8742 and 8042 can be used in 8041AH, 8741A sockets. The 8041AH, 8042 has ROM memory which is mask programmed to user specification during fabrication. The 8741A/8742 are electrically programmed by the user using the Universal PROM Programmer (UPP series) with a UPP-848 or UPP-549 Personality Card. It can be erased using ultraviolet light and reprogrammed at any time.

A program memory map is illustrated in Figure 2-5. Memory is divided into 256 location ‘pages’ and three locations are reserved for special use:

![Program Memory Map](image)

**Figure 2-5. Program Memory Map**

**INTERRUPT VECTORS**

1) Location 0
Following a RESET input to the processor, the next instruction is automatically fetched from location 0.
2) **Location 3**
An interrupt generated by an Input Buffer Full (IBF) condition (when the IBF interrupt is enabled) causes the next instruction to be fetched from location 3.

3) **Location 7**
A timer overflow interrupt (when enabled) will cause the next instruction to be fetched from location 7.

Following a system **RESET**, program execution begins at location 0. Instructions in program memory are normally executed sequentially. Program control can be transferred out of the main line of code by an input buffer full (IBF) interrupt or a timer interrupt, or when a jump or call instruction is encountered. An IBF interrupt (if enabled) will automatically transfer control to location 3 while a timer interrupt will transfer control to location 7.

All conditional JUMP instructions and the indirect JUMP instruction are limited in range to the current 256-location page (that is, they alter PC bits 0–7 only). If a conditional JUMP or indirect JUMP begins in location 255 of a page, it must reference a destination on the following page.

Program memory can be used to store constants as well as program instructions. The UPI-41AH, 42 instruction set contains an instruction (MOV3) designed specifically for efficient transfer of look-up table information from page 3 of memory.

**DATA MEMORY**
The UPI-41AH, 42 universal peripheral interface has 64,128 8-bit words of random access data memory. This memory contains two working register banks, an 8-level program counter stack and a scratch pad memory, as shown in Figure 2-6. The amount of scratch pad memory available is variable depending on the number of addresses nested in the stack and the number of working registers being used.

**Addressing Data Memory**
The first eight locations in RAM are designated as working registers R0–R7. These locations (or registers) can be addressed directly by specifying a register number in the instruction. Since these locations are easily addressed, they are generally used to store frequently accessed intermediate results. Other locations in data memory are addressed indirectly by using R0 or R1 to specify the desired address. Since all RAM locations (including the eight working registers) can be addressed by 6 bits, the two most significant bits (6 and 7) of the addressing registers are ignored.

**Figure 2-6. Data Memory Map**

**Working Registers**
Dual banks of eight working registers are included in the UPI-41AH, 42 data memory. Locations 0–7 make up register bank 0 and locations 24–31 form register bank 1. A **RESET** signal automatically selects register bank 0. When bank 0 is selected, references to R0–R7 in UPI-41AH, 42 instructions operate on locations 0–7 in data memory. A “select register bank” instruction is used to select between the banks during program execution. If the instruction **SEL RB1** (Select Register Bank 1) is executed, then program references to R0–R7 will operate on locations 24–31. As stated previously, registers 0 and 1 in the active register bank are used as indirect address registers for all locations in data memory.

Register bank 1 is normally reserved for handling interrupt service routines, thereby preserving the contents of the main program registers. The SEL RB1 instruction can be issued at the beginning of an interrupt service routine. Then, upon return to the main program, an **RETR** (return & restore status) instruction will automatically restore the previously selected bank. During interrupt processing, registers in bank 0 can be accessed indirectly using R0 and R1.

If register bank 1 is not used, registers 24–31 can still serve as additional scratch pad memory.
FUNCTIONAL DESCRIPTION

Program Counter Stack

RAM locations 8–23 are used as an 8-level program counter stack. When program control is temporarily passed from the main program to a subroutine or interrupt service routine, the 10-bit program counter and bits 4–7 of the program status word (PSW) are stored in two stack locations. When control is returned to the main program via an RETR instruction, the program counter and PSW bits 4–7 are restored. Returning via an RET instruction does not restore the PSW bits, however. The program counter stack is addressed by three stack pointer bits in the PSW (bits 0–2).

The following sections provide a detailed description of the Program Counter Stack and the Program Status Word.

PROGRAM COUNTER

The UPI-41AH, 42 microcomputer has a 10-bit program counter (PC) which can directly address any of the 1024 locations in program memory. The program counter always contains the address of the next instruction to be executed and is normally incremented sequentially for each instruction to be executed when each instruction fetches occurs.

When control is temporarily passed from the main program to a subroutine or an interrupt routine, however, the PC contents must be altered to point to the address of the desired routine. The stack is used to save the current PC contents so that, at the end of the routine, main program execution can continue. The program counter is initialized to zero by a RESET signal.

PROGRAM COUNTER STACK

The Program Counter Stack is composed of 16 locations in Data Memory as illustrated in Figure 2-7. These RAM locations (8 through 23) are used to store the 10-bit program counter and 4 bits of the program status word.

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the program counter stack.

The stack allows up to eight levels of subroutine ‘nesting’; that is, a subroutine may call a second subroutine, which may call a third, etc., up to eight levels. Unused stack locations can be used as scratch pad memory. Each unused level of subroutine nesting provides two additional RAM locations for general use.

A 3-bit Stack Pointer which is part of the Program Status Word (PSW) determines the stack pair to be used at a given time. The stack pointer is initialized by a RESET signal to 00H which corresponds to RAM locations 8 and 9.

The first call or interrupt results in the program counter and PSW contents being transferred to RAM locations 8 and 9 in the format shown in Figure 2-7. The stack pointer is automatically incremented by 1 to point to locations 10 and 11 in anticipation of another CALL.

Nesting of subroutines within subroutines can continue up to 8 levels without overflowing the stack. If overflow does occur the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 07H to 00H. Likewise, the stack pointer will underflow from 00H to 07H.

The end of a subroutine is signaled by a return instruction, either RET or RETR. Each instruction will automatically decrement the Stack Pointer and transfer the contents of the proper RAM register pair to the Program Counter.

PROGRAM STATUS WORD

The 8-bit program status word illustrated in Figure 2-8 is used to store general information about program execution. In addition to the 3-bit Stack
FUNCTIONAL DESCRIPTION

Figure 2-8. Program Status Word

Pointer discussed previously, the PSW includes the following flags:

- CY — Carry
- AC — Auxiliary Carry
- F0 — Flag 0
- BS — Register Bank Select

The Program Status Word (PSW) is actually a collection of flip-flops located throughout the machine which are read or written as a whole. The PSW can be loaded to or from the accumulator by the MOV A, PSW or MOV PSW, A instructions. The ability to write directly to the PSW allows easy restoration of machine status after a power-down sequence.

The upper 4 bits of the PSW (bits 4, 5, 6, and 7) are stored in the PC Stack with every subroutine CALL or interrupt vector. Restoring the bits on a return is optional. The bits are restored if an RETR instruction is executed, but not if an RET is executed.

PSW bit definitions are as follows:

- Bits 0–2 Stack Pointer Bits S0, S1, S2
- Bit 3 Not Used
- Bit 4 Working Register Bank
  0 = Bank 0
  1 = Bank 1
- Bit 5 Flag 0 bit (F0)
  This is a general purpose flag which can be cleared or comple-
- Bit 6 Auxiliary Carry (AC)
  The flag status is determined by an ADD instruction and is used by the Decimal Adjustment instruction DAA.
- Bit 7 Carry (CY)
  The flag indicates that a previous operation resulted in overflow of the accumulator.

CONDITIONAL BRANCH LOGIC

Conditional Branch Logic in the UPI-41AH, 42 allows the status of various processor flags, inputs, and other hardware functions to directly affect program execution. The status is sampled in state 3 of the first cycle.

Table 2-2 lists the internal conditions which are testable and indicates the condition which will cause a jump. In all cases, the destination address must be within the page of program memory (256 locations) in which the jump instruction occurs.

Oscillator and Timing Circuits

The 8041A's internal timing generation is controlled by a self-contained oscillator and timing circuit. A choice of crystal, L-C or external clock can be used to derive the basic oscillator frequency.

The resident timing circuit consists of an oscillator, a state counter and a cycle counter as illustrated in Figure 2-9. Figure 2-10 shows instruction cycle timing.

<table>
<thead>
<tr>
<th>Device</th>
<th>Instruction Mnemonic</th>
<th>Jump Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>JZ</td>
<td>Jump if:</td>
</tr>
<tr>
<td>Accumulator bit</td>
<td>JNZ</td>
<td>All bits zero</td>
</tr>
<tr>
<td>Carry flag</td>
<td>JBC</td>
<td>Any bit not zero</td>
</tr>
<tr>
<td>User flag</td>
<td>JFO</td>
<td>Bit &quot;b&quot; = 1</td>
</tr>
<tr>
<td>Timer flag</td>
<td>JTF</td>
<td>Carry flag = 1</td>
</tr>
<tr>
<td>Test Input 0</td>
<td>JTO</td>
<td>Carry flag = 0</td>
</tr>
<tr>
<td>Test Input 1</td>
<td>JTI</td>
<td>F0 flag = 1</td>
</tr>
<tr>
<td>Input Buffer flag</td>
<td>JNIBF</td>
<td>FT flag = 1</td>
</tr>
<tr>
<td>Output Buffer flag</td>
<td>JOBF</td>
<td>Timer flag = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T0 = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T0 = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T1 = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T1 = 0</td>
</tr>
</tbody>
</table>
FUNCTIONAL DESCRIPTION

Oscillator

The on-board oscillator is a series resonant circuit with a frequency range of 1 to 12 (8041AH-2/8042/8742) MHz. Pins XTAL 1 and XTAL 2 are input and output (respectively) of a high gain amplifier stage. A crystal or inductor and capacitor connected between XTAL 1 and XTAL 2 provide the feedback and proper phase shift for oscillation. Recommended connections for crystal or L-C are shown in Figure 2-11.

State Counter

The output of the oscillator is divided by 3 in the state counter to generate a signal which defines the state times of the machine.

Each instruction cycle consists of five states as illustrated in Figure 2-10 and Table 2-3. The overlap of address and execution operations illustrated in Figure 2-10 allows fast instruction execution.

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>S1 CODE</th>
<th>S2 CODE</th>
<th>S3 CODE</th>
<th>S4 CODE</th>
<th>S5 CODE</th>
<th>S1 CODE</th>
<th>S2 CODE</th>
<th>S3 CODE</th>
<th>S4 CODE</th>
<th>S5 CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN A,P0</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>—</td>
<td>—</td>
<td>Read Port</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>OUTL P0,A</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment/ Timer</td>
<td>Output To Port</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>ANL P0, DATA</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>Read Port</td>
<td>Fetch Immediate Data</td>
<td>—</td>
<td>Increment Program Counter</td>
<td>Output To Port</td>
<td>—</td>
</tr>
<tr>
<td>ORL P0, DATA</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>Read Port</td>
<td>Fetch Immediate Data</td>
<td>—</td>
<td>Increment Program Counter</td>
<td>Output To Port</td>
<td>—</td>
</tr>
<tr>
<td>MOVX A,P0</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>Output Opcode/Address</td>
<td>Increment Timer</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>MOVX P0,A</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>Output Opcode/Address</td>
<td>Increment Timer</td>
<td>Output Data To P2 Lower</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>ANLD P0,A</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>Output Opcode/Address</td>
<td>Increment Timer</td>
<td>Output Data</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>ORLD P0,A</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>Output Opcode/Address</td>
<td>Increment Timer</td>
<td>Output Data</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>J (Conditional)</td>
<td>Fetch Instruction</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>Sample Condition</td>
<td>Increment Timer</td>
<td>—</td>
<td>Fetch Immediate Data</td>
<td>—</td>
<td>Update Program Counter</td>
</tr>
<tr>
<td>MOV STS, A</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>—</td>
<td>—</td>
<td>Update Status Register</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>IN A,D0B</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>OUT D0B,A</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Increment Timer</td>
<td>Output To Port</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>STRT T</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Start Counter</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>STOP T</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Stop Counter</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>EN I</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Enable Interrupt</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>DIS I</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>Disable Interrupt</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>EN DMA</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>DMA Enabled</td>
<td>—</td>
<td>DMA Enabled</td>
<td>—</td>
<td>ORG Cleared</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>EN FLAGS</td>
<td>Fetch</td>
<td>Increment Program Counter</td>
<td>—</td>
<td>ORG, BF</td>
<td>Output Enabled</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Figure 2-9. Oscillator Configuration

Figure 2-10. Instruction Cycle Timing

Table 2-3. Instruction Timing Diagram
FUNCTIONAL DESCRIPTION

Cycle Counter
The output of the state counter is divided by 5 in the cycle counter to generate a signal which defines a machine cycle. This signal is called SYNC and is available continuously on the SYNC output pin. It can be used to synchronize external circuitry or as a general purpose clock output. It is also used for synchronizing single-step.

Frequency Reference
The external crystal provides high speed and accurate timing generation. A crystal frequency of 5.9904 MHz is useful for generation of standard communication frequencies by the 8041AH/8741, 8042/8742. However, if an accurate frequency reference and maximum processor speed are not required, an inductor and capacitor may be used in place of the crystal as shown in Figure 2-11.

A recommended range of inductance and capacitance combinations is given below:
- \( L = 130 \mu H \) corresponds to 3 MHz
- \( L = 45 \mu H \) corresponds to 5 MHz

An external clock signal can also be used as a frequency reference to the 8741AH, 8741A, 8742 or 8042; however, the levels are not TTL compatible. The signal must be in the 1–12 MHz frequency range and must be connected to pins XTAL 1 and XTAL 2 by buffers with a suitable pull-up resistor to guarantee that a logic “1” is above 3.8 volts. The recommended connection is shown in Figure 2-12.

INTERVAL TIMER/EVENT COUNTER
The 8041AH, 8042 has a resident 8-bit timer/counter which has several software selectable modes of operation. As an interval timer, it can generate accurate delays from 80 microseconds to 20.48 milliseconds without placing undue burden on the processor. In the counter mode, external events such as switch closures or tachometer pulses can be counted and used to direct program flow.

Timer Configuration
Figure 2-13 illustrates the basic timer/counter configuration. An 8-bit register is used to count pulses from either the internal clock and prescaler or from an external source. The counter is presettable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice-versa (i.e. MOV T, A and MOV A, T). The counter is stopped by a RESET or STOP TCNT instruction and remains stopped until restarted either as a timer (START T instruction) or as a counter (START CNT instruction). Once started, the counter will increment to its maximum count (FFH) and overflow to zero continuing its count until stopped by a STOP TCNT instruction or RESET.

The increment from maximum count to zero (overflow) results in setting the Timer Flag (TF) and generating an interrupt request. The state of the overflow flag is testable with the conditional jump

Figure 2-11. Recommended Crystal and L-C Connections

Figure 2-12. Recommended Connection For External Clock Signal
FUNCTIONAL DESCRIPTION

The timer interrupt request is stored in a latch and ORed with the input buffer full interrupt request. The timer interrupt can be enabled or disabled independent of the IBF interrupt by the EN TCNTI and DIS TCTNI instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer service routine is stored. If the timer and Input Buffer Full interrupts occur simultaneously, the IBF source will be recognized and the call will be to location 3. Since the timer interrupt is latched, it will remain pending until the DBBIN register has been serviced and will immediately be recognized upon return from the service routine. A pending timer interrupt is reset by the initiation of a timer interrupt service routine.

Event Counter Mode
The STRT CNT instruction connects the TEST 1 input pin to the counter input and enables the counter. Note this instruction does not clear the counter. The counter is incremented on high to low transitions of the TEST 1 input. The TEST 1 input must remain high for a minimum of one state in order to be registered (250 ns at 12 MHz). The maximum count frequency is one count per three instruction cycles (267 kHz at 12 MHz). There is no minimum frequency limit.

Timer Mode
The STRT T instruction connects the internal clock to the counter input and enables the counter. The input clock is derived from the SYNC signal of the internal oscillator and the divide-by-32 prescaler. The configuration is illustrated in Figure 2-13. Note this instruction does not clear the timer register. Various delays and timing sequences between 40 μsec and 10.24 msec can easily be generated with a minimum of software timing loops (at 12 MHz).

Times longer than 10.24 msec can be accurately measured by accumulating multiple overflows in a register under software control. For time resolution less than 40 μsec, an external clock can be applied to the TEST 1 counter input (see Event Counter Mode). The minimum time resolution with an external clock is 3.75 μsec (267 kHz at 12 MHz).

TEST 1 Event Counter Input
The TEST 1 pin is multifunctional. It is automatically initialized as a test input by a RESET signal and can be tested using UPI-41A conditional branch instructions.

In the second mode of operation, illustrated in Figure 2-13, the TEST 1 pin is used as an input to the internal 8-bit event counter. The Start Counter (STRT CNT) instruction controls an internal switch which connects TEST 1 through an edge detector to the 8-bit internal counter. Note that this instruction does not inhibit the testing of TEST 1 via conditional Jump instructions.

In the counter mode the TEST 1 input is sampled once per instruction cycle. After a high level is detected, the next occurrence of a low level at TEST 1
FUNCTIONAL DESCRIPTION

will cause the counter to increment by one.

The event counter functions can be stopped by the Stop Timer/Counter (STOP TCNT) instruction. When this instruction is executed the TEST 1 pin becomes a test input and functions as previously described.

TEST INPUTS

There are two multifunction pins designated as Test Inputs, TEST 0 and TEST 1. In the normal mode of operation, status of each of these lines can be directly tested using the following conditional Jump instructions:

- JT0  Jump if TEST 0 = 1
- JNT0  Jump if TEST 0 = 0
- JT1  Jump if TEST 1 = 1
- JNT1  Jump if TEST 1 = 0

The test inputs are TTL compatible. An external logic signal connected to one of the test inputs will be sampled at the time the appropriate conditional jump instruction is executed. The path of program execution will be altered depending on the state of the external signal when sampled.

INTERRUPTS

The 8041AH/8741A, 8042/8742 has the following internal interrupts:

- Input Buffer Full (IBF) interrupt
- Timer Overflow interrupt

The IBF interrupt forces a CALL to location 3 in program memory; a timer-overflow interrupt forces a CALL to location 7. The IBF interrupt is enabled by the EN I instruction and disabled by the DIS I instruction. The timer-overflow interrupt is enabled and disabled by the EN TCNTI and DIS TCNTI instructions, respectively.

Figure 2-14 illustrates the internal interrupt logic. An IBF interrupt request is generated whenever WR and CS are both low, regardless of whether interrupts are enabled. The interrupt request is cleared upon entering the IBF service routine only. That is, the DIS I instruction does not clear a pending IBF interrupt.

Interrupt Timing Latency

When the IBF interrupt is enabled and an IBF interrupt request occurs, an interrupt sequence is initiated as soon as the currently executing instruction is completed. The following sequence occurs:

- A CALL to location 3 is forced.
- The program counter and bits 4-7 of the Program Status Word are stored in the stack.
- The stack pointer is incremented.

Figure 2-14. Interrupt Logic
Location 3 in program memory should contain an unconditional jump to the beginning of the IBF interrupt service routine elsewhere in program memory. At the end of the service routine, an RETR (Return and Restore Status) instruction is used to return control to the main program. This instruction will restore the program counter and PSW bits 4–7, providing automatic restoration of the previously active register bank as well. RETR also re-enables interrupts.

A timer-overflow interrupt is enabled by the EN TCNTI instruction and disabled by the DIS TCNTI instruction. If enabled, this interrupt occurs when the timer/counter register overflows. A CALL to location 7 is forced and the interrupt routine proceeds as described above.

The interrupt service latency is the sum of current instruction time, interrupt recognition time, and the internal call to the interrupt vector address. The worst case latency time for servicing an interrupt is 7 clock cycles. Best case latency is 4 clock cycles.

Interrupt Timing

Interrupt inputs may be enabled or disabled under program control using EN I, DIS I, EN TCNTI and DIS TCNTI instructions. Also, a RESET input will disable interrupts. An interrupt request must be removed before the RETR instruction is executed to return from the service routine, otherwise the processor will re-enter the service routine immediately. Thus, the WR and CS inputs should not be held low longer than the duration of the interrupt service routine.

The interrupt system is single level. Once an interrupt is detected, all further interrupt requests are latched but are not acted upon until execution of an RETR instruction re-enables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. If an IBF interrupt and a timer-overflow interrupt occur simultaneously, the IBF interrupt will be recognized first and the timer-overflow interrupt will remain pending until the end of the interrupt service routine.

External Interrupts

An external interrupt can be created using the UPI-41AH, 42 timer/counter in the event counter mode. The counter is first preset to FFH and the EN TCNTI instruction is executed. A timer-overflow interrupt is generated by the first high to low transition of the TEST 1 input pin. Also, if an IBF interrupt occurs during servicing of the timer/counter interrupt, it will remain pending until the end of the service routine.

Host Interrupts And DMA

If needed, two external interrupts to the host system can be created using the EN FLAGS instruction. This instruction allocates two I/O lines on PORT 2 (P24 and P25). P24 is the Output Buffer Full interrupt request line to the host system; P25 is the Input Buffer empty interrupt request line. These interrupt outputs reflect the internal status of the OBF flag and the IBF inverted flag. Note, these outputs may be inhibited by writing a “0” to these pins. Reenabling interrupts is done by writing a “1” to these port pins. Interrupts are typically enabled after power on since the I/O ports are set in a “1” condition. The EN FLAG’s effect is only cancelled by a device RESET.

DMA handshaking controls are available from two pins on PORT 2 of the UPI-41A microcomputer. These lines (P26 and P27) are enabled by the EN DMA instruction. P26 becomes DMA request (DRQ) and P27 becomes DMA acknowledge (DACK). The UPI program initiates a DMA request by writing a “1” to P26. The DMA controller transfers the data into the DDBIN data register using DACK which acts as a chip select. The EN DMA instruction can only be cancelled by a chip RESET.

RESET

The RESET input provides a means for internal initialization of the processor. An automatic initialization pulse can be generated at power-on by simply connecting a 1 μf capacitor between the RESET input and ground as shown in Figure 2-15. It has an internal pull-up resistor to charge the capacitor and a Schmitt trigger circuit to generate a clean transition. A 2-stage synchronizer has been added to support reliable operation up to 12 MHz.

If automatic initialization is used, RESET should be held low for at least 10 milliseconds to allow the power supply to stabilize. If an external RESET signal is used, RESET may be held low for a minimum of 8 instruction cycles. Figure 2-15 illustrates a configuration using an external TTL gate to generate the RESET input. This configuration can be used to derive the RESET signal from the 8224 clock generator in an 8080 system.

The RESET input performs the following functions:

- Disables Interrupts
- Clears Program Counter to Zero
- Clears Stack Pointer
- Clears Status Register and Flags
- Clears Timer and Timer Flag
- Stops Timer
- Selects Register Bank 0
- Sets PORTS 1 and 2 to Input Mode

5-650
DATA BUS BUFFER
Two 8-bit data bus buffer registers, DBBIN and DBBOUT, serve as temporary buffers for commands and data flowing between it and the master processor. Externally, data is transmitted or received by the DBB registers upon execution of an INput or OUTput instruction by the master processor. Four control signals are used:

- A0 Address input signifying control or data
- CS Chip Select
- RD Read strobe
- WR Write strobe

Transfer can be implemented with or without UPI program interference by enabling or disabling an internal UPI interrupt. Internally, data transfer between the DBB and the UPI accumulator is under software control and is completely asynchronous to the external processor timing. This allows the UPI software to handle peripheral control tasks independent of the main processor while still maintaining a data interface with the master system.

Configuration
Figure 2-16 illustrates the internal configuration of the DBB registers. Data is stored in two 8-bit buffer registers, DBBIN and DBBOUT. DBBIN and DBBOUT may be accessed by the external processor using the WR line and the RD line, respectively. The data bus is a bidirectional, three-state bus which can be connected directly to an 8-bit microprocessor system. Four control lines (WR, RD, CS, A0) are used by the external processor to transfer data to and from the DBBIN and DBBOUT registers.
An 8-bit register containing status flags is used to indicate the status of the DBB registers. The eight status flags are defined as follows:

- **OBF Output Buffer Full** This flag is automatically set when the UPI-Microcomputer loads the DBBOUT register and is cleared when the master processor reads the data register.
- **IBF Input Buffer Full** This flag is set when the master processor writes a character to the DBBIN register and is cleared when the UPI inputs the data register contents to its accumulator.
- **F0** This is a general purpose flag which can be cleared or toggled under UPI software control. The flag is used to transfer UPI status information to the master processor.
- **F1 Command/Data** This flag is set to the condition of the A0 input line when the master processor writes a character to the data register. The F1 flag can also be cleared or toggled under UPI-Microcomputer program control.
- **ST4 Through ST7** These bits are user defined status bits. They are defined by the MOV STS,A instruction.

All flags in the status register are automatically cleared by a RESET input.

### SYSTEM INTERFACE

Figure 2-17 illustrates how an UPI-Microcomputer can be connected to a standard 8080-type bus system. Data lines D0–D7 form a three-state, bidirectional port which can be connected directly to the system data bus. The UPI bus interface has sufficient drive capability (400 μA) for small systems, however, a larger system may require buffers.

Four control signals are required to handle the data and status information transfer:

- **WR** I/O WRITE signal used to transfer data from the system bus to the UPI DBBIN register and set the F1 flag in the status register.
- **RD** I/O READ signal used to transfer data from the DBBOUT register or status register to the system data bus.
- **CS** CHIP SELECT signal used to enable one 8041A out of several connected to a common bus.
- **A0** Address input used to select either the 8-bit status register or DBBOUT register during an I/O READ. Also, the signal is used to set the F1 flag in the status register during an I/O WRITE.

![Figure 2-17. Interface to 8080 System Bus](image-url)
The WR and RD signals are active low and are standard MCS-80 peripheral control signals used to synchronize data transfer between the system bus and peripheral devices.

The CS and A0 signals are decoded from the address bus of the master system. In a system with few I/O devices a linear addressing configuration can be used where A0 and A1 lines are connected directly to A0 and CS inputs (see Figure 2-17).

**Data Read**

Table 2-4 illustrates the relative timing of a DBBOUT Read. When CS, A0, and RD are low, the contents of the DBBOUT register is placed on the three-state Data lines D0-D7 and the OBF flag is cleared.

The master processor uses CS, A0, WR, and RD to control data transfer between the DBBOUT register and the master system. The following operations are under master processor control:

**Table 2-4. Data Transfer Controls**

<table>
<thead>
<tr>
<th>CS</th>
<th>RD</th>
<th>WR</th>
<th>A0</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Read DBBOUT register</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Read STATUS register</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Write DBBIN data register</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Write DBBIN command register</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Disable DBB</td>
</tr>
</tbody>
</table>

**Status Read**

Table 2-4 shows the logic sequence required for a STATUS register read. When CS and RD are low with A0 high, the contents of the 8-bit status register appears on Data lines D0-D7.

**Data Write**

Table 2-4 shows the sequence for writing information to the DBBIN register. When CS and WR are low, the contents of the system data bus is latched into DBBIN. Also, the IBF flag is set and an interrupt is generated, if enabled.

**Command Write**

During any write (Table 2-4), the state of the A0 input is latched into the status register in the F1 (command/data) flag location. This additional bit is used to signal whether DBBIN contents are command (A0 = 1) or data (A0 = 0) information.

**INPUT/OUTPUT INTERFACE**

The UPI-41A has 16 lines for input and output functions. These I/O lines are grouped as two 8-bit TTL compatible ports: PORTS 1 and 2. The port lines can individually function as either inputs or outputs under software control. In addition, the lower 4 lines of PORT 2 can be used to interface to an 8243 I/O expander device to increase I/O capacity to 28 or more lines. The additional lines are grouped as 4-bit ports: PORTS 4, 5, 6, and 7.

**PORTS 1 and 2**

PORTS 1 and 2 are each 8 bits wide and have the same I/O characteristics. Data written to these ports by an OUTL Pp,A instruction is latched and remains unchanged until it is rewritten. Input data is sampled at the time the IN, A,Pp instruction is executed. Therefore, input data must be present at the PORT until read by an INPut instruction. PORT 1 and 2 inputs are fully TTL compatible and outputs will drive one standard TTL load.

**Circuit Configuration**

The PORT 1 and 2 lines have a special output structure (shown in Figure 2-18) that allows each line to serve as an input, an output, or both, even though outputs are statically latched.

Each line has a permanent high impedance pull-up (50KΩ) which is sufficient to provide source current for a TTL high level, yet can be pulled low by a standard TTL gate drive. Whenever a “1” is written to a line, a low impedance pull-up (5K) is switched in momentarily (500 ns) to provide a fast transition from 0 to 1. When a “0” is written to the line, a low impedance pull-down (300Ω) is active to provide TTL current sinking capability.

To use a particular PORT pin as an input, a logic “1” must first be written to that pin.

**NOTE:** A \textbf{RESET} initializes all PORT pins to the high impedance logic “1” state.

An external TTL device connected to the pin has sufficient current sinking capability to pull-down the pin to the low state. An IN A,Pp instruction will sample the status of PORT pin and will input the proper logic level. With no external input connected, the IN A,Pp instruction inputs the previous output status.

This structure allows input and output information on the same pin and also allows any mix of input and output lines on the same port. However, when inputs and outputs are mixed on one PORT, a PORT write will cause the strong internal pull-ups to turn on at all inputs. If a switch or other low impedance device is connected to an input, a PORT write (“1” to an input) could cause current limits on internal lines to
be exceeded. Figure 2-19 illustrates the recommended connection when inputs and outputs are mixed on one PORT.

The bidirectional port structure in combination with the UPI-41AH, 42 logical AND and OR instructions provides an efficient means for handling single line inputs and outputs within an 8-bit processor.

**PORTS 4, 5, 6, and 7**

By using an 8243 I/O expander, 16 additional I/O lines can be connected to the UPI-41AH, 42 and directly addressed as 4-bit I/O ports using UPI-41AH, 42 instructions. This feature saves program space and design time, and improves the bit handling capability of the UPI-41AH, 42.

The lower half of PORT 2 provides an interface to the 8243 as illustrated in Figure 2-20. The PROG pin is used as a strobe to clock address and data information via the PORT 2 interface. The extra 16 I/O lines are referred to in UPI software as PORTS 4, 5, 6, and 7. Each PORT can be directly addressed and can be ANDed and ORed with an immediate data mask. Data can be moved directly to the accumulator from the expander PORTS (or vice-versa).

The 8243 I/O ports, PORTS 4, 5, 6, and 7, provide more drive capability than the UPI-41AH, 42 bidirectional ports. The 8243 output is capable of driving about 5 standard TTL loads.
FUNCTIONAL DESCRIPTION

Multiple 8243's can be connected to the PORT 2 interface. In normal operation, only one of the 8243's would be active at the time an Input or Output command is executed. The upper half of PORT 2 is used to provide chip select signals to the 8243's. Figure 2-21 shows how four 8243's could be connected. Software is needed to select and set the proper PORT 2 pin before an INPUT or OUTPUT command to PORTS 4-7 is executed. In general, the software overhead required is very minor compared to the added flexibility of having a large number of I/O pins available.

Figure 2-20. 8243 Expander Interface

Figure 2-21. Multiple 8243 Expansion
CHAPTER 3
INSTRUCTION SET

The UPI-41AH, 42 Instruction Set is opcode-compatible with the MCS-48 set except for the elimination of external program and data memory instructions and the addition of the data bus buffer instructions. It is very straightforward and efficient in its use of program memory. All instructions are either 1 or 2 bytes in length (over 70% are only 1 byte long) and over half of the instructions execute in one machine cycle. The remainder require only two cycles and include Branch, Immediate, and I/O operations.

The UPI-41AH, 42 Instruction Set efficiently handles the single-bit operations required in control applications. Special instructions allow port bits to be set or cleared individually. Also, any accumulator bit can be directly tested via conditional branch instructions. Additional instructions are included to simplify loop counters, table look-up routines and N-way branch routines.

The UPI-41AH, 42 Microcomputer handles arithmetic operations in both binary and BCD for efficient interface to peripherals such as keyboards and displays.

The instruction set can be divided into the following groups:
- Data Moves
- Accumulator Operations
- Flags
- Register Operations
- Branch Instructions
- Control
- Timer Operations
- Subroutines
- Input/Output Instructions

Data Moves
(See Instruction Summary)

The 8-bit accumulator is the control point for all data transfers within the UPI-41AH, 42. Data can be transferred between the 8 registers of each working register bank and the accumulator directly (i.e., with a source or destination register specified by 3 bits in the instruction). The remaining locations in the RAM array are addressed either by R0 or R1 of the active register bank. Transfers to and from RAM require one cycle.

Constants stored in Program Memory can be loaded directly into the accumulator or the eight working registers. Data can also be transferred directly between the accumulator and the on-board timer/counter, the Status Register (STS), or the Program Status Word (PSW). Transfers to the STS register alter bits 4-7 only. Transfers to the PSW alter machine status accordingly and provide a means of restoring status after an interrupt or of altering the stack pointer if necessary.

Accumulator Operations

Immediate data, data memory, or the working registers can be added (with or without carry) to the accumulator. These sources can also be ANDed, ORed, or exclusive ORed to the accumulator. Data may be moved to or from the accumulator and working registers or data memory. The two values can also be exchanged in a single operation.

The lower 4 bits of the accumulator can be exchanged with the lower 4 bits of any of the internal RAM locations. This operation, along with an instruction which swaps the upper and lower 4-bit halves of the accumulator, provides easy handling of BCD numbers and other 4-bit quantities. To facilitate BCD arithmetic a Decimal Adjust instruction is also included. This instruction is used to correct the result of the binary addition of two 2-digit BCD numbers. Performing a decimal adjust on the result in the accumulator produces the desired BCD result.

The accumulator can be incremented, decremented, cleared, or complemented and can be rotated left or right 1 bit at a time with or without carry.

A subtract operation can be easily implemented in UPI-41AH, 42 software using three single-byte, single-cycle instructions. A value can be subtracted from the accumulator by using the following instructions:
- Complement the accumulator
- Add the value to the accumulator
- Complement the accumulator

Flags

There are four user accessible flags:
- Carry
- Auxiliary Carry
- F0
- F1

The Carry flag indicates overflow of the accumulator, while the Auxiliary Carry flag indicates overflow between BCD digits and is used during decimal adjust operations. Both Carry and Auxiliary Carry are part of the Program Status Word (PSW) and are stored in the stack during subroutine calls. The F0 and F1 flags are general-purpose flags which can be cleared or complemented by UPI instructions. F0 is accessible via the Program Status Word and is stored in the stack with the Carry flags. F1 reflects the condition of the A0 line, and caution must be used when setting or clearing it.
Register Operations
The working registers can be accessed via the accumulator as explained above, or they can be loaded with immediate data constants from program memory. In addition, they can be incremented or decrement directly, or they can be used as loop counters as explained in the section on branch instructions.

Additional Data Memory locations can be accessed with indirect instructions via R0 and R1.

Branch Instructions
The UPI-41AH, 42 Instruction Set includes 17 jump instructions. The unconditional jump instruction allows jumps anywhere in the 1K words of program memory. All other jump instructions are limited to the current page (256 words) of program memory.

Conditional jump instructions can test the following inputs and machine flags:
- TEST 0 input pin
- TEST 1 input pin
- Input Buffer Full flag
- Output Buffer Full flag
- Timer flag
- Accumulator zero
- Accumulator bit
- Carry flag
- FO flag
- F1 flag

The conditions tested by these instructions are the instantaneous values at the time the conditional jump instruction is executed. For instance, the jump on accumulator zero instruction tests the accumulator itself, not an intermediate flag.

The decrement register and jump if not zero (DJNZ) instruction combines decrement and branch operations in a single instruction which is useful in implementing a loop counter. This instruction can designate any of the 8 working registers as a counter and can effect a branch to any address within the current page of execution.

A special indirect jump instruction (JMP @A) allows the program to be vectored to any one of several different locations based on the contents of the accumulator. The contents of the accumulator point to a location in program memory which contains the jump address. As an example, this instruction could be used to vector to any one of several routines based on an ASCII character which has been loaded into the accumulator. In this way, ASCII inputs can be used to initiate various routines.

Control
The UPI-41AH, 42 Instruction Set has six instructions for control of the DMA, interrupts, and selection of working register banks.

The UPI-41AH, 42 provides two instructions for control of the external microcomputer system. IBF and OBF flags can be routed to PORT 2 allowing interrupts of the external processor. DMA handshaking signals can also be enabled using lines from PORT 2.

The IBF interrupt can be enabled and disabled using two instructions. Also, the interrupt is automatically disabled following a RESET input or during an interrupt service routine.

The working register bank switch instructions allow the programmer to immediately substitute a second 8 register bank for the one in use. This effectively provides either 16 working registers or the means for quickly saving the contents of the first 8 registers in response to an interrupt. The user has the option of switching register banks when an interrupt occurs. However, if the banks are switched, the original bank will automatically be restored upon execution of a return and restore status (RETR) instruction at the end of the interrupt service routine.

Timer
The 8-bit on-board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting.

The counter can be started as a timer with an internal clock source or as an event counter or timer with an external clock applied to the TEST 1 pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

Subroutines
Subroutines are entered by executing a call instruction. Calls can be made to any address in the 1K word program memory. Two separate return instructions determine whether or not status (i.e., the upper 4 bits of the PSW) is restored upon return from a subroutine.

Input/Output Instructions
Two 8-bit data bus buffer registers (DBBIN and DBOOT) and an 8-bit status register (STS) enable the UPI-41A universal peripheral interface to communicate with the external microcomputer system. Data can be INPutted from the DBBIN register to
INSTRUCTION SET

The accumulator. Data can be OUTputted from the accumulator to the DBBOUT register.

The STS register contains four user-definable bits (ST4-ST7) plus four reserved status bits (IBF, OBF, F0, and F1). The user-definable bits are set from the accumulator.

The UPI-41AH, 42 peripheral interface has two 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs to the ports are sampled at the time an IN instruction is executed. In addition, immediate data from program memory can be ANDed and ORed directly to PORTS 1 and 2 with the result remaining on the port. This allows “masks” stored in program memory to be used to set or reset individual bits on the I/O ports. PORTS 1 and 2 are configured to allow input on a given pin by first writing a “1” to the pin.

Four additional 4-bit ports are available through the 8243 I/O expander device. The 8243 interfaces to the UPI-41AH, 42 peripheral interface via four PORT 2 lines which form an expander bus. The 8243 ports have their own AND and OR instructions like the on-board ports, as well as move instructions to transfer data in or out. The expander AND or OR instructions, however, combine the contents of the accumulator with the selected port rather than with immediate data as is done with the on-board ports.

INSTRUCTION SET DESCRIPTION

The following section provides a detailed description of each UPI instruction and illustrates how the instructions are used.

For further information about programming the UPI, consult the 8048/8041A Assembly Language Manual.

Table 3-1. Symbols and Abbreviations Used

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Accumulator</td>
</tr>
<tr>
<td>C</td>
<td>Carry</td>
</tr>
<tr>
<td>DBBIN</td>
<td>Data Bus Buffer Input</td>
</tr>
<tr>
<td>DBBOUT</td>
<td>Data Bus Buffer Output</td>
</tr>
<tr>
<td>F0,F1</td>
<td>FLAG 0, FLAG 1 (C/D flag)</td>
</tr>
<tr>
<td>I</td>
<td>Interrupt</td>
</tr>
<tr>
<td>P</td>
<td>Mnemonic for “in-page” operation</td>
</tr>
<tr>
<td>PC</td>
<td>Program Counter</td>
</tr>
<tr>
<td>Pp</td>
<td>Port designator (p = 1, 2, or 4-7)</td>
</tr>
<tr>
<td>PSW</td>
<td>Program Status Word</td>
</tr>
<tr>
<td>Rr</td>
<td>Register designator (r = 0-7)</td>
</tr>
<tr>
<td>SP</td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>STS</td>
<td>Status register</td>
</tr>
<tr>
<td>T</td>
<td>Timer</td>
</tr>
<tr>
<td>TF</td>
<td>Timer Flag</td>
</tr>
<tr>
<td>T0,T1</td>
<td>TEST 0, TEST 1</td>
</tr>
<tr>
<td>#</td>
<td>Immediate data prefix</td>
</tr>
<tr>
<td>@</td>
<td>Indirect address prefix</td>
</tr>
<tr>
<td>()</td>
<td>Double parentheses show the effect of @, that is, @RO is shown as ((RO)).</td>
</tr>
<tr>
<td>()</td>
<td>Contents of</td>
</tr>
</tbody>
</table>

Table 3-2. Instruction Set Summary

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operation Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>A,Rr</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD</td>
<td>A,@Rr</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD</td>
<td>A,#data</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC</td>
<td>A,Rr</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ADDC</td>
<td>A,@Rr</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC</td>
<td>A,#data</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL</td>
<td>A,Rr</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL</td>
<td>A,@Rr</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL</td>
<td>A,#data</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL</td>
<td>A,Rr</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL</td>
<td>A,@Rr</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL</td>
<td>A,#data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XRL</td>
<td>A,Rr</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL</td>
<td>A,@Rr</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL</td>
<td>A,#data</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INC</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DA</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RR</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### Table 3-2. Instruction Set Summary (Con't.)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operation Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INPUT/OUTPUT</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>A,Pp</td>
<td>Input port to A</td>
<td>1</td>
</tr>
<tr>
<td>OUTL</td>
<td>Pp,A</td>
<td>Output A to port</td>
<td>1</td>
</tr>
<tr>
<td>ANL</td>
<td>Pp,#data</td>
<td>And immediate to port</td>
<td>2</td>
</tr>
<tr>
<td>ORL</td>
<td>Pp,#data</td>
<td>Or immediate to port</td>
<td>2</td>
</tr>
<tr>
<td>IN</td>
<td>A, DBB</td>
<td>Input DBB to A, clear IBF</td>
<td>1</td>
</tr>
<tr>
<td>OUT</td>
<td>DBB,A</td>
<td>Output A to DBB, Set OBF</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>STS,A</td>
<td>A4–A7 to bits 4–7 of status</td>
<td>1</td>
</tr>
<tr>
<td>MOVD</td>
<td>A,Pp</td>
<td>Input Expander port to A</td>
<td>1</td>
</tr>
<tr>
<td>MOVD</td>
<td>Pp,A</td>
<td>Output A to Expander port</td>
<td>1</td>
</tr>
<tr>
<td>ANLD</td>
<td>Pp,A</td>
<td>And A to Expander port</td>
<td>1</td>
</tr>
<tr>
<td>ORLD</td>
<td>Pp,A</td>
<td>Or A to Expander port</td>
<td>1</td>
</tr>
<tr>
<td><strong>DATA MOVES</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>A,Rr</td>
<td>Move register to A</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>A,@Rr</td>
<td>Move data memory to A</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>A,#data</td>
<td>Move immediate to A</td>
<td>2</td>
</tr>
<tr>
<td>MOV</td>
<td>Rr,A</td>
<td>Move A to register</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>@Rr,A</td>
<td>Move A to data memory</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>Rr,#data</td>
<td>Move immediate to register</td>
<td>2</td>
</tr>
<tr>
<td>MOV</td>
<td>@Rr,#data</td>
<td>Move immediate to data memory</td>
<td>2</td>
</tr>
<tr>
<td>MOV</td>
<td>A,PSW</td>
<td>Move PSW to A</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>PSW,A</td>
<td>Move A to PSW</td>
<td>1</td>
</tr>
<tr>
<td>XCH</td>
<td>A,Rr</td>
<td>Exchange A and registers</td>
<td>1</td>
</tr>
<tr>
<td>XCH</td>
<td>A,@Rr</td>
<td>Exchange A and data memory</td>
<td>1</td>
</tr>
<tr>
<td>XCHD</td>
<td>A,@Rr</td>
<td>Exchange digit of A and register</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>A,@A</td>
<td>Move to A from current page</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>A@A</td>
<td>Move to A from Page 3</td>
<td>1</td>
</tr>
<tr>
<td><strong>TIMER/COUNTER</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>A,T</td>
<td>Read Timer/Counter</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>T,A</td>
<td>Load Timer/Counter</td>
<td>1</td>
</tr>
<tr>
<td>STRT</td>
<td>T</td>
<td>Start Timer</td>
<td>1</td>
</tr>
<tr>
<td>STRT</td>
<td>CNT</td>
<td>Start Counter</td>
<td>1</td>
</tr>
<tr>
<td>STOP</td>
<td>TCNT</td>
<td>Stop Timer/Counter</td>
<td>1</td>
</tr>
<tr>
<td>EN</td>
<td>TCNTI</td>
<td>Enable Timer/Counter Interrupt</td>
<td>1</td>
</tr>
<tr>
<td>DIS</td>
<td>TCNTI</td>
<td>Disable Timer/Counter Interrupt</td>
<td>1</td>
</tr>
<tr>
<td><strong>CONTROL</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>DMA</td>
<td>Enable DMA Handshake Lines</td>
<td>1</td>
</tr>
<tr>
<td>EN</td>
<td>I</td>
<td>Enable IBF interrupt</td>
<td>1</td>
</tr>
<tr>
<td>DIS</td>
<td>I</td>
<td>Disable IBF interrupt</td>
<td>1</td>
</tr>
<tr>
<td>EN</td>
<td>FLAGS</td>
<td>Enable Master Interrupts</td>
<td>1</td>
</tr>
<tr>
<td>SEL</td>
<td>RB0</td>
<td>Select register bank 0</td>
<td>1</td>
</tr>
<tr>
<td>SEL</td>
<td>RB1</td>
<td>Select register bank 1</td>
<td>1</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td>No Operation</td>
<td>1</td>
</tr>
<tr>
<td><strong>REGISTERS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>Rr</td>
<td>Increment register</td>
<td>1</td>
</tr>
<tr>
<td>INC</td>
<td>@Rr</td>
<td>Increment data memory</td>
<td>1</td>
</tr>
<tr>
<td>DEC</td>
<td>Rr</td>
<td>Decrement register</td>
<td>1</td>
</tr>
<tr>
<td><strong>SUBROUTINE</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>addr</td>
<td>Jump to subroutine</td>
<td>2</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td>Return</td>
<td>1</td>
</tr>
<tr>
<td>RETR</td>
<td></td>
<td>Return and restore status</td>
<td>1</td>
</tr>
<tr>
<td><strong>FLAGS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR C</td>
<td></td>
<td>Clear Carry</td>
<td>1</td>
</tr>
<tr>
<td>CPL C</td>
<td></td>
<td>Complement Carry</td>
<td>1</td>
</tr>
<tr>
<td>CLR F0</td>
<td></td>
<td>Clear Flag 0</td>
<td>1</td>
</tr>
<tr>
<td>CPL F0</td>
<td></td>
<td>Complement Flag 0</td>
<td>1</td>
</tr>
<tr>
<td>CLR F1</td>
<td></td>
<td>Clear F1 Flag</td>
<td>1</td>
</tr>
<tr>
<td>CPL F1</td>
<td></td>
<td>Complement F1 Flag</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 3-2. Instruction Set Summary (Con't.)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operation Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BRANCH</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>addr</td>
<td>Jump unconditional</td>
<td>2</td>
</tr>
<tr>
<td>JMPP</td>
<td>@A</td>
<td>Jump indirect</td>
<td>1</td>
</tr>
<tr>
<td>DJNZ</td>
<td>Rr,addr</td>
<td>Decrement register and jump on non-zero</td>
<td>2</td>
</tr>
<tr>
<td>JC</td>
<td>addr</td>
<td>Jump on Carry=1</td>
<td>2</td>
</tr>
<tr>
<td>JNC</td>
<td>addr</td>
<td>Jump on Carry=0</td>
<td>2</td>
</tr>
<tr>
<td>JZ</td>
<td>addr</td>
<td>Jump on A Zero</td>
<td>2</td>
</tr>
<tr>
<td>JNZ</td>
<td>addr</td>
<td>Jump on A not Zero</td>
<td>2</td>
</tr>
<tr>
<td>JT0</td>
<td>addr</td>
<td>Jump on T0=1</td>
<td>2</td>
</tr>
<tr>
<td>JNT0</td>
<td>addr</td>
<td>Jump on T0=0</td>
<td>2</td>
</tr>
<tr>
<td>JT1</td>
<td>addr</td>
<td>Jump on T1=1</td>
<td>2</td>
</tr>
<tr>
<td>JNT1</td>
<td>addr</td>
<td>Jump on T1=0</td>
<td>2</td>
</tr>
<tr>
<td>JF0</td>
<td>addr</td>
<td>Jump on F0 Flag=1</td>
<td>2</td>
</tr>
<tr>
<td>JF1</td>
<td>addr</td>
<td>Jump on F1 Flag=1</td>
<td>2</td>
</tr>
<tr>
<td>JTF</td>
<td>addr</td>
<td>Jump on Timer Flag=1</td>
<td>2</td>
</tr>
<tr>
<td>JNBF</td>
<td>addr</td>
<td>Jump on IBF Flag=0</td>
<td>2</td>
</tr>
<tr>
<td>JOBF</td>
<td>addr</td>
<td>Jump on OBF Flag=1</td>
<td>2</td>
</tr>
<tr>
<td>JBB</td>
<td>addr</td>
<td>Jump on Accumulator Bit</td>
<td>2</td>
</tr>
</tbody>
</table>

ALPHABETIC LISTING

ADD A,Rr  Add Register Contents to Accumulator

Opcode: \[0 \ 1 \ 1 \ 0 \ \ r_2 \ r_1 \ r_0\]

The contents of register 'r' are added to the accumulator. Carry is affected.

\[(A) \rightarrow (A) + (Rr)\]

Example: ADDRREG: ADD A,R6 ;ADD REG 6 CONTENTS ;TO ACC

ADD A,@Rr  Add Data Memory Contents to Accumulator

Opcode: \[0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ r\]

The contents of the standard data memory location addressed by register 'r' bits 0–5 are added to the accumulator. Carry is affected.

\[(A) \leftarrow (A) + ((Rr))\]

Example: ADDM: MOV RO,#47 ;MOVE 47 DECIMAL TO REG 0 ADD A,@R0 ;ADD VALUE OF LOCATION ;47 TO ACC

ADD A,#data  Add Immediate Data to Accumulator

Opcode: \[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ d_7 \ d_6 \ d_5 \ d_4 \ d_3 \ d_2 \ d_1 \ d_0\]

This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected.

\[(A) \leftarrow (A) + \text{data}\]

Example: ADDID: ADD A,#ADDER ;ADD VALUE OF SYMBOL ;'ADDER' TO ACC

5-660
### INSTRUCTION SET

**ADDC A,Rr**  Add Carry and Register Contents to Accumulator

**Opcode:**

| 0 | 1 | 1 | 1 | r2 | r1 | r0 |

The content of the carry bit is added to accumulator location 0. The contents of register 'r' are then added to the accumulator. Carry is affected.

\[(A) \leftarrow (A) + ((Rr) + (C)) \quad r=0-7\]

**Example:**

| ADDRG C: ADDC A,R4 | ;ADD CARRY AND REG 4 |
| ;CONTENTS TO ACC |

**ADDC A,@Rr**  Add Carry and Data Memory Contents to Accumulator

**Opcode:**

| 0 | 1 | 1 | 0 | 0 | 0 | r |

The content of the carry bit is added to accumulator location 0. Then the contents of the standard data memory location addressed by register 'r' bits 0-5 are added to the accumulator. Carry is affected.

\[(A) \leftarrow (A) + (\text{Rr}) + (C) \quad r=0-1\]

**Example:**

| ADDM C: MOV R1,#40 | ;MOV '40' DEC TO REG 1 |
| ADDC A,@R1 | ;ADD CARRY AND LOCATION 40 |
| ;CONTENTS TO ACC |

**ADDC A,#data**  Add Carry and Immediate Data to Accumulator

**Opcode:**

| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0. Then the specified data is added to the accumulator. Carry is affected.

\[(A) \leftarrow (A) + \text{data} + (C)\]

**Example:**

| ADDC A,#255 | ;ADD CARRY AND '225' DEC |
| ;TO ACC |

**ANL A,Rr**  Logical AND Accumulator With Register Mask

**Opcode:**

| 0 | 1 | 0 | 1 | r2 | r1 | r0 |

Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

\[(A) \leftarrow (A) \text{ AND (Rr)} \quad r=0-7\]

**Example:**

| ANDREG: ANL A,R3 | ;'AND' ACC CONTENTS WITH MASK |
| ;MASK IN REG 3 |

**ANL A,@Rr**  Logical AND Accumulator With Memory Mask

**Opcode:**

| 0 | 1 | 0 | 1 | 0 | 0 | 0 | r |

Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'r', bits 0-5.

\[(A) \leftarrow (A) \text{ AND ((Rr))} \quad r=0-1\]

**Example:**

| ANDDM: MOV R0,#OFFH | ;MOVE 'FF' HEX TO REG 0 |
| ANL A,#0AFH | ;'AND' ACC CONTENTS WITH |
| ;MASK IN LOCATION 63 |
INSTRUCTION SET

ANL A,#data  Logical AND Accumulator With Immediate Mask

Opcode: \[ \begin{array}{c} \text{O} \text{C} \text{d} \text{e} \text{x} \\ \text{0} \text{1} \text{0} \text{1} \text{0} \text{0} \text{1} \text{1} \end{array} \] \quad \begin{array}{c} d_7 \ d_6 \ d_5 \ d_4 \ d_3 \ d_2 \ d_1 \ d_0 \end{array} \]

This is a 2-cycle instruction. Data in the accumulator is logically ANDed with an immediately-specified mask.

\((A) \leftarrow (A) \text{ AND data} \)

Example:

\begin{align*}
\text{ANLD: ANL A,} \#0AFH &; \text{AND' ACC CONTENTS} \\
\text{ANL A,} \#3+X/Y &; \text{AND' ACC CONTENTS}
\end{align*}

\begin{align*}
\text{ANLD Pp,A} &; \text{Logical AND Port 4-7 With Accumulator Mask} \\
\text{Opcode:} & \begin{array}{c} \text{O} \text{C} \text{d} \text{e} \text{x} \\ \text{1} \text{0} \text{0} \text{1} \text{1} \text{1} \text{p} \text{1} \text{p} \text{0} \end{array} \end{align*}

This is a 2-cycle instruction. Data on port 'Pp' on the 8243 expander is logically ANDed with the digit mask contained in accumulator bits 0-3.

\( \text{PORT Pp \rightarrow (Pp) \text{ AND (A0-3)}} \quad p=4-7 \)

Note:
The mapping of Port 'Pp' to opcode bits \( p_1,p_0 \) is as follows:

\begin{array}{c|c|c|c}
\text{P1} & \text{P0} & \text{Port} \\
\hline
\text{0} & \text{0} & \text{4} \\
\text{0} & \text{1} & \text{5} \\
\text{1} & \text{0} & \text{6} \\
\text{1} & \text{1} & \text{7} \\
\end{array}

Example:

\begin{align*}
\text{ANLP4: ANLD P4,A} &; \text{AND' PORT 4 CONTENTS} \\
\text{;WITH ACC BITS 0-3}
\end{align*}
CALL address Subroutine Call

Opcode: 0 ag ag 1 0 1 0 0

This is a 2-cycle instruction. The program counter and PSW bits 4–7 are saved in the stack. The stack pointer (PSW bits 0–2) is updated. Program control is then passed to the location specified by ‘address’.

Execution continues at the instruction following the CALL upon return from the subroutine.

\[
\begin{align*}
((SP)) &\leftarrow (PC), (PSW_{4-7}) \\
(SP) &\leftarrow (SP) + 1 \\
(PC_{8-g}) &\leftarrow (addr_{8-g}) \\
(PC_{0-7}) &\leftarrow (addr_{0-7})
\end{align*}
\]

Example: Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```
MOV R0,#50 ; MOVE '50' DEC TO ADDRESS
BEGADD: MOV A,R1 ; MOVE CONTENTS OF REG 1 TO ACC
ADD A,R2 ; ADD REG 2 TO ACC
CALL SUBTOT ; CALL SUBROUTINE 'SUBTOT'
ADD A,R3 ; ADD REG 3 TO ACC
ADD A,R4 ; ADD REG 4 TO ACC
CALL SUBTOT ; CALL SUBROUTINE 'SUBTOT'
ADD A,R5 ; ADD REG 5 TO ACC
ADD A,R6 ; ADD REG 6 TO ACC
CALL SUBTOT ; CALL SUBROUTINE 'SUBTOT'

SUBTOT: MOV @R0,A ; MOVE CONTENTS OF ACC TO LOCATION ADDRESSED BY
; REG 0
INC R0 ; INCREMENT REG 0
RET ; RETURN TO MAIN PROGRAM
```

CLR A Clear Accumulator

Opcode: 0 0 1 0 0 1 1 1

The contents of the accumulator are cleared to zero.

(A) ← 00H

CLR C Clear Carry Bit

Opcode: 1 0 0 1 0 1 1 1

During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPLC, RRC, and DAA instructions. This instruction resets the carry bit to zero.

(C) ← 0

CLR F1 Clear Flag 1

Opcode: 1 0 1 0 0 1 0 1

The F1 flag is cleared to zero.

(F1) ← 0
INSTRUCTION SET

CLR F0  Clear Flag 0

Opcode: 1 0 0 0 | 0 1 0 1

Flag 0 is cleared to zero.
(F0) ← 0

CPL A  Complement Accumulator

Opcode: 0 0 1 1 | 0 1 1 1

The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.

Example: Assume accumulator contains 01101010.
CPLA: CPL A ;ACC CONTENTS ARE COMPLEMENTED TO 10010101

CPL C  Complement Carry Bit

Opcode: 1 0 1 0 | 0 1 1 1

The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.

Example: Set C to one; current setting is unknown.
CT01: CLR C ;C IS CLEARED TO ZERO
CPL C ;C IS SET TO ONE

CPL F0  Complement Flag 0

Opcode: 1 0 0 1 | 0 1 0 1

The setting of Flag 0 is complemented; one is changed to zero, and zero is changed to one.
F0 ← NOT (F0)

CPL F1  Complement Flag 1

Opcode: 1 0 1 1 | 0 1 0 1

The setting of the F1 Flag is complemented; one is changed to zero, and zero is changed to one.
(F1) ← NOT (F1)
INSTRUCTION SET

DA A  Decimal Adjust Accumulator

Opcode: 01010111

The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0–3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4–7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one; otherwise, it is cleared to zero.

Example: Assume accumulator contains 9AH.

DA A ;ACC ADJUSTED TO 01H with C set
C AC ACC
0 0 9AH INITIAL CONTENTS
06H ADD SIX TO LOW DIGIT
0 0 A1H
60H ADD SIX TO HIGH DIGIT
1 0 01H RESULT

DEC A  Decrement Accumulator

Opcode: 00000111

The contents of the accumulator are decremented by one.

(A) ← (A) − 1

Example: Decrement contents of data memory location 63.

MOV R0,#3FH ;MOVE '3F' HEX TO REG 0
MOV A,@R0 ;MOVE CONTENTS OF LOCATION 63 ;TO ACC
DEC A ;DECREMENT ACC
MOV @R0,A ;MOVE CONTENTS OF ACC TO LOCATION 63

DEC Rr  Decrement Register

Opcode: 1100010

The contents of working register 'r' are decremented by one.

(r) ← (r) − 1

Example: DECR1: DEC R1 ;DECREMENT ADDRESS REG 1

DIS I  Disable IBF Interrupt

Opcode: 00010101

The input Buffer Full interrupt is disabled. The interrupt sequence is not initiated by WR and CS, however, an IBF interrupt request is latched and remains pending until an EN I (enable IBF interrupt) instruction is executed.

Note: The IBF flag is set and cleared independent of the IBF interrupt request so that handshaking protocol can continue normally.
## INSTRUCTION SET

### DIS TCNTI Disable Timer/Counter Interrupt

**Opcode:** 00110101

The timer/counter interrupt is disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

### DJNZ Rr, address Decrement Register and Test

**Opcode:** 11 1 1 0 1 0 1

This is a 2-cycle instruction. Register ‘r’ is decremented and tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified address within the current page.

\[
(Rr) \leftarrow (Rr) - 1
\]

If \(R \neq 0\), then;

\[
(PC) \leftarrow addr
\]

**Note:** A 10-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it will jump to a target address on the following page. Otherwise, it is limited to a jump within the current page.

**Example:** Increment values in data memory locations 50–54.

```assembly
MOV R0, #50 ; MOVE '50' DEC TO ADDRESS
MOV R3, #05 ; MOVE '5' DEC TO COUNTER

INCR; INC @R0 ; INCREMENT CONTENTS OF LOCATION ADDRESSED BY REG 0
INC R0 ; INCREMENT ADDRESS IN REG 0
DJNZ R3, INCRT ; DECREMENT REG 3 ––– JUMP TO 'INCRT' IF REG 3 NONZERO
NEXT ––– ; 'NEXT' ROUTINE EXECUTED IF R3 IS ZERO
```

### EN DMA Enable DMA Handshake Lines

**Opcode:** 11100101

DMA handshaking is enabled using P26 as DMA request (DRQ) and P27 as DMA acknowledge (DACK). The DACK line forces CS and A0 low internally and clears DRQ.

### EN FLAGS Enable Master Interrupts

**Opcode:** 11110101

The Output Buffer Full (OBF) and the Input Buffer Full (IBF) flags (IBF is inverted) are routed to P24 and P25. For proper operation, a “1” should be written to P25 and P24 before the EN FLAGS instruction. A “0” written to P24 or P25 disables the pin.
INSTRUCTION SET

**Enable IBF Interrupt**

Opcode: 0 0 0 0 0 1 0 1

The Input Buffer Full interrupt is enabled. A low signal on WR and CS initiates the interrupt sequence.

**Enable Timer/Counter Interrupt**

Opcode: 0 0 1 0 0 1 0 1

The timer/counter interrupt is enabled. An overflow of this register initiates the interrupt sequence.

**Input Data Bus Buffer Contents to Accumulator**

Opcode: 0 0 1 0 0 0 1 0

Data in the DBBIN register is transferred to the accumulator and the Input Buffer Full (IBF) flag is set to zero.

(A) ← (DBB)

(IBF) ← 0

Example: INDBB: IN A, DBB ; INPUT DBBIN CONTENTS TO ACCUMULATOR

**Input Port 1–2 Data to Accumulator**

Opcode: 0 0 0 0 1 0 p1 p0

This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator.

(A) ← (Pp)

p = 1–2 (see ANL instruction)

Example: INP12: IN A, P1 ; INPUT PORT 1 CONTENTS TO ACC

**Increment Accumulator**

Opcode: 0 0 0 1 0 1 1 1

The contents of the accumulator are incremented by one.

(A) ← (A) + 1

Example: Increment contents of location 10 in data memory.

INCA: MOV R0, #10 ; MOV '10' DEC TO ADDRESS

MOV A, @R0 ; MOVE CONTENTS OF LOCATION

MOV @R0, A ; MOVE ACC CONTENTS TO LOCATION 10
INSTRUCTION SET

INC Rr  Increment Register

Opcode: 0001 1 r2 r1 r0

The contents of working register 'r' are incremented by one.

Example: INCR0: INC R0 ;INCREMENT ADDRESS REG 0

INC @Rr  Increment Data Memory Location

Opcode: 0001 000 r

The contents of the resident data memory location addressed by register 'r' bits 0–5 are incremented by one.

Example: INCDM: MOV R1,#OFFH ;MOVE ONES TO REG 1
          INC @R1 ;INCREMENT LOCATION 63

JBb address  Jump If Accumulator Bit is Set

Opcode: b2 b1 b0 1 0 0 1 0 • a7 a6 a5 a4 a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

Example: JB4S1: JB4 NEXT ;JUMP TO ‘NEXT’ ROUTINE
          ;IF ACC BIT 4=1

JC address  Jump If Carry Is Set

Opcode: 1 1 1 1 0 1 1 0 • a7 a6 a5 a4 a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

Example: JC1: JC OVERFLOW ;JUMP TO ‘OVERFLOW’ ROUTINE
          ;IF C=1

JFO address  Jump If Flag 0 Is Set

Opcode: 1 0 1 1 0 1 1 0 • a7 a6 a5 a4 a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

Example: JFOIS1: JFO TOTAL ;JUMP TO ‘TOTAL’ ROUTINE
          ;IF F0=1
INSTRUCTION SET

**JF1 address**  Jump If C/D Flag (F1) Is Set

| Opcode: | 0 1 1 1 0 1 1 0 | a7 a6 a5 a4 a3 a2 a1 a0 |

This is a 2-cycle instruction. Control passes to the specified address if the C/D flag (F1) is set to one.

*(PC0-7) ← baddr if F1 = 1*

**Example:**

JF 1S1: JF1 FILBUF ;JUMP TO 'FILBUF'

;ROUTINE IF F1 = 1

**JMP address**  Direct Jump Within 1K Block

| Opcode: | a10 a9 a8 0 0 1 0 0 | a7 a6 a5 a4 a3 a2 a1 a0 |

This is a 2-cycle instruction. Bits 0–9 of the program counter are replaced with the directly-specified address.

*(PC9-9) ← baddr 8-9 (PC0-7) ← baddr 0-7*

**Example:**

JMP SUBTOT ;JUMP TO SUBROUTINE 'SUBTOT'

JMP $-6 ;JUMP TO INSTRUCTION SIX LOCATIONS

;BEFORE CURRENT LOCATION

JMP 2FH ;JUMP TO ADDRESS ‘2F’ HEX

**JMPP @A**  Indirect Jump Within Page

| Opcode: | 1 0 1 1 0 0 1 1 |

This is a 2-cycle instruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC 0–7).

*(PC0-7) ← (A)*

**Example:**

Assume accumulator contains OFH

JMPPAG: JMPP @A ;JMP TO ADDRESS STORED IN LOCATION 15 IN CURRENT PAGE

**JNC address**  Jump If Carry Is Not Set

| Opcode: | 1 1 1 0 0 1 1 0 | a7 a6 a5 a4 a3 a2 a1 a0 |

This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

*(PC0-7) ← baddr if C = 0*

**Example:**

JCO: JNC NOVFLO ;JUMP TO 'NOVFLO' ROUTINE

;IF C = 0

**JNIBF address**  Jump If Input Buffer Full Flag Is Low

| Opcode: | 1 1 0 1 0 1 1 0 | a7 a6 a5 a4 a3 a2 a1 a0 |

This is a 2-cycle instruction. Control passes to the specified address if the Input Buffer Full flag is low (IBF=0).

*(PC0-7) ← baddr if IBF = 0*

**Example:**

LOC 3:JNIBF LOC 3 ;JUMP TO SELF IF IBF=0

;OTHERWISE CONTINUE

5-669
INSTRUCTION SET

JNTO address  Jump If TEST 0 is Low

 Opcode: 0 0 1 0 0 1 1 0  \( \bullet \) a7 a6 a5 a4 a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address, if the TEST 0 signal is low. Pin is sampled during SYNC.

\( (P_{C0-7}) \leftarrow \text{addr} \) if \( T_0 = 0 \)

Example: JTNLOW: JNTO 60 ; JUMP TO LOCATION 60 DEC ; IF \( T_0 = 0 \)

JNT1 address  Jump If TEST 1 is Low

 Opcode: 0 1 0 0 0 1 1 0  \( \bullet \) a7 a6 a5 a4 a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if the TEST 1 signal is low. Pin is sampled during SYNC.

\( (P_{C0-7}) \leftarrow \text{addr} \) if \( T_1 = 0 \)

Example: JNT1LOW: JNT1 OBBH ; JUMP TO LOCATION 'BB' HEX ; IF \( T_1 = 0 \)

JNZ address  Jump If Accumulator Is Not Zero

 Opcode: 1 0 0 1 0 1 1 0  \( \bullet \) a7 a6 a5 a4 a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if the accumulator contents are nonzero at the time this instruction is executed.

\( (P_{C0-7}) \leftarrow \text{addr} \) if \( A \neq 0 \)

Example: JACCNO: JNZ OABH ; JUMP TO LOCATION 'AB' HEX ; IF ACC VALUE IS NONZERO

JOBF Address  Jump If Output Buffer Full Flag Is Set

 Opcode: 1 0 0 0 0 1 1 0  \( \bullet \) a7 a6 a5 a4 a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if the Output Buffer Full (OBF) flag is set (= 1) at the time this instruction is executed.

\( (P_{C0-7}) \leftarrow \text{addr} \) if \( OBF = 1 \)

Example: JOBFHI: JOBF OAAH ; JUMP TO LOCATION 'AA' HEX ; IF \( OBF = 1 \)

JTF address  Jump If Timer Flag Is Set

 Opcode: 0 0 0 1 0 1 1 0  \( \bullet \) a7 a6 a5 a4 a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register overflows to zero. The timer flag is cleared upon execution of this instruction. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)

\( (P_{C0-7}) \leftarrow \text{addr} \) if \( TF = 1 \)

Example: JTF1: JTF TIMER ; JUMP TO 'TIMER' ROUTINE ; IF \( TF = 1 \)
## INSTRUCTION SET

### JTO address  Jump If TEST 0 is High

**Opcode:**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>a7</th>
<th>a6</th>
<th>a5</th>
<th>a4</th>
<th>a3</th>
<th>a2</th>
<th>a1</th>
<th>a0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is a 2-cycle instruction. Control passes to the specified address if the TEST 0 signal is high (= 1). Pin is sampled during SYNC.

\[(PC_0-7) \leftarrow \text{addr} \quad \text{if } T_0 = 1\]

**Example:**

JTOHl: JTO 53

; JUMP TO LOCATION 53 DEC

; IF \(T_0 = 1\)

### JT1 address  Jump If TEST 1 is High

**Opcode:**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>a7</th>
<th>a6</th>
<th>a5</th>
<th>a4</th>
<th>a3</th>
<th>a2</th>
<th>a1</th>
<th>a0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is a 2-cycle instruction. Control passes to the specified address if the TEST 1 signal is high (= 1). Pin is sampled during SYNC.

\[(PC_0-7) \leftarrow \text{addr} \quad \text{if } T_1 = 1\]

**Example:**

JTIHI: JT1 COUNT

; JUMP TO 'COUNT' ROUTINE

; IF \(T_1 = 1\)

### JZ address  Jump If Accumulator is Zero

**Opcode:**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>a7</th>
<th>a6</th>
<th>a5</th>
<th>a4</th>
<th>a3</th>
<th>a2</th>
<th>a1</th>
<th>a0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

\[(PC_0-7) \leftarrow \text{addr} \quad \text{if } A = 0\]

**Example:**

JACCO: JZ OA3H

; JUMP TO LOCATION 'A3' HEX

; IF ACC VALUE IS ZERO

### MOV A,#data  Move Immediate Data to Accumulator

**Opcode:**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>d7</th>
<th>d6</th>
<th>d5</th>
<th>d4</th>
<th>d3</th>
<th>d2</th>
<th>d1</th>
<th>d0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

\[(A) \leftarrow \text{data}\]

**Example:**

MOV A,#OA3H

; MOV 'A3' HEX TO ACC

### MOV A,PSW  Move PSW Contents to Accumulator

**Opcode:**

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The contents of the program status word are moved to the accumulator.

\[(A) \leftarrow (PSW)\]

**Example:**

Jump to 'RB1SET' routine if bank switch, PSW bit 4, is set.

BSCHK: MOV A,PSW

; MOV PSW CONTENTS TO ACC

JB4 RB1 SET

; JUMP TO 'RB1SET' IF ACC

; BIT 4 = 1
### INSTRUCTION SET

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Opcode</th>
<th>Example</th>
</tr>
</thead>
</table>
| MOV A, Rr   | Move Register Contents to Accumulator | 1111111r2 r1 r0 | Eight bits of data are moved from working register ‘r’ into the accumulator. 
(A) ← (Rr) 
Example: MAR: MOV A,R3 ; MOVE CONTENTS OF REG 3 ; TO ACC |
| MOV A,@Rr   | Move Data Memory Contents to Accumulator | 1111000r | The contents of the data memory location addressed by bits 0–5 of register ‘r’ are moved to the accumulator. 
(A) ← ((Rr)) 
Example: 
Assume R1 contains 00110110. 
MADM: MOV A,@R1 ; MOVE CONTENTS OF DATA MEM ; LOCATION 54 TO ACC |
| MOV A,T     | Move Timer/Counter Contents to Accumulator | 01000010 | The contents of the timer/event-counter register are moved to the accumulator. The timer/event-counter is not stopped. 
(A) ← (T) 
Example: Jump to "EXIT" routine when timer reaches ‘64’, that is, when bit 6 is set—assuming initialization to zero. 
TIMCHK: MOV A,T ; MOVE TIMER CONTENTS TO 
ACC 
JB6 EXIT ; JUMP TO 'EXIT' IF ACC BIT 6 = 1 |
| MOV PSW,A   | Move Accumulator Contents to PSW | 11010111 | The contents of the accumulator are moved into the program status word. All condition bits and the stack pointer are affected by this move. 
(PSW) ← (A) 
Example: Move up stack pointer by two memory locations, that is, increment the pointer by one. 
INCPTR: MOV A,PSW ; MOVE PSW CONTENTS TO ACC 
INC A ; INCREMENT ACC BY ONE 
MOV PSW,A ; MOVE ACC CONTENTS TO PSW |
## INSTRUCTION SET

### MOV Rr,A  Move Accumulator Contents to Register

**Opcode:**

```
  1 0 1 0 1 r2 r1 r0
```

The contents of the accumulator are moved to register 'r'.

(Rr) ← (A)  \(r=0-7\)

**Example:**

```
MRA MOV R0,A ;MOVE CONTENTS OF ACC TO REG 0
```

### MOV Rr,#data  Move Immediate Data to Register

**Opcode:**

```
  1 0 1 1 1 r2 r1 r0  d7 d6 d5 d4 d3 d2 d1 d0
```

This a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.

(Rr) ← data  \(r=0-7\)

**Example:**

```
MIR4: MOV R4,#HEXTEN ;THE VALUE OF THE SYMBOL 'HEXTEN' IS MOVED INTO REG 4
MIR5: MOV R5,#PI*(R*R) ;THE VALUE OF THE EXPRESSION 'PI*(R*R)' IS MOVED INTO REG 5
MIR6: MOV R6,#OADH ;'AD' HEX IS MOVED INTO REG 6
```

### MOV @Rr,A  Move Accumulator Contents to Data Memory

**Opcode:**

```
  1 0 1 0 0 0 0 r
```

The contents of the accumulator are moved to the data memory location whose address is specified by bits 0–5 of register 'r'. Register 'r' contents are unaffected.

((Rr)) ← (A)  \(r=0-1\)

**Example:**

```
Assume R0 contains 11000111.
MDMA: MOV @R,A ;MOVE CONTENTS OF ACC TO LOCATION 7 (REG)
```

### MOV @Rr,#data  Move Immediate Data to Data Memory

**Opcode:**

```
  1 0 1 1 0 0 0 r  d7 d6 d5 d4 d3 d2 d1 d0
```

This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the standard data memory location addressed by register 'r', bit 0–5.

((Rr)) ← data  \(r=0-1\)

**Example:**

```
Move the hexadecimal value AC3F to locations 62–63.
MIDM: MOV R0,#62 ;MOVE '62' DEC TO ADDR REG0
    MOV @RO,#OACH ;MOVE 'AC' HEX TO LOCATION 62
    INC R0 ;INCREMENT REG 0 TO '63'
    MOV @RO,#3FH ;MOVE '3F' HEX TO LOCATION 63
```
INSTRUCTION SET

MOV STS,A  Move Accumulator Contents to STS Register

Opcode: 1 0 0 1 0 0 0 0

The contents of the accumulator are moved into the status register. Only bits 4–7 are affected.

(STS4–7) ← (A4–7)

Example: Set ST4–ST7 to "1".
MSTS: MOV A,#0F0H ;SET ACC
      MOV STS,A ;MOVE TO STS

MOV T,A  Move Accumulator Contents to Timer/Counter

Opcode: 0 1 1 0 0 0 1 0

The contents of the accumulator are moved to the timer/event-counter register.

(T) ← (A)

Example: Initialize and start event counter.
INITEC: CLR A ;CLEAR ACC TO ZEROS
         MOV T,A ;MOVE ZEROS TO EVENT COUNTER
         STRT CNT ;START COUNTER

MOVD A,Pp  Move Port 4–7 Data to Accumulator

Opcode: 0 0 0 0 1 1 p1 p0

This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0–3. Accumulator bits 4–7 are zeroed.

(A0–3) ← Pp   p=4–7
(A4–7) ← 0

Note: Bits 0–1 of the opcode are used to represent PORTS 4–7. If you are coding in binary rather than assembly language, the mapping is as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>p1</th>
<th>p0</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

Example: INPPT5: MOVD A,P5 ;MOVE PORT 5 DATA TO ACC ;BITS 0–3, ZERO ACC BITS 4–7

MOVD Pp,A  Move Accumulator Data to Port 4, 5, 6 and 7

 Opcode: 0 0 1 1 1 1 p1 p0

This is a 2-cycle instruction. Data in accumulator bits 0–3 is moved (written) to 8243 port 'p'. Accumulator bits 4–7 are unaffected. (See NOTE above regarding port mapping.)

(Pp) ← (A0–3)   p=4–7

Example: Move data in accumulator to ports 4 and 5.
OUTP45: MOVD P4,A ;MOVE ACC BITS 0–3 TO PORT 4
        SWAP A ;EXCHANGE ACC BITS 0–3 AND 4–7
        MOVD P5,A ;MOVE ACC BITS 0–3 TO PORT 5
INSTRUCTION SET

MOVP A, @A  Move Current Page Data to Accumulator

Opcode: 1 0 1 0 0 0 1 1

This is a 2-cycle instruction. The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0–7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored following this operation.

(A) ← (A)

Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the following page.

Example:

MOV128: MOV A, #128
       MOV A, @A
       ;MOVE '128' DEC TO ACC
       ;CONTENTS OF 129TH LOCATION
       ;IN CURRENT PAGE ARE MOVED TO
       ;ACC

MOVP3 A, @A  Move Page 3 Data to Accumulator

Opcode: 1 1 1 0 0 0 1 1

This is a 2-cycle instruction. The contents of the program memory location within page 3, addressed by the accumulator, are moved to the accumulator. The program counter is restored following this operation.

(A) ← (A) within page 3

Example:

Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

TABSCH: MOV A, #OB8H
        ANL A, #7FH
        MOV P3, A, @A
        ;MOVE CONTENTS OF LOCATION
        ;'38' HEX IN PAGE 3 TO ACC
        ;(ASCII '8')

Access contents of location in page 3 labelled TAB1. Assume current program location is not in page 3.

TABSCH: MOV A, #TAB1
        ;ISOLATE BITS 0–7
        ;OF LABEL
        ;ADDRESS VALUE
        MOV P3, A, @A
        ;MOVE CONTENT OF PAGE 3
        ;LOCATION LABELED 'TAB1'
        ;TO ACC

NOP  The NOP Instruction

Opcode: 0 0 0 0 0 0 0 0

No operation is performed. Execution continues with the following instruction.

ORL A,Rr  Logical OR Accumulator With Register Mask

Opcode: 0 1 0 0 1 r2 r1 r0

Data in the accumulator is logically ORed with the mask contained in working register 'r'.

(A) ← (A) OR (Rr)  r = 0–7

Example:

ORREG: ORL A, R4  ;'OR' ACC CONTENTS WITH
        ;MASK IN REG 4

5-675
## INSTRUCTION SET

### ORL A, @Rr  Logical OR Accumulator With Memory Mask

**Opcode:**

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>r</th>
</tr>
</thead>
</table>

Data in the accumulator is logically ORed with the mask contained in the data memory location referenced by register 'r', bits 0-5.

\[(A) \leftarrow (A) \text{ OR } ((R_r))\]  \[r=0-1\]

**Example:**

ORLM: MOVE R0, #3FH  ; MOVE '3F' HEX TO REG 0

ORL A, @R0  ; 'OR' ACC CONTENTS WITH MASK

; IN LOCATION 63

### ORL A, #data  Logical OR Accumulator With Immediate Mask

**Opcode:**

| 1 | 0 | 0 | 0 | 1 | 1 | d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.

\[(A) \leftarrow (A) \text{ OR } \text{data}\]

**Example:**

ORID: ORL A, #’X’  ; 'OR' ACC CONTENTS WITH MASK

; 01011000 (ASCII VALUE OF 'X')

### ORL Pp, #data  Logical OR Port 1-2 With Immediate Mask

**Opcode:**

| 1 | 0 | 0 | 0 | 1 | 0 | 1 | p1 | p0 | d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

This is a 2-cycle instruction. Data on port 'p' is logically ORed with an immediately-specified mask.

\[(P_p) \leftarrow (P_p) \text{ OR } \text{data}\]  \[p=1-2 \text{ (see OUTL instruction)}\]

**Example:**

ORP1: ORL P1, #OFFH  ; 'OR' PORT 1 CONTENTS WITH

; MASK 'FF' HEX (SET PORT 1

; TO ALL ONES)

### ORLD Pp, A  Logical OR Port 4-7 With Accumulator Mask

**Opcode:**

| 1 | 0 | 0 | 0 | 1 | 1 | p1 | p0 |

This is a 2-cycle instruction. Data on 8243 port 'p' is logically ORed with the digit mask contained in accumulator bits 0-3.

\[(P_p) \leftarrow (P_p) \text{ OR } (A_{0-3})\]  \[p=4-7 \text{ (See MOVD instruction)}\]

**Example:**

ORP7: ORLD P7, A  ; 'OR' PORT 7 CONTENTS

; WITH ACC BITS 0-3

### OUT DBB, A  Output Accumulator Contents to Data Bus Buffer

**Opcode:**

| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Contents of the accumulator are transferred to the Data Bus Buffer Output register and the Output Buffer Full (OBF) flag is set to one.

\[(DBB) \leftarrow (A)\]  \[OBF \leftarrow 1\]

**Example:**

OUTDBB: OUT DBB, A  ; OUTPUT THE CONTENTS OF

; THE ACC TO DBBOUT
OUTL Pp,A  Output Accumulator Data to Port 1 and 2

Opcode: [0 0 1 1 1 0 p1 p0]

This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to port ‘p’ and latched.

(Pp) ← (A)  P=1-2

Note: Bits 0–1 of the opcode are used to represent PORT 1 and PORT 2. If you are coding in binary rather than assembly language, the mapping is as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>p1</th>
<th>p0</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Example: OUTLP: MOV A,R7 ;MOVE REG 7 CONTENTS TO ACC
          OUTL P2,A ;OUTPUT ACC CONTENTS TO PORT2
          MOV A,R6 ;MOVE REG 6 CONTENTS TO ACC
          OUTL P1,A ;OUTPUT ACC CONTENTS TO PORT 1

RET  Return Without PSW Restore

Opcode: [1 0 0 0 0 0 0 1 1]

This is a 2-cycle instruction. The stack pointer (PSW bits 0–2) is decremented. The program counter is then restored from the stack. PSW bits 4–7 are not restored.

(SP) ← (SP) − 1
(PC) ← ((SP))

RETR  Return With PSW Restore

Opcode: [1 0 0 1 0 0 1 1]

This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4–7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine.

(SP) ← (SP) − 1
(PC) ← ((SP))
(PSW4–7) ← ((SP))

RL A  Rotate Left Without Carry

Opcode: [1 1 1 1 0 0 1 1 1]

The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.

(An+1) ← (An)  n=0–6
(An0) ← (A7)

Example: Assume accumulator contains 10110001.
RLNC: RL A ;NEW ACC CONTENTS ARE 01100011
RLC A  Rotate Left Through Carry

Opcode: \[1 1 1 1 0 1 1 1\]

The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotated into the bit 0 position.

\[(A_{n+1}) \leftarrow (A_n)\]  
\[(A_0) \leftarrow (C)\]  
\[(C) \leftarrow (A_7)\]

Example: Assume accumulator contains a 'signed' number; isolate sign without changing value.

RLTC: CLR C  ;CLEAR CARRY TO ZERO
RLC A  ;ROTATE ACC LEFT, SIGN
RR A  ;ROTATE ACC RIGHT — VALUE
\( (A_n) \leftarrow (A_{n+1}) \)  
\( (A_7) \leftarrow (A_0) \)

RR A  Rotate Right Without Carry

Opcode: \[0 1 1 1 0 1 1 1\]

The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position.

\[(A_n) \leftarrow (A_{n+1})\]  
\[(A_7) \leftarrow (A_0)\]

Example: Assume accumulator contains 10110001.

RRNC: RRA  ;NEW ACC CONTENTS ARE 11011000

RRC A  Rotate Right Through Carry

Opcode: \[0 1 1 0 0 1 1 1\]

The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.

\[(A_n) \leftarrow (A_{n+1})\]  
\[(A_7) \leftarrow (C)\]  
\[(C) \leftarrow (A_0)\]

Example: Assume carry is not set and accumulator contains 10110001.

RRTC: RRCA  ;CARRY IS SET AND ACC CONTAINS 01011000

SEL RB0  Select Register Bank 0

Opcode: \[1 1 0 0 0 1 0 1\]

PSW BIT 4 is set to zero. References to working registers 0–7 address data memory locations 0–7. This is the recommended setting for normal program execution.

\( (BS) \leftarrow 0 \)
INSTRUCTION SET

SEL RB1  Select Register Bank 1

Opcode:  \[1 1 0 1 | 0 1 0 1\]

PSW bit 4 is set to one. References to working registers 0–7 address data memory locations 24–31. This is the recommended setting for interrupt service routines, since locations 0–7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

Example: Assume an IBF interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

LOC3: JMP INIT ;JUMP TO ROUTINE 'INIT'

INIT: MOV R7,A ;MOV ACC CONTENTS TO
SEL RB1 ;SELECT REG BANK 1
MOV R7,#OFAH ;MOVE 'FA' HEX TO LOCATION 31

STOP TCNT  Stop Timer/Event Counter

Opcode:  \[0 1 1 0 | 0 1 0 1\]

This instruction is used to stop both time accumulation and event counting.

Example: Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

START: DIS TCNTI ;DISABLE TIMER INTERRUPT
CL R, A ;CLEAR ACC TO ZERO
MOV A,T ;MOV ZERO TO TIMER
MOV R7,A ;MOVE ZERO TO REG 7
STRT T ;START TIMER
MAIN: JTF COUNT ;JUMP TO ROUTINE 'COUNT'
JMP MAIN ;IF TF=1 AND CLEAR TIMER FLAG

COUNT: INC R7 ;INCREMENT REG 7
MOV A,R7 ;MOVE REG 7 CONTENTS TO ACC
JB3 INT ;JUMP TO ROUTINE 'INT' IF ACC
BIT 3 IS SET (REG 7 = 8)
JMP MAIN ;OTHERWISE RETURN TO ROUTINE

INT: STOP TCNT ;STOP TIMER
JMP 7H ;JUMP TO LOCATION 7 (TIMER INTERRUPT ROUTINE)
## INSTRUCTION SET

### STRT CNT  Start Event Counter

<table>
<thead>
<tr>
<th>Opcode:</th>
<th><code>01000101</code></th>
</tr>
</thead>
</table>

The TEST 1 (T1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high to low transition on the T1 pin.

**Example:** Initialize and start event counter. Assume overflow is desired with first T1 input.

```
STARTC: EN TCNTI ; ENABLE COUNTER INTERRUPT
MOV A,#OFFH ; MOVE 'FF' HEX (ONES) TO ACC
MOV T,A ; MOVE ONES TO COUNTER
STRT CNT ; INPUT AND START
```

### STRT T  Start Timer

<table>
<thead>
<tr>
<th>Opcode:</th>
<th><code>01010101</code></th>
</tr>
</thead>
</table>

Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

**Example:** Initialize and start timer.

```
STARTT: EN TCNTI ; ENABLE TIMER INTERRUPT
CLR A : CLEAR ACC TO ZEROS
MOV T,A ; MOVE ZEROS TO TIMER
STRT T ; START TIMER
```

### SWAP A  Swap Nibbles Within Accumulator

<table>
<thead>
<tr>
<th>Opcode:</th>
<th><code>01000111</code></th>
</tr>
</thead>
</table>

Bits 0–3 of the accumulator are swapped with bits 4–7 of the accumulator. 

\[(A_{4-7}) \leftrightarrow (A_{0-3})\]

**Example:** Pack bits 0–3 of locations 50–51 into location 50.

```
PCKDIG: MOV R0,#50 ; MOVE '50' DEC TO REG 0
MOV R1,#51 ; MOVE '51' DEC TO REG 1
XCHD A,@R0 ; EXCHANGE BIT 0–3 OF ACC AND LOCATION 50
SWAP A ; SWAP BITS 0–3 AND 4–7 OF ACC
XCHD A,@R1 ; EXCHANGE BITS 0–3 OF ACC AND LOCATION 51
MOV @R0,A ; MOVE CONTENTS OF ACC TO LOCATION 51
```

### XCH A,Rr  Exchange Accumulator-Register Contents

<table>
<thead>
<tr>
<th>Opcode:</th>
<th><code>0010111</code></th>
</tr>
</thead>
</table>

The contents of the accumulator and the contents of working register 'r' are exchanged.

\[(A) \leftrightarrow (R_r)\]

**Example:** Move PSW contents to Reg 7 without losing accumulator contents.

```
XCHAR7: XCH A,R7 ; EXCHANGE CONTENTS OF REG 7 AND ACC
MOV A,PSW ; MOVE PSW CONTENTS TO ACC
XCH A,R7 ; EXCHANGE CONTENTS OF REG 7 AND ACC AGAIN
```

5-680
INSTRUCTION SET

XCH A,@Rr Exchange Accumulator and Data Memory Contents

<table>
<thead>
<tr>
<th>Opcode:</th>
<th>0 0 1 0 0 0 0 r</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>The contents of the accumulator and the contents of the data memory location addressed by bits 0–5 of register 'r' are exchanged. Register 'r' contents are unaffected.</td>
<td></td>
</tr>
<tr>
<td>(A) ←→ (Rr)</td>
<td>r=0–1</td>
</tr>
<tr>
<td>Example:</td>
<td>Decrement contents of location 52.</td>
</tr>
<tr>
<td>DEC52: MOV R0,#52; MOVE '52' DEC TO ADDRESS</td>
<td></td>
</tr>
<tr>
<td>XCH A,@R0; EXCHANGE CONTENTS OF ACC</td>
<td></td>
</tr>
<tr>
<td>DEC A; DECREMENT ACC CONTENTS</td>
<td></td>
</tr>
<tr>
<td>XCH A,@R0; EXCHANGE CONTENTS OF ACC</td>
<td></td>
</tr>
<tr>
<td>AND LOCATION 52 AGAIN</td>
<td></td>
</tr>
</tbody>
</table>

XCHD A,@Rr Exchange Accumulator and Data Memory 4-bit Data

<table>
<thead>
<tr>
<th>Opcode:</th>
<th>0 0 1 1 0 0 0 r</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>This instruction exchanges bits 0–3 of the accumulator with bits 0–3 of the data memory location addressed by bits 0–5 of register 'r'. Bits 4–7 of the accumulator, bits 4–7 of the data memory location, and the contents of register 'r' are unaffected.</td>
<td></td>
</tr>
<tr>
<td>(A0–3) ←→ ((Rr0–3))</td>
<td>r=0–1</td>
</tr>
<tr>
<td>Example:</td>
<td>Assume program counter contents have been stacked in locations 22–23.</td>
</tr>
<tr>
<td>XCHNIB: MOV R0,#23; MOVE '23' DEC TO REG 0</td>
<td></td>
</tr>
<tr>
<td>CLR A; CLEAR ACC TO ZEROS</td>
<td></td>
</tr>
<tr>
<td>XCHD A,@R0; EXCHANGE BITS 0–3 OF ACC</td>
<td></td>
</tr>
<tr>
<td>AND LOCATION 23 (BITS 8–11; OF PC ARE ZEROED, ADDRESS; REFERS TO PAGE 0)</td>
<td></td>
</tr>
</tbody>
</table>

XRL A,Rr Logical XOR Accumulator With Register Mask

<table>
<thead>
<tr>
<th>Opcode:</th>
<th>1 1 0 1 1 0 1 r</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register 'r'.</td>
<td></td>
</tr>
<tr>
<td>(A) ← (A) XOR (Rr)</td>
<td>r=0–7</td>
</tr>
<tr>
<td>Example:</td>
<td>XORREG: XRL A,R5; 'XOR' ACC CONTENTS WITH</td>
</tr>
<tr>
<td>;MASK IN REG 5</td>
<td></td>
</tr>
</tbody>
</table>

XRL A,@Rr Logical XOR Accumulator With Memory Mask

<table>
<thead>
<tr>
<th>Opcode:</th>
<th>1 1 0 1 0 0 0 r</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register 'r', bits 0–5.</td>
<td></td>
</tr>
<tr>
<td>(A) ← (A) XOR ((Rr))</td>
<td>r=0–1</td>
</tr>
<tr>
<td>Example:</td>
<td>XORDM: MOV R1,#20H; MOVE '20' HEX TO REG 1</td>
</tr>
<tr>
<td>XRL A,@R1; 'XOR' ACC CONTENTS WITH MASK</td>
<td></td>
</tr>
<tr>
<td>;IN LOCATION 32</td>
<td></td>
</tr>
</tbody>
</table>
**INSTRUCTION SET**

<table>
<thead>
<tr>
<th>XRL A,#data</th>
<th>Logical XOR Accumulator With Immediate Mask</th>
</tr>
</thead>
</table>

**Opcode:**

```
1 1 0 1 0 0 1 1 • d7 d6 d5 d4 d3 d2 d1 d0
```

This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

\[(A) ← (A) \text{ XOR data}\]

**Example:**

```
XORID: XOR A, #HEXTEN ; XOR CONTENTS OF ACC WITH
        ; MASK EQUAL VALUE OF SYMBOL
        ; 'HEXTEN'
```
SINGLE-STEP
The UPI family has a single-step mode which allows the user to manually step through his program one instruction at a time. While stopped, the address of the next instruction to be fetched is available on PORT 1 and the lower 2 bits of PORT 2. The single-step feature simplifies program debugging by allowing the user to easily follow program execution.

Figure 4-1 illustrates a recommended circuit for single-step operation, while Figure 4-2 shows the timing relationship between the SYNC output and the SS input. During single-step operation, PORT 1 and part of PORT 2 are used to output address information. In order to retain the normal I/O functions of PORTS 1 and 2, a separate latch can be used as shown in Figure 4-3.

Figure 4-1. Single-Step Circuit

Figure 4-2. Single-Step Timing
Timing

The sequence of single-step operation is as follows:

1) The processor is requested to stop by applying a low level on SS. The SS input should not be brought low while SYNC is high. (The UPI samples the SS pin in the middle of the SYNC pulse).

2) The processor responds to the request by stopping during the instruction fetch portion of the next instruction. If a double cycle instruction is in progress when the single-step command is received, both cycles will be completed before stopping.

3) The processor acknowledges it has entered the stopped state by raising SYNC high. In this state, which can be maintained indefinitely, the 10-bit address of the next instruction to be fetched is present on PORT 1 and the lower 2 bits of PORT 2.

4) SS is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing SYNC low.

5) To stop the processor at the next instruction SS must be brought low again before the next SYNC pulse—the circuit in Figure 4-1 uses the trailing edge of the previous pulse. If SS is left high, the processor remains in the “RUN” mode.

Figure 4-1 shows a schematic for implementing single-step. A single D-type flip-flop with preset and clear is used to generate SS. In the RUN mode SS is held high by keeping the flip-flop preset (preset has precedence over the clear input). To enter single-step, preset is removed allowing SYNC to bring SS low via the clear input. Note that SYNC must be buffered since the SN7474 is equivalent to 3 TTL loads.

The processor is now in the stopped state. The next instruction is initiated by clocking “1” into the flip-flop. This “1” will not appear on SS unless SYNC is high (i.e., clear must be removed from the flip-flop). In response to SS going high, the processor begins an instruction fetch which brings SYNC low. SS is then reset through the clear input and the processor again enters the stopped state.
SINGLE-STEP, PROGRAMMING, & POWER-DOWN MODES

PROGRAMMING, VERIFYING AND ERASING EPROM (8741A, 8742 EPROM ONLY)
The internal Program Memory of the 8741A and 8742 may be erased and reprogrammed by the user as explained in the following sections. See the data sheet for more detail.

Programming
The programming procedure consists of the following: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. Figure 4-4 illustrates the programming and verifying sequence. The following is a list of the pins used for programming and a description of their functions:

- XTAL 1, Clock Input
- XTAL 2
- RESET Initialization and Address Latching
- TEST 0 Selection of Program or Verify Mode
- EA Activation of Program/Verify Modes
- D0-D7 Address and Data Input
- DATA OUT Address 0-7
- DATA OUT Data Output During Verify

- P20, P21 Address Input
- VDD Programming Power Supply
- PROG Program Pulse Input

NOTE: All set-up and hold times are 4 cycles.

The detailed Programming sequence (for one byte) is as follows:

1) Initial Conditions: $V_{CC} = VDD = 5V$; Clock Running; $A_0 = 0V$, $CS = 5V$; $EA = 5V$; $D_0-D_7$ and PROG Floating.

2) $\overline{RESET} = 0V$, TEST 0 = 0V (Select Programming Mode).

3) $EA = 23V$ for 8741A
   $EA = 18V$ for 8742

4) Address applied to $D_0-D_7$ and PORTS 20-22.

5) $\overline{RESET} = 5V$ (Latch Address).

6) Data applied to $D_0-D_7$.

7) $VDD = 25V$ for 8741A
   $VDD = 21V$ for 8742 (Programming Power).

Figure 4-4. Programming Sequence
SINGLE-STEP, PROGRAMMING, & POWER-DOWN MODES

8) \( PROG = 0V \) followed by one 50 msec pulse of 23V for 8741A
   \( PROG = 0V \) followed by one 50 msec pulse of 18V for 8742.

9) \( VDD = 5V \).

10) \( TEST \; 0 = 5V \) (Select Verify Mode).

11) Read data on \( D0-D7 \) and verify EPROM cell contents.

WARNING
An attempt to program a mis-socketed 8741A or 8742 will result in severe damage to the part.
An indication of a properly socketed part is the appearance of the SYNC clock output. The lack of this clock may be used to disable the programmer.

Verification
Verification is accomplished by latching in an address as in the Programming Mode and then applying “1” to the TEST 0 input. The word stored at the selected address then appears on the \( D0-D7 \) lines. Note that verification can be applied to both ROM’s and EPROM’s independently of the programming procedure. See the data sheet.

The detailed Verifying sequence (for one byte) is as follows:

1) Initial Conditions: \( VCC = VDD = 5V \); Clock Running; \( A0 = 0V \), \( CS = 5V \); \( EA = 5V \); \( D0-D7 \) and \( PROG \) Floating.

2) \( \text{RESET} = 0V \), \( \text{TEST} \; 0 = 5V \) (Verify Mode).

3) \( EA = 23V \) for 8741A
   \( EA = 18V \) for 8742

4) Address applied to \( D0-D7 \) and \( PORTS \) 20–22.

5) \( \text{RESET} = 5V \) (Latch Address)

6) Read data on \( D0-D7 \) and verify EPROM cell contents.

Erasing
The program memory of the 8741A or 8742 may be erased to zeros by exposing its translucent lid to shortwave ultraviolet light.

EPROM Light Sensitivity
The erasure characteristics of the 8741A or 8742 EPROM are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms. It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Angstrom range. Data shows that constant exposure to room level fluorescent lighting could erase the typical 8741A in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8741A or 8742 is to be exposed to these types of lighting conditions for extended periods of time, opaque labels (available from Intel) should be placed over the 8741A or 8742 window to prevent unintentional erasure.

The recommended erasure procedure for the 8741A or 8742 is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms. The integrated dose (i.e., UV intensity \( \times \) exposure time) for erasure should be a minimum of 15W-sec/cm\(^2\) power rating. The erasure time with this dosage is approximately 15 minutes using an ultraviolet lamp with a 12,000 \( \mu \)W/cm\(^2\) power rating. The 8741A or 8742 should be placed within 1 inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

EXTERNAL ACCESS
The UPI family has an External Access mode (EA) which puts the processor into a test mode. This mode allows the user to disable the internal program memory and execute from external memory. External Access mode is useful in testing because it allows the user to test the processor’s functions directly. It is only useful for testing since this mode uses \( D0-D7 \), \( PORTS \) 10–17 and \( PORTS \) 20–22.

This mode is invoked by connecting the EA pin to 5V. The 11-bit current program counter contents then come out on \( PORTS \) 10–17 and \( PORTS \) 20–22 after the SYNC output goes high. (PORT 10 is the least significant bit.) The desired instruction opcode is placed on \( D0-D7 \) before the start of state S1. During state S1, the opcode is sampled from \( D0-D7 \) and subsequently executed in place of the internal program memory contents.

The program counter contents are multiplexed with the I/O port data on \( PORTS \) 10–17 and \( PORTS \) 20–22. The I/O port data may be demultiplexed using an external latch on the rising edge of SYNC. The program counter contents may be demultiplexed similarly using the trailing edge of SYNC.

Reading and/or writing the Data Bus Buffer registers is still allowed although only when \( D0-D7 \) are not being sampled for opcode data. In practice, since this sampling time is not known externally, reads or
writes on the system bus are done during SYNC high time. Approximately 600ns are available for each read or write cycle.

**POWER DOWN MODE**

*(8041AH/8042 ROM ONLY)*

Extra circuitry is included in the ROM version to allow low-power, standby operation. Power is removed from all system elements except the internal data RAM in the low-power mode. Thus the contents of RAM can be maintained and the device draws only 10 to 15% of its normal power.

The VCC pin serves as the 5V power supply pin for all of the ROM version's circuitry except the data RAM array. The VDD pin supplies only the RAM array. In normal operation, both VCC and VDD are connected to the same 5V power supply.

To enter the Power-Down mode, the RESET signal to the UPI is asserted. This ensures the memory will not be inadvertently altered by the UPI during power-down. The VCC pin is then grounded while VDD is maintained at 5V. Figure 4-5 illustrates a recommended Power-Down sequence. The sequence typically occurs as follows:

1) Imminent power supply failure is detected by user defined circuitry. The signal must occur early enough to guarantee the 8041AH or 8042 can save all necessary data before VCC falls outside normal operating tolerance.

2) A “Power Failure” signal is used to interrupt the processor (via a timer overflow interrupt, for instance) and call a Power Failure service routine.

3) The Power Failure routine saves all important data and machine status in the RAM array. The routine may also initiate transfer of a backup supply to the VDD pin and indicate to external circuitry that the Power Failure routine is complete.

4) A RESET signal is applied by external hardware to guarantee data will not be altered as the power supply falls out of limits. RESET must be low until VCC reaches ground potential.

Recovery from the Power-Down mode can occur as any other power-on sequence. An external 1 μfd capacitor on the RESET input will provide the necessary initialization pulse.

![Figure 4-5. Power-Down Sequence](image-url)
CHAPTER 5
SYSTEM OPERATION

BUS INTERFACE
The UPI-41AH, 42 Microcomputer functions as a peripheral to a master processor by using the data bus buffer registers to handle data transfers. The DBB configuration is illustrated in Figure 5-1. The UPI-41AH, 42 Microcomputer's 8 three-state data lines (D7–D0) connect directly to the master processor's data bus. Data transfer to the master is controlled by 4 external inputs to the UPI:
- **A0** Address Input signifying command or data
- **CS** Chip Select
- **RD** Read strobe
- **WR** Write strobe

![Figure 5-1. Data Bus Register Configuration](image)

The master processor addresses the UPI-41AH, 42 Microcomputer as a standard peripheral device. Table 5-1 shows the conditions for data transfer:

<table>
<thead>
<tr>
<th>CS</th>
<th>A0</th>
<th>RD</th>
<th>WR</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Read DBBOUT</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Read STATUS</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Write DBBIN data, set F1 = 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Write DBBIN command set F1 = 1</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Disable DBB</td>
</tr>
</tbody>
</table>

Reading the DBBOUT Register
The sequence for reading the DBBOUT register is shown in Figure 5-2. This operation causes the 8-bit contents of the DBBOUT register to be placed on the system Data Bus. The OBF flag is cleared automatically.

**Reading STATUS**
The sequence for reading the UPI-41AH, 42 Microcomputer's 8 STATUS bits is shown in Figure 5-3. This operation causes the 8-bit STATUS register contents to be placed on the system Data Bus as shown.

Write Data to DBBIN
The sequence for writing data to the DBBIN register is shown in Figure 5-4. This operation causes the system Data Bus contents to be transferred to the DBBIN register and the IBF flag is set. Also, the F1 flag is cleared (F1 = 0) and an interrupt request is generated. When the IBF interrupt is enabled, a jump to location 3 will occur. The interrupt request is cleared upon entering the IBF service routine or by a system RESET input.

![Figure 5-2. DBBOUT Read](image)

![Figure 5-3. Status Read](image)
SYSTEM OPERATION

Writing Commands to DBBIN

The sequence for writing commands to the DBBIN register is shown in Figure 5-5. This sequence is identical to a data write except that the A0 input is latched in the F1 flag (F1 = 1). The IBF flag is set and an interrupt request is generated when the master writes a command to DDB.

Operations of Data Bus Registers

The UPI-41AH, 42 Microcomputer controls the transfer of DDB data to its accumulator by executing INPut and OUTput instructions. An IN A, DBB instruction causes the contents to be transferred to the UPI accumulator and the IBF flag is cleared.

The OUT DBB, A instruction causes the contents of the accumulator to be transferred to the DDBOUT register. The OBF flag is set.

The UPI’s data bus buffer interface is applicable to a variety of microprocessors including the 8086, 8088, 8085AH, 8080, and 8048.

A description of the interface to each of these processors follows.

DESIGN EXAMPLES

8085AH Interface

Figure 5-6 illustrates an 8085AH system using a UPI-41AH, 42. The 8085AH system uses a multiplexed address and data bus. During I/O the 8 upper address lines (A9–A15) contain the same I/O address as the lower 8 address/data lines (A0–A7); therefore I/O address decoding is done using only the upper 8 lines to eliminate latching of the address. An 8205 decoder provides address decoding for both the UPI-41AH, 42 and the 8237. Data is transferred using the two DMA handshaking lines of PORT 2. The 8237 performs the actual bus transfer operation. Using the UPI-41AH, 42’s OBF master interrupt, the UPI-41A notifies the 8085AH upon transfer completion using the RST 5.5 interrupt input. The IBF master interrupt is not used in this example.

8088 Interface

Figure 5-7 illustrates a UPI-41AH, 42 interface to an 8088 minimum mode system. Two 8-bit latches are used to demultiplex the address and data bus. The address bus is 20-lines wide. For I/O only, the lower 16 address lines are used, providing an addressing range of 64K. UPI address selection is accomplished using an 8205 decoder. The A0 address line of the bus is connected to the corresponding UPI input for register selection. Since the UPI-41A is polled by the 8088, neither DMA nor master interrupt capabilities of the UPI-41AH, 42 are used in the figure.

8086 Interface

The UPI-41AH, 42 can be used on an 8086 maximum mode system as shown in figure 5-8. The address and data bus is demultiplexed using three 8282 latches providing separate address and data buses. The address bus is 20-lines wide and the data bus is 16-lines wide. Multiplexed control lines are decoded by the 8288. The UPI’s CS input is provided by linear selection. Note that the UPI-41AH, 42 is both I/O mapped and memory mapped as a result of the linear addressing technique. An address decoder may be used to limit the UPI-41AH, 42 to a specific I/O mapped address. Address line A1 is connected to the UPI’s Aq input. This insures that the registers of the UPI will have even I/O addresses. Data will be transferred on D0–D7 lines only. This allows the I/O registers to be accessed using byte manipulation instructions.
SYSTEM OPERATION

Figure 5-6. 8085AH-UPI System

Figure 5-7. 8088-UPI Minimum Mode System
8080 Interface
Figure 5-9 illustrates the interface to an 8080A system. In this example, a crystal and capacitor are used for UPI-41AH, 42 timing reference and power-on RESET. If the 2-MHz 8080A 2-phase clock were used instead of the crystal, the UPI-41AH, UPI-42 would run at only 16% full speed.

The A0 and CS inputs are direct connections to the 8080 address bus. In larger systems, however, either of these inputs may be decoded from the 16 address lines.

The RD and WR inputs to the UPI can be either the IOR and IOW or the MEMR and MEMW signals depending on the I/O mapping technique to be used.

The UPI can be addressed as an I/O device using INPUT and OUTput instructions in 8080 software.

8048 Interface
Figure 5-10 shows the UPI interface to an 8048 master processor.

The 8048 RD and WR outputs are directly compatible with the UPI. Figure 5-11 shows a distributed processing system with up to seven UPI's connected to a single 8048 master processor.

In this configuration the 8048 uses PORT 0 as a data bus. I/O PORT 2 is used to select one of the seven UPI's when data transfer occurs. The UPI's are programmed to handle isolated tasks and, since they operate in parallel, system throughput is increased.

GENERAL HANDSHAKING PROTOCOL
1) Master reads STATUS register (RD, CS, A0 = (0, 0, 1)) in polling or in response to either an IBF or an OBF interrupt.

2) If the UPI DDBBIN register is empty (IBF flag = 0), Master writes a word to the DDBIN register (WR, CS, A0 = (0, 0, 1) or (0, 0, 0)). If A0 = 1, write command word, set F1. If A0 = 0, write data word, F1 = 0.
3) If the UPI DBBOUT register is full (OBF flag = 1), Master reads a word from the DBBOUT register (RD, CS, A0 = (0, 0, 0)).

4) UPI recognizes IBF (via IBF interrupt or JNIBF). Input data or command word is processed, depending on F1; IBF is reset. Repeat step 1 above.

5) UPI-41AH, 42 recognizes OBF flag = 0 (via JOBF). Next word is output to DBBOUT register, OBF is set. Repeat step 1 above.
Figure 5-11. Distributed Processor System
Chapter 6
APPLICATIONS

ABSTRACTS
The UPI-41A is designed to fill a wide variety of low to medium speed peripheral interface applications where flexibility and easy implementation are important considerations. The following examples illustrate some typical applications.

Keyboard Encoder
Figure 6-1 illustrates a keyboard encoder configuration using the UPI and the 8243 I/O expander to scan a 128-key matrix. The encoder has switch matrix scanning logic, N-key rollover logic, ROM look-up table, FIFO character buffer, and additional outputs for display functions, control keys or other special functions.

PORT 1 and PORTs 4–7 provide the interface to the keyboard. PORT 1 lines are set one at a time to select the various key matrix rows.

When a row is energized, all 16 columns (i.e., PORTs 4–7 inputs) are sampled to determine if any switch in the row is closed. The scanning software is code efficient because the UPI instruction set includes individual bit set/clear operations and expander PORTs 4–7 can be directly addressed with single, 2-byte instructions. Also, accumulator bits can be tested in a single operation. Scan time for 128 keys is about 10 ms. Each matrix point has a unique binary code which is used to address ROM when a key closure is detected. Page 3 of ROM contains a look-up table with useable codes (i.e., ASCII, EBCDIC, etc.) which correspond to each key. When a valid key closure is detected the ROM code corresponding to that key is stored in a FIFO buffer in data memory for transfer to the master processor. To avoid stray noise and switch bounce, a key closure must be detected on two consecutive scans before it is considered valid and loaded into the FIFO buffer. The FIFO buffer allows multiple keys to be processed as they are depressed without regard to when they are released, a condition known as N-key rollover.

The basic features of this encoder are fairly standard and require only about 500 bytes of memory. Since the UPI is programmable and has additional memory capacity it can handle a number of other functions. For example, special keys can be programmed to give an entry on closing as well as opening. Also, I/O lines are available to control a 16-digit, 7-segment display. The UPI can also be programmed to recognize special combinations of characters such as commands, then transfer only the decoded information to the master processor.

A complete keyboard application has been developed for the UPI-41A. A description is included in this section. The code for the application is available in the Intel Insite Library (program AB 147).

Figure 6-1. Keyboard Encoder Configuration
Matrix Printer Interface

The matrix printer interface illustrated in Figure 6-2 is a typical application for the UPI-41A. The actual printer mechanism could be any of the numerous dot-matrix types and similar configurations can be shown for drum, spherical head, daisy wheel or chain type printers.

The bus structure shown represents a generalized, 8-bit system bus configuration. The UPI's three-state interface port and asynchronous data buffer registers allow it to connect directly to this type of system for efficient, two-way data transfer.

The UPI's two on-board I/O ports provide up to 16 input and output signals to control the printer mechanism. The timer/event counter is used for generating a timing sequence to control print head position, line feed, carriage return, and other sequences. The on-board program memory provides character generation for 5 x 7, 7 x 9, or other dot matrix formats. As an added feature a portion of the 64 x 8-bit data memory can be used as a FIFO buffer so that the master processor can send a block of data at a high rate. The UPI can then output characters from the buffer at a rate the printer can accept while the master processor returns to other tasks.

The 8295 Printer Controller is an example of an 8041A preprogrammed as a dot matrix printer interface.

Tape Cassette Controller

Figure 6-3 illustrates a digital cassette interface which can be implemented with the UPI-41A. Two sections of the tape transport are controlled by the UPI: digital data/command logic, and motor servo control.

The motor servo requires a speed reference in the form of a monostable pulse whose width is proportional to the desired speed. The UPI monitors a prerecorded clock from the tape and uses its onboard interval timer to generate the required speed reference pulses at each clock transition.

Recorded data from the tape is supplied serially by the data/command logic and is converted to 8-bit words by the UPI, then transferred to the master processor. At 10 ips tape speed the UPI can easily handle the 8000 bps data rate. To record data, the UPI uses the two input lines to the data/command logic which control the flux direction in the recording head. The UPI also monitors 4 status lines from the tape transport including: end of tape, cassette

![Figure 6-2. Matrix Printer Controller](image-url)
inserted, busy, and write permit. All control signals can be handled by the UPI's two I/O ports.

**Universal I/O Interface**

Figure 6-4 shows an I/O interface design based on the UPI. This configuration includes 12 parallel I/O lines and a serial (RS232C) interface for full duplex data transfer up to 1200 baud. This type of design can be used to interface a master processor to a broad spectrum of peripheral devices as well as to a serial communication channel.

PORT 1 is used strictly for I/O in this example while PORT 2 lines provide five functions:

- P23-P20 I/O lines (bidirectional)
- P24 Request to send (RTS)
- P25 Clear to Send (CTS)
- P26 Interrupt to master
- P27 Serial data out

The parallel I/O lines make use of the bidirectional port structure of the UPI. Any line can function as an input or output. All port lines are automatically initialized to 1 by a system RESET pulse and remain
latched. An external TTL signal connected to a port line will override the UPI's 50K-ohm internal pull-up so that an INPUT instruction will correctly sample the TTL signal.

Four PORT 2 lines function as general I/O similar to PORT 1. Also, the RTS signal is generated on PORT 2 under software control when the UPI has serial data to send. The CTS signal is monitored via PORT 2 as an enable to the UPI to send serial data. A PORT 2 line is also used as a software generated interrupt to the master processor. The interrupt functions as a service request when the UPI has a byte of data to transfer or when it is ready to receive. Alternatively, the EN FLAGS instruction could be used to create the OBF and IBF interrupts on P24 and P25.

The RS232C interface is implemented using the TEST 0 pin as a receive input and a PORT 2 pin as a transmit output. External packages (A0, A1) are used to provide RS232C drive requirements. The serial receive software is interrupt driven and uses the on-chip timer to perform time critical serial control. After a start bit is detected the interval timer can be preset to generate an interrupt at the proper time for sampling the serial bit stream. This eliminates the need for software timing loops and allows the processor to proceed to other tasks (i.e., parallel I/O operations) between serial bit samples. Software flags are used so the main program can determine when the interrupt driven receive program has a character assembled for it.

This type of configuration allows system designers flexibility in designing custom I/O interfaces for specific serial and parallel I/O applications. For instance, a second or third serial channel could be substituted in place of the parallel I/O if required. The UPI's data memory can buffer data and commands for up to 4 low-speed channels (110 baud typewriter, etc.)

Application Notes

The following application notes illustrate the various applications of the UPI family. Other related publications including the 8048 Family Application Handbook are available through the Intel Literature Department.
INTRODUCTION TO THE UPI-41A™

Introduction

Since the introduction in 1974 of the second generation of microprocessors, such as the 8080, a wide range of peripheral interface devices have appeared. At first, these devices solved application problems of a general nature; i.e., parallel interface (8255), serial interface (8251), timing (8253), interrupt control (8259). However, as the speed and density of LSI technology increased, more and more intelligence was incorporated into the peripheral devices. This allowed more specific application problems to be solved, such as floppy disk control (8271), CRT control (8275), and data link control (8273). The advantage to the system designer of this increased peripheral device intelligence is that many of the peripheral control tasks are now handled externally to the main processor in the peripheral hardware rather than internally in the main processor software. This reduced main processor overhead results in increased system throughput and reduced software complexity.

In spite of the number of peripheral devices available, the pervasiveness of the microprocessor has been such that there is still a large number of peripheral control applications not yet satisfied by dedicated LSI. Complicating this problem is the fact that new applications are emerging faster than the manufacturers can react in developing new, dedicated peripheral controllers. To address this problem, a new microcomputer-based Universal Peripheral Interface (UPI-41A) device was developed.

In essence, the UPI-41A acts as a slave processor to the main system CPU. The UPI contains its own processor, memory, and I/O, and is completely user programmable; that is, the entire peripheral control algorithm can be programmed locally in the UPI, instead of taxing the master processor's main memory. This distributed processing concept allows the UPI to handle the real-time tasks such as encoding keyboards, controlling printers, or multiplexing displays, while the main processor is handling non-real-time dependent tasks such as buffer management or arithmetic. The UPI relies on the master only for initialization, elementary commands, and data transfers. This technique results in an overall increase in system efficiency since both processors—the master CPU and the slave UPI—are working in parallel.

This application note presents three UPI-41A applications which are roughly divided into two groups: applications whose complexity and UPI code space requirements allow them to either stand alone or be incorporated as just one task in a “multi-tasking” UPI, and applications which are complete UPI applications in themselves. Applications in the first group are a simple LED display and sensor matrix controllers. A combination serial/parallel/ I/O device is an application in the second group. Each application illustrates different UPI configurations and features. However, before the application details are presented, a section on the UPI/master protocol requirements is included. These protocol requirements are key to UPI software development. For convenience, the UPI block diagram is reproduced in Figure 1 and the instruction set summary in Table 1.

UPI-41 vs. UPI-41A

The UPI-41A is an enhanced version of the UPI-41. It incorporates several architectural features not found on the “non-A” device:

- Separate Data In and Data Out data bus buffer registers
- User-definable STATUS register bits
- Programmable master interrupts for the OBF and IBF flags
- Programmable DMA interface to external DMA controller.

The separate Data In (DBBIN) and Data Out (DBBOUT) registers greatly simplify the master/UPI protocol compared to the UPI-41. The master need only check IBF before writing to DBBIN and OBF before reading DBBOUT. No data bus buffer lock-out is required.

The most significant nibble of the STATUS register, undefined in the UPI-41, is user-definable in UPI-41A. It may be loaded directly from the most significant nibble of the Accumulator (MOV STS,A). These extra four STATUS bits are useful for transferring additional status information to the master. This application note uses this feature extensively.

A new instruction, EN FLAGS, allows OBF and IBF to be reflected on PORT 2 BIT 4 and PORT 2 BIT 5 respectively. This feature enables interrupt-driven data transfers when these pins are interrupt sources to the master.

By executing an EN DMA instruction PORT 2 BIT 6 becomes a DRQ (DMA Request) output and PORT 2 BIT 7 becomes DACK (DMA Acknowledge). Setting DRQ requests a DMA cycle to an external DMA controller. When the cycle is granted, the DMA controller returns DACK plus either RD (Read) or WR (Write). DACK automatically forces
CS and A0 low internally and clears DRQ. This selects the appropriate data buffer register (DBBOUT for DACK and RD, DBBIN for DACK and WR) for the DMA transfer.

Like the “non-A”, the UPI-41A is available in both ROM (8041A) and EPROM (8741A) Program Memory versions. This application note deals exclusively with the UPI-41A since the applications use the “A”s enhanced features.

UPI/MASTER PROTOCOL

As in most closely coupled multiprocessor systems, the various processors communicate via a shared resource. This shared resource is typically specific locations in RAM or in registers through which status and data are passed. In the case of a master processor and a UPI-41A, the shared resource is 3 separate, master-addressable, registers internal to the UPI. These registers are the status register (STATUS), the Data Bus Buffer Input register (DBBIN), and the Data Bus Output register (DBBOUT). [Data Bus Buffer direction is relative to the UPI]. To illustrate this register interface, consider the 8085A/UPI system in Figure 2.

Looking into the UPI from the 8085A, the 8085A sees only the three registers mentioned above. If the 8085A wishes to issue a command to the UPI, it does so by writing the command to the DBBIN register according to the decoding of Table 2. Data for the UPI is also passed via the DBBIN register. (The UPI differentiates commands and data by examining the A0 pin. Just how this is done is covered shortly.) Data from the UPI for the 8085A is passed in the DBBOUT register. The 8085A may interrogate the UPI’s status by reading the UPI’s STATUS register. Four bits of the STATUS register act as flags and are used to handshake data and commands into and out of the UPI. The STATUS register format is shown in Figure 3.

BIT 0 is OBF (Output Buffer Full). This flag indicates to the master when the UPI has placed data in the DBBOUT register. OBF is set when the UPI writes to DBBOUT and is reset when the master reads DBBOUT. The master finds meaningful data in the DBBOUT register only when OBF is set.

The Input Buffer Full (IBF) flag is BIT 1. The UPI uses this flag as an indicator that the master has written to the DBBIN register. The master uses IBF
APPLICATIONS

Figure 1C. UPI-41A Block Diagram

to indicate when the UPI has accepted a particular command or data byte. The master should examine IBF before outputting anything to the UPI. IBF is set when the master writes to DBBIN and is reset when the UPI reads DBBIN. The master must wait until IBF=0 before writing new data or commands to DBBIN. Conversely, the UPI must ensure IBF=1 before reading DBBIN.

The third STATUS register bit is F0 (FLAG 0). This is a general purpose flag that the UPI can set, reset, and test. It is typically used to indicate a UPI error or busy condition to the master.

FLAG 1 (F1) is the final dedicated STATUS bit. Like F0 the UPI can set, reset, and test this flag. However, in addition, F1 reflects the state of the A0 pin whenever the master writes to the DBBIN register. The UPI uses this flag to delineate between master command and data writes to DBBIN.

The remaining four STATUS register bits are user definable. Typical uses of these bits are as status indicators for individual tasks in a multitasking UPI or as UPI generated interrupt status. These bits find a wide variety of uses in the upcoming applications.

Looking into the 8085A from the UPI, the UPI sees the two DBB registers plus the IBF, OBF, and F1 flags. The UPI can write from its accumulator to DBBOUT or read DBBIN into the accumulator. The UPI cannot read OBF, IBF, or F1 directly, but these flags may be tested using conditional jump

Figure 2. Register Interface
### Table 1. Instruction Set Summary

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A,Ry</td>
<td>Add address register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A,@Ry</td>
<td>Add address memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A,ddata</td>
<td>Add immediate to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ADDC A,Ry</td>
<td>Add address register to A with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A,@Ry</td>
<td>Add address memory to A with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A,ddata</td>
<td>Add immediate to A with carry</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL A,Ry</td>
<td>AND register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A,@Ry</td>
<td>AND data memory to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL A,ddata</td>
<td>AND immediate to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ORL A,Ry</td>
<td>OR register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A,@Ry</td>
<td>OR data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A,ddata</td>
<td>OR immediate to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>XRL A,Ry</td>
<td>Exclusive OR register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A,@Ry</td>
<td>Exclusive OR data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A,ddata</td>
<td>Exclusive OR immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INCA</td>
<td>Increment A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal Adjust A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap digits of A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate A left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate A left through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate A right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A right through carry</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Accumulator

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN A,P</td>
<td>Input port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>OUTL P,A</td>
<td>Output A to port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL P,ddata</td>
<td>AND immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL P,ddata</td>
<td>OR immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>IN A,DBBB</td>
<td>Input DBB to A, clear IFB</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OUT DBBA,A</td>
<td>Output A to DBB, set OBFF</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV STS A</td>
<td>A4-A7 Bits to Bits 4-7 of Status</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOVDP P,A</td>
<td>Input Expander port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVDP P,A</td>
<td>Output A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANLD P,A</td>
<td>AND A to Expander port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORLD P,A</td>
<td>OR A to Expander port</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Data Moves

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,Ry</td>
<td>Move register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,@Ry</td>
<td>Move memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,ddata</td>
<td>Move immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV A,Ry</td>
<td>Move A to register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,Ry</td>
<td>Move A to data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,ddata</td>
<td>Move immediate to register</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV A,ddata</td>
<td>Move immediate to data memory</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV A,PSW,A</td>
<td>Move PSW to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,PSW,A</td>
<td>Move A to PSW</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A,Ry</td>
<td>Exchange A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A,@Ry</td>
<td>Exchange A and data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCHD A,@Ry</td>
<td>Exchange digit of A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,@A</td>
<td>Move A to current page</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVPS3 A,@A</td>
<td>Move A from page 3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 2. Register Decoding

<table>
<thead>
<tr>
<th>CS AO RD WR</th>
<th>REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1</td>
<td>READ DBBOUT</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>READ STATUS</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>WRITE DBBIN (DATA)</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>WRITE DBBIN (COMMAND)</td>
</tr>
<tr>
<td>1 X X X</td>
<td>NO ACTION</td>
</tr>
</tbody>
</table>

### Figure 3. Status Register Format

- **STATUS REGISTER**
- **OSF — DBBOUT FULL**
- **IBF — DBBIN FULL**
- **FO — FLAG 0**
- **F1 — FLAG 1**
- **USER DEFINED**

---

5-701
instructions. The UPI should make sure that OBF is reset before writing new data into DBBOUT to ensure that the master has read previous DBBOUT data. IBF should also be tested before reading DBBIN since DBBIN data is valid only when IBF is set. As was mentioned earlier, the UPI uses F₁ to differentiate between command and data contents in DBBIN when IBF is set. The UPI may also write the upper 4-bits of its accumulator to the upper 4-bits of the STATUS register. These bits are thus user definable.

The UPI can test the flags at any time during its internal program execution. It essentially “polls” the STATUS register for changes. If faster response is needed to master commands and data, the UPI’s internal interrupt structure can be used. If IBF interrupts are enabled, a master write to DBBIN (either command or data) sets IBF which generates an internal CALL to location 03H in program memory. At this point, working register contents can be saved using bank switching, the accumulator saved in a spare working register, and the DBBIN register read and serviced. The interrupt logic for the IBF interrupt is shown in Figure 4. A few observations concerning this logic are appropriate. Note that if the master writes to DBBIN while the UPI is still servicing the last IBF interrupt (a RETR instruction has not been executed), the IBF Interrupt Pending line is made high which causes a new CALL to 03H as soon as the first RETR is executed. No EN I (Enable Interrupt) instruction is needed to rearm the interrupt logic as is needed in an 8080 or 8085A system; the RETR performs this function. Also note that executing a DIS I to disable further IBF interrupts does not clear a pending interrupt. Only a CALL to location 03H or RESET clears a pending IBF interrupt.

Keeping in mind that the actual master/UIPI protocol is dependent on the application, probably the best way to illustrate correct protocol is by example. Let’s consider using the UPI as a simple parallel I/O device. (This is a trivial application but it embodies all of the important protocol considerations.) Since the UPI may be either interrupt or non-interrupt driven internally, both cases are considered.

Let’s take the easiest configuration first; using the UPI PORT 1 as an 8-bit output port. From the UPI’s point-of-view, this is an input-only application since all that is required is that the UPI input data from the master. Once the master writes data to the UPI, the UPI reads the DBBIN register and transfers the data to PORT 1. No testing for commands versus data is needed since the UPI “knows” it only performs one task—no commands are needed.

Figure 4. UPI-41A Interrupt Structure
Non-interrupt driven UPI software is shown in Figure 5A while Figure 5B shows interrupt based software. For Figure 5A, the UPI simply waits until it sees IBF go high indicating the master has written a data byte to DDBBIN. The UPI then reads DDBBIN, transfers it to PORT 1, and returns to waiting for the next data. For the interrupt-driven UPI, Figure 5B, once the EN I instruction is executed, the UPI simply waits for the IBF interrupt before handling the data. The UPI could handle other tasks during this waiting time. When the master writes the data to DDBBIN, an IBF interrupt is generated which performs a CALL to location 03H. At this point the UPI reads DDBBIN (no testing of IBF is needed since an IBF interrupt implies that IBF is set), transfers the data to PORT 1, and executes an RETR which returns program flow to the main program.

Software for the master 8085A is included in Figure 5C. The only requirement for the master to output data to the UPI is that it check the UPI to be sure the previous data had been taken before writing new data. To accomplish this the master simply reads the STATUS register looking for IBF=0 before writing the next data.

Figure 6A illustrates the case where UPI PORT 2 is used as an 8-bit input port. This configuration is termed UPI output-only as the master does not write (input) to the UPI but simply reads either the STATUS or the DDBBOUT registers. In this example only the OBF flag is used. OBF signals the master that the UPI has placed new port data in DDBBOUT. The UPI loops testing OBF. When OBF is clear, the master has read the previous data and UPI then reads its input port (PORT 2) and places this data in DDBBOUT. It then waits on OBF until the master reads DDBBOUT before reading the input port again. When the master wishes to read the input port data, Figure 6B, it simply checks for OBF being set in the STATUS register before reading DDBBOUT. While this technique illustrates proper protocol, it should be noted that it is not meant to be a good method of using the UPI as an input port since the master would never get the newest status of the port.

The above examples can easily be combined. Figure 7 shows UPI software to use PORT 1 as an output port simultaneously with PORT 2 as an input port. The program starts with the UPI checking IBF to see if the master has written data destined for the output port into DDBBIN. If IBF is set, the UPI reads DDBBIN and transfers the data to the output port (PORT 1). If IBF is not set or once the data is transferred to the output port if it was, OBF is tested. If OBF is reset (indicating the master has read DDBBOUT), the input port (PORT 2) is read and transferred to DDBBOUT. If OBF is set, the master has yet to read DDBBOUT so the program just loops back to test IBF.

The master software is identical to the separate input/output examples; the master must test IBF.
and OBF before writing output port data into DBBIN or before reading input port from DBBOUT respectively.

In all of the three examples above, the UPI treats information from the master solely as data. There has been no need to check if DBBIN information is a command rather than data since the applications do not require commands. But what if both PORTs 1 and 2 were used as output ports? The UPI needs to know into which port to put the data. Let’s use a command to select which port.

Recall that both commands and data pass through DBBIN. The state of the AO pin at the time of the write to DBBIN is used to distinguish commands from data. By convention, DBBIN writes with AO=0 are for data, and those with AO=1 are commands. When DBBIN is written into, F1 (FLAG 1) is set to the state of AO. The UPI tests F1 to determine if the information in the DBBIN register is data or command.

For the case of two output ports, let’s assume that the master selects the desired port with a command prior to writing the data. (We could just use F1 as a port select but that would not illustrate the subtle differences between commands and data). Let’s define the port select commands such that BIT 1=1 if the next data is for PORT 1 (Write PORT 1=0000 0010) and BIT 2=1 if the next data is for PORT 2 (Write PORT 2=0000 0100). (The number of the set bit selects the port.) Any other bits are ignored. This assignment is completely arbitrary; we could use any command structure, but this one has the advantage of being simple.

Note that the UPI must “remember” from DBBIN write to write which port has been selected. Let’s use F0 (FLAG 0) for this purpose. If a Write PORT 1 command is received, F0 is reset. If the command is Write PORT 2, F0 is set. When the UPI finds data in DBBIN, F0 is interrogated and the data is loaded into the previously selected port. The UPI software is shown in Figure 8A.

Initially, the UPI simply waits until IBF is set indicating the master has written into DBBIN. Once IBF is set, DBBIN is read and F1 is tested for a command. If F1=1, the DBBIN byte is a command. Assuming a command, BIT 1 is tested to see if the command selected PORT 1. If so, F0 is cleared and the program returns to wait for the data. If BIT 1=0, BIT 2 is tested. If BIT 2 is set, PORT 2 is selected so F0 is set. The program then loops back waiting for the next master input. This input is the desired port data. If BIT 2 was not set, F0 is not changed and no action is taken.

When IBF=1 is again detected, the input is again tested for command or data. Since it is necessarily data, DBBIN is read and F0 is tested to determine which port was previously selected. The data is then output to that port, following which the program waits for the next input. Note that since F0 still selects the previous port, the next input could be more data for that port. The port selection command could be thought of as a port select flip-flop control; once a selection is made, data may be repeatedly written to that port until the other port is selected. Master software, Figure 8B, simply must check IBF before writing either a command or data to DBBIN. Otherwise, the master software is straightforward.
error indicator. If the master happened to issue an invalid command (a command without either BIT 1 or 2 set), F0 is set to notify the master that the UPI did not know how to interpret the command. F0 is also set if the master commanded a port read before it had read DBBOUT from the previous command. The UPI simply tests OBF just prior to loading DBBOUT and if OBF=1, F0 is set to indicate the error.

All of the above examples are, in themselves, rather trivial applications of the UPI although they could easily be incorporated as one of several tasks in a UPI handling multiple small tasks. We have covered them primarily to introduce the UPI concept and to illustrate some master/UPI protocol. Before moving on to more realistic UPI applications, let’s discuss two UPI features that do not directly relate to the master/UPI protocol but greatly enhance the UPI's transfer capability.

In addition to the OBF and IBF bits in the STATUS register, these flags can also be made available directly on two port pins. These port pins can then be used as interrupt sources to the master. By executing an EN FLAGS instruction, PORT 2 pin 4 reflects the condition of OBF and PORT 2 pin 5 reflects the inverted condition of IBF (IBF). These dedicated outputs can then be enabled or disabled via their respective port bit values; i.e., P24 reflects OBF as long as an instruction is executed which sets P24 (i.e., ORL P2,#10H). The same action applies to the IBF output except P25 is used. Thus P24 may serve as a DATA AVAILABLE interrupt output. Likewise for P25 as a READY-TO-ACCEPT-DATA interrupt. This greatly simplifies interrupt-driven master-slave data transfers.

(DACK) input. Any instruction which would normally set P26 now sets DRQ. DRQ is cleared when DACK is low and either RD or WR is low. When DACK is low, CS and A0 are forced low internally which allows data bus transfers between DBBOUT or DBBIN to occur, depending upon whether WR or RD is true. Of course, the function requires the use of an external DMA controller.

Now that we have discussed the aspects of the UPI protocol and data transfer interfaces, let’s move on to the actual applications.

EXAMPLE APPLICATIONS

Each of the following three sections presents the hardware and software details of a UPI application. Each application utilizes one of the protocols mentioned in the last section. The first example is a simple 8-digit LED display controller. This application requires only that the UPI perform input operations from the DBBIN; DBBOUT is not used. The reverse is true for the second application: a sensor matrix controller. The final application involves both DBBOUT and DBBIN operations: a combination serial/parallel I/O device.

The core master processor system used in these applications is the isbc 80/30 single board computer. This board provides an especially convenient UPI environment since it contains a dedicated socket specifically interfaced for the UPI-41A. The 80/30 uses the 8085A as the master processor. The I/O and peripheral complement on the 80/30 include 12 vectored priority interrupts (8 on an 8259 Programmable Interrupt Controller and 4 on the 8085A itself), an 8253 Programmable Interval Timer supplying three 16-bit programmable timers (one is dedicated as a programmable baud rate generator), a high speed serial channel provided by an 8251 Programmable USART, and 24 parallel I/O...
lines implemented with an 8255A Programmable Parallel Interface. The memory complement contains 16K bytes of RAM using 2117 16K bit Dynamic RAMs and the 8202 Dynamic RAM Controller, and up to 8K bytes of ROM/EPROM with sockets compatible with 2716, 2758, or 2332 devices. The 80/30's RAM uses a dual port architecture. That is, the memory can be considered a global system resource, accessible from the on-board 8085A as well as from remote CPUs and other devices via the MULTIBUS. The 80/30 contains MULTIBUS control logic which allows up to 16 80/30s or other bus masters to share the same system bus. (More detailed information on the iSBC 80/30 and other iSBC products may be found in the latest Intel Systems Data Catalog.)

A block diagram of the iSBC 80/30 is shown in Figure 10. Details of the UPI interface are shown in Figure 11. This interface decodes the UPI registers in the following format:

<table>
<thead>
<tr>
<th>Register</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read STATUS</td>
<td>IN E5H</td>
</tr>
<tr>
<td>Write DBBIN (command)</td>
<td>OUT E5H</td>
</tr>
<tr>
<td>Read DBBOUT (data)</td>
<td>IN E4H</td>
</tr>
<tr>
<td>Write DBBIN (data)</td>
<td>OUT E4H</td>
</tr>
</tbody>
</table>

8-Digit Multiplexed LED Display

The traditional method of interfacing an LED display with a microprocessor is to use a data latch along with a BDC-to-7-segment decoder for each digit of the display. Thus two ICs, seven current limiting resistors, and about 45 connections are required for each digit. These requirements are, of course, multiplied by the total number of digits desired. The obvious disadvantages of this method are high parts count and high power dissipation since each digit is "ON" continuously. Instead, a scheme of time multiplexing the display can be used to decrease both parts count and power dissipation.

Display multiplexing basically involves connecting the same segment (a, b, c, d, e, f, or g) of each digit in parallel and driving the common digit element (anode or cathode) of each digit separately. This is shown schematically in Figure 12. The various digits of the display are not all on at once; rather, only one digit at a time is energized. As each digit is energized, the appropriate segments for that digit are turned on. Each digit is enabled in this way, in sequence, at a rate fast enough to ensure that each digit appears to be "ON" continuously. This implies that the display must be "refreshed" at periodic intervals to keep the digits flicker-free. If the CPU had to handle this task, it would have to suspend normal processing, go update the display, and then return to its normal flow. This extra burden is ideally handled by a UPI. The master CPU could simply give characters to the UPI and let the UPI do the actual segment decoding, display multiplexing, and refreshing.

As an example of this technique, Figure 13 shows the UPI controlling an 8-digit LED display. All digit segments are connected in parallel and are driven through segment drivers by the UPI PORT 1. The lower 3 bits of PORT 2 are inputs to a 3-to-8 decoder which selects an individual digit through a digit driver. A fourth PORT 2 line is used as a decoder enable input. The remaining PORT 2 lines plus the TEST 0 and TEST 1 inputs are available for other tasks.

Internally, the UPI uses the counter/timer in the interval timer mode to define the interval between display refreshes. Once the timer is loaded with the desired interval and started, the UPI is free to handle other tasks. It is only when a timer overflow interrupt occurs that the UPI handles the short display multiplexing routine. The display multiplexing can be considered a background task which is entirely interrupt-driven. The amount of time spent multiplexing is such that there is ample time to handle a non-timer task in the UPI foreground. (We'll discuss this timing shortly.)

When a timer interrupt occurs, the UPI turns off all digits via the decoder enable. The next digit's segment contents are retrieved from the internal data memory and output via PORT 1 to the segment drivers. Finally, the next digit's location is placed on PORT 2 (P20-P22) and the decoder enabled. This displays the digit's segment information until the next interrupt. The timer is then restarted for the next interval. This process continues repeatedly for each digit in sequence.

As a prelude to discussing the UPI software, let's examine the internal data memory structure used in this application, Figure 14. This application requires only 14 of the 64 total data memory locations. The top eight locations are dedicated to the Display Map; one location for each digit. These locations contain the segment and decimal point information for each character. Just how characters are loaded into this section of memory is covered shortly. Register R7 of Register Bank 1 is used as the temporary Accumulator store during the interrupt service routines. Register R3 stores the digit number of the next digit to be displayed. R2 is a temporary storage register for characters during input routine. R0 is
the offset pointer pointing to the Display Map location of the next digit. That makes 12 locations so far. The remaining two locations are the two stack locations required to store the return address plus status during the timer and input interrupt service routines. The remaining unused locations, all of Register Bank 0, 14 bytes of stack, 4 in Register Bank 1, and 24 general purpose RAM locations, are all available for use by any foreground task.

The UPI software consists of only three short routines. One, INIT, is used strictly during initialization. DISPLA is the multiplexing routine called at a timer interrupt. INPUT is the character input handler called at an IBF interrupt. The flow charts for these routines are shown in Figures 14A through 14C.

INIT initializes the UPI by simply turning off all segment and digit drivers, filling the Display Map with blank characters, loading and starting the timer, and enabling both timer and IBF interrupts. Although the flow chart shows the program looping at this point, it is here that the code for any foreground task is inserted. The only restrictions on this foreground task are that it not use I/O lines dedicated to the display and that it not require dedicated use of the timer. It could share the timer if precautions are taken to ensure that the display will still be refreshed at the required interval.
Figure 11. UPI Interface on iSBC 80/30

Figure 12. LED Multiplexing
APPLICATIONS

Figure 13. UPI Controlled 8-Digit LED Display

Figure 14. LED Display Controller Data Memory Allocation

Figure 14A. INIT Routine Flow
The INPUT routine handles the character input. It is called when an IBF interrupt occurs. After the usual swapping of register banks and saving of the accumulator, DBBIN is read and stored in register R2. DBBIN contains the Display Data Word. The format for this word, Figure 15, has two fields: Digit Select and Character Select. The Digit Select field selects the digit number into which the character from the Character Select field is placed. Notice that the character set is not limited strictly to numerics, some alphanumeric capability is provided. Once DBBIN is read, the offset for the selected digit is computed and placed in the Display Map Pointer Rg. Next the segment information for the selected character is found through a look-up table starting in page 3 of the program memory. This segment information is then stored at the location pointed at by the Display Map Pointer. If the Character Select field specified a decimal point, the segment corresponding to the decimal point is ANDed into the present segment information for that digit. After the accumulator is restored, execution is returned to the main program.

The DISPLA routine simply implements the multiplexing actions described earlier. It is called whenever a timer interrupt occurs. After saving pre-interrupt status by switching register banks and storing the Accumulator, all digit drivers are turned off. The Display Map Pointer is then updated using the Current Digit Register to point at that digit’s segment information in the Display Map. This information is output to PORT 1; the segment drivers. The number of the current digit, R3, is then sent to the digit select decoder and the decoder is enabled. This turns on the current digit. The digit counter is incremented and tested to see if all eight digits have been refreshed. If so, the digit counter is reset to zero. If not, nothing is done. Finally, the timer is loaded and restarted, the Accumulator is restored, and the routine returns execution to the main program. Thus DISPLA refreshes one digit each time it is called by the timer interrupt. The digit remains on until the next time DISPLA is executed.

The UPI software listing is included as Appendix A1. Appendix A2 shows the 8085A test routine used...
to display the contents of a display buffer on the display. The 8085A software takes care of the display digit numbering. Since the application is input-only for the UPI, the only protocol required is that the master must test IBF before writing a Display Data Word into DBBIN.

On the iSBC 80/30, the UPI frequency is at 5.5296 MHz. To obtain a flicker-free display, the whole display must be refreshed at a rate of 50 Hz or greater.

If we assume a 50 Hz refresh rate and an 8-digit display, this means the DISPLA routine must be CALLed 50×8 or 400 times/sec. This transfers, using the timer interval of 87 μs at 5.5296 MHz, to a timer count of 227. (Recall from the UPI-41A User's Manual that the timer is an “8-bit up-counter”.) Hence the TIME equate of 227D in the UPI listing. Obviously, different frequency sources or display lengths would require that this equate be modified.

With the UPI running at 5.5296 MHz, the instruction cycle time is 2.713 μs. The DISPLA routine requires 28 instruction cycles, therefore, the routine executes in 76 μs. Since DISPLA is CALLed 400 times/sec, the total time spent refreshing the display during one second is then 30 ms or 3% of the total UPI time. This leaves 97.0% for any foreground tasks that could be added.

While the basic UPI software is useful just as it stands, there are several enhancements that could be incorporated depending on the application. Auto-incrementing of the digit location could be added to the input routine to alleviate the need for the master to keep track of digit numbers. This could be optionally either right-handed or left-handed entry a la TI or HP calculators. The character set could be easily modified by simply changing the lookup table. The display could be expanded to 16 digits at the expense of one additional PORT 2 digit select line, the replacement of the 3-to-8 decoder with a 4-to-16 decoder, and 8 more Display Map locations.

Now let's move on to a slightly more complex application that is UPI output-only—a sensor matrix controller.

**Sensor Matrix Controller**

Quite often a microprocessor system is called upon to read the status of a large number of simple SPST switches or sensors. This is especially true in a process or industrial control environment. Alarm systems are also good examples of systems with a large sensor population. If the number of sensors is small, it might be reasonable to dedicate a single input port pin for each sensor. However, as the number of sensors increase, this technique becomes very wasteful. A better arrangement is to configure the sensors in a matrix organization like that shown in Figure 16. This arrangement of 16 sensors requires only 4 input and 4 output lines; half the number needed if dedicated inputs were used. The line saving becomes even more substantial as the number of sensors increases.
In Figure 16, the basic operation of the matrix involves scanning individual row select lines in sequence while reading the column return lines. The state of any particular sensor can then be determined by decoding the row and column information. The typical configuration pulls up the column return lines and the selected row is held low. De-selected rows are held high. Thus a return line remains high for an open sensor on the selected row and is pulled low for a closed sensor. Diode isolation is used to prevent a phantom closure which would occur when a sensor is closed on a selected row and there are two or more closures on a deselected row. Germanium diodes are used to provide greater noise margin at the return line input.

If the main processor was required to control such a matrix it would periodically have to output at the row port and then read the column return port. The processor would need to maintain in memory a map of the previous state of the matrix. A comparison of the new return information to the old information would then be made to determine whether a sensor change had occurred. Any changes would be processed as needed. A row counter and matrix map pointer also require maintenance each scan. Since in most applications sensors change very slowly compared to most processing actions, the processor probably would scan the rows only periodically with other tasks being processed between scans.

Rather than require the processor to handle the rather mundane tasks of scanning, comparing, and decoding the matrix, why not use a dedicated processor? The UPI is perfect.

Figure 17 shows a UPI configuration for controlling up to 128 sensors arranged in a 16x8 matrix. The 4-to-16 line decoder is used as the row selector to save port pins and provides the expansion to 128 sensors over the maximum of 64 sensors if the port had been used directly. It also helps increase the port drive capability. The column return lines go directly into PORT 1. Features of this design include complete matrix management. As the UPI scans the matrix it compares its present status to the previous scan. If any change is detected, the location of the change is decoded and loaded, along with the sensor's present state, into DBBOUT. This byte is called a Change Word. The Master processor has only to read one byte to determine the status and coordinate of a changed sensor. If the master had not read a previous Change Word in DBBOUT (OBF=1) before a new sensor change is detected, the new Change Word is automatically loaded.
APPLICATIONS

Word is loaded into an internal FIFO. This FIFO buffers up to 40 changes before it fills. The status of the FIFO and OBF is made available to the master either by polling the UPI STATUS register, Figure 18A, or as interrupt sources on port pins P24 and P25 respectively, Figure 17. The FIFO NOT EMPTY pin and bit are true as long as there are changes not yet read in the FIFO. As long as the FIFO is not empty, the UPI monitors OBF and loads new Change Words from the FIFO into DBBOUT. Thus, the UPI provides complete FIFO management.

Figure 18A. Sensor Matrix Status Register Format

Figure 18B. Sensor Matrix Change Word Format

Internally, the matrix scanning software is programmed to run as a foreground task. This allows the timer/counter to be used by any background task although the hardware configuration leaves only 2 inputs (TEST 0 and TEST 1) plus 2 I/O port pins available. Also, to add a background task, the FIFO would have to be made smaller to accommodate the needed register and data memory space. (It would be possible however to turn the table here and make the scanning software timer/counter interrupt-driven where the timer times the scan interval.)

The data memory organization for this application is shown in Figure 19. The upper 16 bytes form the Matrix Map and store the sensor states from the previous scan; one bit for each sensor. The Change Word FIFO occupies the next 40 locations. (The top and bottom addresses of this FIFO are treated as equate variables in the program so that the FIFO size may easily be changed to accommodate the register needs of other tasks.) Register R9 serves as a pointer into the matrix map area for comparisons and updates of the sensor status. R1 is a general FIFO pointer. The FIFO is implemented as a circular buffer with In and Out pointer registers which are stored in R4 and R5 respectively. These registers are moved into FIFO pointer R1 for actual transfers into or out of the FIFO. R2 is the Row Select Counter. It stores the number of the row being scanned.

Figure 19. Sensor Matrix Data Memory Map

Register R3 is the Column Counter. This counter is normally set to 00H; however, when a change is detected somewhere in a particular row, it is used to inspect each sensor status bit individually for a change. When a changed counter sensor bit is found, the Row Select Counter and Column Counter are combined to give the sensor's matrix coordinate. This coordinate is temporarily stored in the Change Word Store, register R6. Register R7 is the Compare Result. As each row is scanned, the return information is Exclusive-OR'd with the return information from the previous scan of that row. The result of this operation is stored in R7. If R7 is zero, there have been no changes on that row. A non-zero result indicates at least one changed sensor.

The basic program operation is shown in the flow chart of Figure 20. At RESET, the software initializes the working registers, the ports, and clears the STATUS register. To get a starting point from which to perform the sensor comparisons, the current status of the matrix is read and stored in the Matrix Map. At this point, the UPI begins looking for changed sensors starting with the first row.
Before delving further into the flow, let's pause to describe the general format of the operation. The UPI scans the matrix one row at a time. If no changes are detected on a particular row, the UPI simply moves to the next row after checking the status of DBBOUT and the FIFO. If a change is detected, the UPI must check each bit (sensor) within the row to determine the actual sensor location. (More than one sensor on the scanned row could have changed.) Rather than test all 8 bits of the row before checking the DBBOUT and FIFO status again, the UPI performs the status check in between each of the bit tests. This ensures the fastest response to the master reading previous Change Words from DBBOUT and the FIFO.

With this general overview in mind, let's go first thru the flow chart assuming we are scanning a row where no changes have occurred. Starting at the Scan-and-Compare section, the UPI first checks if the entire matrix has been scanned. If it has, the various pointers are reset. If not, the address of the next row is placed on PORTs 20 thru 23. This selects the desired row. The state of the row is then read on PORT 1; the column return lines. This present state is compared to the previous state by retrieving the previous state from the matrix map and performing an Exclusive-OR with the present state. Since we are assuming that no change has occurred, the result is zero. No coordinate decoding is needed and the flow branches to the FIFO-DBBOUT Management section.

The FIFO-DBBOUT Management section simply maintains the FIFO and loads DBBOUT whenever Change Words are present in the FIFO and DBBOUT is clear (OBF=0). The section first tests if the FIFO is full. (If we assume our “no-change” row is the first row scanned, the FIFO obviously would not be full.) If it is, the UPI waits until OBF=0, at which point the next Change Word is retrieved from the FIFO and placed in DBBOUT. This “unfills” the FIFO making room for more Change Words. At this point, the Column Counter, R3, is checked. For rows with no changes, the Column Counter is always zero so the test simply falls through. (We cover the case for changes shortly.) Now the FIFO is tested for being empty. If it is, there is no sense in any further tests so the flow simply goes back up to scan the next row. If the FIFO is not empty, DBBOUT is tested again through OBF. If a Change Word is in DBBOUT waiting for the master to read it, nothing can be done and the flow likewise branches up for the next row. However, if the DBBOUT is free and remembering that the previous test showed that the FIFO was not empty, DBBOUT is loaded with the next Change Word and the last two conditional tests repeat.
APPLICATIONS

Now let’s assume the next row contains several changed sensors. Like before, the row is selected, the return lines read, and the sensor status compared to the previous scan. Since changes have occurred, the Exclusive-OR result is now non-zero. Any 1’s in the result reflect the positions of the changed sensors. This non-zero result is stored in the Compare Result register, R7. At this point, the Column Counter is preset to 8. To determine the changed sensors’ locations, the Compare Result register is shifted bit-by-bit to the left while decrementing the Column Counter. After each shift, BIT 7 of the result is tested. If it is a one, a changed sensor has been found. The Column Counter then reflected the sensor’s matrix column position while the Scan Row Select register holds it row position. These registers are then combined in R6, the Change Word Store, to form the sensor’s matrix coordinate section of the Change Word. The 8th bit of the Change Word Store is coded with the sensor’s present state (Figure 18). This byte forms the complete Change Word. It is loaded into the next available FIFO position. If BIT 7 of the Compare Result had been zero, that particular sensor had not changed and the coordinate decoding is not performed.

In between each shift, test, and coordinate encode (if necessary), the FIFO-DBBOUT Management is performed. It is the Column Counter test within this section that routes the flow back up to the Change Word Encoding section if the entire Compare Result (row) has not been shifted and tested.

The FIFO is implemented as a circular buffer with IN and OUT pointers (R4 and R5 respectively). The operations of the FIFO is best understood using an example, Figure 21. This series of figures show how the FIFO, DBBOUT, and OBF interact as changes are detected and Change Words are read by the master. The letters correspond to sequential Change Words being loaded into the FIFO. Note that the figures show only a 4x8 FIFO however, the principles are the same in the 40x8 FIFO.

Figure 21A shows the condition where no Change Words have been loaded into the FIFO or DBBOUT. In Figure 21B a change, “A”, has been detected, decoded, and loaded into the FIFO at the location equal to the value of the FIFO-IN pointer. The FIFO-OUT pointer is reset to the bottom of the FIFO since it had reached the FIFO top. Now that a Change Word is in the FIFO, OBF is checked to see if DBBOUT is empty. Because OBF=0, DBBOUT is empty and the Change Word is loaded from the FIFO location pointed at by the FIFO-OUT pointer. This is shown in Figure 21C. Loading DBBOUT automatically sets OBF. OBF remains set until the master reads DBBOUT. Figures 21D and 21E show two more Change Words loaded into the FIFO. In Figure 21F the first Change Word is finally read by the master resetting OBF. This allows the next Change Word to be loaded into DBBOUT. Note that each time the FIFO is loaded, the FIFO-IN pointer increments. Each time DBBOUT is read the FIFO-OUT pointer increments unless there are no more Change Words in the FIFO. Both pointers wrap-around to the bottom once they reach the FIFO top. The remaining figures show more Change Words being loaded into the FIFO. When the entire FIFO fills and DBBOUT can not be loaded (OBF=1), scanning stops until the master reads DBBOUT making room for more Change Words.

As was mentioned earlier, two interrupt outputs to the master are available: Change Word Ready (P25, OBF) and FIFO NOT EMPTY (P24). The Change Word Ready interrupt simply reflects OBF and is handled automatically by the UPI since an EN FLAGS instruction is executed during initialization. The FIFO NOT EMPTY interrupt is generated and cleared as appropriate, each pass through the FIFO management code.

No debouncing is provided although it could be added. Rather, the scan time is left as an equate variable so that it could be varied to account for both debounce time and expected sensor change rates. The minimum scan time for this application is 2msec when using a 6MHz clock. Since the matrix controller is coded as a foreground task, scan time simply uses a software delay loop.

The UPI software is included as Appendix B1. Appendix B2 is 8085A test software which builds a Change Word buffer starting at BUFSRT. This software simply polls the STATUS register looking for Change Word Ready to go true. DBBOUT is then read and loaded into the buffer. Now let’s move on to an application which combines both the foreground and background concepts.

Combination I/O Device

The final UPI application was designed especially to add additional serial and parallel I/O ports to the iSBC 80/30. This UPI simulates a full-duplex UART (Universal Asynchronous Receiver/Transmitter) combined with an 8-bit parallel I/O port. Features of the UART include: software selectable baud rates (110, 300, 600, or 1200 baud), double buffering for both the transmitter and receiver, and receiver testing for false start bit, framing, and overrun errors. For parallel I/O, one 8-bit port is programmable for either input or output. The output port is statically latched and the input port is sampled.

5-715
APPLICATIONS

Figure 21A-J. FIFO Operation Example
APPLICATIONS

Figure 22 shows the interface of this combination I/O device to the dedicated UPI socket on the iSBC 80/30. The only external requirement is a 76.8 kHz source which serves as the baud rate standard. The internal baud rates are generated as multiples of this external clock. This clock is obtained from one of the 8253 counters. Otherwise, an RS-232 driver and receiver already available for UPI use in serial I/O applications. Sockets are also provided for termination of the parallel port.

![Figure 22. Combination I/O Device](image)

There are three commands for this application. Their format is shown in Figure 23. The CONFIGURE command specifies the serial baud rate and the parallel I/O direction. Normally this command is issued once during system initialization. The I/O command causes a parallel I/O operation to be performed. If the parallel port direction is out, the UPI expects the data byte immediately following an I/O command to be data for the output port. If the port is in the input direction, an I/O command causes the port to be read and the data placed in DBBOUT. The RESET ERROR command resets the serial receiver error bits in the STATUS register.

![Figure 23. Combination I/O Command Format](image)

The STATUS register format is shown in Figure 24. Looking at each bit, BIT 0 (OBF) is the DATA AVAILABLE flag. It is set whenever the UPI places data into DBBOUT. Since the data may come from either the receiver or the parallel input port, the F0 and F1 flags (BITS 2 and 3) code the source. Thus, when the master finds OBF set, it must decode F0 and F1 to determine the source.

![Figure 24. STATUS Register Format](image)

BIT 1 (IBF) functions as a busy bit. When IBF is set, no writes to DBBIN are allowed. BIT 5 is the TxINT (Transmitter Interrupt) bit. It is asserted whenever the transmitter buffer register is empty. The master uses this bit to determine when the transmitter is ready to accept a data character.

BITS 6 and 7 are receiver error flags. The framing error flag, BIT 6, is set whenever a character is received with an invalid stop bit. BIT 7, overrun error, is set if a character is received before the master has read a previous character. If an overrun occurs, the previous character is overwritten and lost. Once an error occurs, the error flag remains set until reset by a RESET ERROR command. A set error flag does not inhibit receiver operation however.

Figure 25 shows the port pin definition for this application. PORT 1 is the parallel I/O port. The UART uses PORT 2 and the Test inputs. P20 is the transmitter data out pin. It is set for a mark and reset for a space. P23 is a transmitter interrupt output. This pin has the same timing as the TxINT bit in the STATUS register. It is normally used in interrupt-driven systems to interrupt the master processor when the transmitter is ready to accept a new data character.

The OBF flag is brought out on P24 as a master interrupt when data is available in DBBOUT. P26 is a diagnostic pin which pulses at four times the selected baud rate. (More about this pin later.) The receiver data input uses the TEST 0 input. One of the PORT 2 pins could have been used, however, the

5-717
software can test the TEST 0 in one instruction without first reading a port.

The TEST 1 input is the baud rate external source. The UART divides this input to determine the timing needed for the selected baud rate. The input is a non-synchronous 76.8 kHz source.

Internally, when the CONFIGURE command is received and the selected baud rate is determined, the internal timer/counter is loaded with a baud rate constant and started in the event counter mode. Timer/counter interrupts are then enabled. The baud rate constant is selected to provide a counter interrupt at four times the desired baud rate. At each interrupt, both the transmitter and receiver are handled. Between interrupts, any new commands and data are recognized and executed.

As a prelude to discussing the flow charts, Figure 26 shows the register definition. Register Bank 0 serves the UART receiver and parallel I/O while Register Bank 1 handles the UART transmitter and commands. Looking at RB0 first, R3 is the receiver status register, RxSTS. Reflected in the bits of this register is the current receiver status in sequential order. Figure 27 shows this bit definition. BIT 0 is the Rx flag. It is set whenever a possible start bit is received. BIT 1 signifies that the start bit is good and character construction should begin with the next received bit. BIT 1 is the Good Start flag. BIT 2 is the Byte Finished flag. When all data bits of a character are received, this flag is set. When all the bits, data and stop bits are received, the assembled character is loaded into the holding register (R4 in Figure 27) BIT 3, the Data Ready flag, is set. The foreground routine which looks for commands and data continuously, looks at this bit to determine when the receiver has received a character. BITS 4 and 5 signify any error conditions for a particular character.

The parallel I/O port software uses BITS 6 and 7. BIT 6 codes the I/O direction specified by the last CONFIGURE command, BIT 7 is set whenever an I/O command is received. The foreground routine tests this bit to determine when an I/O operation has been requested by the master.

As was mentioned, R4 is the receiver holding register. Assembled characters are held in this register until the foreground routine finds DBBOU free, at which time the data is transferred from R4 to DBBOU. R5 is the receiver tick counter. Recall that counter interrupts occur at four times the baud rate. Therefore, once a start bit is found, the receiver only needs to look at the data every four interrupts or tick counts. R5 holds the current tick count.

R6 is the receiver de-serializing register. Data characters are assembled in this register. R6 is preset to 80H when a good start bit is received. As each bit is
sampled every four timer ticks, they are rotated into the leftmost bit of R6. The software knows the character assembly is complete when the original preset bit rotates into the carry.

An image of the upper 4 bits of the STATUS register is stored in R7. These bits are the TxINT, Framing and Overrun bits. This image is needed since the UPI may load the upper 4 STATUS register bits from its accumulator; however, it cannot read STATUS directly.

In Register Bank 1 (Figure 26), R1 holds the baud rate constant which is found from decoding the baud rate select bits of the CONFIGURE command. The counter is reloaded with this constant every timer tick. Like the receiver, the transmitter only needs to update the transmitter output every four ticks. R2 holds the transmitter tick count. The value of R2 determines which portion of the data is being transmitted; start bit, data bits, or stop bit. The transmit serializer is R3. R3 holds the data character as each character bit is transmitted.

R4 is the transmitter holding register. It provides the double buffering for the transmitter. While transmitting one character, it is possible to load the next character into R4 via DBBIN. The TxINT bit in STATUS and pin on PORT 2 reflect the “fullness” of R4. If the holding register is empty, the interrupt bit and pin are set. They are reset when the master writes a new data byte for the transmitter into DBBIN. The transmitter status register (TxSTS) is R5. Like RxSTS, TxSTS contains flag bits which indicate the current state of the transmitter. This flag bit format is shown in Figure 28.

TxSTS BIT 0 is the Tx flag. It is set whenever the transmitter is transmitting a character. It is set from the beginning of the start bit until the end of the stop bit. BIT 1 is the Tx request flag. This bit is set by the foreground routine when it transfers a new character from DBBIN to the Tx holding register, R4. The transmitter software uses this flag to tell if new data is available. It is reset when the transmitter transfers the character from the holding register to the serializer.

BIT 2 is the pipelined Tx data bit. The transmitter uses a pipelining technique which sets up the next output level in BIT 2 after processing the current timer tick. The output level is always changed at the same point after a timer tick interrupt. This technique ensures that no bit timing distortion results from different length processing paths through the receiver and transmitter routines.

BIT 3 of TxSTS is the Start Bit flag. It is set by the transmitter when the start bit space is set up in the pipelined data bit. This allows the transmitter to differentiate between the start bit and the data bits on following timer ticks.

The flow charts for this application are shown in Figures 29A–F. At reset, the INIT routine is executed which initializes the registers and port pins. After initialization, IBF and OBF are tested in MNLOOP. These flags are tested continually in this loop. If IBF is set, F1 is tested for command or data and execution is transferred to the appropriate routine (CMD or DATA). If IBF=0, OBF is checked. If OBF=0 (DBBOUT is free), the Rx data ready and I/O flags in RxSTS are tested. If Rx data ready is set, the received data is retrieved from the Rx holding register and transferred to DBBOUT. Any error flags associated with that data are also transferred to STATUS. If the I/O flag is set and the I/O direction is input, PORT 1 is read and the data transferred to DBBOUT. In either case, F0 and F1 are set to indicate the data source.

If IBF is set by a command write to DBBIN, CMD reads the command and decodes the desired operation. If an I/O operation is specified, the I/O flag is set to indicate to the MNLOOP and DATA routines that an I/O operation is to be performed. If the command is a CONFIGURE command, the constant for the selected baud rate is loaded into both Baud Rate Constant register and the timer/counter. The timer/counter is started in the event counter mode and timer/counter interrupts are enabled. In addition, the I/O port is initialized to all 1’s if the I/O direction bit specifies an input port. If the command is a RESET ERROR command, the two error flags in STATUS are cleared.

If the IBF flag is set by a data write, the DATA routine reads DBBIN and places the data in the appropriate place. If the I/O flag is set, the data is for the output port so the port is loaded. If the I/O flag is reset, the data is for the UART transmitter. Data for the transmitter resets the TxINT bit and pin plus sets the Tx request flag in TxSTS. The data is transferred to the Tx holding register, R4.
Once a CONFIGURE command is received and the counter started, timer/counter interrupts start occurring at four times the selected baud rate. These interrupts cause a vector to the TIMINT routine, Figure 29D. A 76.8 kHz counter input provides a 13.02 μs counter resolution. Since it requires several UPI instruction cycles to reload the counter, the counter is set to two counts less than the desired baud rate and the counter is reloaded in TIMINT synchronous with the second low-going transition after the interrupt. Once the counter is reloaded, an output port (P26) is toggled to give an external indication of internal counter interval. This is a helpful diagnostic feature. After the tick sample output, the pipelined transmitter data in TxSTS is output to the TxD pin. Although this occurs every timer tick, the pipelined data is changed only every fourth tick.

The receiver is now handled, Figure 29E. The Rx flag in RxSTS is examined to see if the receiver is currently in the process of receiving a character. If it is not, the RxD input is tested for a space condition which might indicate a possible start bit. If the input is a mark, no start bit is possible and execution...
branches to the transmitter flow, XMIT. If the input is a space, the Rx flag is set before proceeding with XMIT.

If the Rx flag is found set when entering RCV, the receiver is in the process of receiving a character. If so, the start bit flag is then tested to determine if a good start bit was received. The Rx tick counter is initialized to 4 and the Rx deserializer is set to 80H. A mark indicates a bad start bit; the Rx flag is reset to abort the reception.

If the start bit flag is set, the program is somewhere in the middle of the received character. Since the data should be sampled every fourth timer tick, the tick counter is decremented and tested for zero. If non-zero no sample is needed and execution continues with XMIT. If zero, the tick counter is reset to four. Now the byte finished flag is tested to determine if the data sample is a data or stop bit. If reset, the sample is a data bit. The sample is done and the new bit rotated into the Rx deserializer. If this rotate
APPLICATIONS

Figure 29D. TIMINT Flow Chart

sets the carry, that data bit was the last so the byte finished flag is set. If the carry is reset, the data bit is not the last so execution simply continues with XMIT.

Had the byte finished flag been set, this sample is for the stop bit. The RxD input is tested and if a space, the framing error flag is set. Otherwise, it is reset. Next, the Rx data ready flag is tested. If it is set, the master has not read the previous character so the overrun error flag is set. Then the Rx data ready flag is set and the received data character is transferred into the Rx holding register. The Rx, start bit, and byte finished flags are reset to get ready for the next character.

Execution of the transmitter routine, XMIT, follows the receiver, Figure 29F. The transmitter starts by checking the start bit flag in TxSTS. Recall that the actual transmit data is output at the beginning of the timer routine. The start bit flag indicates whether the current timer tick interrupt started the start bit. If it is set, the pipelined data output earlier in the routine was the start of the start bit so the flag is reset and the Tx tick counter is initialized. Nothing else is done this timer tick so the routine returns to the foreground.

If the start bit flag is reset, the Tx tick counter is incremented and tested. The test is performed modulo 4. If the counter mod 4 is not zero, it has not been four ticks since the transmitter was handled last so the routine simply returns. If the counter mod 4 is zero, it is time to handle the transmitter and the Tx flag is tested.

The Tx flag indicates whether the transmitter is active. If the transmitter is inactive, no character is currently being transmitted so the Tx request flag is tested to see if a new character is waiting in the Tx buffer. If no character is waiting (Tx request flag=0), the Tx interrupt pin and bit are set before returning to the foreground. If there is a character waiting, it is retrieved from the buffer and placed in the Tx serializer. The Tx request flag is reset while the Tx and start bit flags are set. A space is placed in the Tx pipelined data bit so a start bit will be output on the next tick. Since the Tx buffer is now empty, the Tx interrupt bit and pin are set to indicate the availability of the buffer to the master. The routine then returns to the foreground.

If the tick counter mod 4 is zero and the Tx flag indicates the transmitter is in the middle of a character, the tick counter is checked to see what transmitter operation is needed. If the counter is 28H (40D), all data bits plus the stop bits are complete. The character is therefore done and the Tx flag is reset. If the counter is 24H (36D), the data bits are complete and the next output should be a mark for the stop bit so a mark is loaded into the Tx pipelined data bit.

If neither of the above conditions are met for the counter, the transmitter is some place in the data field, so the next data bit is rotated out of the Tx serializer into the pipelined data bit. The next tick outputs this bit.

At this point the program execution is returned to the foreground.

That completes the discussion of the combination I/O device flow charts. The UPI software listing is shown in Appendix C1. Appendix C2 is example 8085A driver software.

Several observations concerning the drivers are appropriate. Notice that since the receiver and input port of the UPI use the OBF flag and interrupt output, the interrupt and flag are cleared when the master reads DBBOUT. This is not true for the transmitter. There is always some time after a master write of new transmitter data before the transmitter bit and pin are cleared. Thus in an interrupt-driven system, edge-sensitive interrupts should be
used. For polled-systems, the software must wait after writing new data for IBF=0 before re-examining the Tx interrupt flag in STATUS.

Notice that this application uses none of the user data memory above Register Bank 1 and only 361 bytes of program memory. This leaves the door open for many improvements. Improvements that come to mind are increased buffering of the transmit or received data, modem control pins, and parallel port handshaking inputs.

This completes our discussion of specific UPI applications. Before concluding, let's look briefly at two debug techniques used during the development of these applications that you might find useful in your own designs.

DEBUG TECHNIQUES

Since the UPI is essentially a single-chip microcomputer, the classical data, address, and control buses are not available to the outside world during normal operation. This fact normally makes debugging a UPI design difficult; however, certain “tricks” can be included in the UPI software to ease this task.

If a UPI is handling multiple tasks, it is usually easier to code and debug each task individually. This is fairly standard procedure. Since each task usually utilizes only a subset of the total number of I/O pins,
coding only one task leaves some I/O pins free. Port output instructions can then be added in the task code being debugged which toggle these unused pins to determine which section of task code is being executed at any particular time. The task can also be made to "wait" at various points by using an extra pin as an input and adding code to loop until a particular input condition is met.

One example of using an extra pin as an output is included in the combination serial/parallel device code. During initial development the receiver was not receiving characters correctly. Since this could be caused by incorrect sampling, three lines of code were added to toggle BIT 6 of PORT 2 at each tick of the sample clock. This code is at lines 184 and 185 of the listing. Thus by looking at the location of the tick sample pulse with respect to the received bit, the UPI sampling interval can be observed. The tick sample time was incorrect and the code was modified accordingly. Similar techniques could be applied at other locations in the program.

The EPROM version of the UPI (8741A) also contains another feature to aid in debug: the capability to single step thru a program. The user may step thru the program instruction-by-instruction. The address of the next instruction to be fetched is available on PORT 1 and the lower 2 bits of PORT 2. Figure 30 shows the timing used in the discussion below. When the single step input, SS, is brought low, the internal processor responds by stopping during the fetch portion of the next instruction. This action is acknowledged by the processor raising the SYNC
output. The address of the instruction to be fetched is then placed on the port pins. This state may be held indefinitely. To step to the next instruction, SS is raised high, which causes SYNC to go low, which is then used to return SS low. This allows the processor to advance to the next instruction. If SS is left high, the processor continues to execute at normal speed until SS goes low.

To preserve port functionality, port data is valid while SYNC is low. Figure 31 shows the external circuitry required to implement single step while preserving port functionality. S1 is the RUN/STOP switch. When in the RUN position, the 7474 is held preset so SS is high and the UPI executes normally. When switched to STOP, the preset is removed and the next low-going transition of SYNC causes the 7474 to clear, lowering SS. While sync is low, the port data is valid and the current instruction is executing. Low SYNC is also used to enable the tri-state buffers when the ports are used as inputs. When execution is complete, SYNC goes high. This transition latches the valid port data in the 74LS374s. SYNC going high also signifies that the address of the next instruction will appear on the port pins. This state can be held indefinitely with the address data displayed on the LEDs.

When the S2 is depressed, the 7474 is set which causes SS to go high. This allows the processor to fetch and execute the instruction whose address was displayed. SYNC going low during execution, clears
the 7474 lowering SS. Thus the processor again stops when execution is complete and the next fetch is started.

All UPI functions continue to operate while single stepping (the processor is actually executing NOPs internally while stopped). Both IBF and timer/counter interrupts can be serviced. The only change is that the interval timer is prescaled on single stepped instructions and, of course, will not indicate the correct intervals in real time. The total number of instruction which would have been executed during a given interval is the same however.

The single step circuitry can be used to step through a complete program; however, this might be a time-consuming job if the program is long or if only a portion is to be examined. The circuitry could easily be modified to incorporate the output toggling technique to determine when to run and stop. If you would like to step thru a particular section of code, an extra port pin could replace switch S1. Extra instructions would then be added to lower the port when entering the code section and raise the port when exiting the section. The program would then stop when that section of code is reached allowing it to be stepped through. At the end of the section, the program would execute at normal speed.

**CONCLUSION**

Well, that's it. Machine readable (floppy disk or paper tape) source listings of UPI software for these applications are available in Insite, the Intel library of user-donated programs. Also available in Insite are the source listings for some of Intel's pre-programmed UPI products.

For information about Insite, write to:

Insite
Intel Corp.
3065 Bowers Ave.
Santa Clara, Ca 95051
APPENDIX A
### APPLICATIONS

**F1 ASM48 F3 LED PRINT (LP) NOOBJECT**

ISIS-II MCS-48/UP-41 macro assembler, V3 0

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NOD41A</td>
<td>2</td>
<td>***************************</td>
</tr>
<tr>
<td>3</td>
<td># UPI-41A 8-DIGIT LED DISPLAY CONTROLLER #</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>THIS PROGRAM USES THE UPI-41A AS A LED DISPLAY CONTROLLER</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>WHICH SCANS AND REFRESHES EIGHT SEVEN-SEGMENT LED DISPLAYS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>THE CHARACTERS ARE DEFINED BY INPUT FROM A MASTER CPU IN THE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>FORM OF ONE EIGHT BIT WORD PER DIGIT-CHARACTER SELECTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>***************************</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>REGISTER DEFINITIONS</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>R0 DISPLAY MAP POINTER NOT USED</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>R2 DATA WORD AND CHARACTER STORAGE NOT USED</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>R3 DIGIT COUNTER NOT USED</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>27</td>
<td>R5 NOT USED NOT USED</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>R7 ACCUMULATOR STORAGE NOT USED</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>31</td>
<td>PORT PIN DEFINITIONS</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>35</td>
<td>#EJECT</td>
<td></td>
</tr>
</tbody>
</table>

PORT 1 FUNCTION | PORT 2 FUNCTION | SEGMENT DRIVER CONTROL | DIGIT DRIVER CONTROL

- PORT 0-7

5-728
## Applications

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>Source Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td></td>
<td>37</td>
<td><strong>DISPLAY DATA WORD BIT DEFINITION</strong></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td>39</td>
<td>BIT FUNCTION</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>41</td>
<td>0-7 CHARACTER SELECT</td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
<td>5-7 DIGIT SELECT</td>
</tr>
<tr>
<td>43</td>
<td></td>
<td></td>
<td>CHARACTER SELECT:</td>
</tr>
<tr>
<td>44</td>
<td></td>
<td>45</td>
<td>D4 D3 D2 D1 D0 CHARACTER</td>
</tr>
<tr>
<td>46</td>
<td></td>
<td>47</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td>49</td>
<td>0 0 0 1 1 1 3</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>51</td>
<td>0 0 1 0 1 5 5</td>
</tr>
<tr>
<td>52</td>
<td></td>
<td>53</td>
<td>0 1 0 0 0 0 8</td>
</tr>
<tr>
<td>54</td>
<td></td>
<td>55</td>
<td>0 1 0 0 1 9</td>
</tr>
<tr>
<td>56</td>
<td></td>
<td>57</td>
<td>0 1 0 1 1 B</td>
</tr>
<tr>
<td>58</td>
<td></td>
<td>59</td>
<td>0 1 1 0 1 D</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td>61</td>
<td>0 1 1 1 1 F</td>
</tr>
<tr>
<td>62</td>
<td></td>
<td>63</td>
<td>1 0 0 0 1 G</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>65</td>
<td>1 0 0 1 0 H</td>
</tr>
<tr>
<td>66</td>
<td></td>
<td>67</td>
<td>1 0 1 1 0 L</td>
</tr>
<tr>
<td>68</td>
<td></td>
<td>69</td>
<td>1 0 1 1 1 D</td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>71</td>
<td>1 1 0 0 0 P</td>
</tr>
<tr>
<td>72</td>
<td></td>
<td>73</td>
<td>1 1 0 0 1 R</td>
</tr>
<tr>
<td>74</td>
<td></td>
<td>75</td>
<td>1 1 1 0 1 T</td>
</tr>
<tr>
<td>76</td>
<td></td>
<td>77</td>
<td>1 1 1 1 1 &quot;BLANK&quot;</td>
</tr>
<tr>
<td>78</td>
<td></td>
<td>79</td>
<td>DIGIT SELECT</td>
</tr>
<tr>
<td>80</td>
<td></td>
<td>81</td>
<td>D7 D6 D5 DIGIT NUMBER</td>
</tr>
<tr>
<td>82</td>
<td></td>
<td>83</td>
<td>0 0 0 1 1</td>
</tr>
<tr>
<td>84</td>
<td></td>
<td>85</td>
<td>0 1 0 1 2</td>
</tr>
<tr>
<td>86</td>
<td></td>
<td>87</td>
<td>0 1 1 0 3</td>
</tr>
<tr>
<td>88</td>
<td></td>
<td>89</td>
<td>0 1 1 0 4</td>
</tr>
</tbody>
</table>

**5-729**
LOC  OBJ  LINE  SOURCE STATEMENT
90  ;***************************************************************
91  EQUATES
92  ;THE FOLLOWING CODE DESIGNATES "TIME" AS A VARIABLE THIS
93  ;ADJUSTS THE AMOUNT OF CYCLES THE TIMER COUNTS BEFORE
94  ;A TIMER INTERRUPT OCCURS AND REFRESHES THE DISPLAY APPROXIMATELY
95  ;50 TIMES PER SECOND
FFFI  96  TIME  EQU  OFH  ;TIMER VALUE 2 SMSEC
97  ;***************************************************************
98  ; INTERRUPT BRANCHING
99  ;THIS PORTION OF MEMORY IS DEDICATED FOR USE OF RESET AND
100 ; INTERRUPT BRANCHING WHEN THE INTERRUPTS ARE ENABLED THE
101 ; CODE AT THE FOLLOWING DESIGNATED SPOTS ARE EXECUTED WHEN A
102 ;RESET OR A INTERRUPT OCCURS
0000 103  ORG  0
0000 0409 104  JMP  START ;RESET
0002 00 105  NOP  ;
0003 0436 106  JMP  INPUT ;IBF INTERRUPT
0005 00 107  NOP  ;
0006 00 108  NOP  ;
0007 041D 109  JMP  DISPLAY ;TIMER INTERRUPT
110  ;***************************************************************
111  ; INITIALIZATION
112  ; THE FOLLOWING CODE SETS UP THE UPI-41 AND DISPLAY HARDWARE
113  ; INTO OPERATIONAL FORMAT. THE DISPLAY IS TURNED OFF, THE DISPLAY
114  ; MAP IS FILLED WITH "BLANK" CHARACTERS, THE TIMER SET AND THE
115  ; INTERRUPTS ARE ENABLED
116
0009 D5 117  START;  SEL  RB1
000A B00 118  ORL  P2. #0BH ;TURN DIGIT DRIVERS OFF
000C B03B 119  MOV  RO. #3BH ;DISPLAY MAP POINTER BOTTOM OF DISPLAY MAP
000E 23FF 120  BLKMAP. MOV  A. #OFFH ;FF= "BLANK"
0010 A0 121  MOV  @RO. A ;BLANK TO DISPLAY MAP
0011 18 122  INC  RO ;INCREMENT DISPLAY MAP POINTER
0012 F8 123  MOV  A. RO ;DISPLAY MAP POINTER TO ACCUMULATOR
0013 B20E 124  JSB  BLKMAP ;BLANK DISPLAY MAP TILL FILLED
0015 B00 125  MOV  R3. #0OH ;SET DIGIT COUNTER TO 0
0017 23F1 126  MOV  A. #TIME ;TIMER VALUE
0019 62 127  MOV  T. A ;LOAD TIMER
001A 55 128  STRT  T ;START TIMER
001B 25 129  EN  TCNT1 ;ENABLE TIMER INTERRUPT
001C 05 130  EN  I ;ENABLE IBF INTERRUPT
131  ;***************************************************************
132  ; USER PROGRAM
133 ; A USERS PROGRAM WOULD INITIALIZE AT THIS POINT THE FOLLOWING
134 ; CODE IS UN CONCLUDED WITH
135 ;SYNC CHARACTERS (OAAH) A CHECKSUM BYTE IMMEDIATELY PRECEEDS THE
136 ;FINAL SYNC WHEN READING, THE CONTROLLER*********************************
### APPLICATIONS

**ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0**

**PAGE 4**

<table>
<thead>
<tr>
<th>LOC OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010 D5</td>
<td>138</td>
<td>DISPLAY ROUTINE</td>
</tr>
<tr>
<td>0010 AF</td>
<td>139</td>
<td>THIS PORTION OF THIS PROGRAM IS AN INTERRUPT ROUTINE WHICH IS</td>
</tr>
<tr>
<td>0011 FB</td>
<td>140</td>
<td>ACTED UPON WHEN THE TIMER COUNT IS COMPLETED THE ROUTINE UPDATES</td>
</tr>
<tr>
<td>0022 433B</td>
<td>141</td>
<td>ONE DISPLAY DIGIT FROM THE DISPLAY MAP PER INTERRUPT SEQUENTIALLY,</td>
</tr>
<tr>
<td>0022 A8</td>
<td>142</td>
<td>THUS EIGHT TIMER INTERRUPTS WILL HAVE REFRESHED THE ENTIRE DISPLAY</td>
</tr>
<tr>
<td>0022 39</td>
<td>143</td>
<td>REGISTER BANK 1 IS SELECTED AND THE ACCUMULATOR IS SAVED UPON</td>
</tr>
<tr>
<td>0022 18</td>
<td>144</td>
<td>ENTERING THE ROUTINE ONCE THE DISPLAY HAS BEEN REFRESHED THE TIMER</td>
</tr>
<tr>
<td>0022 D207</td>
<td>145</td>
<td>IS RESET AND THE ACCUMULATOR AND PRE-INTERRUPT REGISTER BANK IS RESTORED</td>
</tr>
<tr>
<td>0022 9630</td>
<td>146</td>
<td>DISPLAY ROUTINE</td>
</tr>
<tr>
<td>0022 B000</td>
<td>147</td>
<td>REGISTER BANK 1</td>
</tr>
<tr>
<td>0022 3F1</td>
<td>148</td>
<td>SAVE ACCUMULATOR</td>
</tr>
<tr>
<td>0022 03F1</td>
<td>149</td>
<td>TURN DIGIT DRIVERS OFF</td>
</tr>
<tr>
<td>0022 0208</td>
<td>150</td>
<td>DIGIT COUNTER TO ACCUMULATOR</td>
</tr>
<tr>
<td>0022 03F1</td>
<td>151</td>
<td>OR&quot; TO GET DISPLAY MAP ADDRESS</td>
</tr>
<tr>
<td>0022 0460</td>
<td>152</td>
<td>DISPLAY MAP POINTER</td>
</tr>
<tr>
<td>0022 0460</td>
<td>153</td>
<td>GET CHARACTER FROM DISPLAY MAP</td>
</tr>
<tr>
<td>0022 0460</td>
<td>154</td>
<td>OUTPUT CHARACTER TO SEGMENT DRIVERS</td>
</tr>
<tr>
<td>0022 0460</td>
<td>155</td>
<td>DISPLAY MAP ON ADDRESS</td>
</tr>
<tr>
<td>0022 0460</td>
<td>156</td>
<td>GET CHARACTER FROM DISPLAY MAP</td>
</tr>
<tr>
<td>0022 0460</td>
<td>157</td>
<td>OUTPUT TO DIGIT DRIVERS</td>
</tr>
<tr>
<td>0022 0460</td>
<td>158</td>
<td>INCREMENT DIGIT COUNTER</td>
</tr>
<tr>
<td>0022 0460</td>
<td>159</td>
<td>CHECK IF AT LAST DIGIT</td>
</tr>
<tr>
<td>0022 0460</td>
<td>160</td>
<td>SETIME</td>
</tr>
<tr>
<td>0022 0460</td>
<td>161</td>
<td>RESET TIMER IN NOT LAST DIGIT</td>
</tr>
<tr>
<td>0022 0460</td>
<td>162</td>
<td>MOV R3, #00H</td>
</tr>
<tr>
<td>0022 0460</td>
<td>163</td>
<td>LOAD TIMER</td>
</tr>
<tr>
<td>0022 0460</td>
<td>164</td>
<td>START T</td>
</tr>
<tr>
<td>0022 0460</td>
<td>165</td>
<td>RESTORE ACCUMULATOR</td>
</tr>
<tr>
<td>0022 0460</td>
<td>166</td>
<td>RETR</td>
</tr>
<tr>
<td>0022 0460</td>
<td>167</td>
<td>RETURN</td>
</tr>
<tr>
<td>0022 0460</td>
<td>168</td>
<td>SELECT</td>
</tr>
</tbody>
</table>

**5-731**
INPUT CHARACTER AND DIGIT ROUTINE

THIS PORTION OF THE PROGRAM IS AN INTERRUPT ROUTINE WHICH IS ACTED UPON WHEN THE 18F BIT IS SET. THE ROUTINE GETS THE DISPLAY DATA WORD FROM THE DBB AND DEFINES BOTH THE DIGIT AND THE CHARACTER TO BE DISPLAYED. THIS IS DONE BY MEANS OF A CHARACTER LOOK-UP TABLE AND A DISPLAY MAP FOR DIGIT AND CHARACTER LOCATION. SPECIAL CONSIDERATION IS TAKEN FOR A DECIMAL POINT WHICH IS SIMPLY ADDED TO THE EXISTING CHARACTER IN THE DISPLAY MAP REGISTER.

INPUT ONCE THE DATA WORD HAS BEEN FULLY DEFINED THE ACCUMULATOR AND THE PRE-INTERRUPT REGISTER BANK IS RESTORED

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0036</td>
<td>D5</td>
<td>183</td>
<td>INPUT. SEL. RB1, REGISTER BANK 1</td>
</tr>
<tr>
<td>0037</td>
<td>AF</td>
<td>184</td>
<td>MOV R7, A, SAVE ACCUMULATOR</td>
</tr>
<tr>
<td>0039</td>
<td>AA</td>
<td>186</td>
<td>MOV R2, A, SAVE DATA WORD</td>
</tr>
<tr>
<td>003A</td>
<td>A7</td>
<td>187</td>
<td>SWAP A, DEFINE DIGIT LOCATION</td>
</tr>
<tr>
<td>003B</td>
<td>77</td>
<td>188</td>
<td>RR A</td>
</tr>
<tr>
<td>003C</td>
<td>5307</td>
<td>189</td>
<td>ANL A, #07H</td>
</tr>
<tr>
<td>003E</td>
<td>432B</td>
<td>190</td>
<td>ORL A, #38H</td>
</tr>
<tr>
<td>0040</td>
<td>AE</td>
<td>191</td>
<td>MOV R0, A, DIGIT LOCATION IN DIGIT POINTER</td>
</tr>
<tr>
<td>0041</td>
<td>FA</td>
<td>192</td>
<td>MOV A, R2, SAVED DATA WORD TO ACCUMULATOR</td>
</tr>
<tr>
<td>0042</td>
<td>531F</td>
<td>193</td>
<td>ANL A, #1FH, DEFINE CHARACTER LOOK-UP-TABLE LOC</td>
</tr>
<tr>
<td>0044</td>
<td>E3</td>
<td>194</td>
<td>MOV R3, A, GET CHARACTER</td>
</tr>
<tr>
<td>0045</td>
<td>AA</td>
<td>195</td>
<td>MOV R2, A, SAVE CHARACTER</td>
</tr>
<tr>
<td>0046</td>
<td>D37F</td>
<td>196</td>
<td>XRL A, #7FH, IS CHARACTER DECIMAL POINT</td>
</tr>
<tr>
<td>0048</td>
<td>C64E</td>
<td>197</td>
<td>DJNZ DPOINT</td>
</tr>
<tr>
<td>004A</td>
<td>FA</td>
<td>198</td>
<td>MOV A, R2, SAVED CHARACTER TO ACCUMULATOR</td>
</tr>
<tr>
<td>004B</td>
<td>A0</td>
<td>199</td>
<td>MOV R0, A, CHARACTER TO DISPLAY MAP</td>
</tr>
<tr>
<td>004C</td>
<td>0451</td>
<td>200</td>
<td>JMP RETURN</td>
</tr>
<tr>
<td>004B</td>
<td>FA</td>
<td>201</td>
<td>DPOINT MOV A, R2, SAVED CHARACTER TO ACCUMULATOR</td>
</tr>
<tr>
<td>004F</td>
<td>50</td>
<td>202</td>
<td>ANL A, #RO, &quot;AND&quot; WITH OLD CHARACTER</td>
</tr>
<tr>
<td>0050</td>
<td>A0</td>
<td>203</td>
<td>MOV R0, A, BACK TO DISPLAY MAP</td>
</tr>
<tr>
<td>0051</td>
<td>FF</td>
<td>204</td>
<td>RETURN MOV A, R7, RESTORE ACCUMULATOR</td>
</tr>
<tr>
<td>0052</td>
<td>93</td>
<td>205</td>
<td>RETR</td>
</tr>
</tbody>
</table>

**********************************************************************
APPLICATIONS

210 IF "ACRO ASSEMBLER, Y30 PAGE LOC DB!

200 :******************************************************************************

211 MEMORY IT IS USED TO DEFINE THE CORRECT LEVEL OF EACH SEGMENT

212 AND DECIMAL POINT FOR A SELECTED CHARACTER FROM THE INPUT ROUTINE

213 INVERSE LOGIC IS USED BECAUSE OF THE SPECIFIC DRIVER CIRCUITRY. THUS

214 A 1 ON A GIVEN SEGMENT MEANS IT IS OFF AND A 0 MEANS IT IS ON

215 ;

216

0300 217 ORG 300H ,DP G F E D C B A

0300 C0 21B CH0 DB 0C0H ,1 1 1 1 1 1 0 0 1

0301 F9 219 CH1 DB 0F9H ,1 1 1 1 1 1 0 0 1

0302 A4 220 CH2 DB 0A4H ,1 0 1 0 0 1 0 0

0303 B0 221 CH3 DB 0B0H ,1 0 1 1 0 0 0 0

0304 99 222 CH4 DB 099H ,1 0 0 1 1 0 0 1

0305 92 223 CH5 DB 092H ,1 0 0 1 0 0 1 0

0306 82 224 CH6 DB 082H ,1 0 0 0 0 0 1 0

0307 FB 225 CH7 DB 0FBH ,1 1 1 1 1 0 0 0

0308 B0 226 CH8 DB 0B0H ,1 0 0 0 0 0 0 0

0309 8B 227 CH9 DB 08BH ,1 1 1 1 0 0 0 0

030A BB 228 CHA DB 0BBH ,1 0 0 0 1 0 0 0

030B BE 229 CHB DB 0BETH ,1 0 0 0 0 0 0 1

030C C6 230 CHC DB 0C6H ,1 1 1 0 0 0 1 0

030D AF 231 CHD DB 0AFH ,1 1 1 1 1 1 0 0

030E B7 232 CHE DB 0B7H ,1 1 1 1 1 1 0 0

030F E1 233 CHF DB 0E1H ,1 1 1 1 1 1 1 1

209 : ++SEGMENTS+++ 0310 7F 234 CHG DB 07FH ,0 1 1 1 1 1 1 1

0311 C2 235 CHH DB 0C2H ,1 1 1 0 0 0 1 0

0312 B9 236 CHI DB 0B9H ,1 1 1 0 0 0 1 0

0313 FB 237 CHO DB 0FBH ,1 1 1 1 1 1 1 1

0314 E1 238 CHA DB 0E1H ,1 1 1 1 1 1 1 1

0315 C7 239 CHB DB 0C7H ,1 1 1 1 1 1 1 1

0316 AB 240 CHC DB 0A8H ,1 0 1 0 1 0 1 1

0317 A3 241 CHD DB 0A3H ,1 0 1 0 0 0 1 1

0318 BC 242 CHF DB 0BCH ,1 1 0 0 0 1 0 0

0319 AF 243 CHG DB 0AFH ,1 0 1 0 0 0 1 1

031A B7 244 CHH DB 0B7H ,1 0 0 0 0 1 1 1

0322 EF 245 CHI DB 0CH ,1 1 1 0 0 0 0 1

031C 91 246 CHJ DB 091H ,1 1 1 0 0 0 0 1

031D BF 247 CHKASH DB 0BFH ,1 1 1 1 1 1 1 1

031E FD 248 CHAPOS DB 0FFH ,1 1 1 1 1 1 1 0

031F 18 249 BLANK DB 0FFH ,1 1 1 1 1 1 1 1

250 :******************************************************************************

251 END

USER SYMBOLS

BLANK 031F BLKMAP 000E CH0 0300 CH1 0301 CH2 0302 CH3 0303 CH4 0304 CH5 0305

CH6 0306 CH7 0307 CH8 0308 CH9 0309 CHA 030A CHAPDS 031E CHB 030B CHC 030C

CHD 030D CHDASH 0310 CHDP 0310 CHE 030E CHF 030F CHG 0311 CHH 0312 CHJ 0313

CHO 0314 CHL 0313 CHN 0316 CHO 0319 CHT 031A CHU 031B

CHY 031C DISPLA 0010 DPINT 004E INPUT 0036 RETURN 0051 SETIME 0030 START 0009 TIME FFI:

ASSEMBLY COMPLETE, NO ERRORS

5-733
THIS PROGRAM USES THE UPI-41A AS A SENSOR MATRIX CONTROLLER.
IT HAS MONITORING CAPABILITIES OF UP TO 128 SENSORS. THE COORDINATE
AND SENSOR STATUS OF EACH DETECTED CHANGE IS AVAILABLE TO THE MASTER.
MICROPROCESSOR IN A SINGLE BYTE A 40x8 FIFO QUEUE IS PROVIDED FOR
DATA BUFFERING. BOTH HARDWARE OR POLLED INTERRUPT METHODS CAN BE USED
TO NOTIFY THE MASTER OF A DETECTED SENSOR CHANGE.

**REGISTER DEFINITIONS**

- **R0**: MATRIX MAP POINTER
- **R1**: FIFO POINTER
- **R2**: SCAN ROW SELECT
- **R3**: COLUMN COUNTER
- **R4**: FIFO-IN
- **R5**: FIFO-OUT
- **R6**: CHANGE WORD
- **R7**: COMPARE

**PORT PIN DEFINITIONS**

- **P0-7**: COLUMN LINE INPUTS
- **P0-3**: ROW SELECT OUTPUTS
- **P4**: FIFO NOT EMPTY INTERRUPT
- **P5**: OBF INTERRUPT
- **P6-7**: NOT USED
APPLICATIONS

ISIS-II MCS-48/UP-41 MACRO ASSEMBLER. V3 0

LOC OBJ LINE SOURCE STATEMENT

41 .*****************************************************************************
42 .
43 . CHANGE WORD BIT DEFINITION
44 .
45 . BIT FUNCTION
46 . ---
47 . DO-6 SENSOR COORDINATE
48 . D7 SENSOR STATUS
49 .
50 .*****************************************************************************
51 .
52 . STATUS REGISTER BIT DEFINITION
53 .
54 . BIT FUNCTION
55 . ---
56 . DO DBF
57 . D1-3 IBF, FO, FI (NOT USED)
58 . D4 FIFO NOT EMPTY
59 . DS-7 USED DEFINED (NOT USED)
60 .
61 .*****************************************************************************
62 .
63 . EQUATES
64 .
65 . THE FOLLOWING CODE DESIGNATES THREE VARIABLES. SCANTM, FIFOBA
66 . AND FIFOA. SCANTM ADJUSTS THE LENGTH OF A DELAY BETWEEN
67 . SCANNING SWITCH THIS SIMULATES DEBOUNCE FUNCTIONS FIFOA
68 . IS THE BOTTOM ADDRESS OF THE FIFO. FIFOA IS THE TOP ADDRESS
69 . OF THE FIFO THIS MAKES IT POSSIBLE TO HAVE A FIFO 3 TO 40
70 . BYTES IN LENGTH
71 .
72 .*****************************************************************************
73 .
74 SCANTM EQU 0FH .SCAN TIME ADJUST
75 FIFOBA EQU 08H .FIFO BOTTOM ADDRESS
76 FIFOA EQU 2FH .FIFO TOP ADDRESS
77 
78 *EJECT

5-735
APPLICATIONS

ISIs

LOC OBJ

PAGE 3

SOURCE STATEMENT

79 .***************************************************************
B0 ,
B1 ,
INITIALIZATION
B2 ,
B3 .THE PROGRAM STARTS AT THE FOLLOWING CODE UPON RESET WITHIN
B4 .THIS INITIALIZATION SECTION THE REGISTERS THAT MAINTAIN THE MATRIX
B5 .MAP,FIFO AND ROW SCANING ARE SET UP. PORT 1 IS SET HIGH FOR USE
B6 .AS AN INPUT PORT FOR THE COLUMN STATUS, BIT 4 OF STATUS REGISTER IS
B7 .WRITTEN TO CONVEY A FIFO EMPTY CONDITION. THE INITIAL COLUMN STATUS
B8 .OF ALL THE ROWS IN THE SENSOR MATRIX IS THEN READ INTO THE MATRIX
B9 .MAP ONCE THE MATRIX MAP IS FILLED THE OBF INTERRUPT (PORT 2-4) IS
B0 .ENABLED
B1 ,
B2 .***************************************************************
B3 ,

0000 0000 BB3F 94 ORG 0
0000 0000 8A0F 95 INITMX MOV R0,#0FH .MATRIX MAP POINTER REGISTER, TOP ADDRESS
0000 0000 8C0B 96 MOV R2,#0FH .SCAN ROW SELECT REGISTER, TOP ROW
0000 0000 8D2F 97 MOV R4,#FIFOBA .FIFO INPUT ADDRESS REGISTER, BOTTOM OF FIFO
0000 0000 8BFF 98 MOV R5,#FIFOBA .FIFO OUTPUT ADDRESS REGISTER, TOP OF FIFO
0000 0000 2300 99 ORL P1,#OFFH .INITIALIZE PORT 1 HIGH FOR INPUTS
0000 0000 90 100 MOV A,#00H .INITIALIZE STATUS REGISTER, FIFO EMPTY
0000 0000 90 101 MOV STS,A .WRITE TO STATUS REGISTER, BITS 4-7
0000 0000 90 102 FILLMX MOV A,R2 .SCAN ROW SELECT TO ACCUMULATOR
0000 0000 90 103 OUTL P2,A .OUTPUT SCAN ROW SELECT TO PORT 2
0000 0000 90 104 IN A,P1 .INPUT COLUMN STATUS PORT 1
0000 0000 90 105 MOV R0,A .LOAD MATRIX MAP WITH COLUMN STATUS
0000 0000 90 106 MOV A,R2 .CHECK SCAN ROW SELECT REGISTER VALUE FOR 0
0000 0000 90 107 CJZ OBFINT .IF 0 ENABLE DBF INTERRUPT
0000 0000 90 108 DEC R0 .DECREMENT TO NEXT MATRIX MAP ADDRESS
0000 0000 90 109 DEC R2 .DECREMENT TO SCAN NEXT ROW
0000 0000 90 10A JMP FILLMX .FILL NEXT MATRIX MAP ADDRESS
0000 0000 90 10B OBFINT MOV R2,#0H .BIT 4 HIGH IN ROW SCAN SELECT REGISTER
0000 0000 90 10C MOV A,R2 .ROW SCAN SELECT VALUE TO ACCUMULATOR
0000 0000 90 10D FILLMX MOV A,R2 .LOAD ADDRESS TO SCAN PORT 1
0000 0000 90 10E OUTL P2,A .INITIALIZE PORT 2. BIT 4 FOR "EN FLAGS"
0000 0000 90 10F EN FLAGS .ENABLE DBF INTERRUPT PORT 2. BIT 4
0000 0000 90 110 EJECT .

5-736
SCAN AND COMPARE

THE FOLLOWING CODE IS THE SCAN AND COMPARE SECTION OF THE PROGRAM.
UPON ENTERING THIS SECTION A CHECK IS MADE TO SEE IF THE ENTIRE MATRIX
HAS BEEN SCANNED. IF SO THE REGISTERS THAT MAINTAIN THE MATRIX MAP AND ROW
BITS 0-3 FOR SCANNING AND BITS 4 AND 5 FOR THE EXTERNAL INTERRUPTS. THUSLY
ALL USAGE OF THE REGISTERS IS DONE BY LOGICALLY MASKING IT SO AS TO ONLY
AFFECT THE FUNCTION DESIRED. ONCE THE REGISTERS ARE RESET, ONE ROW OF THE
SENSOR MATRIX IS SCANNED. A DELAY IS EXECUTED TO ADJUST FOR SCAN TIME
(Debounce) A BYTE OF COLUMN STATUS IS THEN READ INTO THE MATRIX MAP
AT THE TIME THE NEW COLUMN STATUS IS COMPARED TO THE OLD. THE RESULT IS
STORED IN THE COMPARE REGISTER. THE PROGRAM IS THEN ROUTED ACCORDING TO
WHETHER OR NOT A CHANGE WAS DETECTED.

LOC OBJ
LINE
117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158
LOC OBJ
PAGE 4
SOURCE STATEMENT
BASED MOV A.R2 SCAN ROW SELECT TO ACCUMULATOR
ANL A.0FH CHECK FOR 0 SCAN VALUE ONLY. NOT INTERRUPT
JZ RSETRG IF 0 RESET REGISTERS
DEC R0 DECREMENT MATRIX MAP POINTER
DEC R2 DECREMENT SCAN ROW SELECT
JMP SCANMX SCAN MATRIX
RSETRG MOV R0.3FH RESET MATRIX MAP PTORIZER. TOP ADDRESS
MOV A.R2 SCAN ROW SELECT TO ACCUMULATOR
MOV A.0FH RESET SCAN ROW SELECT. NO INTERRUPT CHANGE
INC R2.A SCAN ROW SELECT REGISTER
SCANMX MOV A.R2 SCAN ROW SELECT TO ACCUMULATOR
OUTL P2.A OUTPUT SCAN ROW SELECT TO PORT 2
MOV R3.4SCANM SET DELAY FOR OUTPUT SCAN TIME
DJNZ R3.DELAY2 DELAY
IN A.P1 INPUT COLUMN STATUS FROM PORT 1 TO ACCUMULATOR
XCH A.0RO STORE NEW COLUMN STATUS SAVE OLD IN ACCUMULATOR
XRL A.0R0 COMPARE OLD WITH NEW COLUMN STATUS
MOV R7.A SAVE COMPARE RESULT IN COMPARE REGISTER
DJNZ CHFPU IF THE SAME, CHECK IF FIFO IS FULL
$EJECT
159. ...........................................................................................................
160. CHANGE WORD ENCODING
161. THE FOLLOWING CODE IS THE CHANGE WORD ENCODING SECTION. THIS
162. SECTION IS ONLY EXECUTED IF A CHANGE WAS DETECTED. THE COLUMN COUNTER
163. IS SET AND DECREMENTED TO DESIGNATE EACH OF THE 8 COLUMNS. THE COMPARE
164. REGISTER IS LOOKED AT ONEBIT AT A TIME TO FIND THE EXACT LOCATION OF
165. THE CHANGE(S) WHEN A CHANGE IS FOUND IT IS ENCODED BY GIVING IT A
166. COORDINATE FOR ITS LOCATION. THIS IS DONE BY COMBINING THE PRESENT VALUE
167. IN THE ROW SCAN SELECT REGISTER AND THE COLUMN COUNTER. THE ACTUAL STATUS
168. OF THAT SENSOR IS ESTABLISHED BY LOOKING AT THE CORRESPONDING BYTE IN
169. THE MATRIX MAP. THIS STATUS IS COMBINED WITH THE COORDINATE TO ESTABLISH
170. THE CHANGE WORD. THE CHANGE WORD IS THEN STORED IN THE CHANGE WORD REGISTER
171. 172. 173. 174. 175.

003B B80B 176. MOV R3, #08H .SET COLUMN COUNTER REGISTER TO 8
003A C8 177. RRL, R3 .DECREMENT COLUMN COUNTER
003B F0 178. MOV A, #R0 .COLUMN STATUS TO ACCUMULATOR
003C 77 179. RR A .ROTATE COLUMN STATUS RIGHT
003D 90 180. MOV #R0, A .ROTATED COLUMN STATUS BACK TO MATRIX MAP
003E FF 181. MOV A, R7 .COMPARE REGISTER VALUE TO ACCUMULATOR
003F 77 182. RR A .ROTATE COMPARE VALUE RIGHT
0040 AF 183. MOV R7, A .ROTATED COMPARE VALUE TO COMPARE REGISTER
0041 F245 184. JB7 ENCODE .TEST BIT 7. IF CHANGE DETECTED ENCODE CHANGE WORD
0042 0469 185. JMP CHFFUL .IF NO CHANGE IS DETECTED CHECK FOR FIFO FULL
0043 04F5 186. ENCODE MOV A, R2 .SCAN ROW SELECT TO ACCUMULATOR 0000XXXX
0044 550F 187. ANL A, #0FH .ROTATE ONLY SCAN VALUE
0045 E7 188. RL A .ROTATE LEFT 0000XXX0
0044 E7 189. RL A .ROTATE LEFT 0000XXX0
0042 88 190. ORL A, R3 .ESTABLISH MATRIX COORDINATE 0XXXXXXX
0043 AE 191. MOV A, R6 .SAVE COORDINATE IN CHANGE WORD REGISTER
0044 F0 194. MOV A, #R0 .COLUMN STATUS FROM MATRIX MAP TO ACCUMULATOR
0044 5380 195. ANL A, #B8H .0 ALL BITS BUT BIT 7
0045 4E 196. ORL A, R6 .(OR) SENSOR STATUS WITH COORDINATE FOR COMPLETED CHANGE WORD
0046 AE 197. MOV A, R6 .SAVE CHANGE WORD XXXXXX
198.
199. **EJECT
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0052</td>
<td>FC</td>
<td>217</td>
<td>LOADF MV A.R4 FIFO INPUT ADDRESS TO ACCUMULATOR</td>
</tr>
<tr>
<td>0053</td>
<td>A9</td>
<td>218</td>
<td>MV R1.A FIFO POINTER USED FOR INPUT</td>
</tr>
<tr>
<td>0054</td>
<td>FE</td>
<td>219</td>
<td>MV R1.A CHANGE WORD TO ACCUMULATOR</td>
</tr>
<tr>
<td>0055</td>
<td>A1</td>
<td>220</td>
<td>MV @R1.A LOAD FIFO AT FIFO INPUT ADDRESS</td>
</tr>
<tr>
<td>0056</td>
<td>230</td>
<td>221</td>
<td>STATNE MV A.#10H BIT 4 FOR FIFO NOT EMPTY</td>
</tr>
<tr>
<td>0058</td>
<td>90</td>
<td>222</td>
<td>MV STA.L WRITE TO STATUS REGISTER FIFO NOT EMPTY</td>
</tr>
<tr>
<td>0059</td>
<td>BA20</td>
<td>223</td>
<td>INTRH DRL P2.#20H FIFO NOT EMPTY INTERRUPT PORT 2-5 HIGH</td>
</tr>
<tr>
<td>005B</td>
<td>FA</td>
<td>224</td>
<td>MV A.R2 ROW SCAN SELECT TO ACCUMULATOR</td>
</tr>
<tr>
<td>005C</td>
<td>4320</td>
<td>225</td>
<td>DRL A.#20H SAVE INTERRUPT NO CHANGE TO SCAN VALUE</td>
</tr>
<tr>
<td>005E</td>
<td>AA</td>
<td>226</td>
<td>MV R2.A ROW SCAN SELECT REGISTER</td>
</tr>
<tr>
<td>005F</td>
<td>232F</td>
<td>227</td>
<td>ADJFIN MV A.#FIFOTA FIFO TOP ADDRESS TO ACCUMULATOR</td>
</tr>
<tr>
<td>0061</td>
<td>DC</td>
<td>228</td>
<td>XR R A.R4 COMPARE WITH CURRENT FIFO INPUT ADDRESS</td>
</tr>
<tr>
<td>0062</td>
<td>C667</td>
<td>229</td>
<td>JZ RSFFIN IF THE SAME RESET FIFO INPUT REGISTER</td>
</tr>
<tr>
<td>0064</td>
<td>1C</td>
<td>230</td>
<td>INC R4 NEXT FIFO INPUT ADDRESS</td>
</tr>
<tr>
<td>0065</td>
<td>0469</td>
<td>231</td>
<td>JMP CHFUL CHECK FIFO FULL</td>
</tr>
<tr>
<td>0067</td>
<td>B808</td>
<td>232</td>
<td>RSFFIN MV R4.#FIFOBA RESET FIFO INPUT REGISTER BOTTOM OF FIFO</td>
</tr>
<tr>
<td>0069</td>
<td>FC</td>
<td>233</td>
<td>CHFUL MV A.R4 FIFO INPUT ADDRESS TO ACCUMULATOR</td>
</tr>
<tr>
<td>006A</td>
<td>DD</td>
<td>234</td>
<td>XR A.R5 COMPARE INPUT WITH OUTPUT FIFO ADDRESS</td>
</tr>
<tr>
<td>006B</td>
<td>967D</td>
<td>235</td>
<td>JNZ CHCNTR IF NOT SAME CHECK COLUMN COUNTER VALUE</td>
</tr>
<tr>
<td>006D</td>
<td>B8D0</td>
<td>236</td>
<td>CHBF1 JDBF CHBF1 IF OBF IS 1 THEN CHECK OBF</td>
</tr>
<tr>
<td>006F</td>
<td>232F</td>
<td>237</td>
<td>ADJFOT MV A.#FIFOBA FIFO TOP ADDRESS TO ACCUMULATOR</td>
</tr>
<tr>
<td>0071</td>
<td>DD</td>
<td>238</td>
<td>XR A.R5 COMPARE TOP TO OUTPUT FIFO ADDRESS</td>
</tr>
<tr>
<td>0072</td>
<td>C677</td>
<td>239</td>
<td>JZ RSFFDT IF THE SAME RESET FIFO OUTPUT REGISTER</td>
</tr>
<tr>
<td>0074</td>
<td>1D</td>
<td>240</td>
<td>INC R5 NEXT FIFO OUTPUT ADDRESS</td>
</tr>
<tr>
<td>0075</td>
<td>0479</td>
<td>241</td>
<td>JMP LOADDB LOAD DBB</td>
</tr>
<tr>
<td>0077</td>
<td>B088</td>
<td>242</td>
<td>RSFFGT MV R5.#FIFOBA RESET FIFO OUTPUT ADDRESS TO BOTTOM OF FIFO</td>
</tr>
<tr>
<td>0079</td>
<td>FD</td>
<td>243</td>
<td>LOADDW MV A.R5 OUTPUT FIFO ADDRESS TO ACCUMULATOR</td>
</tr>
<tr>
<td>007A</td>
<td>A9</td>
<td>244</td>
<td>MV R1.A FIFO POINTER USED FOR OUTPUT</td>
</tr>
<tr>
<td>007B</td>
<td>F1</td>
<td>245</td>
<td>MV A.#R1 CHANGE WORD TO ACCUMULATOR</td>
</tr>
<tr>
<td>007C</td>
<td>02</td>
<td>246</td>
<td>DBB.A CHANGE WORD TO DBB</td>
</tr>
<tr>
<td>007D</td>
<td>FB</td>
<td>247</td>
<td>CHCNTR MV A.R3 COLUMN COUNTER TO ACCUMULATOR</td>
</tr>
<tr>
<td>007E</td>
<td>963A</td>
<td>248</td>
<td>JNZ RRLOK IF NOT 0 FINISH CHANGE WORD ENCODING</td>
</tr>
<tr>
<td>0080</td>
<td>2308</td>
<td>249</td>
<td>CHFFEM MV A.#FIFOBA FIFO BOTTOM ADDRESS TO ACCUMULATOR</td>
</tr>
</tbody>
</table>

5-739
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0082 DC</td>
<td>252</td>
<td>XRL</td>
<td>A. R4, COMPARE FIFO INPUT ADDRESS WITH FIFO BOTTOM ADDRESS</td>
</tr>
<tr>
<td>0083 C68C</td>
<td>253</td>
<td>JZ</td>
<td>ADFJEM, IF THE SAME, ADJUST TO CHECK FOR FIFO EMPTY</td>
</tr>
<tr>
<td>0083 FC</td>
<td>254</td>
<td>MOV</td>
<td>A. R4, FIFO INPUT ADDRESS TO ACCUMULATOR</td>
</tr>
<tr>
<td>0086 07</td>
<td>255</td>
<td>DEC</td>
<td>A. R3, DECREMENT FIFO INPUT ADDRESS IN ACCUMULATOR</td>
</tr>
<tr>
<td>0087 DD</td>
<td>256</td>
<td>XRL</td>
<td>A. R3, COMPARE INPUT TO OUTPUT FIFO ADDRESSES</td>
</tr>
<tr>
<td>0088 C691</td>
<td>257</td>
<td>JZ</td>
<td>STATMT, IF SAME, WRITE STATUS REGISTER FOR FIFO EMPTY</td>
</tr>
<tr>
<td>008A 049C</td>
<td>258</td>
<td>JMP</td>
<td>CHDBF2, CHECK DBF</td>
</tr>
<tr>
<td>008C 232F</td>
<td>259</td>
<td>ADFJEM MOV</td>
<td>A. #FIFOTA, FIFO TOP ADDRESS TO ACCUMULATOR</td>
</tr>
<tr>
<td>008E DD</td>
<td>260</td>
<td>XRL</td>
<td>A. R5, COMPARE TOP TO OUTPUT FIFO ADDRESS</td>
</tr>
<tr>
<td>008F 969C</td>
<td>261</td>
<td>JN I</td>
<td>CHDBF2, IF NOT SAME THEN FIFO IS NOT EMPTY, CHECK DBF</td>
</tr>
<tr>
<td>0091 2300</td>
<td>262</td>
<td>STATMT MOV</td>
<td>A. #00H, CLEAR BIT 0 FOR FIFO EMPTY</td>
</tr>
<tr>
<td>0093 90</td>
<td>263</td>
<td>MOV</td>
<td>STS. A, WRITE TO STATUS REGISTER</td>
</tr>
<tr>
<td>0094 9ADF</td>
<td>264</td>
<td>INTRLO ANL</td>
<td>P. #0DFH, FIFO EMPTY, INTERRUPT PORT 2-5 LOW</td>
</tr>
<tr>
<td>0096 FA</td>
<td>265</td>
<td>MOV</td>
<td>A. R2, SCAN ROW SELECT TO ACCUMULATOR</td>
</tr>
<tr>
<td>0097 53DF</td>
<td>266</td>
<td>ANL</td>
<td>A. #0DFH, SAVE INTERRUPT, NO CHANGE TO SCAN VALUE</td>
</tr>
<tr>
<td>0099 AA</td>
<td>267</td>
<td>MOV</td>
<td>R. R2.A, SCAN ROW SELECT REGISTER</td>
</tr>
<tr>
<td>009A 041D</td>
<td>268</td>
<td>JMP</td>
<td>ADJREG, ADJUST REGISTERS</td>
</tr>
<tr>
<td>009C 861D</td>
<td>269</td>
<td>CHDBF2 JMP</td>
<td>ADJREG, IF OBF=1 THEN ADJUST REGISTERS</td>
</tr>
<tr>
<td>009E 046F</td>
<td>270</td>
<td>JMP</td>
<td>ADJFOT, ADJUST FIFO OUT ADDRESS TO LOAD DBBOUT</td>
</tr>
<tr>
<td>271</td>
<td></td>
<td></td>
<td>END</td>
</tr>
</tbody>
</table>

**USER SYMBOLS**

- ADFJEM: 008C
- ADJFIN: 005F
- ADJFOT: 006F
- ADJREG: 001D
- CHCNTR: 007D
- CHFFEM: 0080
- CHFFUL: 0069
- CHDBF1: 006D
- CHDBF2: 009C
- DELAY2: 0030
- ENCODE: 0045
- FIFOBA: 000B
- FIFOA: 002F
- FILLMX: 000D
- INITMX: 0000
- INTRH1: 0039
- INTRL0: 0004
- LOADDB: 0079
- LOADDF: 0032
- DBFIN: 001B
- RRLDOK: 003A
- RBETRG: 0026
- RSFIN: 0067
- RSFFOT: 0077
- SCANMX: 002C
- SCANTM: 000F
- STATMT: 0091
- STATNE: 0056

**ASSEMBLY COMPLETE, NO ERRORS**
APPLICATIONS

PROGRAMMABLE KEYBOARD INTERFACE

- Simultaneous Keyboard and Display Operations
- Interface Signals for Contact and Capacitive Coupled Keyboards
- 128-Key Scanning Logic
- 10.7msec Matrix Scan Time for 128 Keys and 6MHz Clock
- Eight Character Keyboard FIFO

This application is a general purpose programmable keyboard and display interface device designed for use with 8-bit microprocessors like the MCS-80 and MCS-85. The keyboard portion can provide a scanned interface to 128-key contact or capacitive-coupled keyboards. The keys are fully debounced with N-key rollover and programmable error generation on multiple new key closures. Keyboard entries are stored in an 8-character FIFO with overrun status indication when more than 8 characters are entered. Key entries set an interrupt request output to the master CPU.

The display portion of the UPI-41A provides a scanned display interface for LED, incandescent and other popular display technologies. Both numeric displays and simple indicators may be used. The UPI-41A has a 16×4 display RAM which can be

Figure 1. Pin Configuration  
Figure 2. Block Diagram
APPLICATIONS

loaded or interrogated by the CPU. Both right entry calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto increment of the display RAM address.

ORDERING INFORMATION:
This part may be ordered as an 8041A with ROM code number 8278. The source code is available through Insite.

Throughout this application of the UPI-41A, it will be referred to by its ROM code number, 8278. The 8278 is packaged in a 40-pin DIP. The following is a brief functional description of each pin.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Pin. No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0-D7</td>
<td>12-19</td>
<td>I/O</td>
<td>Data Bus: Three-state, bi-directional data bus lines used to transfer data and commands between the CPU and the 8278.</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>I</td>
<td>Write: Write strobe which enables the master CPU to write data and commands between the CPU and the 8278.</td>
</tr>
<tr>
<td>RD</td>
<td>8</td>
<td>I</td>
<td>Read: Read strobe which enables the master CPU to read data and status from the 8278 internal registers.</td>
</tr>
<tr>
<td>CS</td>
<td>6</td>
<td>I</td>
<td>Chip Select: Chip select input used to enable reading and writing to the 8278.</td>
</tr>
<tr>
<td>A0</td>
<td>9</td>
<td>I</td>
<td>Control/Data: Address input used by the CPU to indicate control or data.</td>
</tr>
<tr>
<td>RESET</td>
<td>4</td>
<td>I</td>
<td>Reset: A low signal on this pin resets the 8278.</td>
</tr>
<tr>
<td>X1, X2</td>
<td>2,3</td>
<td>I</td>
<td>Freq. Reference Inputs: Inputs for crystal, L-C or external timing signal to determine internal oscillator frequency.</td>
</tr>
<tr>
<td>IRQ</td>
<td>23</td>
<td>O</td>
<td>Interrupt Request: Interrupt Request Output to the master CPU. In the keyboard mode the IRQ line goes low with each FIFO read and returns high if there is still information in the FIFO or an ERROR has occurred.</td>
</tr>
<tr>
<td>M0-M6</td>
<td>27-33</td>
<td>O</td>
<td>Matrix Scan Lines: Matrix scan outputs. These outputs control a decoder which scans the key matrix columns and the 16 display digits. Also, the Matrix scan outputs are used to multiplex the return lines from the key matrix.</td>
</tr>
<tr>
<td>RL</td>
<td>1</td>
<td>I</td>
<td>Keyboard Return Line: Input from the multiplexer which indicates whether the key currently being scanned is closed.</td>
</tr>
<tr>
<td>HYS</td>
<td>22</td>
<td>O</td>
<td>Hysteresis: Hysteresis output to the analog detector. (Capacitive keyboard configuration). A &quot;0&quot; means the key currently being scanned has already been recorded.</td>
</tr>
<tr>
<td>KCL</td>
<td>34</td>
<td>O</td>
<td>Key Clock: Key Clock output to the analog detector (capacitive keyboard configuration) used to reset the detector before scanning a key.</td>
</tr>
<tr>
<td>SYNC</td>
<td>11</td>
<td>O</td>
<td>Output Clock: High frequency (400 kHz) output signal used in the key scan to detect a closed key (capacitive keyboard configuration).</td>
</tr>
<tr>
<td>B0-B3</td>
<td>35-38</td>
<td>O</td>
<td>Display Outputs: These four lines contain binary coded decimal display information synchronized to the keyboard column scan. The outputs are for multiplexed digital displays.</td>
</tr>
<tr>
<td>ERROR</td>
<td>24</td>
<td>O</td>
<td>Error Signal: This line is high whenever two new key closures are detected during a single scan or when too many characters are entered into the keyboard FIFO. It is reset by a system RESET pulse or by a &quot;1&quot; input on the CLR pin or by the CLEAR ERROR command.</td>
</tr>
<tr>
<td>CLR</td>
<td>39</td>
<td>I</td>
<td>Clear Error: Input used to clear an ERROR condition in the 8278.</td>
</tr>
<tr>
<td>BP</td>
<td>21</td>
<td>O</td>
<td>Tone Enable: Tone enable output. This line is high for 10ms following a valid key closure; it is set high and remains high during an ERROR condition.</td>
</tr>
<tr>
<td>VCC, VDD</td>
<td>40,26</td>
<td>I</td>
<td>Power: +5 volt power input: +5V ± 10%.</td>
</tr>
<tr>
<td>GND</td>
<td>20,7</td>
<td>I</td>
<td>Ground: Signal ground.</td>
</tr>
</tbody>
</table>

PRINCIPLES OF OPERATION
The following is a description of the major elements of the Programmable Keyboard/Display interface device. Refer to the block diagram in Figure 1.

I/O Control and Data Buffers
The I/O control section uses the CS, A0, RD, and WR lines to control data flow to and from the various internal registers and buffers (see Table 2). All data flow to and from the 8278 is enabled by CS. The 8-bits of information being transferred by the CPU is identified by AO. A logic one means information is command or status. A logic zero means the information is data. RD and WR determine the direction of data flow through the Data Bus Buffer (DBB). The
DBB register is a bi-directional 8-bit buffer register which connects the internal 8278 bus buffer register to the external bus. When the chip is not selected (CS = 1) the DBB is in the high impedance state. The DBB acts as an input when (RD, WR, CS) = (1, 0, 0) and an output when (RD, WR, CS) = (0, 1, 0).

Table 2. I/O Control and Data Buffers

<table>
<thead>
<tr>
<th>CS</th>
<th>A0</th>
<th>WR</th>
<th>RD</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Read DBB Data</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Read STATUS</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Write Data to DBB</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Write Command to DBB</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Disable 8278 Bus, High Impedance</td>
</tr>
</tbody>
</table>

Scan Counter

The scan counter provides the timing to scan the keyboard and display. The four MSB's (M3-M6) scan the display digits and provide column scan to the keyboard via a 4 to 16 decoder. The three LSB's (M0-M2) are used to multiplex the row return lines into the 8278.

Keyboard Debounce and Control

The 8278 system configuration is shown in Figure 3. The rows of the matrix are scanned and the outputs are multiplexed by the 8278. When a key closure is detected, the debounce logic waits about 12 msec to check if the key remains closed. If it does, the address of the key in the matrix is transferred into a FIFO buffer.

FIFO and FIFO Status

The 8278 contains an 8x8 FIFO character buffer. Each new entry is written into a successive FIFO location and each is then read out in the order of entry. A FIFO status register keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or key entries will be recognized as an error. The status can be read by a RD with CS low and A0 high. The status logic also provides an IRQ signal to the master processor whenever the FIFO is not empty.

Display Address Registers and Display RAM

The Display Address registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles being displayed. The read/write addresses are programmed by CPU command. They also can be set to auto increment after each read or write. The display RAM can be directly read by the CPU after the correct mode and address is set. Data entry to the display can be set to either left or right entry.

Figure 3. System Configuration for Capacitive-Coupled Keyboard
APPLICATIONS

Figure 4. System Configuration for Contact Keyboard

COMMANDS

The 8278 operating mode is programmed by the master CPU using the A0, WR and D0-D7 inputs as shown below:

Where the mode set bits are defined as follows:
K—the keyboard mode select bit
0—normal key entry mode
1—special function mode: Entry on key closure and on key release
D—the display entry mode select bit
0—left display entry
1—right display entry
I—the interrupt request (IRQ) output enable bit.
0—enable IRQ output
1—disable IRQ output
E—the error mode select bit
0—error on multiple key depression
1—no error on multiple key depression
N—the number of display digits select
0—16 display digits
1—8 display digits

The master CPU presents the proper command on the D0-D7 data lines with A0 =1 and then sends a WR pulse. The command is latched by the 8278 on the rising edge of the WR and is decoded internally to set the proper operating mode. See the 8041A/8741A data sheet for timing details.

Command Summary

KEYBOARD/DISPLAY MODE SET

Where the mode set bits are defined as follows:
K—the keyboard mode select bit
0—normal key entry mode
1—special function mode: Entry on key closure and on key release
D—the display entry mode select bit
0—left display entry
1—right display entry
I—the interrupt request (IRQ) output enable bit.
0—enable IRQ output
1—disable IRQ output
E—the error mode select bit
0—error on multiple key depression
1—no error on multiple key depression
N—the number of display digits select
0—16 display digits
1—8 display digits

NOTE:
The default mode following a RESET input is all bits zero:

READ FIFO COMMAND

CODE

READ DISPLAY COMMAND

NOTE:
The default mode following a RESET input is all bits zero:

CODE

5-744
APPLICATIONS

Where AI indicates Auto Increment and A3-A0 is the address of the next display character to be read out.

\[
\text{AI} = 1 \quad \text{AUTO increment} \\
\text{AI} = 0 \quad \text{no AUTO increment}
\]

**WRITE DISPLAY COMMAND**

\[
\text{CODE} \quad 1 \quad 0 \quad 0 \quad \text{AI} \quad \text{A3} \quad \text{A2} \quad \text{A1} \quad \text{A0}
\]

Where AI indicates Auto Increment and A3-A0 is the address of the next display character to be written.

**CLEAR/BLANK COMMAND**

\[
\text{CODE} \quad 1 \quad 0 \quad 1 \quad \text{UD} \quad \text{BD} \quad \text{CD} \quad \text{CF} \quad \text{CE}
\]

Where the command bits are defined as follows:

- **CE** = Clear ERROR
- **CF** = Clear FIFO
- **CD** = Clear Display to all High
- **BD** = Blank Display to all High
- **UD** = Unblank Display

The display is cleared and blanked following a Reset.

**Status Read**

The status register in the 8278 can be read by the master CPU using the A0, RD, and D0-D7 inputs as shown below:

\[
\text{A0, CS} = \text{VALID} \\
\text{RD} \\
\text{D0-D7} = \text{VALID}
\]

The 8278 places 8-bits of status information on the D0-D7 lines following (A0, CS, RD) = 1, 0, 0 inputs from the master.

**Status Format**

\[
S_3 \quad S_2 \quad S_1 \quad S_0 \quad B \quad \text{KE} \quad \text{IBF} \quad \text{OFB}
\]

Where the status bits are defined as follows:

- **IBF** = Input Buffer Full Flag
- **OBF** = Output Buffer Full Flag
- **KE** = Keyboard Error Flag (multiple depression)
- **B** = BUSY Flag
- **S3-S0** = FIFO Status

**STATUS DESCRIPTION**

The S3-S0 status bits indicate the number of entries (0 to 8) in the 8-level FIFO. A FIFO overrun will lock status at 1111. The overrun condition will prevent further key entries until cleared.

A multiple key closure error will set the KE flag and prevent further key entries until cleared.

The IBF and OBF flags signify the status of the 8278 data buffer registers used to transfer information (data, status or commands) to and from the master CPU.

The IBF flag is set when the master CPU writes Data or Commands to the 8278. The IBF flag is cleared by the 8278 during its response to the Data or Command.

The OBF flag is set when the 8278 has output data ready for the master CPU. This flag is cleared by a master CPU Data READ.

The Busy flag in the status register is used as a LOCKOUT signal to the master processor during response to any command or data write from the master.

The master must test the Busy flag before each read (during a sequence) to be sure that the 8278 is ready with valid DATA.

The ERROR and TONE outputs from the 8278 are set high for either type of error. Both types of error are cleared by the CLR input, by the CLEAR ERROR command, or by a reset. The FIFO and Display buffers are cleared independently of the Errors.

FIFO status is used to indicate the number of characters in the FIFO and to indicate whether an error has occurred. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO. The character read will be the last one entered. FIFO status will remain at 0000 and the error condition will not be set.

**Data Read**

The master CPU can read DATA from the 8278 FIFO or Display buffers by using the A0, RD, and D0-D7 inputs.

The master sends a RD pulse with A0 = 0 and CS = 0 and the 8278 responds by outputting data on lines D0-D7. The data is strobed by the trailing edge of RD.
DATA READ SEQUENCE

Before reading data, the master CPU must send a command to select FIFO or Display data. Following the command, the master must read STATUS and test the BUSY flag and the OBF flag to verify that the 8278 has responded to the previous command. A typical DATA READ sequence is as follows:

After the first read following a Read Display or Read FIFO command, successive reads may occur as soon as OBF rises.

Data Write

The master CPU can write DATA to the 8278 Display buffers by using the A0, WR and D0-D7 inputs as follows:

The master CPU presents the Data on the D0-D7 lines with A0=0 and then sends a WR pulse. The data is latched by the 8278 on the rising edge of WR.

DATA WRITE SEQUENCE

Before writing data to the 8278, the master CPU must first send a command to select the desired display entry mode and to specify the address of the next data byte. Following the commands, the master must read STATUS and test the BUSY flag (B) and IBF flag to verify that the 8278 has responded. A typical sequence is shown below.
Figure 5. Keyboard Timing

Figure 6. Key Entry and Error Timing

Figure 7. Display Timing
into specific locations in the display register. A new data character is put out on Bq-B9 each time the Mq-M3 lines change (i.e., once every 0.75ms with a 6 MHz crystal). Data is blanked during the time the column select lines change by raising the display outputs. Output data is positive true.

LEFT ENTRY

The left entry mode is the simplest display format in that each display position in the display corresponds to a byte (or nibble) in the Display RAM. ADDRESS 0 in the RAM is the left-most display character and ADDRESS 15 is the right-most display character. Entering characters from position zero causes the display to fill from the left. The 17th character is entered back in the left-most position and filling again proceeds from there.

RIGHT ENTRY

Right entry is the method used by most electronic calculators. The first entry is placed in the right-most display character. The next entry is also placed in the right-most character after the display is shifted left one character. The left-most character is shifted off the end and is lost.

AUTO INCREMENT

In the Left Entry mode, Auto Incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non-Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Left Entry—Auto Increment mode has no undesirable side effects and the result is predictable:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1ST ENTRY</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2ND ENTRY</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMAND</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ENTER NEXT AT LOCATION 5 AUTO INCREMENT

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>3RD ENTRY</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4TH ENTRY</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the Right Entry mode, Auto Incrementing and non-Incrementing have the same effect as in the Left Entry except that the address sequence is interrupted.

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1ST ENTRY</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2ND ENTRY</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>COMMAND</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

ENTER NEXT AT LOCATION 5 AUTO INCREMENT

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3RD ENTRY</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4TH ENTRY</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Starting at an arbitrary location operates as shown below.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>10010101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DISPLAY RAM ADDRESS**

**ENTER NEXT AT LOCATION 5 AUTO INCREMENT**

<table>
<thead>
<tr>
<th>1ST ENTRY</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2ND ENTRY</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8TH ENTRY</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9TH ENTRY</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Entry appears to be from the initial entry point.
Complex Peripheral Control with the UPI-42
COMPLEX PERIPHERAL CONTROL WITH THE UPI-42

TABLE OF CONTENTS

INTRODUCTION
DOT MATRIX PRINTING
THE PRINTER MECHANISM
HARDWARE INTERFACE
TECHNICAL BACKGROUND
SOFTWARE
  Introduction
  Functional Overview
  Memory and Register Allocation
  Description of Functional Blocks and Flowcharts
CONCLUSION

APPENDICES
  Appendix A. Software Listing
  Appendix B. Printer Enhancements
  Appendix C. Printer Mechanism
  Drive Circuit Schematics

FIGURES

1. UPI-42 Pin Configuration
2. UPI-42 Block Diagram
3. UPI-41A, 42 Functional Block Diagram
4. Character E in 5 x 7 Dot Matrix Format
5. Carriage Stepper Motor Assembly
6. Print Head Solenoid Assembly
7. Hardware Interface Block Diagram
8. Hardware Interface Schematic
9. UPI-42 and 8243 I/O Port Map
10. Stepper Motor Step Sequence Waveforms
11. Carriage Stepper Motor Step Sequence
12. Paper Feed Stepper Motor Step Sequence
13. Carriage Stepper Motor Drive Timing
14. Carriage Stepper Motor Predetermined Time Constants
15. Paper Feed Stepper Motor Predetermined Time Constants
16. PTS Lags PT Timing
17. PTS Leads PT Timing
18. Components of Print Head Assembly Line Motion and Printing
19. Data Memory Allocation Map
20. Register Bank 0 Register Assignment
21. Register Bank 0 Status Byte Flag Assignments
22. Register Bank 1
   Register Assignment
23. Register Bank 1 Status
   Byte Flag Assignments
24. Program Memory Allocation Map
25. ASCII Character Code TEST
    Output and Print Example
26. Carriage Stepper Motor
    Phase/Step Data

FLOW CHARTS

1. Main Program Body
2. Power-On/Reset Initialization
3. Home Print Head Assembly
4. External Status Switch Check
5. Character Buffer Fill
6. Carriage Stepper Motor Drive
   and Line Printing
7. Carriage Stepper Motor
   Acceleration Time Storage
8. Process Characters for Printing
9. Translate Character-to-Dots
10. Decelerate Carriage
    Stepper Motor
11. Paper Feed Stepper Motor Drive

Additional sources of information on Intel's UPI devices:

"UPI User's Manual"
   Includes the following Application Notes;
   Programmable Keyboard Interface
   Using the 8295 Dot Matrix Printer Controller
   An 8741A/8041A Digital Cassette Controller

"8048 Family Applications Handbook"

"1983 Microprocessor and Peripheral Handbook"

"MCS-48 and UPI-41A/42 Assembly Language Manual"

"Specifications for Impact Dot Matrix Printer Model-3210", Epson, Jan 8, 1981
INTRODUCTION

The UPI-42 is the newest member of Intel's Universal Peripheral Interface (UPI) microcomputer family. It represents a significant growth in UPI capabilities resulting in a broader spectrum of applications. The UPI-42 incorporates twice the EPROM/ROM of the UPI-41A, 2048 vs 1024 bytes, twice the RAM, 128 vs 64 bytes, and operates at a maximum speed twice that of the UPI-41A, i.e. 12 MHz vs 6 MHz. The ROM based 8042 and the EPROM based 8742 provide more highly integrated solutions for complex stepping motor and dot matrix printer applications. Those applications previously requiring a microprocessor plus a UPI chip can now be implemented entirely with the UPI-42.

The software features of the UPI-42, such as indirect Data and Program Memory addressing, two independent and selectable 8 byte register banks, and directly software testable I/O pins, greatly simplify the external interface and software flow. The software and hardware design of the UPI-42 allows a complex peripheral to be controlled with a minimum of external hardware.

Until recently, the dedicated control processor approach was usually not cost effective due to the large number of components needed; CPU, RAM, ROM, I/O, and Timer/Counters. To help make the approach more cost effective, in 1977 Intel introduced the UPI-41 family of Universal Peripheral Interface controllers consisting of an 8041 (ROM) device and an 8741 (EPROM) device. These devices integrated the common microprocessor system functions into one 40 pin package. The UPI-42 family, consisting of the 8042 and 8742, further extends the UPI's cost effectiveness through more memory and higher speed.

Another member of the UPI family is the Intel 8243 Input/Output Expander chip. This chip provides the UPI-41A and UPI-42 with up to 16 additional independently programmable I/O lines, and interfaces directly to the UPI-41A/42. Up to seven 8243s can be cascaded to provide over 100 I/O lines.

The UPI is a single chip microcomputer with a standard microprocessor interface. The UPI's architecture, illustrated in Figure 3, features on-chip program memory, ROM (8041A/8042) or EPROM (8741A/8742), data memory (RAM), CPU, timer/counter, and I/O. Special interface registers are provided which enable the UPI to function as a peripheral to an 8-bit central processor.

Using one of the UPI devices, the designer simply codes his proprietary peripheral control algorithm into the UPI device itself, rather than into the main system software. The UPI device then performs the peripheral control task while the host processor simply issues commands and transfers data. With the proliferation of microcomputer systems, the use of UPIs or slave microprocessors to off load the main system microprocessor has become quite common.

This Application Note describes how the UPI-42 can be used to control dot matrix printing and the printer mechanism, using stepper motors for carriage/print head assembly and paper feed motion. Previous Intel Application Notes AP-27, AP-54, and AP-91 describe using intelligent processors and peripherals to control single solenoid driven printer mechanisms with 80 character line buffering and bidirectional printing. This Application Note expands on these previous themes and extends the concept of complex device control by incorporating full 80 character line buffering, bidirectional printing, as well as drive and feedback control of two four phase stepper motors.

The Application Note assumes that the reader is familiar with the 8042/8742 and 8243 Data Sheets, and UPI-41A/42 Assembly Language. Although some background information is included, it also assumes a basic understanding of stepper motors and dot matrix printer mechanisms. A complete software listing is included in Appendix A.
DOT MATRIX PRINTING

A dot matrix printer print head typically consists of seven to nine solenoids, each of which drives a stiff wire, or hammer, to impact the paper through an inked ribbon. Characters are formed by firing the solenoids to form a matrix of “dots” (impacts of the wires). Figure 4 shows how the character “E” is formed using a 5 x 7 matrix. The columns are labeled C1 through C5, and the rows R1 through R7. The print head moves left-to-right across the paper, so that at time T1 the head is over column C1. The character is formed by activating the proper solenoids as the print head sweeps across the character position.

Dot matrix printers are a cost effective way of providing good quality hard copy output for microcomputer systems. There is an ever increasing demand for the moderately priced printer to provide more functionality with improved cost and performance. Using stepper motors to control the paper feed and carriage/print head assembly motion is one way of enabling the dot matrix printer to provide more capabilities, such as expanded or contracted characters, dot or line graphics, variable line and character spacing, and subscript or superscript printing.

However, stepper motors require fairly complex control algorithms. Previous solutions involved the use of a main CPU, UPI, RAM, ROM, and I/O onboard the peripheral. The CPU acted as supervisor and used parallel processing to achieve accurate stepper motor control via a UPI, character buffering via the I/O device, RAM, and ROM. The CPU performed real-time decoding of each character into a dot matrix pattern. This Application Note demonstrates that the increased memory and performance of the UPI-42 facilitates integrating these control functions to reduce the cost and component count.

THE PRINTER MECHANISM

The printer mechanism used in this application is the Epson Model 3210. It consists of four basic sub-assemblies; the chassis or frame, the paper feed mechanism and stepper motor, the carriage motion mechanism and stepper motor, and the print head assembly.

The paper feed mechanism is a tractor feed type. It accommodates up to 8.5 inch wide paper (not including tractor feed portion). There is no platen as such; the paper is moved through the paper guide by two sprocketed wheels mounted on a center sprocket shaft. The sprocket shaft is driven by a four phase stepper motor. The rotation of the stepper motor is transmitted to the sprocket shaft through a series of four reduction gears.
The carriage motion mechanism consists of another four phase stepper motor which controls the left-to-right or right-to-left print head assembly motion. The print speed is 80 CPS maximum. Both the speed of the stepper motor and the movement of the print head assembly are independently controllable in either direction. The rotation of the stepper motor is converted to the linear motion of the print head assembly via a series of reduction gears and a toothed drive belt. The drive belt also controls a second set of reduction gears which advances the print ribbon as the print head assembly moves.

Two optical sensors provide feedback information on the carriage assembly position and speed. The first of these optical sensors, called the 'HOME RESET' or HR, is mounted near the left-most physical position to which the print head assembly can move. As the print head assembly approaches the left-most position, a flange on the print head assembly interferes with the light source and sensor, causing the output of the sensor to shift from a logic level one to zero. The right-most printer position is monitored in software rather than by another optical sensor. The right-most print position is a function of the number of characters printed and the distance required to print them.

The second optical sensor, called the 'PRINT TIMING SIGNAL' or PTS, provides feedback on carriage stepper motor velocity and relative position within a
given step of the motor. The feedback is generated by the optical sensor as an "encoder disk" moves across it. Figure 5 illustrates the carriage stepper motor, optical sensor, encoder disk and reduction gears, and drive belt assembly. The optical sensor outputs a pulse train with the same period as the phase shift signal used to drive the stepper, but slightly out of phase with it when the motor is at a constant speed (see Software Functional Block: Phase Shift Data for additional details). The disk acts as a timing wheel, providing feedback to the UPI software of the carriage speed, position, and optimum position for energizing the print head solenoids. The two optical sensors are monitored under software and provide the critical feedback needed to control the print head assembly and paper feed motion accurately. The process of stepper motor drive and control via feedback signals is called "closed loop" stepper motor control, and is covered in more detail in the software discussion.

The print head assembly consists of nine solenoids and nine wires or hammers. Figure 6 illustrates a print head assembly. The available dot matrix measures 9 x 9. This large matrix enables the Epson 3210 print mechanism to print a variety of character fonts, such as expanded or contracted characters, as well as line or block graphics (see Appendix B, Printer Enhancements). It also facilitates printing lower case ASCII characters with "lower case descenders." That is to say, certain lower case letters (e.g. y, p, etc.) will print below the bottom part of all upper case letters.

Figure 5. Carriage Stepper Motor Assembly

Figure 6. Print Head Solenoid Assembly
HARDWARE DESCRIPTION

Figure 7 shows a block diagram of the UPI-42 and 8243 interface to the printer mechanism drive circuit. A complete schematic is shown in Figure 8. The UPI-42 provides all signals necessary to control character buffering and handshaking, paper feed and carriage motion stepper motor timing, print head solenoid activation, and monitoring of external status switches.

The Epson 3210 printer mechanism manual recommends a specific interface circuit to provide proper drive levels to the stepper motors windings and print head solenoids. The hardware interface used for this Application Note followed those recommendations exactly (see Appendix C, Printer Mechanism Drive Circuit Schematics).

I/O Ports

The lower half of the UPI-42 Port 2, pins 0-3, provides an interface to the 8243 I/O expander. The PROG pin of the UPI-42 is used as a strobe to clock address and data information via the Port 2 interface. The extra 16 I/O lines of the 8243 become PORTS 4, 5, 6, and 7 to the UPI software. Combined, the UPI-42 and 8243 provide a total of 28 independently programmable I/O line. These lines are used as follows:
Figure 8. Hardware Interface Schematic

<table>
<thead>
<tr>
<th>Port</th>
<th>No of lines</th>
<th>Bits</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>0-7</td>
<td>O</td>
<td>Character dot column data to print head solenoids (same)</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
<td>O</td>
<td>Print head solenoid trigger</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>7</td>
<td>O</td>
<td>Print head solenoid trigger</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4,5</td>
<td>O</td>
<td>Host system data transfer handshaking (ACK/BUSY)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0-3</td>
<td>O</td>
<td>Carriage &amp; paper feed stepper motors</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1-3</td>
<td>O</td>
<td>Stepper motor select and current limiting</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>I</td>
<td>Paper End sense</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>O</td>
<td>Print head trigger reset (unused)</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>0,2,3</td>
<td>-</td>
<td>External status switches; (LF, FF, TEST, ON/OFF Line)</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>0-3</td>
<td>I</td>
<td>External status switches; (LF, FF, TEST, ON/OFF Line)</td>
</tr>
</tbody>
</table>

Note: The notation used in the balance of this Application Note, when referring to a port number and a particular pin or bit, is Port 23 rather than Port 2 bit 3.

The two printer mechanism optical sensors, discussed in the Printer Mechanism description, are tied to the two “Test Input” pins, T0 and T1, of the UPI-42 through a buffer circuit for noise suppression. These inputs are directly testable in software.
Host System Interface

The host system interfaces to the printer through a parallel port to the UPI-42 Data Bus. Four handshaking signals are used to control data transfer: Data Strobe (STB/), Acknowledge (ACK), Busy (BUSY), and Online or Select. The Data Strobe line of the host parallel port is tied directly to the UPI-42 WR/ pin. This provides a low going pulse on the UPI-42 WR/ pin whenever a data byte is written to the UPI-42. The ACK and BUSY handshake signals are tied to two UPI-42 1/O port lines for software control of data transfer. The “On Line” handshake signal is tied to a single-pole single-throw fixed position switch, which externally enables or disables character transfer from the host system. Characters transmitted to the UPI-42 by the host are loaded into the UPI-42 Data Bus Buffer In (DBBIN) register, and the Input Buffer Full (IBF) interrupt and UPI-42 status flag are set (see Figure 9. UPI-42 and 8243 1/O Ports).

Stepper Motor Interface

Port 4 (41-43) of the 8243, provides both carriage and paper feed stepper motor phase shift signals to the printer mechanism drive circuit. Each of the two stepper motors is driven by 2 two phase excitation signals (4 phases). Figure 10 shows the wave form for each stepper motor. Each signal consists of two components (Sig. 1 A/B & Sig. 2 C/D) 180 degrees out of phase with the other. Each of these signal pairs (A/B & C/D) is 90 degrees out of phase with the other pair. For each signal pair, one port line supplies both halves by using an inverter.

Each of the resulting eight stepper motor drive signals is interfaced to a discrete drive transistor through an inverter. The emitter of the drive transistor is tied to the open collector of the inverter to provide high current sinking capability for the drive transistor. Each half of the motor winding is tied to the collector of the drive transistor (see Appendix C, Printer Mechanism Drive Circuit Schematic).

Each stepper motor requires two current levels for operation. These levels are called “Rush” current and “Hold” current. Rush current refers to the high current required to cause the rotor to rotate within its windings as the polarity of the power applied to the windings is changing. Each change in the polarity of the power applied to the motor windings is called a step or phase shift. Hold current refers to the low level of current required to stabilize and maintain the rotor in a fixed position when the the polarity applied to the windings is not changing. Hold current is simply Rush current with a current limiting transistor switched in. Switching from Hold to Rush current “selects” or enables that stepper motor to move with the next step signal output. In the balance of this Application Note, the term “select” will be used to refer to turning on Rush current, and “deselect” will refer to switching to Hold current.

Three 8243 port lines are dedicated to the select/deselect control of the two stepper motors. One line is for the paper feed stepper motor, and two lines are for the carriage motion stepper motor (80 and 132 column). These lines are labeled SLF, 80Col, and 132Col, and are 8243 PORT 53, 52, and 51, respectively.

By varying the voltage applied to the stepper motor biasing circuit and the current, it is possible to vary the distance the motor moves the print head assembly with each step. Enabling one of two different voltage biasing levels, and changing the timing rate at which the motor is stepped, facilitates either 80 or 132 character column printing. Only 80 character column printing is implemented in the software design. Appendix B, Printer Enhancements, details the software algorithm for handling 132 character printing.

Print Head Interface

A total of eleven I/O lines are used to control the print head solenoids and solenoid firing (see Figure 9 above). Nine are used for character dot data, one for the Print Head Trigger, and one for Reset of the Print Head Trigger circuit. Each of the nine character dot data lines is buffered by an open collector hex inverter.
The Print Head Trigger output is tied to the Trigger input of a 555 Monostable Multivibrator. The output pulse generated by the 555 triggers the print head solenoids to fire. The 555 Output pulse width is independent of the input trigger waveform. The pulse width is determined by an RC network across the 555 inputs and the voltage level applied to the Control Voltage 555 input. The 555 Output is tied to the base of a PNP transistor through an inverter, biased in a normally off configuration. The PNP transistor supplies enough drive to pull up the open collector inverter on each print head solenoid line, Port 10-17 and 26. The 555 output pulse momentarily enables the print head solenoid line open collector inverter output, turning on the solenoid drive transistor, and firing the print head hammer. The 555 Output pulse width is approximately 400 us. Further details of the print head firing operation can be found in the software description below.

Miscellaneous Interface Signals

The 8243 Port 5 pin 0 is tied to the Paper End Detector, a reed switch located on the printer paper guide. This sensor detects when the paper is nearly exhausted.

Three LED status lights complete the hardware interface design. One status light is used for each of the following: Power ON/OFF, On/Off Line, and Out of Paper.

BACKGROUND

Before a detailed discussion of the software begins, a few terms and software functions referenced throughout the software need introduction.

A. What is a Stepper Motor?

A stepper motor has the ability to rotate in either direction as well as start and stop at predetermined angular positions. The stepper motor's shaft (rotor) moves in precise angular increments for each input step. The displacement is repeated for each input step command, accurately positioning the rotor for a given number and sequence of steps.

The stepper motor controls position, velocity, and direction. The accuracy of stepper motors is generally 5 percent of one step. The number of steps in each revolution of the shaft varies, depending on the intended application.

B. Open/Closed Loop Stepper Motor Drive and Control

The carriage stepper motor is closed loop driven. The paper feed stepper motor is open loop driven.

There are two major types of stepper motor control known by the broad headings of open and closed loop.

Open loop is simply continuous pulses to drive the motor at a predetermined rate based on the voltage, current, and the timing of the step pulses applied. Closed loop control is characterized by continuous monitoring of the stepper motor, through feedback signals, and adjusting the motor's operation based upon the feedback received.

C. Stepper Motor Drive Phase Shift or Step Sequence

Each change in the polarity of the power applied to the motor windings is called a step or phase shift. The sequence of the steps or phase shifts, and the pattern of polarity changes output to the stepper motor, determines the direction of rotation.

Figure 10 shows the waveforms for each of the two stepper motors. Figure 11 lists the step sequence for carriage motor clockwise rotation, which moves the print head assembly Left-to-Right. Figure 11 also lists the step sequence for counterclockwise rotations; the print head assembly moves Right-to-Left. Figure 12 lists the step sequence for the paper feed stepper motor clockwise drive. The phase sequence, for either stepper motor, may begin at any point within the sequence list, but must then continue in order.

### Carriage stepper motor rotates clockwise
Print head assembly moves from left to right

<table>
<thead>
<tr>
<th>Step No.</th>
<th>A-Step</th>
<th>B-Step</th>
<th>C-Step</th>
<th>D-Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
</tr>
<tr>
<td>2</td>
<td>On</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>3</td>
<td>Off</td>
<td>On</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>4</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
<td>On</td>
</tr>
</tbody>
</table>

### Carriage stepper motor rotates counter clockwise
Print head assembly moves from right to left

<table>
<thead>
<tr>
<th>Step No.</th>
<th>A-Step</th>
<th>B-Step</th>
<th>C-Step</th>
<th>D-Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>On</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>2</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
</tr>
<tr>
<td>3</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
<td>On</td>
</tr>
<tr>
<td>4</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
<td>On</td>
</tr>
</tbody>
</table>

Figure 11. Carriage Stepper Motor Step Sequence

5-760
C. Acceleration and Deceleration of Stepper Motors

The carriage stepper motor starts from a fixed position, accelerates to a constant speed, maintains constant speed, and then decelerates to a fixed position. Printing may occur from the time and position the print head assembly reaches constant speed, until the time and position the print head assembly begins to decelerate from constant speed. Whether printing occurs during any carriage stepper motor drive sequence is controlled by software. Figure 18, below, illustrates these components of print head assembly line motion.

Due to inertia, a finite time interval and angular displacement is required to accelerate a stepper motor to its full speed. Conversely, deceleration must begin some time before the final angular position. The time interval and angular displacement of the carriage stepper motor translates into the distance the print head assembly travels before it reaches a constant speed. The distance traveled during acceleration is constant. The distance the print head assembly travels during deceleration must be the same as the distance traveled during acceleration in order to accurately align the character dot columns from one line to the next.

E. Stepper Motor Predetermined Time Constant

Whenever the stepper motor is stepped, or energized, the angular velocity of the rotor is greater than the constant speed which is ultimately required. This is called "overshoot." The frictional load of the carriage assembly (motor rotor, reduction gears, drive belt and print head assembly, or paper feed sprocket shaft and wheels) provides damping or frictional load. Damping slows the motor to less than the required constant speed and is called "undershoot" (see Figure 13, Carriage Stepper Motor Drive Timing). A constant rate of speed is achieved through the averaging of the overshoot and undershoot within each step.

Figure 12. Paper Feed Stepper Motor Step Sequence

<table>
<thead>
<tr>
<th>Step No.</th>
<th>A-Step</th>
<th>B-Step</th>
<th>C-Step</th>
<th>D-Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>On</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>2</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
</tr>
<tr>
<td>3</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
<td>On</td>
</tr>
<tr>
<td>4</td>
<td>Off</td>
<td>On</td>
<td>On</td>
<td>Off</td>
</tr>
</tbody>
</table>

Figure 13. Carriage Stepper Motor Drive Timing
The Predetermined Time (PT) Constant is the time required to average the overshoot and undershoot of the particular stepper motor for a desired constant rate of speed. The PT also is the time required to move the print head assembly a specific distance, accounting for both overshoot and undershoot of the stepper motor.

Changing the Predetermined Time Constant changes the angular displacement of the stepper motor rotor, this in turn changes the output. Figure 14 lists the Time Constants for both standard and condensed character printing. Figure 15 lists the paper feed stepper motor Time Constants used for various line spacing formats. This Application Note implements standard character print and paper feed (6 lines per inch) Time Constants. See Appendix B, Printer Enhancements, for details on implementing non-standard Time Constants.

<table>
<thead>
<tr>
<th>Character mode</th>
<th>Predetermined time</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard or Enlarged</td>
<td>2.08ms +10%</td>
<td>-4%</td>
</tr>
<tr>
<td>Character</td>
<td>4.16ms +10%</td>
<td>-4%</td>
</tr>
</tbody>
</table>

**Figure 14. Carriage Stepper Motor Predetermined Time Constants**

<table>
<thead>
<tr>
<th>Paper feed pitch</th>
<th>Approximated values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.12mm (1/16&quot;) /1 pulse</td>
<td></td>
</tr>
<tr>
<td>4.23mm (1/6&quot;) /36 pulses</td>
<td></td>
</tr>
<tr>
<td>3.18mm (1/8&quot;) /27 pulses</td>
<td></td>
</tr>
<tr>
<td>2.82mm (1/9&quot;) /24 pulses</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Paper feed time</th>
<th>Approximated values</th>
</tr>
</thead>
<tbody>
<tr>
<td>150ms/4 23mm</td>
<td>Approx. 6.6 lines/s (continuous feed)</td>
</tr>
<tr>
<td>113ms/3 18mm</td>
<td>Approx. 8.8 lines/s (continuous feed)</td>
</tr>
<tr>
<td>100ms/2 82mm</td>
<td>Approx. 10 lines/s (continuous feed)</td>
</tr>
</tbody>
</table>

**Figure 15. Paper Feed Stepper Motor Predetermined Time Constants**

D. Relationship Between PTS and PT

Figure 13 illustrates how PTS lags PT at the start of acceleration, and moves to lead PT as the motor achieves constant speed. Figure 13 also illustrates the relationship between HR, PTS, PT, acceleration, constant speed, and printing. Figure 16 and 17 illustrate the relationship between PTS and PT during acceleration and at constant speed.

PTS is the point of peek angular velocity within a step of the motor. PTS is a function of the slot spacing on the encoder disk, shown in Figure 5. The spacing is determined by the mechanics of the printer mechanism. When the carriage stepper motor is accelerated from a fixed position, the effects of damping slows the angular velocity of energizing the stepper motor. This causes PTS to occur after the PT, or PTS lags PT. When PTS lags PT, the next step signal is output at PTS rather than at PT. If the step signal is outputted at PTS, the rotor could be midway through a rotation. Energizing the motor at PT could cause it to bind or shift in the wrong direction. When the carriage stepper motor is at a constant rate of speed, PTS leads PT and the step signal is output at PT (see Figure 13).
The time between each step, for a constant number of steps, required for the motor to reach a constant speed, is calculated and stored in Data Memory during acceleration. The values stored are used, in reverse order, during deceleration as the Predetermined Time (PT) Constants. This ensures that the acceleration and deceleration distance traveled by the print head assembly is the same, and that it accurately aligns character dot columns from one line to the next during printing. The time values stored are called “Stored Time Constants.” Steps T1 through T11 in Figure 13, represent the Stored Time Constants.

The equations for the Stored Time Constants are given at the bottom of Figure 13, Carriage Stepper Motor Drive Timing.

**H. Print Head Assembly “Home” Position**

The “logical” Home position for the print head assembly is the left-most position at which printing begins (for L-to-R motion) or ends (for R-to-L motion). The “physical” Home position is the logical HOME position, plus the distance required by the carriage stepper motor to fully accelerate the print head assembly to a constant speed. Printing can only occur when the print head is moving at a constant speed. The printer mechanism manual stipulates eleven step time periods are required to ensure the the print head assembly is at a constant speed. These eleven step time periods are the Stored Time Constants described above. Figure 18 illustrates the components of print head assembly line motion and character printing.

---

**Figure 18. Components of Print Head Assembly Line Motion and Printing**
SOFTWARE

Introduction

The software description is presented in three sections. First, a brief overview of the software to familiarize the reader with the interdependencies and overall program flow. Second, data and program memory allocation and status register flag definitions. And third, each of the ten software blocks is presented with its own flowchart.

Software Overview

The software is written in Intel UPI-41A/42 Assembly Language. A block structure approach is used for ease of development, maintenance, and comprehension. The software is divided into five principal parts.

1. Initialization
2. Character Buffering or Input
3. Stepper Motor Drive and Control
4. Character Processing
5. Character Printing or Output

Flow Chart No. 1 illustrates the overall software algorithm. Below, is a description of the algorithm.
Upon power-on or reset, a software and hardware initialization is performed. This stabilizes and sets inactive the printer hardware and electronics. The print head assembly is then moved to establish its HOME position. The default status registers are set for character buffering, carriage, and paper feed stepper motor drive. The External Status switches are checked; FORMFEED, LINEFEED, ON/OFF LINE, and Character Print TEST. If the printer is ON LINE, the software will loop on filling the Data Memory Character Buffer.

Character or data input to the UPI-42 is interrupt driven. Characters sent by the host system set the Input Buffer Full (IBF) interrupt and the IBF Program Status flag. Character input servicing (completed during the paper feed and carriage stepper motor drive end Delay subroutine) tests for various ASCII character codes, loads characters into the Character Buffer (CB), and repeats until one of several conditions sets the CB Full status flag. Once the CB Full flag is set, further character transmission by the host system is inhibited and printing can begin.

The carriage stepper motor is initialized, and drive begins for the direction indicated. The motor is accelerated to constant speed, printable character codes are translated to dot patterns and printed (if printing is enabled), and the motor is decelerated to a stop. Two timing loops guarantee both constant speed and protection (Failsafe Time) against stepper motor burn out due to high current overload. The two optical sensors, described in the Printer Mechanism section above, are constantly monitored to maintain constant speed, and trigger print head solenoid firing.

Once the line is printed and the carriage stepper motor drive routine has been completed, a Linefeed is forced. The paper feed stepper motor drive subroutine tests the number of lines to move, and energizes the paper feed stepper motor for the required distance. The lines per page default is 66; if 66 lines have been received, a Formfeed to Top-of-Next-Page is performed. The Top-Of-Page is set at Power On/Reset.

When the EOF code is received, the EOF status flag is set. When the last line has been printed, the EOF check will force the print head assembly to the HOME position. The EOF flag is tested following each Paper Feed stepper motor drive. The next entry to the External Status Check subroutine begins a loop which waits for input from either the external status switches or the host system.

The software character dot matrix used in this application is 5 x 7 of the available 9 x 9 print head solenoid matrix. Although lower case descenders and block/line graphics characters are not implemented, Appendix B, Printer Enhancements, discusses how and where these enhancements could be added. The software implements the full 95 ASCII printable characters set.

### Memory and Register Allocation

#### Data Memory Allocation (RAM)

The UPI-42 has 128 bytes of Data Memory. Sixteen bytes are used by the two 8 byte register banks (RBO and RBl). Sixteen additional bytes are used for the Program Stack. The Stored Time Constants utilize 11 bytes, while the stepper motor phase storage requires 4 bytes. Below is a detailed description of Data and Program Memory.

<table>
<thead>
<tr>
<th>Hex Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2F-2FH</td>
<td>80 Character Line Buffer (80 Bytes)</td>
</tr>
<tr>
<td>25-2EH</td>
<td>Stored Time Constants Buffer (11 Bytes)</td>
</tr>
<tr>
<td>20H</td>
<td>Unused</td>
</tr>
<tr>
<td>23H</td>
<td>Character Print Test ASCII Code</td>
</tr>
<tr>
<td>22H</td>
<td>Pseudo Register, Paper Feed Stepper Motor Last Phase Indirect Address</td>
</tr>
<tr>
<td>21H</td>
<td>Pseudo Register, Carriage Stepper Motor Forward/Reverse Last Phase</td>
</tr>
<tr>
<td>18H</td>
<td>Pseudo Register, Last Phase of Stepper Motor Not Being Driven</td>
</tr>
<tr>
<td>1-17H</td>
<td>Register Bank 1 Character Processing</td>
</tr>
<tr>
<td>5-07H</td>
<td>8 Level Stack</td>
</tr>
<tr>
<td></td>
<td>Register Bank 0 Stepper Motor Forward/Reverse Acceleration/Drive</td>
</tr>
</tbody>
</table>

#### Figure 19. Data Memory Allocation Map

Register Bank 0 is used for stepper motor drive functions. Register Bank 1 is used for character processing. Each register bank's register assignments is listed in Figure 20 and 22, respectively. Each register bank has one register allocated as a Status Register. Figure 21 and 23 detail the Status Register flag assignments. Note that bit 7 of each Status Byte is used as a print head assembly motion direction flag. This saves coding of the Select Register Bank (SEL RBl) instruction at each point the flag is checked.

<table>
<thead>
<tr>
<th>Register</th>
<th>Program Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>TmpR0</td>
<td>RB0 Temporary Register</td>
</tr>
<tr>
<td>R1</td>
<td>TStrR0</td>
<td>Store Time Register</td>
</tr>
<tr>
<td>R2</td>
<td>GStrR0</td>
<td>General Status Register</td>
</tr>
<tr>
<td>R3</td>
<td>PhzR30</td>
<td>Stepper Motor Step Register</td>
</tr>
<tr>
<td>R4</td>
<td>CntR40</td>
<td>Count Register</td>
</tr>
<tr>
<td>R5</td>
<td>TConR0</td>
<td>Time Constant Register</td>
</tr>
<tr>
<td>R6</td>
<td>LncL0</td>
<td>Line Count Register</td>
</tr>
<tr>
<td>R7</td>
<td>Opnl70</td>
<td>Available, Scratch</td>
</tr>
</tbody>
</table>

#### Figure 20. Register Bank 0 Register Assignment
Program Memory Allocation (EPROM/ROM)

The UPI-42 has 2048 bytes of Program Memory divided into eight pages, each 256 bytes. Figure 24 illustrates the Program Memory allocation map by page.

Software Functional Blocks

Below is a description and flow chart for each of the ten software blocks listed above.

1. Power-On/Reset Initialization

The first operational part in Flow Chart No. 1 is the Power-On or Reset Initialization. Flowchart No. 2 illustrates the Initialization sequence in detail.
Initialization first disables both interrupts. This is done as a precaution to prevent the system software from hanging-up should an interrupt occur before the proper registers and Data Memory values are initialized.

Initialization then deactivates the system electronics. This is also a precaution to protect the printer mechanism and includes the print head solenoid (trigger and data) lines and the stepper motor select lines. The host system handshake signals are activated to inhibit data transfer from the host until the printer is ready to accept data.

Next, Data Memory is cleared from 20H to 7FH. This includes; the 80 byte Character Buffer, the 11 byte Stored Time Constants buffer, and the 4 bytes used as pseudo registers. The pseudo registers are Data Memory locations used as if they were registers. They serve as storage locations for step data used in accurately reversing the direction of the carriage stepper motor, and stabilizing either of the stepper motors not being driven.

The Data Memory locations 00H through 1FH are not cleared. These locations are Register Bank 0 (00H-07H), Program Stack (08H-17H), and Register Bank 1 (18H-1FH) (see Figure 19). Clearing the Program Registers or Stack would cause the initialization subroutine to become lost. The registers are used from the beginning of the program. Care is taken to initialize the registers and stack accurately prior to each program subroutine as required.

Upon power-on, it is necessary to initialize the two stepper motors, verify their operation, and locate the print head assembly in the left-most ‘HOME’ position. This sequence serves as a system checkout. If a failure occurs, the motors are deselected and the external status light is turned on. Each stepper motor is selected and energized for a sequence of four steps. This serves to align and stabilize each stepper motor’s rotor position, preventing the rotor from skipping or binding when the first drive sequence begins.

At the end of each stepper motor’s initialization, the last step data address is stored in one of the Data Memory pseudo registers. The last step data address is recalled at the beginning of the next corresponding stepper motor drive sequence, and used as the basis of the next step sequence. This ensures that the stepper motor always receives the exact next step data, in sequence, to guarantee smooth stepper motor motion. This also guarantees the motor never skips or jerks, which would misalign the start, stop, and character dot column positions. A stepper motor not being driven has its last phase data output held constant to stabilize it.

Following any stepper motor drive sequence of either motor, a delay of 30-60 ms occurs by switching the current to Hold Current, stabilizing the motor before it is deselected.

2. Home Print Head Assembly

At the end of the carriage stepper motor four step initialization, the output of the HR optical sensor is tested. The level of the HR signal indicates which drive sequence will be required to ‘HOME’ the print head assembly. If the print head assembly is to the right of HR, HR is high, the print head assembly need only be moved to from Right-to-Left until HR is low, then decelerated to locate the physical home position. If HR is low, the print head assembly must be moved first Left-to-Right until HR is high, then Right-to-Left to locate both the logical and physical ‘HOME’ positions. In each case, the software accelerates the carriage stepper motor, generating the Stored Time Constants then decelerates the stepper motor using the Stored Time Constants (see Background section above). Flow Chart No. 3 details the HOME print head assembly subroutine. Figures 13 and 18 illustrate the components of acceleration and print head assembly line motion.
Flow Chart No. 3. HOME Print Head Assembly

The carriage stepper motor drive subroutines used to HOME the print head assembly and to print, are the same. A status flag, called Do-Not-Print, determines whether the Character Processing subroutine is called. The flag is set by the subroutine which calls the Carriage Stepper Motor Drive subroutine. Details of the carriage and paper feed stepper motor drive and character processing subroutines are covered separately below.

3. External Status Switch Check

Once the system is initialized and the print head is at the HOME position, the software enters a loop which continually monitors the four external status switches, and exits if any one is active. Flow Chart No. 4 details the External Status Switch Check subroutine.

Flow Chart No. 4. External Status Switch Check

If the LINEFEED or FORMFEED switch is set, the Paper Feed subroutine is called. The Paper Feed subroutine is discussed in detail below. If the ONLINE switch is set, the Character Buffer (CB) Fill subroutine is called.

If the Character Print TEST switch is set, the Data Memory Character Buffer (CB) is automatically loaded with the ASCII code sequence, beginning at 20H (a Space character), the first ASCII printable character code. The software then proceeds as if the CB had been filled by characters received from the host system. The External Status Switch Check subroutine is exited and character printing begins. When the line has finished printing, a linefeed occurs (as shown in the main program Flow Chart No.1) and the program returns to the External Status Switch Check subroutine. If the TEST switch remains active, the ASCII character code is incremented and program continues as before. This will eventually print all 95 ASCII printable characters. An example of the TEST printer output, the complete ASCII character code printed, is shown in Figure 25.
4. Character Buffer Fill

The Character Buffer (CB) Fill subroutine is called from three points within the main program; External Status Switch subroutine, and the Delay subroutine following the carriage and paper feed stepper motor drive subroutines. Flowchart No. 5 details the Character Buffer Fill subroutine operation.

![Flowchart](flowchart.png)

**Flow Chart No. 5. Character Buffer Fill**

The approximate 80 ms total pre-deselect delay at the end of each stepper motor drive sequence, 40 ms carriage and 40 ms paper feed stepper motor pre-deselect delay, is sufficient to load an entire 80 character line. Half the CB is filled at the end of printing the current line, and the second half is filled at the end of a paper feed. There is no time lost in printing throughput due to filling the character buffer.

Character input is interrupt driven. When the IBF interrupt is enabled, a transmitted character sets the IBF interrupt and IBF Program Status flag. Three instructions make up the IBF interrupt service routine. This short routine disables further interrupts, sets the BUSY handshake line active, inhibiting further transmission by the host, and returns. The subroutine can be executed at virtually any point in the software flow without effecting the printer mechanism operation. Processing of the received character takes place during one of the three program segments mentioned above. The BUSY line remains active until the character is processed by the CB Fill subroutine.

The CB is 80 bytes from the top of Data Memory (30H–7FH). It is a FIFO for forward, left-to-right printing, and a LIFO for reverse, right-to-left, printing. Loading the CB always begins at the top, 7FH. One character may be loaded into the buffer each time the CB Fill subroutine is called.

The CB is always filled with 80 bytes of data prior to printing. If the total number of characters input up to a Carriage Return (CR)/Linefeed (LF), does not completely fill the CB, the CR code is loaded into the CB and the balance of the CB is padded with 20H (Space Character) until the CB is full. A Linefeed (LF) character following a Carriage Return is ignored. A LF is always forced at the end of a printed line. When the CB is full, the CB Full status byte flag is set and printing can begin.

A LF character alone is treated as a CR/ LF at the end of a full 80 character line. This is a special case of a printed line and is handled during character processing for printing (see No. 7, Processing Characters for Printing, below). A Formfeed (FF) character sets the FF status byte flag. The flag is tested at each paper feed stepper motor drive subroutine entry.

When the software is available to load the CB with a character, entry to the CB Fill subroutine checks three status flags; CB Full, CB Pad, and IBF flag. If the CB Full flag is set, the program returns without entering the body of the CB Fill subroutine. The CB Pad flag will cause another Space character to be loaded. If the IBF flag is not set, the program returns. If the IBF flag is set, the character is read from the Data Bus Buffer register, tested for printable or nonprintable ASCII code, and, if printable, loaded into the CB. If the character is a non-printable ASCII code and not an acceptable ASCII control code (CR, LF, FF, EOF), a 20H (Space Character) is loaded into the CB.

Exiting the CB Full subroutine with the CB Full or CB

---

*Note: The content above is a close transcription of the original document, preserving the structure and layout as accurately as possible.*
The carriage stepper motor drive subroutine controls both L-to-R and R-to-L print head assembly motion. Upon entering the subroutine, the HR signal level is tested to determine the direction of print head assembly motion and the Direction status flag is set. The default control register values are loaded and balance of the default status flags are set for stepper motor control and character processing. The default control register values include PT and the step sequence look-up table start address for the direction indicated.

5. Carriage Stepper Motor Drive and Line Printing
The carriage stepper motor drive subroutine controls
The direction flag is tested throughout the carriage stepper motor drive and character processing subroutines. This enables the same subroutines to control activities for either direction, simplifying and shortening the overall program. Flow Chart No. 6 illustrates the carriage stepper motor drive subroutine.

Next, the carriage and paper feed stepper motor step data is initialized. The last step data output to the paper feed stepper motor is loaded into the Last Phase pseudo register. This data is masked with each step data output to the carriage stepper motor. Masking the step data in this manner guarantees the paper feed motor signals do not change as the carriage stepper motor is being driven.

Figure 26 illustrates the carriage stepper motor step sequence versus the actual step data output for clockwise rotation, Left-to-Right motion, and counterclockwise rotation, Right-to-Left print head assembly motion. An eight step sequence is depicted in the figure. Note that the sequence for Right-to-Left motion is the reverse of the sequence for Left-to-Right motion. Note also, that for the L-to-R sequence step 4 is the same as step 6, step 5 the same as step 1, etc., through step 7 matching step 3. The four step sequence simply repeats itself until the motor is stopped via the Deceleration subroutine.

Flow Chart No. 6. Carriage Stepper Motor Drive/Line Printing

Figure 26. Carriage Stepper Motor Phase/Step Data

When the carriage stepper motor is driven for a specific direction of print head assembly motion, the step sequence must be consistent for the motion to be smooth and accurate. The same holds true for the transition from one direction of motion to the other. Since the sequence for one direction is the opposite for the other direction, incrementing the sequence for L-to-R and decrementing for R-to-L provides the needed step data flow. For example, referring to Figure 26, if the print head assembly moved L-to-R and the last step output was #1, the first step for R-to-L motion would be #7. Thus, when the carriage stepper motor is initialized for a clockwise (L-to-R) or counterclockwise (R-to-L) rotation, the last step sequence number is incremented or decremented to obtain the proper next step. In this way, the smooth motion of the stepper motors is assured.

The step data is referenced indirectly via the step sequence number. The step data is stored in a Program Memory look-up table whose addresses correspond to the step sequence numbers. For example, as shown in
Figure 26, at location 0 the step data "1001" is stored. This method is particularly well suited to the UPI-42 software. The UPI-42 features a number of instructions which perform an indirect move or data handling operation. One of these instructions, MOV3 A, @A, unlike the others, allows data to be moved from Page 3 of Program Memory to any other page of Program Memory. This instruction allows the step data to be centrally located on Page 3 of Program Memory and accessed by various subroutines.

Each time the carriage stepper motor step data is output, the step data lookup table address is incremented or decremented, depending upon the direction of rotation, and tested for restart of the sequence. The address is tested because the actual step data, Figure 26, is not a linear sequence and thus is not an easily testable condition for restarting the sequence. The sequence number is tested for rollover of the sequence count from 03H to 04H and clockwise motor rotation via the Jump on Accumulator Bit instruction (JBA), with 00H loaded to restart the sequence. The same bit is tested when decrementing the sequence count for counterclockwise motor rotation, R-to-L motion, because the count rolls over from 00H to 0FFH, with 03H loaded to restart the sequence.

At this point the UPI-42 Timer/Counter is loaded, the step signal is output, and the timer started. The next step data to be output has been determined and the At-Speed flag is tested for entry to one of two subroutines; Stepper Motor Acceleration Time Storage or Character Processing.

The first entry to the Acceleration Time Storage subroutine initializes the subroutine and returns. All other entries to one of the two subroutines perform the necessary operations, detailed below (Blocks 6 and 7), and returns. The program loops until the PT times out or the PT level change is detected. PT is tied to TO of the UPI-42. The level present on TO is directly tested via conditional jump instructions. The software loops on polling the timer Time Out Program Status flag and the TO input level.

As described in the Background section above (shown in Figure 13), if PT times out before PT is detected, the software waits for PT before outputting the next step signal. If PT times out before PT, a second timer count value is loaded into the UPI-42 timer. The timer value is called "Failsafe." This is the maximum time the stepper motor can be selected, with no rotor motion, and not damage the motor. If PT is not detected, either the carriage stepper motor is not rotating or the optical sensor is defective. In either case, program execution halts, the motor is deselected, and the external status light is turned on to indicate a malfunction. A system reset is required to recover from this condition. The Failsafe time is approximately 20 milliseconds, including PT.

The Failsafe time loop also serves as a means of tracking the elapsed time between PT time out and PT.

Entry to the Failsafe time loop sets the Failsafe/Constant Time Window status flag. This flag is tested by the Acceleration Time Storage subroutine for branching to the proper time storage calculation to be performed (see Figure 13 and Block 6 below for further description).

During the Failsafe timer loop, if PTS is detected and verified as true, the Failsafe timer value is read and stored in the Time Storage register. This value is used during the next Acceleration Time Storage subroutine call to calculate the Stored Time Constant (see Block 6 below). If PTS is invalid, the flow returns to the timer loop just exited, again waiting for PTS or Failsafe time out.

During the PT time loop, if PTS is detected and verified, the Sync flag is tested for entry to the print head solenoid firing subroutine. This flag is set by the first entry to the Character Processing subroutine. The flag synchronizes the solenoid firing with character processing. Only if characters are processed for printing will the solenoids be enabled, via the Sync flag, for firing. This prevents solenoids from being fired without a valid character dot data present.

As described in the Background section "Relationship Between PTS and PT," PTS is the point of peak angular velocity within a step of the motor. After PTS is detected, the motor speed ramps down, compensating for the overshoot of the rotor motion. PTS is the optimum time for print head solenoid firing, as shown in Figure 13. This is the most stable point of motor rotation and, thus, the print head assembly motion. If PTS is detected during PT, printing is enabled, the Sync flag is set, and the solenoid trigger is fired.

The firing of the solenoid trigger, following PTS, is very time critical. The time between PTS and solenoid firing must be consistent for accurate dot column alignment throughout the printed line. The software is designed to meet this requirement by placing all character processing and motor control overhead before the solenoid firing subroutine is called. The actual instruction sequence which fires the print head solenoid trigger is plus or minus one instruction for any call to the subroutine.

Once the timer loop is complete, the software tests for Exit conditions. If the Exit conditions fail, the software loops to output the next step signal, starts the PT timer, and continues to accelerate the carriage stepper motor, or process, and print characters. If the Exit test is true, the carriage stepper motor is decelerated to a fixed position, and the program returns to the main program flow (see Flowchart 1).

The exit conditions are different for the two directions of print head assembly motion. For L-to-R printing, if a Carriage Return (CR) character code is read from CB, the carriage stepper motor drive terminates and the motor is decelerated to a fixed position. There are two conditions for terminating carriage stepper motor drive upon detecting a CR during L-to-R motion. If less than half a character line (40 characters) has been printed,
the print head assembly returns to the HOME position to start the next printed line. Otherwise, the print head assembly continues to the right-most position for a full 80 character line, and then begins printing the next line from R-to-L. R-to-L printing always returns the print head assembly to the HOME position before the next line is printed L-to-R. When HR is high, character printing always stops and the carriage stepper motor drive subroutine exits to the deceleration subroutine.

6. Accelerate Stepper Motor Time Storage

As described above, when the carriage stepper motor is accelerated the step time required to guarantee the motor is at a constant rate of speed translates to a specific distance traveled by the print head assembly (see Figure 18). In order to position the print head assembly accurately for bi-directional printing, the distance traveled during deceleration must be the same as during acceleration. The Carriage Motor Acceleration Time Storage subroutine calculates the step times needed to accelerate the carriage stepper motor, and stores them in Data Memory for use as PT during deceleration.

The first call of the Carriage Stepper Motor Acceleration Time Storage subroutine initializes the required registers and status flags. The time calculation begins with the second carriage stepper motor step signal output. The program returns to the carriage motor drive subroutine and loops on PT. Each subsequent call of the Acceleration Time Storage subroutine tests the Failsafe/Constant flag and branches accordingly (see Flow Chart 7). The Acceleration Time Storage subroutine has two parts which correspond to PTS leading or PTS lagging PT.

If the Failsafe/Constant flag is set, PTS lagged PT. The time from PT time out to PTS, Tx (see Figure 13), must be added to the PT and stored in Data Memory. As described above, if PT lagged PT, the Failsafe time is loaded and PTS is again polled during the time loop. When PTS occurs within the Failsafe time, the timer is stopped and the timer value stored. The UPI-42 timer is an up timer, which means that the value stored is the time remaining of the Failsafe time when PTS occurred. The elapsed time must be calculated by subtracting the time remaining (the value stored) from the Failsafe time constant. This is done in software by using two's complement arithmetic. If the Failsafe flag is not set PTS led PT, and PT is the Stored Time Constant stored.

Indirect addressing of Data Memory is used to reference the Stored Time Constant Data Memory location. The Data Memory location address is decremented each time the Acceleration Time Storage subroutine is exited and a Stored Time Constant has been generated.

The last Acceleration Time Storage subroutine exit sets the At-Speed status flag and initializes the character processing registers and flags.

3. Process Characters for Printing

The Character Processing subroutine is entered only if the Home Reset (HR) optical sensor signal is high and printing is enabled. Otherwise, the software simply returns to the Carriage Stepper Motor Drive subroutine. There are two cases when printing is not enabled; during the HOME subroutine operation, and when the print head assembly returns to the HOME position after printing less than half an 80 character line. If printing is enabled, the Sync status flag is set.

All character processing operations use the second UPI-42 Data Memory Register Bank, RBI. Register Bank 1 is independent of Data Memory Register Bank 0, used for stepper motor control. The use of two independent register banks greatly simplifies the software flow, and helps to ensure the accuracy of event sequences that must be handled in parallel. Each register bank must be initialized only once for any entry to either the Carriage Stepper Motor Drive or Character Processing subroutines. A single UPI-42 Assembly Language instruction selects the appropriate register bank. Initializing the character processing registers includes loading the maximum character count (80), dot matrix size count (6), and CB start address. The CB start address is print direction dependant, as described in Block 4, above.

Character processing reads a character from the CB, tests for control codes, translates the character to dots, and conditionally exits, returning to the Carriage Stepper Motor Drive subroutine. Flow Chart 8 details the character processing subroutine.
Each character requires six steps of the carriage stepper motor to print; five for the 5 character dot columns and I for the blank dot column between each character. Reading a character from the CB and character-to-dot pattern translation takes place during the last character dot column, or blank column, time.

The first character line entry to the Character Processing subroutine appears to the software as if a last character dot column (blank column) had been entered. The next character, in this case the first character in the line, is translated and printing can begin. This method of initializing the Character Processing subroutine utilizes the same software for both start-up and normal character flow. Once a character code has been translated to a dot matrix pattern starting address in the look-up table, all subsequent entries to the Character Processing subroutine simply advance the dot column data address and outputs the data.

The decision to translate the character to dots during the blank column time was an arbitrary one. As was the choice of the blank column following rather than preceding the actual character dot matrix printing.

### 4. Translate Character-to-Dots

Character-to-dot pattern translation involves converting the ASCII code into a look-up table address, where the first of the five bytes of character dot column data is stored. The address is then incremented for the next column of dot pattern data until the full character has been printed.

The dot pattern look-up table occupies two pages, or approximately 512 bytes of Program Memory. A printable ASCII character is tested for its dot pattern location page and the offset address, from zero, on that page. Both the page test and page offset calculations use two's complement arithmetic, with a jump on carry or not carry causing the appropriate branching. Once the pattern page and address are determined the indirect addressing and data move instructions are used to read and output the data to the print head solenoids. Flowchart 9 details the Character-to-Dots Translation subroutine.

In the case of R-to-L printing, although the translation operation is the same, the character is printed in reverse. This requires that the character dot pattern address be incremented by five, before printing begins, so that the first dot column data output is the last dot column data of the character. The dot pattern look-up table address is then decremented rather than incremented, as in L-to-R printing, for the balance of the character. Translation still takes place during the last character dot column, the blank column, and the blank column follows the character matrix.

Only one control code, a Carriage Return (CR), is encountered by the character translation subroutine. Linefeed (LF) characters are stripped off by the CB Fill subroutine. If a CR code is detected the software tests for a mid-line exit condition; less than half the line printed exits the stepper motor drive subroutine and HOMEs the print head assembly before printing the next line. If the test fails, more than half the line has been printed, the CR is replaced by a 20H (Space character) and printing continues for the balance of the line; the space characters padding the CB are printed.
As mentioned above, the character dots are printed and the print head trigger is fired when the PTS signal is detected and verified and the carriage stepper motor is At Speed.

When the character to print test fails the CB Buffer size count equals zero, the Carriage Stepper Motor Drive subroutine exit flags are set, and the flow passes to the Deceleration and Delay subroutines and programs returns to the main program flow.

9. Decelerate Carriage Stepper Motor

The transition from the Carriage Stepper Motor Drive subroutine to the Deceleration subroutine outputs the next step signal in sequence, and then initializes the Deceleration subroutine registers; Stored Time Constants Data Memory buffer end address and size. The Stored Time Constant Buffer is a LIFO for deceleration of the carriage stepper motor. The buffer size is used as the step count. When the step count decrements to zero, the step signal output is terminated, and the last step sequence number is stored in the carriage stepper motor Next Step pseudo register. The last step sequence number is recalled, during initialization of the next carriage stepper motor drive, as the basis of the next step data signal to be output. See Flow Chart 10.

When the carriage stepper motor is decelerated, Fail-safe protection and PTS monitoring are not necessary. The Deceleration subroutine acts as its own fail-safe mechanism. Should the stepper motor hang-up, the subroutine would exit and deselected the motor in sufficient time to protect the motor from burnout. Since neither FailSafe nor print head solenoid firing take place during deceleration, PTS is not needed. PT is replaced by the Stored Time Constant values in Data Memory. The Deceleration subroutine determines the next step signal to output, loads the Timer with the Stored Time Constant, starts the UPI-42 Timer, and loops until time out. The subroutine loops to output the next step until all of the Stored Time Constants have been used. The program returns to the Carriage Stepper Motor Drive subroutine and the motor is deselected following the Delay subroutine execution. The Delay subroutine is called to stabilize the stepper motor before it is deselected. During the DELAY subroutine, the IBF interrupt is enabled and characters are processed. A paper feed is forced following the carriage stepper motor being deselected.

10. Paper Feed Stepper Motor Drive

The paper feed stepper motor subroutine outputs a predefined number of step signals to advance the paper, in one line increments, for the required number of lines. The number of step signals per line increment is a function of the defined number of lines per inch, given the distance the paper moves in one step. Figure 16 lists three step (or pulse) count and line spacing configura-
tions, as well as the distance the paper moves in one step. Standard 6 lines per inch spacing has been implemented in this Application Note (Appendix B details how variable line spacing could be implemented). Flowchart 11 illustrates the Paper Feed subroutine.

If the Formfeed flag has been set in the Character Buffer Fill subroutine, the software calculates the number of lines needed for a top of next page paper feed. The resulting line count is loaded in the Line Count Register. The Paper Feed subroutine loops on the line count until done and then returns to the main program body.

Once the Paper Feed subroutine is complete, the software loops to test the End of File (EOF) Flag (see Flow Chart I). If EOF is set, the print head assembly is moved to the HOME position, the program again enters the External Status Switch Test subroutine, and begins polling the external status switches. If EOF is not set, the program directly calls the External Status Switch Check subroutine, and the program repeats for the next line.

**CONCLUSION**

Although the full speed, 12 MHz, of the UPI-42 was used, the actual speed required is approximately 8-9 MHz. 1400 bytes of the available 2K bytes of Program Memory were used; 500 bytes for the 95 character ASCII code dot pattern look-up table, 900 bytes for operational software. This means that the UPI-42 has excess processing power and memory space for implementing the additional functions such as those listed below and discussed in Appendix B.

- Special Characters or Symbols
- Lower Case Descenders
- Inline Control Codes
- Different Character Formats
- Variable Line Spacing

The software developed for this Application Note was not fully optimized and could be further packed by combining functions. This would require creating another status register, which could also serve to implement some of the features listed above. Since the full 16 byte stack is not used for subroutine nesting, there are 6-8 bytes of Program Stack Data Memory that could be used for this purpose. In several places, extra code was added for clarity of the Application Note. For example, each status byte flag is set with a separate instruction, using a equate label, rather than setting several flags simultaneously at the same point in the code.

This Application Note has demonstrated that the UPI-42 is easily capable of independently controlling a complex peripheral device requiring real time event monitoring. The moderate size of the program required to implement this application attests to the effectiveness of the UPI-42 for peripheral control.
APPENDIX A.
SOFTWARE LISTING

1 $MOD42 TITLE('UPI 42 APP NOTE');
2 $MACROFILE NOSYMBOLS NOGEN DEBUG
3 $INCLUDE(:F1:ANECD.OVI)

5  
6  
7  Complex Peripheral Control With the UPI-42
8  
9  Intel Corporation
10  
11  
12  
13  
14  
15  Written By Christopher Scott

20  Notes and Comments
21  
22  Three Assembly Language files comprise the full Application
23  Note source code:
24  
25  1 ANECD.OVI App Note Equates, Constants, Declarations . Overlay
26  
27  2. 42ANC.SRC UPI-42 App Note Code Source
28  
29  3 CHRTBL.OVI Character Table . Overlay (Character Lookup Tables)
30  
31  
32  
33  
34  PG
35  
36  Equates, Constants and System Definitions
37  
38  
39  Data & Program Memory Allocations
40  
41  Program Memory
42  
43  
44  
45  
46  Page 7 1792-2047 Char to Dot pattern lookup table
47  
48  Page 6 1536-1791 Char to Dot pattern lookup table
49  
50  Page 5 1280-1535 Misc called routines:
51  
52  
53  
54  
55  
56  
57  
58  Page 4 1024-1279 PaperFeed Stpr Mtr Init and Drive
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70
= 71 ; PG
= 72 ;
= 73 ; Data Memory
= 74 ;
= 75 ;
= 76 ;
= 77 ;
= 78 ;
= 79 ;
= 80 ;
= 81 ;
= 82 ;
= 83 ;
= 84 ;
= 85 ;
= 86 ;
= 87 ;
= 88 ;
= 89 ;
= 90 ;
= 91 ;
= 92 ;
= 93 ; Data Memory Equates.
= 94 ;
= 95 ;
= 96 ;
= 97 ;
= 98 ;
= 99 ;
= 100 ;
= 101 ;
= 102 ;
= 103 ;
= 104 ;
= 105 ;
= 106 ;
= 107 ;
= 108 ;
= 109 ;
= 110 ;
= 111 ;
= 112 ;
= 113 ;
= 114 ;
= 115 ;
= 116 ;
= 117 ; PG
= 118 ;
= 119 ; Register allocation
= 120 ;
= 121 ;
= 122 ; All Indirect Data Memory Addressing via @Rn inst must use
= 123 ; only registers 0 & 1 of either register bank. Any other will
= 124 ; be rejected by the Assembler
= 125 ; Last character in table indicates Register Bank referenced
= 126 ;
= 127 ; Register Bank O
= 128 ;
= 129 ;
= 130 ;
= 131 ;
= 132 ;
= 133 ;
= 134 ;
= 135 ;
= 136 ;
= 137 ;
= 138 ;
= 139 ;
= 140 ; Register Bank O Data Memory Address
= 141 ;
0000  = 142 TmpAOO EQU OOH  ; Temporary Register DM address
0001  = 143 TStrAO EQU O1H  ; Time Store Register DM address
0002  = 144 GStrAs EQU O2H  ; RBO Char Status Reg DM address
0003  = 145 PhnAOE EQU O3H  ; Stpr Mtr Phase Register DM address
0004  = 146 CntRmE EQU O4H  ; Cnt Reg. Phase count-Stpr Mtr loops
0005  = 147  
0006  = 148 LcRmAO EQU O5H  ; Time constant reg DM address
0007  = 151 OpnA70 EQU O7H  ; Available

= 154 ; PG

---------------------------------------------------------------------
= 156 ; RBO Status Byte Bit Definition

---------------------------------------------------------------------
= 159 ; Stepper Motor control bit masks function on GStr10

---------------------------------------------------------------------
= 160 ;

---------------------------------------------------------------------
= 161 ; Bit Definition

---------------------------------------------------------------------
= 162 ; Stpr Mtr Direction. L-to-R = 1,

---------------------------------------------------------------------
= 163 ; R-to-L = 0

---------------------------------------------------------------------
= 164 ; Stpr Mtr at speed and CR not left of Home

---------------------------------------------------------------------
= 165 ; Accel/Decel Init. 1 = Done / 0 = Not Done

---------------------------------------------------------------------
= 166 ; 1 = FailSafe / 0 = Constant. Time Window

---------------------------------------------------------------------
= 167 ; 2 = Form Feed / 0 = Line Feed

---------------------------------------------------------------------
= 168 ; 1 = Do Not Print / 0 = Print

---------------------------------------------------------------------
= 169 ; FAccel/DAccel drive Ready = 1/NotRdy = 0 (exit

---------------------------------------------------------------------
= 170 ; drive & decel stpr mtr)

---------------------------------------------------------------------
= 171 ;

---------------------------------------------------------------------
= 172 ; Bit Masks: RBO

---------------------------------------------------------------------
= 173 ;

---------------------------------------------------------------------
= 174 ;

---------------------------------------------------------------------
= 175 ;

---------------------------------------------------------------------
= 176 ;

---------------------------------------------------------------------
= 177 ;

---------------------------------------------------------------------
= 178 ;

---------------------------------------------------------------------
= 179 ;

---------------------------------------------------------------------
= 180 ;

---------------------------------------------------------------------
= 181 ;

---------------------------------------------------------------------
= 182 ;

---------------------------------------------------------------------
= 183 ;

---------------------------------------------------------------------
= 184 ;

---------------------------------------------------------------------
= 185 ;

---------------------------------------------------------------------
= 186 ;

---------------------------------------------------------------------
= 187 ;

---------------------------------------------------------------------
= 188 ;

---------------------------------------------------------------------
= 189 ;

---------------------------------------------------------------------
= 190 ;

---------------------------------------------------------------------
= 191 ;

---------------------------------------------------------------------
= 192 ;

---------------------------------------------------------------------
= 193 ;

---------------------------------------------------------------------
= 194 ;

---------------------------------------------------------------------
= 195 ; PG

---------------------------------------------------------------------
= 196 ;

---------------------------------------------------------------------
= 197 ;

---------------------------------------------------------------------
= 198 ;

---------------------------------------------------------------------
= 199 ;

---------------------------------------------------------------------
= 200 ;

---------------------------------------------------------------------
= 201 ; Register Bank 1

---------------------------------------------------------------------
= 202 ;

---------------------------------------------------------------------
= 203 ;

---------------------------------------------------------------------
= 204 ;

---------------------------------------------------------------------
= 205 ;

---------------------------------------------------------------------
= 206 ;

---------------------------------------------------------------------
= 207 ;

---------------------------------------------------------------------
= 208 ;

---------------------------------------------------------------------
= 209 ;

---------------------------------------------------------------------
= 210 ;

---------------------------------------------------------------------
= 211 ;

---------------------------------------------------------------------
= 212 ;

---------------------------------------------------------------------
= 213 ;

---------------------------------------------------------------------
= 214 ;

---------------------------------------------------------------------
= 215 ;

---------------------------------------------------------------------
= 216 ;

---------------------------------------------------------------------
= 217 ;

---------------------------------------------------------------------
= 218 ;

---------------------------------------------------------------------
= 219 ;

---------------------------------------------------------------------
= 220 ;

---------------------------------------------------------------------
= 221 ;

---------------------------------------------------------------------
= 222 ;

---------------------------------------------------------------------
= 223 ;

---------------------------------------------------------------------
= 224 ;

---------------------------------------------------------------------
= 225 ;

---------------------------------------------------------------------
= 226 ;

---------------------------------------------------------------------
= 227 ;

---------------------------------------------------------------------
= 228 ;

---------------------------------------------------------------------
= 229 ;

---------------------------------------------------------------------
= 230 ;

---------------------------------------------------------------------
= 231 ;

---------------------------------------------------------------------
= 232 ;

---------------------------------------------------------------------
= 233 ;

---------------------------------------------------------------------
= 234 ;

---------------------------------------------------------------------
= 235 ;

---------------------------------------------------------------------
= 236 ;

---------------------------------------------------------------------
= 237 ;

---------------------------------------------------------------------
= 238 ;

---------------------------------------------------------------------
= 239 ;

---------------------------------------------------------------------
= 240 ;

---------------------------------------------------------------------
= 241 ;

---------------------------------------------------------------------
= 242 ;

---------------------------------------------------------------------
= 243 ;

---------------------------------------------------------------------
= 244 ;

---------------------------------------------------------------------
= 245 ;

---------------------------------------------------------------------
= 246 ;

---------------------------------------------------------------------
= 247 ;

---------------------------------------------------------------------
= 248 ;

---------------------------------------------------------------------
= 249 ;

---------------------------------------------------------------------
= 250 ;

---------------------------------------------------------------------
= 251 ;

---------------------------------------------------------------------
= 252 ;

---------------------------------------------------------------------
= 253 ;

---------------------------------------------------------------------
= 254 ;

---------------------------------------------------------------------
= 255 ;

---------------------------------------------------------------------
= 256 ;
277 278

Character printing bit masks function on ChnStr1

0080 = 246 ChnPrn EQU 90H ;Str Mtr Direction: L-to-R = 1, R-to-L = 0
007F = 247 CifPrf EQU 7FH ;Str Mtr Direction. R-to-L = 0
0040 = 248 ChnIntD EQU 040H ;Set Char Init Done
008F = 249 ChnIntND EQU 0B8H ;Reset Char Init Not Done
0020 = 250 ChnPrP1 EQU 20H ;Page 1 char. out entry bit (ORL)
00DF = 251 ChnPrP2 EQU 0DFH ;Page 2 char. reset entry bit (ANL)
0010 = 252 TstPrn EQU 10H ;Char print test
00EF = 253 NmPrn EQU 0E8H ;Normal char input
0024 = 254 Bit Masks. ;RB1
0006 = 255 EDF EQU 0BH ;set EDF flag
0007 = 256 CilEDF EQU 078H ;clear EDF flag = Not EDF
0004 = 257 CRLF EQU 04H ;CR/LF
0008 = 258 CrlCR EQU 08H ;Clear CR/LF
0002 = 259 CBFLn EQU 02H ;Full Line in Char Buffer
000D = 260 CBFLn EQU 02FH ;Not Full Line in Char Buffer
0001 = 261 IntCBR EQU 01H ;Init of CB registers done
000E = 262 CInfCBR EQU 0FEH ;Init of CB registers not done
0263 = 264
0265 = 265 ; PG
0266 = 266 ; Equates (cont)
0267 = 267
0268 = 268
0269 = 269
0270 = 270 ; Misc
0271 = 271
0004 = 272 RLPShf EQU 04H ;R-to-L print lookup table addr shift
0020 = 274 Ascii EQU 20H ;hex nnmbr of first Ascii Char
007F = 275 AscList EQU 7FH ;hex nnmbr of last Ascii Char
0026 = 276
00F3 = 277 CRCpl EQU 0F3H ;ASCII control code 2's complement
00F6 = 278 LFCpl EQU 0FH ;
00F4 = 279 FFCpl EQU 0FH ;
00E5 = 280 EscCpl EQU 0E5H ;
00E0 = 281 AscCpl EQU 0E0H ;
00C8 = 282 TFCpl EQU 0CH ;
00D0 = 283 CR EQU 0DH ;Ascii code (hex)
0020 = 284 Space EQU 2OH ;Ascii code (hex)
00B1 = 285 LAsEnd EQU 81H ;Ascii End 2's cpl - test line start
00B2 = 287 PasEnd EQU 82H ;Ascii End 2's cpl - within line print
007F = 288 AscStp EQU 7FH ;Ascii mask. strip off MBB
0042 = 289 PgLnCt EQU 66 ;Page Line Count: Default = 66.
00C4 = 290 PgLCt EQU 04H ;Printed lines per page test
0018 = 291 EDFCpl EQU 18H ;EDF ascii code cpl
292 = 292
293 = 293 ; Loop count values
294 = 294

5-780
<table>
<thead>
<tr>
<th>Offset</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0004</td>
<td>297 PHCn1</td>
<td>EQU 04H</td>
</tr>
<tr>
<td>0004</td>
<td>297 PHCn1</td>
<td>NUMBER OF SM PHASES ON INIT</td>
</tr>
<tr>
<td>0004</td>
<td>299 ILFCnt</td>
<td>EQU 01H</td>
</tr>
<tr>
<td>0004</td>
<td>300 LP1Sp6</td>
<td>EQU 36H</td>
</tr>
<tr>
<td>0004</td>
<td>301 LP1Sp8</td>
<td>EQU 27H</td>
</tr>
<tr>
<td>0004</td>
<td>302 LP1Sp10</td>
<td>EQU 24H</td>
</tr>
<tr>
<td>0004</td>
<td>303</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>304 LineCt</td>
<td>EQU 01H</td>
</tr>
<tr>
<td>0004</td>
<td>305 FmFdcT</td>
<td>EQU 66H</td>
</tr>
<tr>
<td>0004</td>
<td>306 Status</td>
<td>EQU 03H</td>
</tr>
<tr>
<td>0004</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>308 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>309 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>310 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>311 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>312 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>313 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>314 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>315 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>316 DLYCL</td>
<td>EQU 80H</td>
</tr>
<tr>
<td>0004</td>
<td>317 DLYCS</td>
<td>EQU 30H</td>
</tr>
<tr>
<td>0004</td>
<td>318 DtgTim</td>
<td>EQU 256-52</td>
</tr>
<tr>
<td>0004</td>
<td>319 FailTim</td>
<td>EQU 256-256</td>
</tr>
<tr>
<td>0004</td>
<td>320 CrTmr1</td>
<td>EQU 256-52</td>
</tr>
<tr>
<td>0004</td>
<td>321 CrTmr2</td>
<td>EQU 256-70</td>
</tr>
<tr>
<td>0004</td>
<td>322 CrTmr3</td>
<td>EQU 256-110</td>
</tr>
<tr>
<td>0004</td>
<td>323 IntTmr2</td>
<td>EQU 256-64</td>
</tr>
<tr>
<td>0004</td>
<td>324 LFTMR1</td>
<td>EQU 256-104</td>
</tr>
<tr>
<td>0004</td>
<td>325 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>326 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>327 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>328 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>329 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>330 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>331 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>332 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>333 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>334 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>335 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>336 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>337 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>338 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>339 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>340 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>341 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>342 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>343 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>344 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>345 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>346 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>347 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>348 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>349 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>350 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>351 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>352 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>353 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>354 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>355 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>356 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>357 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>358 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>359 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>360 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>361 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>362 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>363 Status</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>364 Status</td>
<td></td>
</tr>
</tbody>
</table>

5-781
PORT BIT ASSIGNMENT:

- S S S -
- L C C
- F R R
- B I
- O 3
- 2

3 S 5 5 5
3 2 1 0

CODING:

- SLF 0 1 0 0 O6H
- SCR80 1 0 0 0 OAH
- SCR132 1 1 0 0 OCH
- SMOFF 1 1 1 0 OEH
- W/SCR80 & SCR132 'O' [BOTH SELECTED]
- [DO NOT KNOW WHETHER SCR80='O' WILL
- SELECT 80 COL ONLY] - REQUIRES TEST.

401: PG

MAIN PROGRAM BODY

Power On / Reset Program Entry

PROGRAM START

0000 0408
010 Org 00H
011
412 START: JMP RESET
413
414 INPUT BUFFER FULL INTERRUPT CALL ENTRY AND VECTOR

0003 019 ORG 03H
0003 1429
0005 93
0005 RTR
0007 Timer
0007 1429 Call TMIS
0009 DC 421 SEL R80
000A 83
0022 Ret
0023
424 : INITIALIZATION

425

0008 13 426 Reset: Dis I
000C 35 427 Dis Tcnt1
000D B40F 428 Call InitAl
000F B42F 429 Call CIRDM
030 Call InitCR
0011 B44B 431 Call InitLF
0013 9400 432 CALL CR SM POWER ON INIT
433
434 : MAIN PROGRAM LOOP

435

0015 B422 436 All program segments are called from here
437

438

0017 B400 439
0019 1420 440 CBInpt: Call BVBFF
441 test for: CB full/ff, LF, FF,
442 Char Prnt Test

5-782 230795-001
0012 3400 443 Repeat: Call SMDrivr ;Call Forward Strp Mtr Drive
0010 940D 444 Call LFDrivr ;Call Linefeed Strp Mtr Drive
001F D5 445 SEL RB1
0020 FA 446 Mov A,ChStr1 ;get the Char Status Register RB1
0021 7215 447 JB3 Home ;jump to CR SM Home if EDF bit set
0023 0419 448 Jmp CBInstr ;loop to Char Buffer Input test

450 ; PG
451 ; INTERRUPT SERVICE ROUTINE
452 ;
453 ;
454 ;
455 ;
456 ; INPUT BUFFER FULL INTERRUPT SERVICE ROUTINE
457 ;
458 ;
459 ; IBFIS:
460 ;
461 ; Acknowledge Char input and set Hold/Busy Active
462 GOm P2,#Busy ;get & set DBS ACK/Busy Bits
463 Dis I ;disable IBF interrupts
464 Ret
465
466 ;
467 ; TIMER / COUNTER INTERRUPT SERVICE ROUTINE
468 ;
469 ; ITF interrupt service routine disables all intr during
470 ; strp mtr phase shifting
471 TMRRS. Dis I ;disable IBF interrupts
472 Dis TCnt1 ;disable ITF interrupts
473 Ret
474
475 ; PG

476 ; EXTERNAL STATUS SWITCH CHECK/CHAR. BUFFER FILL
477 ;
478 ; ESCBF: ;Prep for normal character handling/input
479 ;
480 002C D5 480 SEL RB1
481 002D FA 481 Mov A,ChStr1 ;get the character stat reg byte
482 0025 334F 482 ANL A,#NmrPrn ;set normal character input
483 0030 AA 483 Mov ChStr1.A ;store the stat byte
484 0031 C5 484 SEL RB0
485
486 ;
487 0032 OF 487 MovD A,P7 ;get the stat switch port bits
488 0033 123D 488 JB0 FormFd ;service Formfeed
489 0035 3245 489 JB1 LinFd ;service Linefeed
490 0037 5249 490 JB2 ChTst ;service Character TEST
491 0039 723E 491 JB3 Online ;service Char Buffer Check/Fill
492 0038 042C 492 Jmp ESCBF ;loop
493
494 ;
495 003A FA 494 FormFd. Mov A,GStr20 ;get the status byte
496 003E 4304 495 ORL A,#FrmFd ;set the formfeed stat flag
497 0040 AA 496 Mov GStr20.A ;store the status byte
498 0041 940D 497 Call LFDrivr ;do a formfeed
499 0043 042C 498 Jmp ESCBF ;

500 0045 940D 500 LinFd Call LFDrivr ;do a line drive
501 0047 042C 501 ESCBF Jmp
502
503 0049 D5 502 ChrTst SEL RB1
504 004A FA 503 Mov A,ChStr1 ;get the character stat reg byte
505 0048 4310 504 ORL A,#TstPrn ;set character test flag
506 0040 AA 505 Mov ChStr1.A ;store the stat byte
507 0046 BB23 506 Mov TmpR10.#TasCls ;load the pseudo Ascii code tmp reg addr
508 0050 F0 507 Mov A,TmpR10 ;get the inc'd ascii code
509 0051 0381 508 ADD A,#LAsEnd ;test for code end
50A 0053 9E57 509 JNZ AsClCd ;if not code end jump to load
510 0055 B020 510 Jmp TmpR10.#Ascl ;if end restart ascii at beginning
511 0057 F0 511 AsClCd: Mov A,TmpR10 ;get the ascii code again
512 0059 AF 512 Mov DprR71.A ;place in the empty register
513 0059 10 513 Inc TmpR10 ;Inc start ASCII char in data memory
514 005A B439 514 Call PrnTst ;call the DM load procedure
515 005C C5 515 JEL RB0 ;resellect reg bank 0
516 005D B3 516 Ret

5-783
230795-001
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>005E D5</td>
<td>520</td>
<td>OnLine: SEL RBD</td>
<td>select char buffer registers</td>
</tr>
<tr>
<td>005F 05</td>
<td>521</td>
<td>EN I</td>
<td>enable interrupts</td>
</tr>
<tr>
<td>0060 FA</td>
<td>522</td>
<td>CBfCkI: Mov A.ChStrI</td>
<td>get the Char Stat Byte</td>
</tr>
<tr>
<td>0061 0267</td>
<td>523</td>
<td>JB1 CBfCkEx</td>
<td>if Char Buf has Full line exit</td>
</tr>
<tr>
<td>0063 146D</td>
<td>524</td>
<td>IBfCk: Call CBfIll</td>
<td>read a char into Char Buffer</td>
</tr>
<tr>
<td>0065 0460</td>
<td>525</td>
<td>Jmp CBfCkI</td>
<td>loop to Char Buf Full test</td>
</tr>
<tr>
<td>0067 C5</td>
<td>526</td>
<td>CBfCkEx: SEL RBD</td>
<td></td>
</tr>
<tr>
<td>0068 B3</td>
<td>527</td>
<td>Ret</td>
<td></td>
</tr>
<tr>
<td>529</td>
<td></td>
<td>PN</td>
<td></td>
</tr>
<tr>
<td>530</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>531</td>
<td></td>
<td>Character Input</td>
<td></td>
</tr>
<tr>
<td>532</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>533</td>
<td></td>
<td>Input Buffer Full service routine: test for Char</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>buffer full-exit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>else load char into char buffer</td>
<td></td>
</tr>
<tr>
<td>0069 D5</td>
<td>535</td>
<td>IBFSrv: SEL RBD</td>
<td></td>
</tr>
<tr>
<td>006A FA</td>
<td>536</td>
<td>Mov A.ChStrI</td>
<td>get the RBO stat byte</td>
</tr>
<tr>
<td>006B 32EC</td>
<td>537</td>
<td>JB1 CBfUl</td>
<td>if Do Not Print Bit Set - EXIT</td>
</tr>
<tr>
<td>006D 527C</td>
<td>538</td>
<td>CBfUl: JS2 CBfPad</td>
<td>test for CB padding flag</td>
</tr>
<tr>
<td>539</td>
<td></td>
<td>JS2</td>
<td>if not pad enable char input</td>
</tr>
<tr>
<td>540</td>
<td></td>
<td></td>
<td>tell the host to send char's</td>
</tr>
<tr>
<td>006F 05</td>
<td>541</td>
<td>EN I</td>
<td></td>
</tr>
<tr>
<td>0070 66EC</td>
<td>542</td>
<td>JN1IBF CBfILE</td>
<td></td>
</tr>
<tr>
<td>543</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0072 FA</td>
<td>544</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0073 127C</td>
<td>545</td>
<td>Mov A.ChStrI</td>
<td>get the RBO Char Stat Byte</td>
</tr>
<tr>
<td>0074 BA10</td>
<td>546</td>
<td>UBO SkpInt</td>
<td>test for CB has been initialized</td>
</tr>
<tr>
<td>547</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0075 4301</td>
<td>548</td>
<td>ORL A.IntCBR</td>
<td>set CB Reg skip Initialization stat bit</td>
</tr>
<tr>
<td>0077 AA</td>
<td>549</td>
<td>Mov ChStrI.A</td>
<td>save the altered stat byte</td>
</tr>
<tr>
<td>007B B7F7</td>
<td>550</td>
<td>Mov CDmr1.#FCFBst</td>
<td>load char reg w/char buffer str</td>
</tr>
<tr>
<td>007A BD50</td>
<td>551</td>
<td>Mov CCntR1.#ChFS#</td>
<td>load char cnt reg w/char buffer size</td>
</tr>
<tr>
<td>007C ED86</td>
<td>552</td>
<td>CBPad: DJNZ CCntR1.LdChar</td>
<td>DECREMENT BUFFER SIZE</td>
</tr>
<tr>
<td>007E FA</td>
<td>553</td>
<td></td>
<td></td>
</tr>
<tr>
<td>007F 4302</td>
<td>554</td>
<td>Mov A.ChStrI</td>
<td>get the status byte</td>
</tr>
<tr>
<td>0080 53FB</td>
<td>555</td>
<td>ORL A.#CBFLn</td>
<td>set Char Buffer Full Line stat bit</td>
</tr>
<tr>
<td>0081 53FE</td>
<td>556</td>
<td>ANL A.#ClrCr</td>
<td>clear the CR/(LF) stat bit</td>
</tr>
<tr>
<td>0083 53FB</td>
<td>557</td>
<td>ANL A.#ClrCBR</td>
<td>reset CB Init bit: init CB reg on entry</td>
</tr>
<tr>
<td>0085 AA</td>
<td>558</td>
<td>Mov ChStrI.A</td>
<td>store the status byte</td>
</tr>
<tr>
<td>0086 FA</td>
<td>559</td>
<td>LeChar: JS2 ChStrI</td>
<td>get the status byte</td>
</tr>
<tr>
<td>0087 52E1</td>
<td>560</td>
<td>JS2 CBPadI</td>
<td>CB not full but CR/LF previously</td>
</tr>
<tr>
<td>0089 9AEF</td>
<td>561</td>
<td>ANL JS2</td>
<td>received so pad CB</td>
</tr>
<tr>
<td>008B 22</td>
<td>562</td>
<td>In A.DBB</td>
<td>read the Char</td>
</tr>
<tr>
<td>008C 537F</td>
<td>563</td>
<td>ANL A.#AscStp</td>
<td>strip off MBR</td>
</tr>
<tr>
<td>008D 566</td>
<td>564</td>
<td>Mov TmpRIO.A</td>
<td>temp save char</td>
</tr>
<tr>
<td>008F BA10</td>
<td>565</td>
<td>ORL JS2</td>
<td>output DDB ACK High</td>
</tr>
<tr>
<td>0091 03E0</td>
<td>566</td>
<td>Jmp JS2</td>
<td>output DDB ACK</td>
</tr>
<tr>
<td>0093 F697</td>
<td>567</td>
<td>ADD A.#ABCCpl</td>
<td>test for ASCII printable character</td>
</tr>
<tr>
<td>0095 049C</td>
<td>568</td>
<td>JC Asc1IC</td>
<td>test for Carriage Return</td>
</tr>
<tr>
<td>0097 97</td>
<td>569</td>
<td>Jmp ChrChk</td>
<td>jmp to service</td>
</tr>
<tr>
<td>0099 52E1</td>
<td>570</td>
<td>ASC1IC: ChrCl</td>
<td>clear carry flag</td>
</tr>
<tr>
<td>009B FB</td>
<td>571</td>
<td>Mov A.TmpRIO</td>
<td>get the char back</td>
</tr>
<tr>
<td>0099 A1</td>
<td>572</td>
<td>Mov @AdDrR1.A</td>
<td>load data memory w/Char</td>
</tr>
<tr>
<td>009A 4043</td>
<td>573</td>
<td>Jmp IBFSrv</td>
<td>test for FormFeed</td>
</tr>
<tr>
<td>009C FB</td>
<td>574</td>
<td>ChrChk: Mov A.TmpRIO</td>
<td>get the char back</td>
</tr>
<tr>
<td>009D 03F3</td>
<td>575</td>
<td>ADD A.#MCFCpl</td>
<td>test for Carriage Return</td>
</tr>
<tr>
<td>009F C6C3</td>
<td>576</td>
<td>JS2 ChrCl</td>
<td>if CR go service it</td>
</tr>
<tr>
<td>00A1 FB</td>
<td>577</td>
<td>Mov A.TmpRIO</td>
<td>get the char back</td>
</tr>
<tr>
<td>00A2 0319</td>
<td>578</td>
<td>ADD A.#EDFCpl</td>
<td>test for End Of File</td>
</tr>
<tr>
<td>00A4 9AFA</td>
<td>579</td>
<td>JSN ChrCl</td>
<td>if not EOF jmp to CB Pad</td>
</tr>
<tr>
<td>00A6 FB</td>
<td>580</td>
<td>Mov A.TmpRIO</td>
<td>if EOF, place it in CB</td>
</tr>
<tr>
<td>00A7 A1</td>
<td>581</td>
<td>Mov @AdDrR1.A</td>
<td>load data memory w/CR Char</td>
</tr>
<tr>
<td>00AB 048B</td>
<td>582</td>
<td>Jmp ExtSet</td>
<td>Exit</td>
</tr>
<tr>
<td>00AA 5F8</td>
<td>583</td>
<td>ChrChk: Mov A.TmpRIO</td>
<td>get the status byte</td>
</tr>
<tr>
<td>00A9 03F3</td>
<td>584</td>
<td>ADD A.#FEFCpl</td>
<td>test for FormFeed</td>
</tr>
<tr>
<td>00AD 9E11</td>
<td>585</td>
<td>JNZ CBPadI</td>
<td>if not FF Pad the CB</td>
</tr>
<tr>
<td>00AF C5</td>
<td>586</td>
<td>SEL RBD</td>
<td></td>
</tr>
<tr>
<td>00B0 FA</td>
<td>587</td>
<td>Mov A.GSR20</td>
<td>get the status byte</td>
</tr>
<tr>
<td>00B1 4304</td>
<td>588</td>
<td>ORL A.# FrmFnd</td>
<td>set the formfeed flag</td>
</tr>
<tr>
<td>00B3 AA</td>
<td>589</td>
<td>Mov GSR20,A</td>
<td>store the status byte</td>
</tr>
<tr>
<td>00B4 D5</td>
<td>590</td>
<td>SEL RBD</td>
<td></td>
</tr>
<tr>
<td>00B5 FA</td>
<td>591</td>
<td>Mov A.ChStrI</td>
<td>get the status byte</td>
</tr>
<tr>
<td>00B6 4304</td>
<td>592</td>
<td>ORL A.#CRLF</td>
<td>set CRLF stat bit pad balance of CB</td>
</tr>
<tr>
<td>00BB AA</td>
<td>593</td>
<td>Mov ChStrI.A</td>
<td>store the status byte</td>
</tr>
<tr>
<td>00BA FA</td>
<td>594</td>
<td>ExtSet: Mov A.ChStrI</td>
<td>get the status byte</td>
</tr>
</tbody>
</table>

5-784

230795-001
008A 4302 602 ORL A, #CBFLn set Char Buffer Full Line stat bit
008C 53FB 603 ANL A, #CrCrC clear the CR/(LF) stat bit
008E 53FE 604 ANL A, #C11CBR reset CB Init bit, init CB reg on entry
0090 AA 605 Mov ChStR1.A store the status byte
00C1 04EC 606 Jmp CBFlEx
0067
0068 Store CR char read in LF char (assume its always there) and ignore it
00C3 FB 608 Mov CRChar get the char back
00C4 A1 610 Mov @CAdrR1.A load data memory w/CR Char
00C5 C5 611 SEL RBO
00C6 8C 612 INC LncTR0 incr the line count
00C7 FE 613 Mov A, LncTR0 get the line count
00C8 03C4 614 Add A, #PLPCL set for page feed in cnt
00C9 03C1 615 JNC NoFrd if not at end of page skip
00CA E6D0 616 JB2 GStR20 get the status byte
00CD 4304 617 Mov A, #FrmFd get the form feed status flag
00CF A4 618 Mov GStR20.A save the status byte
00D0 D5 619 Mov @FrmFd. SEL R1
00D1 05 620 En I enable the IBF service
00D2 9ADF 621 Mov P2, #NotBsry output a not busy to Host
00D4 06D4 622 Mov LFTest: JNIBF LFTest loop to next char
00D6 9ACF 623 Mov P2, @Back output DB Ack low
00DB 22 624 In A, DBB if next char – assume it’s a LF
00D9 FA 625 Mov @StPad: Mov A, ChStR1 get the status byte
00DA 4304 626 ORL A, #CRLF set CRLF stat bit - pad balance of CB
00DC AA 627 Mov ChStR1.A store the status byte
00DD BA10 628 ORL P2, #ResBack output DB ACK High
00DF 04E3 629 Jmp IBFSrE jump to addr step 6 & exit
00E1 B120 630 Mov @CBPad: Mov A, #CAdrR1.#Space load data memory w/Char
00E3 C9 631 Mov @IBFSe: DEC #CAadr1 Decrease dat memory location
00E4 FA 632 Mov A, ChStR1 get the status byte
00E5 32EC 641 JB1 CBFull test for CB Full
00E7 32EC 642 JBF2 CBFlEx test for CB pad – exit w/Busy set
00E9 05 643 Mov EN I Set Busy Line Low – Not Busy
00EA 9ADF 644 Mov EN I output a not busy to Host
00EB 06D4 645 Mov @IBFSe: DEC #CAadr1 Decrease dat memory location
00ED B3 646 Mov EN I test w/ Busy Still set high
00F0 00 647 Mov CBFull: Ret
00F1 53 648 Mov CBFlEx: Ret
00F2 52 650 PG
00F3 54 651 L-to-R/R-to-L Carriage Stepper Motor Drive
00F4 55 652 and Line Printing
0100
0100 3622 660 SMDrv: JTO RAccel if Print Head at left drive right
0101 3622 661 JTO RAccel: if Print Head at left drive right
0102 FA 662 F==============================================================================
0103 S3BF 663 FACcel: L-to-R Accelerate Stepper Motor
0104 53BF 664 FACcel: L-to-R Accelerate Stepper Motor
0105 53DF 665 Mov A, GStR20 get the status byte
0107 4380 666 Mov A, #ClrSnk set not at speed flag = 0
0109 4301 667 Mov A, #NAtSpd set Not At Speed flag = 0
010B 53EF 668 ORL A, #Rdy Read set stop mtr. ready – Drive On
010D AA 669 ORL A, #LRPrnt set L-to-R print stat bit = 1
010E D5 670 ORL A, #AdRntN set A/D Init Not Done
010F FA 671 Mov GStR20.A store the status byte
0110 43BF 672 CRDr: SEL R8 set the Char Stat Reg Data Mem Addr
0111 4380 673 Mov A, #LRPrnt set L-to-R print bit
0112 AA 674 Mov R8 set the mode
0113 C3 675 Mov ChStR1.A save the Char Stat byte
0114 BB1 676 SEL R80
0115 FA 677 Mov R8 set the mode
0116 F0 678 Mov #TpRRO. #CPAdr set Phz Storage Addr pseudo reg
0117 AB 679 Mov #SelPhzRRO. #get stored CR last phase index addr
0118 680 Mov #PhzR30. A place last LF phase index addr in Phz Reg

0119 681

5-785
23079-001
0118 1B 683 ; Set up for next phase bit output before entering timing loops
0119 FB 684 INC PhR30 ; STEP PHASE DB ADDRESS
011A 321E 685 MOV A,PhR30 ; CHECK THE PHASE COUNT REG
011C 2440 686 JB2 IAFzP ; CHK FDR COUNT BIT ROLLOVER
011E B800 687 JMP SMDFit ; skip addr index reset
0120 2440 688 MOV PhR30,#$btCRP ; ZERO CR SM PHASE REGISTER
0122 FA 689 JMP SMDFit
0123 33BF 690
0125 330F 691 ; Rtes==Rto-L Accelerate Stepper Motor
0127 537F 692 RAccel
0129 FA 693
012D AA 694 ; Set the Reverse acceleration/drive Entry status bits
012F FA 695 Mov A,GSR20 ; get the status byte
0131 53BF 696 ANL A,$ClrsNk ; clear Print Ready bit
0133 530F 697 ANL A,$NatsPd ; set Not At Speed flag = 0
0135 537F 698 ANL A,$RLPnt ; set R-to-L print status bit
0137 4301 699 ORL A,Ready ; set strp mtr ready - Drive On
0139 53EF 700 ANL A,$ADlnTn ; set A/D Init Not Done
013B AA 701 MOV GSR20,A ; store the status byte
013D BB03 702 RCBRD$ SEL RBl
013F FA 703 Mov A,ChStr1 ; get the Char Stat Reg Data Mem Addr
0141 537F 704 ANL A,$RLPnt ; set R-to-L print bit
0143 AA 705 MOV ChStr1,A ; save the Char Stat byte
0145 C5 706 SEL RBO
0147 BB03 707 ; Restore the phase register index address
0149 230B 708 Mov TmpROO,WCPSadr ; get Phz Storage Addr pseudo reg
014B 3C 709 MOV A,$tmpROO ; get stored CR last phase index addr
014D 2221 710 SET up for next phase bit output before entering timing loops
014F BB02 712 Mov PhR30,A ; place last LF phase index addr in Phz Reg
0151 AB 714 Set up for next phase bit output before entering timing loops
0153 BB02 716 Mov TmpROO,$LastPh ; load Last Phz pseudo reg to Temp Reg
0155 330F 717 Mov A,$tmpROO ; store Last Phase bits - indirect
0157 BB03 719 SMDFit;
0159 33BF 720 ;
015B 330F 721 ; for stabilization of unused strp mtr during CR strp mtr drive.
015D 337F 722 ; store the unused strp mtr current phase bits
015F 4301 723 Mov TmpROO,WP$adr ; get the Phz storage addr
0161 53EF 724 Mov A,$tmpROO ; get the byte stored there
0163 4301 725 MovP3 A,EA ; get the phz data byte
0165 B820 726 Mov TmpROO,$LastPh ; load Last Phz pseudo reg to Temp Reg
0167 BB02 727 Mov A,$tmpROO ; store Last Phase bits - indirect
0169 BB03 729 Setup Strp Mtr Time Constant
016B 2308 730 MOV TConRO,#CrTerm$ ; Load time constant Reg
016D 3D 731
016F 3C 732 Select ; Select the Strp Mtr
0171 3D 733 MOV A,$SCRBO ; GET CR SM SELECT BITS
0173 3C 734 MOVDP A,; SELECT SM [SCRBO]
0175 33BF 735 ; Setup Strp Mtr Phase Shift index address register
0177 330F 736 ; Output next phase and init timer to Std Time constant
0179 537F 737 MOV A, TConRO ; get time constant from reg
017B 3C 739 MOV T,A ; load the timer
017D 3C 740 MOV A,PhR30 ; get the phz reg indirect addr index
017F E3 741 MovP3 A,EA ; do indirect get of phz bits
0181 3C 742 ;
0183 3C 743 ; patch together the CR last and LF next phase bits
0185 537F 744 Mov TmpROO,$LastPh ; load Last Phz pseudo reg to Temp Reg
0187 537F 745 MOV A,$tmpROO ; patch together CR existing & new LF
0189 3C 746 ORL A,$tmpROO ; patch together CR existing & new LF
018B 54 747 MOVDP A,4 ; OUTPUT BITS
018D E8 748 STRT T ; START TIMER
018F 749
0191 74C 750 ; At start of timing loop do all Strp Mtr Accel/Decel or
0193 74C 751 ; Character SetUp overhead
0195 74C 752 Call ADPTst ; call Accel/Decel/Print Test
0197 74C 753 ; Set up for next phase bit output before entering timing loops
0199 74C 754 PNrdyl ; test for forward / reverse phase start indirect index to load
019B FA 755 Mov A, GSR20 ; store stat byte
019D F264 756 JB7 ACF2
019F 757 758 ; reverse:
019F 759 ; reverse:
019F 760 ; Set up for next phase bit output before entering timing loops
01A1 CB 761 Dec PhR30 ; STEP PHASE DB ADDRESS
01A3 FB 762 MOV A,PhR30 ; CHECK THE PHASE COUNT REG
01A5 5260 763 JB2 ARzP ; CHK FDR COUNT BIT ROLLOVER
01A7 2462 764 JMP ARnSP
01A9 B803 765 ARzP MOV PhR30,#$btCRP ; ZERO CR SM PHASE REGISTER
01AB 246C 766 ARnSP JMP ANzPh

5-786
230795-001
was Print

0164 1B 770 AclF2: INC PhR30 ;STEP PHASE DB ADDRESS
0165 F8 771 MOV A,PhR30 ;CHECK THE PHASE COUNT REG
0166 52A 772 JB2 AF1roP ;CHK FOR COUNT BIT ROLLOVER
0168 246C 773 JMP ANtPh ;skip addr index reset
016A BB50 774 AF1roP: MOV PhR30.0#$FRoP ZERO CR SM PHASE REGISTER
775 ANtPh:

776 |-----------------------|
777 | stage one timer loop - T occurs before Std timeout |
778 | wait for time out |
779 |-----------------------|
016C 1682 779 TLOOP2: JTF FAILSF; JMP ON TIME-OUT t DOES NOT OCCUR 1ST
016E 5672 780 JT1 tCHK1: JS T HGH-JMP TO tCHK
0170 246C 781 JMP TLOOP2 ;LOOP FOR JT1 OR JT6
0172 00 782 tCHK1: NOP ;delay; then double check T signal
0173 5677 783 JT1 tTRuW1: ;JUMP T TEST TRUE-WAIT FOR JTF
0175 246C 784 JMP TLOOP2

785 tTRuW1:

786 |-----------------------|
787 | test for Print Ready bit - was Print Head Fire Setup Done? |
788 |-----------------------|
0177 FA 788 MOV A,GStr20 ;get the status byte - prep for print
0179 D27C 789 JB6 RdVPr2 ;if Ready Print bit set call PHFire
017A 247E 790 Jmp SkpPHF; else skip Print Head Fire
017C 74CA 791 RdVPr2: Call PHFire ;print head solenoid routine
017D 247D 792 PNtRd2:

793 SkpPHF:

794 |-----------------------|
795 |-----------------------|
017E 169B 794 tTRuW2: JTF NXPtR2; JUMP TO SM ERROR
0180 247E 795 JMP tTRuW2 ;LOOP TO TLOOP3

796 |-----------------------|
797 | Store in FailSafe/startup timer setup - T occurs but does not |
798 | time and wait for FailSafe timeout or T. If T occurs |
799 | output phase immediately after T verify |
0182 2300 800 FAILSF: MOV A,FAILTm ;LOAD TIMER W/15.0mS
0184 62 801 MOV T,A ;SM PROTECTION TIMEOUT
0185 55 802 STRT T ;START TIMER

803 |-----------------------|
804 |-----------------------|
0186 FA 806 MOV A,GStr20 ;get the status byte
0187 4308 807 ORL A,FSCTm ;set FailSafe/constant time flag
0189 A 808 MOV GStr20,A ;store the status byte
018A 5690 809 TLOOP3: JT6 tCHK2: JS T HGH |
018C 16AC 810 JT6 DBSLECT; IF TIME OUT GO SM ERROR |
018E 248A 811 JMP TLOOP3 ;LOOP UNTIL T HGH OR T-OUT |
0190 00 812 tCHK2: NOP ;WAIT |
0191 5695 813 JT1 StrTm1 ;jump out and store elapsed time |
0193 248A 814 JMP TLOOP3 ;JMP TO FAILSF LOOP |
0195 65 815 StrTm1: Stop TCnt ;stop the FailSafe Timer |
0196 42 816 MOV A,T ;read the timer |
0197 A1 817 MOV @TStrR0,A ;Store the time read in indexed addr |
0198 FA 818 |-----------------------|
0199 26AC 819 MOV A,GStr20 ;store stat byte |
0200 B7 820 JB7 FDrive |
0201 F27 821 Test 1s CR Stpr Mtr Drive is finished prior to next phase output |

822 |-----------------------|
823 NXPtR2: |

824 |-----------------------|
0199 FA 825 MOV A,GStr20 ;store stat byte |
0199 F27 826 JB7 FDrive |
0207 |-----------------------|
0198 26AC 827 Reverse -- test for Reverse Stpr Mtr Drive procedure exit |
0208 |-----------------------|
0193 124C 828 ALWAYS drive the CR to the left most HOME position |
0195 FA 829 JNT0 EDLn | test if home position jmp stop |
019D FA 830 MOV A,GStr20 ;get the status byte |
019E 124C 831 JB0 StrT |

832 |-----------------------|
01A0 4302 832 |-----------------------|
01A2 53BF 833 ORL A,#DNoToP ;set the do not print flag |
01A3 26AC 834 MOV GStr20,A ;save the status byte |
01A5 244C 835 Jmp StrT ;continue CR SM drive |
01A7 FA 836 |-----------------------|
01AB 26AC 837 |-----------------------|
01A1 F2B3 838 Forward -- test for Forward Stpr Mtr Drive procedure exit |
01A7 FA 839 FDrive: |

840 |-----------------------|
01A8 124C 840 MOV A,GStr20 ;get the status byte |
01A9 244C 841 JB0 StrT |

842 |-----------------------|
01A1 5437 843 |-----------------------|
01A5 4302 844 |-----------------------|
01AC 53BF 845 |-----------------------|
01AE FA 846 MOV GStr20 ;store stat byte |
01AF F2B3 847 |-----------------------|

5-787

230795-001
Start: 0x0000

INSTRUCTIONS:

1. Initialize the stepper motor.
2. Set the initial speed constant.
3. Set the print flag.
4. Load the print flag.
5. Load the speed constant.
6. Load the status byte.
7. Load the print flag.
8. Load the speed constant.
9. Load the status byte.
10. Load the print flag.
11. Load the speed constant.
12. Load the status byte.
13. Load the print flag.
14. Load the speed constant.
15. Load the status byte.
16. Load the print flag.
17. Load the speed constant.
18. Load the status byte.
19. Load the print flag.
20. Load the speed constant.
21. Load the status byte.
22. Load the print flag.
23. Load the speed constant.
24. Load the status byte.
25. Load the print flag.
26. Load the speed constant.
27. Load the status byte.
28. Load the print flag.
29. Load the speed constant.
30. Load the status byte.
31. Load the print flag.
32. Load the speed constant.
33. Load the status byte.
34. Load the print flag.
35. Load the speed constant.
36. Load the status byte.
37. Load the print flag.
38. Load the speed constant.
39. Load the status byte.
40. Load the print flag.
41. Load the speed constant.
42. Load the status byte.
43. Load the print flag.
44. Load the speed constant.
45. Load the status byte.
46. Load the print flag.
47. Load the speed constant.
48. Load the status byte.
49. Load the print flag.
50. Load the speed constant.
51. Load the status byte.
52. Load the print flag.
53. Load the speed constant.
54. Load the status byte.
55. Load the print flag.
56. Load the speed constant.
57. Load the status byte.
58. Load the print flag.
59. Load the speed constant.
60. Load the status byte.
61. Load the print flag.
62. Load the speed constant.
63. Load the status byte.
64. Load the print flag.
65. Load the speed constant.
66. Load the status byte.
67. Load the print flag.
68. Load the speed constant.
69. Load the status byte.
70. Load the print flag.
71. Load the speed constant.
72. Load the status byte.
73. Load the print flag.
74. Load the speed constant.
75. Load the status byte.
76. Load the print flag.
77. Load the speed constant.
78. Load the status byte.
79. Load the print flag.
80. Load the speed constant.
81. Load the status byte.
82. Load the print flag.
83. Load the speed constant.
84. Load the status byte.
85. Load the print flag.
86. Load the speed constant.
87. Load the status byte.
88. Load the print flag.
89. Load the speed constant.
90. Load the status byte.
91. Load the print flag.
92. Load the speed constant.
93. Load the status byte.
94. Load the print flag.
95. Load the speed constant.
96. Load the status byte.
97. Load the print flag.
98. Load the speed constant.
99. Load the status byte.
100. Load the print flag.
101. Load the speed constant.
102. Load the status byte.
103. Load the print flag.
104. Load the speed constant.
105. Load the status byte.
106. Load the print flag.
107. Load the speed constant.
108. Load the status byte.
109. Load the print flag.
110. Load the speed constant.
111. Load the status byte.
112. Load the print flag.
113. Load the speed constant.
114. Load the status byte.
115. Load the print flag.
116. Load the speed constant.
117. Load the status byte.
118. Load the print flag.
119. Load the speed constant.
120. Load the status byte.
121. Load the print flag.
122. Load the speed constant.
123. Load the status byte.
124. Load the print flag.
125. Load the speed constant.
126. Load the status byte.
127. Load the print flag.
128. Load the speed constant.
129. Load the status byte.
130. Load the print flag.
131. Load the speed constant.
132. Load the status byte.
133. Load the print flag.
134. Load the speed constant.
135. Load the status byte.
136. Load the print flag.
137. Load the speed constant.
138. Load the status byte.
139. Load the print flag.
140. Load the speed constant.
141. Load the status byte.
142. Load the print flag.
143. Load the speed constant.
144. Load the status byte.
145. Load the print flag.
146. Load the speed constant.
147. Load the status byte.
148. Load the print flag.
149. Load the speed constant.
150. Load the status byte.
151. Load the print flag.
152. Load the speed constant.
153. Load the status byte.
154. Load the print flag.
155. Load the speed constant.
156. Load the status byte.
157. Load the print flag.
158. Load the speed constant.
159. Load the status byte.
160. Load the print flag.
161. Load the speed constant.
162. Load the status byte.
163. Load the print flag.
164. Load the speed constant.
165. Load the status byte.
166. Load the print flag.
167. Load the speed constant.
168. Load the status byte.
169. Load the print flag.
170. Load the speed constant.
171. Load the status byte.
172. Load the print flag.
173. Load the speed constant.
174. Load the status byte.
175. Load the print flag.
176. Load the speed constant.
177. Load the status byte.
178. Load the print flag.
179. Load the speed constant.
180. Load the status byte.
181. Load the print flag.
182. Load the speed constant.
183. Load the status byte.
184. Load the print flag.
185. Load the speed constant.
186. Load the status byte.
187. Load the print flag.
188. Load the speed constant.
189. Load the status byte.
190. Load the print flag.
191. Load the speed constant.
192. Load the status byte.
193. Load the print flag.
194. Load the speed constant.
195. Load the status byte.
196. Load the print flag.
197. Load the speed constant.
198. Load the status byte.
199. Load the print flag.
200. Load the speed constant.
201. Load the status byte.
202. Load the print flag.
203. Load the speed constant.
204. Load the status byte.
205. Load the print flag.
206. Load the speed constant.
207. Load the status byte.
208. Load the print flag.
209. Load the speed constant.
210. Load the status byte.
211. Load the print flag.
212. Load the speed constant.
213. Load the status byte.
214. Load the print flag.
215. Load the speed constant.
216. Load the status byte.
217. Load the print flag.
218. Load the speed constant.
219. Load the status byte.
220. Load the print flag.
221. Load the speed constant.
222. Load the status byte.
223. Load the print flag.
224. Load the speed constant.
225. Load the status byte.
226. Load the print flag.
227. Load the speed constant.
228. Load the status byte.
229. Load the print flag.
230. Load the speed constant.
231. Load the status byte.
232. Load the print flag.
233. Load the speed constant.
234. Load the status byte.
235. Load the print flag.
236. Load the speed constant.
237. Load the status byte.
238. Load the print flag.
239. Load the speed constant.
240. Load the status byte.
241. Load the print flag.
242. Load the speed constant.
243. Load the status byte.
244. Load the print flag.
245. Load the speed constant.
246. Load the status byte.
247. Load the print flag.
248. Load the speed constant.
249. Load the status byte.
250. Load the print flag.
251. Load the speed constant.
252. Load the status byte.
253. Load the print flag.
254. Load the speed constant.
255. Load the status byte.
Carriage Stepper Motor Deceleration

Dec 0

Set the Deceleration registers

Mov TstrR0, #SMEnd ;Load the Stpr Mtr Buffer End Addr

0237 B925

Mov CntR40, #DSBFSz ;Load the Buffer Size

0239 BCOA

Mov A, PhzR30 ;get phase index address

023B FA

MovP3 A, #8A ;get phase from indexed address

023C E3

Patch together the CR last and LF next phase bits

023D BB20

Mov TempROO, #LastPh ;load Last Phz psuedo reg to Temp Reg

023F 40

Orl A, TempROO ;patch together CR existing & new LF

0240 3C

Mov0 P0, #A ;OUTPUT BITS

0241 F1

SttR0D: Mov A, #BStrR0 ;get time from indexed data memory

0242 42

Mov T, #A ;load timer

0243 E5

SttR0 T ;START TIMER

0244 19

Inc TstrR0 ;step the Memorized time addr index reg

0245 FA

Test for forward / reverse phase start indirect index to load

0246 F252

Mov A, #BStR20 ;store stat byte

024F 3F2

Dec1ISM;

0243

Dec up

024B CB

Set up for next phase bit output before entering timing loops

Dec PhzR30 ;decrement the phase addr

0249 FA

Mov A, PhzR30 ;get the phz data addr

024A 524E

Jmp DntPh ;CHK FOR COUNT BIT ROLLOVER

024C 445A

DntPh ;skip addr index reset

024E B803

DrzrOp Mov PhzR30, #StCRP ;ZERO CR SM PHASE REGISTER

0250 445A

DntPh ;set up for next phase shift

0257 1B

Load Last Phase

0252 18

DciR2: Mov A, PhzR30 ;get phase from indexed address

0253 FB

Patch together the CR last and LF next phase bits

0254 445B

Mov A, #PhzR30 ; Get the phz data addr

0256 445A

Jmp DntPh ;CHK FOR COUNT BIT ROLLOVER

0259 B800

DntPh, Mov PhzR30, #StCRP ; ZERO CR SM PHASE REGISTER

025B 2E00

DntPh ;set up for next phase shift

025C BB20

Mov TempROO, #LastPh ;load Last Phz psuedo reg to Temp Reg

025E 40

Orl A, TempROO ;patch together CR existing & new LF

025F 1663

TLoopD: Jtf NxtPD2 ;JMP ON TIME OUT TO NEXT PH

0261 445F

Jmp TLoopD ;LOOP UNTIL TIME OUT

0263 3C

Mov0 P0, #A ;OUTPUT BITS

0264 EC41

DntZ; CntR40, StrtR0 ;Exit Test

0267

Set Storage of next phase data in psuedo addr. This insures

0269

next phase is sequence correct for step mtr drive direction

0266 BB21

SetR0N: Mov TempROO, #CPSAddr ;get Phz Storage Addr psuedo reg

0268 FA

Mov A, PhzR30 ;get Phz data

0269 A0

Mov #TempROO, A ;Store CR Next phase index addr

026A B478

DmExit Call DiLyng

026C B490

Call De1ISM

026E B3

Ret

0275

PG

Stepper Motor Phase Shift Definitions

All program procedures call this data

0300

Org 300H

0301

Define Phase Addresses

The phase data is encoded to the address called during the

0305

Stpr mtr energize sequence corresponding to the next phase

0307

of the sequence required.

0309

Carrage Motor encoding FORWARD - Left-to-Right

0310

Reverse - Right-to-Left

5-789
Reverse direction ENCODING is the same bytes accessed in reverse direction.

**Reverse direction ENCODING**

**LF MOTOR PHASE ENCODE & DECODE. FORWARD (CLOCKWISE)**

**Forward direction ENCODING:**

- 0310 04: DB LFMFP1
- 0309 08: DB LFMFP2
- 0308 08: DB LFMFP3
- 0308 00: DB LFMFP4

**Accel/Decel / Character Handling Test**

**Process Characters for Printing**

- 0310 2668: Character dot matrix - normal char
- 0315 D21B: d = Dot Column
- 0315 D21B: b = Blank Column
- 0315 D21B: b d d d d
- 0315 D21B: (Char Matrix)
- 0315 D21B: 0 0 0 0 b
- 0315 D21B: 0 0 0 1 d
- 0315 D21B: 0 0 1 0 d
- 0315 D21B: 0 0 1 1 d
- 0315 D21B: 0 1 0 0 d
- 0315 D21B: 0 1 0 1 d

**Call for Individual character processing: mid line test if CR/(LF)**

- 0310 266B: test for CR/(LF) if it is the test position in the line
- 0310 266B: if CR go service it
- 0310 266B: if bit 7 = 1 then Print L-to-R
if L-to-R printing exit line if less than 1/2 line printed

load char cnt reg w/chbuf size

add the 2's cpl of 1/2 char buf size

if CD1/2 full set CR/LF stat bit for pad

if CBC1/2 set buffer full stat bit

mid-line exit

clear carry flag

insert a space char

char inserted jmp over get char

fetch the char dot column data

page test for balance of char

fix jmp over page boundaries

Ascii char 50 - 7F hex

jump to Matrix Test

Ascii char 20 - 4F Hex

fall thru to print matrix

and CB count tests

test the Char dot column print matrix count and Char buffer count

test for dot col or blank

status byte in A upon entry here

get the status byte

set Char Init NotDone stat flag

store the status byte

dec char cnt-jmp if Not Last Char

set CB Reg Init flag - do Init

save the status byte

get Gen Status register addr

clear the ready bit

store the General Status Byte

fall thru to Get Char

A contains LR/RL bit properly set

get char status register addr

test Char Stat Byte Returned

increment char data memory addr

Decrement char data memory addr

fall thru to Get Char

Re-Entry Exit point for same char:
(before returning step the matrix)

test for L-to-R (forward) or R-to-L (reverse) printing

(see GChar1 ASCII char code translation procedure)

Test for L-to-R (forward) or R-to-L (reverse) printing

(see GChar1 ASCII char code translation procedure)


0368 C5 0369 B3 036A D9 036B FA 036C F27C 036E C5 0370 53BF 0372 B3 0373 D27C 0375 4340 0376 AA 0377 B7 0378 B007 0379 44B8 037C E888 037F FA 0380 FA 0381 AA 0382 C5 0383 FA 0384 53FE 0386 AA 0387 B3 0388 C5 0389 B3 038A FA 038B 53FD 038D 53FE 038F AA 0390 C5 0391 FA 0392 4302 0394 53BF 0396 AA 0397 B3 0398 AE 0399 03E0 0399 F69F 039D 46C9 039F 97 03A0 FE 03A1 0380 03A3 F6AE 03A5 FA 03A6 4320 03A8 AA 03A9 FE 03AA 03E0 03AC 64B8

1170: Character Print Set Up Exit Procedures
1171: Clean Standard Exit
1176: Ret: SEL RBO
1177: Exit - return w/ Reg Bank 0 Reset
1178: Do Not Print exit. set Stpr Mtr drive routine count loop
1179: NPrEt: SEL RBO
1180: Mov A.ChStr1A get the status byte
1181: JB7 SkipNPI test print direction
1182: Reverse
1183: SEL RBO
1184: Mov A.GStr20 get the status byte
1185: ANL A.#CIRSnk reset the print ready bit - skips PHFire call
1186: Ret
1187: Forward
1188: JB6 SkipNPI test for first PHFSet entry reg init
1189: Initialize register variables upon first entry
1190: End of count clears char to print bit in status byte
1191: ORL A. #CntnD set Char Reg Init Done stat bit
1192: Mov ChStr1.A save the status byte
1193: Mov TmpRIO. #07H load CR stpr mtr count during NPrnt
1194: JMP NPExit
1195: DJNZ TmpRIO. NPExit
1196: Mov A.ChStr1 get the status byte
1197: ANL A. #CntnD reset - char init not done
1198: Mov ChStr1.A save the status byte
1199: SEL RBO
1200: Mov A.GStr20 get Gen Status register addr
1201: ANL A. #NotRdy' clear the ready bit
1202: Mov GStr20.A store the General Status Byte
1203: NSetEx Ret
1204: NPExit: SEL RBO
1205: Ret
1206: Mid-Line Exit
1207: EXIT - if CR and not > 1/2 line done during L-to-R print
1208: ORL A. #CntnD get the status byte
1209: ANL A. #C0Fbit if 0 reset status bit Not CB Full Line
1210: Mov A.CStr2L set CB Reg Init Flag - do Init
1211: Mov ChStr1.A save the status byte
1212: SEL RBO
1213: Mov A.GStr20 get the RBO status byte
1214: Mov A. #DStrP set the Do Not Print Flag(for RAcel)
1215: ORL A. #DStrP set the Do Not Print Flag(for RAcel)
1216: Mov A. #ChStr save the status byte
1217: Mov GStr20.A save the status byte
1218: Mov A. #ChStr save the status byte
1219: Mov A. #CIRSnk reset the print ready bit-exit FAcel
1220: MOV StrCRI.
1221: ; STORE THE CHAR
1230: screen for printable char [char +(cpl 20 Hex + 1 = EO Hex)]
1231: ADD A. #EOOH
1232: JC PrintCH: CntlCh jmp to control char lookup table
1233: JMP CntlCh
1234: PrintCH: C1r C clear carry flag
1235: MOV A. #ChStr1 get the char again
1236: MOV A. #ChStr1 get the char again
1237: screen for char page [char +(cpl 50 Hex + 1 = B0 Hex)]
1238: if carry char on page 2 else page 1
1239: ADD A. #B0OH
1240: JC Page2
1241: Page 1 Character -- ASCII 20 Hex thru 4F Hex
1242: Correct offset for lookup table page
1243: (char + EO Hex) * 5 = Page 1 indexaddr
1244: 5-792
03A2 D5 1299 PHFIRE: SEL RB1
03C8 FB 1300 Mov A,CDRC1 ; set the chot dot column cnt
03CC 96D2 1301 JNZ FIPE ; if char cnt not 0 - Fire Head Sol.
03C2 3A 1302 Jmp Retrni ; if Char Dot Cnt 0, reset the
03C3 B806 1303 SETCNT: Mov CDRC1,NORCCT ; char dot column count
03C4 64D8 1304 Jmp Retrn1 ; skip PH Fire
03D2 B3A0 1305 Fire, MOV A,#PTRO Lo ; get the Print Head Trigger byte
03D4 3A 1306 OUT P2,A ; FIRE PRINT HEAD
03D5 23A0 1307 MOV A,#PTrght ; get the Print Head Trigger byte
03D7 3A 1308 OUT P2,A ; FIRE PRINT HEAD
03D8 C5 1309 RETRNI: SEL RB0
03D9 B3 1310 RET A, :EXIT - return w/ Reg Bank 0 Reset
1311 1312 ; PG
1313 ; PaperFeed Stgr Mtr Drive
1314 ;
1315 ***********
1316 1317 ORG 400H
1318
1319
1320
1321 Init psuedo register with LF indirect addr start - subsequent exchanges of the psuedo register will yield correct value
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332 LineFeed / FormFeed Drive

5-793
040B BC1B
040D FA
0410 BE01
0412 B41B
0414 FE
0416 CF
041B 0342
041A AE
041B BB21
041D F0
041E E3
041F BB20
0420 BB22
0422 BD9B
042B 2306
042A 3D
042B F8
042C E3
042D BB20
042F 40
0430 3C
0431 FD
0433 55
0435 FB
0438 523A
043B 843C
043A BB08
043C FB
043D E3
043E BB20
0440 40
0441 1645
0443 BD41
0445 3C
0446 EC31
044B BC1B
044A EE31
044C FA
044D 5FB
044F AA
0450 BB22

LineFeed/FormFeed Test

MOV CntR40.ULPIBpB ;init cnt reg for standard line feed

for stabilization of unused stpr mtr during CR stpr mtr drive.

store the unused stpr mtr current phase bits

MOV PhzR30,A ;get the phase register addr index

MOV P3A,8A ;do indirect get of phz bits

MOV TmpROO.ULPIBpB ;load Last Phz psuedo reg to Temp Reg

MOV TmpROO.A ;store last phase bits - indirect

exchange/store the phase register index addresses

MOV TmpROO.ULPIBpB ;get Phz Indirect Addr psuedo reg

MOV TmpROO.A ;get LF last phase index addr

MOV PhzR30,A ;place last LF phase index addr in Phz Reg

MOV TconRO.ULPIBpB ;Load time constant Reg

Select the Stpr Mtr

MOV A.ULPIBpB ;GET CR SM SELECT BITS

MOV P5.A ;SELECT SM (SCR80)

End of line spacing would go here

Load time constant Reg

Start of phase index address

start timer and step motor

MOV A.ULPIBpB ;get the phase reg indirect addr index

MOV PhzR30,A ;get the phase register index addresses

MOV TmpROO.ULPIBpB ;load Last Phz psuedo reg to Temp Reg

MOV TmpROO.A ;patch together CR existing & new LF

MOV P4.A ;OUTPUT BITS

MOV P5.A ;SELECT SM (SCR80)

MOV TconRO.A ;Load time constant Reg
; get the phase index address
2003 B478 0452 FB
2004 B490 Mov A,PhiR30
2005 0453 AO
2006 0454 A017 Mov @TmpROO.A
2007 0455 A418 Call DlyLng
2008 0456 B319 Call DrsGSM
2009 
2010 ; Check if Char Buffer contains full line (80 char or nChar & CR)
2011 ; exit otherwise continue to read in characters
2012 ; get the status byte
2013 ; if Do Not Print Bit Set - EXIT
2014 ; Call CBFCk
2015 0458 B3 0452 FB
2016 ByPasL: Ret
2017 
2018 ; PG
2019 ; Minor Software Subroutines
2020 ; ORG $500H
2021 ; System initialization
2022 ; subroutine
2023 ; Default:
2024 ; set/reset EDF status flag bit = 0
2025 0500 D5
2026 0501 FA
2027 ; get the char status byte
2028 ; clear the EDF flag bit
2029 ; store the char status byte
2030 ; get the Ascii code tmp store addr
2031 ; load the tmp stor reg w/ascii start
2032 
2033 ; reset/set Ok-to-print status flag bit = 0
2034 0506 B3
2035 ; get the status byte
2036 ; reset print flag - Qk Print
2037 ; save the status byte
2038 
2039 ; InitAl:
2040 ; 110ff:
2041 ; CLEAR all outputs
2042 ; FORCE PORT HI - R/ OF 555
2043 ; TURN ALL PRN'T SOL's OFF
2044 ; print head fire tirgger inactive
2045 ; clear the status registers
2046 ; ink the logical left home CR position
2047 ; delay a long time before continuing
2048 ; RETURN TO INIT ROUTINE
2049 
2050 ; Home Carriage / Print Head Assembly
2051 ; get the status byte
2052 ; set the do not print flag
2053 ; save the status byte
2054 ; test for position of PH assembly
2055 ; drive accordingly
2056 ; idrive CR Stpr Mtr
2057 ; find the logical left home CR position
2058 ; delay a long time before continuing
2059 ; ZERO MEMORY LOCATION
2060 
2061 ; Clear Data Memory
2062 ; At PowerUp or Reset, following CR & LF Stpr Mtr Init, this
2063 ; procedure clears data memory above RBO. Stack and RBO.
2064 ; GET BUFFER START LOCATION [HEX]
2065 052F B87F 052E B3
2066 C1rDM: Mov RO, #DMTop
2067 0531 B95D Mov R1, #DMSize
2068 0533 B000 C1rDM1: Mov R0, #DMSize

5-795
0535 CB 1497 DEC RO ;dec buffer, loop if not zero[end]
0536 E933 1498 DJNZ Ri.ClrDMi ;RETURN TO INIT ROUTINE
0538 B3 1499 RET
1500
1501 ; PG
1502 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
1503 ; Character Print TEST
1504 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
1505 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
1506 PrnTst:
1507 ; TEST --- load the char buffer with successive increments of
1508 ; the ascii code start, test for end of ascii
1509 ; printable chars and reinit the char stream loaded.
1510
1511 0539 B97F 2111 CTInt: Mov CadrR1.#FChFSt :load char reg w/char buff strt
1512 053B BD50 2112 Mov CCntR1.#ChBSz :load char cnt reg w/char buff size
1513 053C C9 2113 ChSts: Test char buffer fill with ASCII Char Code
1514
1515 053D FF 2114 Mov Aapr71 :get the ascii char
1516 053E A1 2115 Mov ECAdr1.A :load data memory w/Char
1517 053F C9 2116 DEC CadrR1 :Increment data memory location
1518 0540 1F 2117 INC opnR71 :Increment Ascii char number
1519 0541 03B2 2118 ADD A.:#PsEnd :test for ascii code end
1520 0542 847 2119 JNZ ChTyng :if not end jmp over code restart
1521 0545 BF20 2120 Mov OpnR71.#Ascii
1522 0547 ED3D 2121 ChrTgo: DJNZ CCntR1.ChTst :dec buffer, loop if not zero[end]
1523 0549 C5 2122 SEL RBO
1524 054A B3 2123 RET :ELSE RETURN TO INIT ROUTINE
1525 ; PG
1526 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
1527 ; CR Stpr Mtr Power On Initialization and
1528 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
1529 ; This routine drives the CR or LF stpr mtr for four phase
1530 ; shifts for initialization.
1531
1532 0548 BC04 2132 MOV CntR40.#PhCnt1 :load phase cnt reg for INIT
1533 054D 2308 2133 MOV A.:#ScrR0 :GET CR SM SELECT BITS
1534 054F 3D 2134 MOV DvP5.A :SELECT SM (*SCRBO*)
1535 0550 BD0C 2135 MOV TConR0.#Intm2 :Load time constant Reg
1536 0552 BB00 2136 MOV PhzR30.#FStCrP :zero SM phase reg - forward
1537 0554 FB 2137 MOV A.:PhzR30 :get phase index register byte
1538 0555 E3 2138 MovP3 A.:#A :load indexed phase shift byte
1539 0556 3C 2139 MovD P4.A :OUTPUT BITS
1540 0557 FD 2140 STTRT: MOV A.:TConR0 :GET TIMER CONSTANT
1541 0558 62 2141 MOV DvT.A
1542 0559 S5 2142 STRT T :START TIMER
1543 055A 1B 2143 INC PhzR30 :step phase index register
1544 055B FB 2144 MOV A.:PhzR30 :CHECK THE PHASE COUNT REG
1545 055C 3260 2145 JB2 ZroRg2
1546 055E A462 2146 JMP NxtPhR
1547 0560 B800 2147 Mov PhzR30.#FStCrP :zero SM phase reg - forward
1548 0564 NtPhR: MOV A.:PhzR30 :get phase index register byte
1549 0562 FB 2149 MOV DvP3 A.:#A :load indexed phase shift byte
1550 0563 E3 2150 MovP3 A.:#A :load indexed phase shift byte
1551 0564 1669 2151 TLooP: JTF NXPRH1 :JMP ON TIME OUT TO NEXT PH
1552 0566 A464 2152 JMP TLooP :LOOP UNTIL TIME OUT
1553 0568 3C 2153 MovD P4.A :OUTPUT BITS
1554 0569 EC57 2154 NXPRH1: DJNZ CntR40.STRTTR
1555 ;-----------------------------------------------------------------------
1556 ; store the last phase register index addresses
1557 ;-----------------------------------------------------------------------
1558 056B BB21 2157 Mov TapR00.#CPSaRd :Get Phz Storage Addr pseudo reg
1559 056D FB 2158 Mov A.:PhzR30 :place last CR phase index addr in Phz Reg
1560 056E A0 2159 Mov -TapR00.A : store CR last phase index addr
1561 056F B47B 2160 Call Dlyng
1562 0571 B490 2161 Call DeSiSM
1563 0573 B3 2162 RET
1564 ; PG
1565 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
1566 ; Time Delay Subroutines
1567 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
1568 ;-----------------------------------------------------------------------
1569
1570 0574 B87F 2170 DlyVlg: MOV TapR00.#7FH :LOAD DELAY COUNT IN REG.
1571 0576 A47E 2171 Jmp DlyST
1572
1573 ;-----------------------------------------------------------------------
1574 ; Very Long
1575 0574 B880 2174 DlyLng: MOV TapR00.#D1yCL :LOAD DELAY COUNT IN REG.
1576 057A A47E 2175 Jmp DlyST
1577
1578 ;-----------------------------------------------------------------------
Character Table Page 1, contains

Character Dot Generator Look-up Table Page 1

Character Dot Pattern Fetch

Page 1 -- Character Dot Pattern Fetch

Listing below is for reference only, actual code is not listed at assembly time.
Character Dot Pattern Fetch

Character Dot Pattern Fetch

Character Table Page 2, contains:

"NOPQRSTUVWXYZ123_@abcdfghijklmnopqrstuvwxyz!@"
Character Dot Pattern Fetch

this bit fix necessary to not underline each character
this saves fixing each bit in the look up table

char bit fix
output the dot pattern
exit with byte in acc

Program End

ASSEMBLY COMPLETE, NO ERRORS
APPENDIX B.
SOFTWARE PRINTER ENHANCEMENTS

This section describes several software enhancements which could be implemented as additions to the software developed for this Application Note. Space is available for most of the items described. Approximately 5 bytes of Data Memory would be required to implement most of the features. Two bytes would be used for status flags, and two bytes for temporary data or count storage. It is possible to use less than five bytes, but this would require the duplicate use of some flags, or other Data Memory storage, which will significantly complicate the software coding and debug tasks.

Special Characters or Symbols

Dot matrix printing lends itself well to the creation of custom characters and symbols. There are two aspects to implementing special characters. First, a character look-up table, and second, additional software for decoding and processing the special characters or symbols. Special characters might be scientific notation, mathematical symbols, unique language characters, or block and line graphics characters.

The character look-up table could be an additional page of Program Memory dedicated to the special characters, or replace part, or all, of the existing look-up tables. If an additional look-up table is used, a third page test would be needed at the beginning of the Character Translation subroutine. There is fundamentally no difference between the processing of special characters and standard ASCII printable characters. If the characters require the same 5 x 7 dot matrix, the balance of the software would remain the same. If, however, the special characters require a different matrix, or the manipulation of the matrix, the software becomes more complex.

In general, the major software modification required to implement special characters is the size of the dot matrix printed or the dot matrix configuration used. In the case of scientific characters, it would often be necessary to shift the 5 x 7 matrix pattern within the available 9 x 9 matrix. Block or line graphics characters, on-the-other-hand, would require using the entire 9 x 9 print head matrix and printing during normally blank dot columns. This would require suspending the blank column blanking mechanism implemented in this Application Note. This would be the most complex aspect of implementing special characters. It would possibly change the number of required instructions, and thus the timing between PTS detection and print head solenoid trigger firing. This could cause the dot columns to be misaligned within a printed line and between lines.

In the case of a matrix change, two approaches are possible: dynamically changing the matrix, in line, as standard ASCII characters are being printed, or isolating the special characters to a separate processing flow where special characters are handled as a unique and complete line of characters only. A discussion of inline matrix changes for special characters is beyond the scope of this Appendix. It is sufficient to say that the changes would require the conditions setting the EOLN flag, character count, and dot column count software be modified during character processing and printing.

Lower Case Descenders

The general principle of implementing lower case descenders is to shift the 5 x 7 character dot matrix within the available 9 x 9 print head solenoid matrix. Implementing lower case descenders requires two software modifications and the creation of status flag for the purpose. First, the detection of characters needing descenders and setting a dedicated status flag during the character code to dot pattern translation subroutine. Second, the character dot column data output to the print head solenoids must be shifted for each dot column of the character. At the end of the character, the flag would be reset.

Inline Control Codes

Inline control codes are two to three characters in length, which indicate special hardware conditions or software flow control and branching. The first character indicates that the control code sequence is beginning and is typically an ASCII Escape character (ESC), 1BH. Termination of the inline code sequence would be indicated by a default number of code sequence characters. This would decrease the buffer size available for characters. Full 80 character line buffering would require loading the Character Buffer with a received character as a character is removed from it and processed. The Inline Control Code test would be performed in two places: in the Character Buffer Fill subroutine and in the Character Processing (translation) subroutine. The test would be performed in the same manner that a Carriage Return (CR) character code test is implemented. Examples are horizontal tabs and expanded or condensed character fonts. In the case of horizontal tabs, 2OH (Space Character) would have to be placed in the Character Buffer for inline processing during character processing and printing. Unless fixed position tabs are used, a minimum of a nibble of Data Memory would be required to maintain a "spaces-to-tab" count. Fixed tab positions could be set via another inline control code, by default of the printer software, or through the use of external hardware switch settings. The control code method of setting the tab positions is the most desirable, but the most complex to implement.

Different Character Formats

Figure B1 illustrates three different character fonts; standard, condensed, and enlarged or expanded characters. As the the figure illustrates, condensed and
enlarged characters are variations in either the number of dots and/or the space used to print them. Thus, each character is a variation of the stepper motor and/or print head solenoid trigger timings. Figure B2 illustrates the timings required to implement the additional character printing.

In addition to the three character fonts shown, it is possible to print each in bold face by printing each dot twice per dot column position. This would require little software modification, but would require a status flag. Again, care must be used to ensure that the delay in retriggering the solenoids is precisely the same for each type of event. Without this precise timing the dot column alignment will not be accurate. The software modifications needed to implement enlarged or condensed characters is essentially the same. The carriage and print head solenoid firing software flow is the same, but the timing for each changes. For condensed characters, the step Time Constant is doubled to approximately 4.08 ms, and the solenoids are fired four times within each step time. The step rate actually becomes a multiple of the solenoid firing time, and a counter incrementing once for each solenoid firing would be needed. At the count of four, the carriage stepper motor is stepped and the counter reset.

In the case of condensed characters, PTS does not play the same roll as in standard or enlarged character printing. PTS is not used to indicate the optimum print head solenoid firing time. Solenoid firing is purely a time function for condensed characters. PTS would only be used for Failsafe protection.

Enlarged characters would require the solenoids be fired twice per dot column data, in two sequential dot columns, at the same rate as standard characters. The character dot column data and dot column count would not be incremented at each output but at every other output. A flag could be used for this purpose.

When printing either condensed or enlarged characters, the maximum character count would have to compensate for the increased or decreased characters per line count. When printing enlarged characters, the maximum characters per line would be 40. The Character Buffer could hold two complete lines of characters. But, condensed characters presents a quite different situation. The available character per line increases to 132, well beyond the 80 character Character Buffer size. The solution is to re-initialize the Character Buffer Size Count register count during condensed character processing. This will effectively inhibit the carriage stepper motor drive EOLN detection.

Two status flags would be required; one for standard or enlarged characters, and the second for condensed characters. A third status flag would be required to implement bold face printing. Activating one of the alternate character fonts could be either through the use of external status switches or through inline control code sequences, as detailed above. Note, that if the alternate character fonts are implemented in such a way that format changing is to occur dynamically during any single line being printed, the same control code problems described above also apply. In addition, the effect on the timing and dot column alignment must also be investigated.

**Variable Line Spacing**

Variable line spacing is another feature which could be implemented either through the use of external status switches or inline control codes. The line spacing is a function of the number of steps the stepper motor rotates for a given line. Figure 15, Paper Feed Stepper Motor Predetermined Time Constants, in the Background section above, lists the Time Constants required for three different line spacings; 6, 8, and 10 lines per inch. At the beginning of the Paper Feed Stepper Motor Drive subroutine, the default line step count is loaded. The software required is a conditional load for the line spacing, indicated by a status flag set in the External Status Switch Check subroutine or the Character Buffer Fill subroutine. Implementing the three different line spacings would require two additional status flags.
APPENDIX C.
PRINTER MECHANISM
DRIVE CIRCUIT

Recommended Solenoid Drive Circuit

<table>
<thead>
<tr>
<th>PARTS NO.</th>
<th>TYPE</th>
<th>MAKER</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC1–IC10</td>
<td>SN7406</td>
<td>TI</td>
</tr>
<tr>
<td>IC11</td>
<td>μA555</td>
<td>Fairchild</td>
</tr>
<tr>
<td>D1–D9</td>
<td>DIODE</td>
<td>S5277B</td>
</tr>
<tr>
<td>Q1–Q9</td>
<td>TRANSISTOR</td>
<td>2SD986</td>
</tr>
<tr>
<td>Q10</td>
<td>TRANSISTOR</td>
<td>2SA1015</td>
</tr>
<tr>
<td>Q11</td>
<td>TRANSISTOR</td>
<td>2SD633</td>
</tr>
<tr>
<td>R1–R9</td>
<td>RESISTOR</td>
<td>1.2kΩ ¼</td>
</tr>
<tr>
<td>R10</td>
<td>RESISTOR</td>
<td>22Ω ¼</td>
</tr>
<tr>
<td>R11</td>
<td>RESISTOR</td>
<td>580Ω 2</td>
</tr>
<tr>
<td>R12</td>
<td>RESISTOR</td>
<td>15kΩ ¼</td>
</tr>
<tr>
<td>R13</td>
<td>RESISTOR</td>
<td>1.2kΩ ¼</td>
</tr>
<tr>
<td>VR1</td>
<td>VARIABLE RESISTOR</td>
<td>20kΩ ¼</td>
</tr>
<tr>
<td>C1</td>
<td>CAPACITOR</td>
<td>1μF 100V</td>
</tr>
<tr>
<td>C2</td>
<td>CAPACITOR</td>
<td>0.01μF</td>
</tr>
<tr>
<td>C3</td>
<td>CAPACITOR</td>
<td>0.001μF</td>
</tr>
<tr>
<td>C4</td>
<td>CAPACITOR</td>
<td>10μF 16V</td>
</tr>
<tr>
<td>C5</td>
<td>CAPACITOR</td>
<td>0.1μF fil=</td>
</tr>
<tr>
<td>ZD1</td>
<td>ZENOR DIODE</td>
<td>HZ24</td>
</tr>
<tr>
<td>ZD2</td>
<td>ZENOR DIODE</td>
<td>HZ5C1</td>
</tr>
</tbody>
</table>

PRINT PULSE 1
500 ± 20μs

PRINT PULSE 9

TRIGGER PULSE
* 200μs OR LESS

RESET PULSE

Vcc TH
OUT

DIS

TRIG
R GND

CONT

C2

C5

C 3

VR1

R11

ZD 2

24V ± 10%

5V ± 5%

R10

Q 9

Q 10

Q 1

D 1

Solenoid 1

Solenoid 9
Recommended Carriage Motor Drive Circuit

HOLD SIGNAL DRIVE SIGNAL

\[ 5V \pm 5\% \]

\[ 24V \pm 10\% \]

\[ \text{(CONDENSED CHARACTER PRINTING), V=14 \pm 20\%} \]

\[ \text{INCASE OF } T_c=4.16\text{ms} \]

<table>
<thead>
<tr>
<th>PARTS NO.</th>
<th>TYPE</th>
<th>MAKER</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Resistor</td>
<td>1kΩ±10% 1/4</td>
<td>1</td>
</tr>
<tr>
<td>R2-R5</td>
<td>Resistor</td>
<td>2200Ω±10% 1/4</td>
<td>4</td>
</tr>
<tr>
<td>R6</td>
<td>Resistor</td>
<td>10kΩ±10% 1/4</td>
<td>1</td>
</tr>
<tr>
<td>R7</td>
<td>Resistor</td>
<td>470Ω±10% 3</td>
<td>1</td>
</tr>
<tr>
<td>R8</td>
<td>Resistor</td>
<td>1300Ω±10% 7</td>
<td>1</td>
</tr>
<tr>
<td>R9</td>
<td>Resistor</td>
<td>330Ω±10% 3</td>
<td>1</td>
</tr>
<tr>
<td>Q1</td>
<td>Transistor</td>
<td>2SC1815</td>
<td>1</td>
</tr>
<tr>
<td>Q2-Q5</td>
<td>Transistor</td>
<td>2SD526-Y</td>
<td>4</td>
</tr>
<tr>
<td>Q6</td>
<td>Transistor</td>
<td>2SB669</td>
<td>1</td>
</tr>
<tr>
<td>D1-D4</td>
<td>Diode</td>
<td>1S954</td>
<td>4</td>
</tr>
</tbody>
</table>
Recommended Paper Feed Motor Drive Circuit

```
PARTS NO.      TYPE           MAKER     QTY
R1            Resistor    1kΩ±10% ¼    1
R2-R5         Resistor    220Ω±10% ¼    4
R6            Resistor    10kΩ±10% ¼    1
R7            Resistor    470Ω±10% 3    1
R8            Resistor    130Ω±10% 7    1
R9            Resistor    330Ω±10% 3    1
Q1            Transistor  2SC1815      Toshiba 1
Q2-Q5         Transistor  2SD526—Y    Toshiba 4
Q6            Transistor  2SB669       Matsushita 1
D1~D4         Diode       1S854       NEC 4
```
An 8741A/8041A Digital Cassette Controller
INTRODUCTION

The microcomputer system designer requiring a low-cost, non-volatile storage medium has a difficult choice. His options have been either relatively expensive, as with floppy discs and bubble memories, or non-transportable, like battery backed-up RAMs. The full-sized digital cassette option was open but many times it was too expensive for the application. Filling this void of low-cost storage is the recently developed digital mini-cassette. These mini-cassettes are similar to, but not compatible with, dictation cassettes. The mini-cassette transports are inexpensive (well under $100 in quantity), small (less than 25 cu. in.), low-power (one watt), and their storage capacity is a respectable 200K bytes of unformatted data on a 100-foot tape. These characteristics make the mini-cassette perfect for applications ranging from remote datalogging to program storage for hobbyist systems.

The only problem associated with mini-cassette drives is controlling them. While these drives are relatively easy to interface to a microcomputer system, via a parallel I/O port, they can quickly overburden a CPU if other concurrent or critical real-time I/O is required. The cleanest and probably the least expensive solution in terms of development cost is to use a dedicated single-chip controller. However, a quick search through the literature turns up no controllers compatible with these new transports. What to do? Enter the UPI-41A family of Universal Peripheral Interfaces.

The UPI-41A family is a group of two user-programmable slave microcomputers plus a companion I/O expander. The 8741A is the “flag-chip” of the line. It is a complete microcomputer with 1024 bytes of EPROM program memory, 64 bytes of RAM data memory, 16 individually programmable I/O lines, an 8-bit event counter and timer, and a complete slave peripheral interface with two interrupts and Direct Memory Access (DMA) control. The 8041A is the masked ROM, pin compatible version of the 8741A. Figure 2 shows a block diagram common to both parts. The 8243 I/O port expander completes the family. Each 8243 provides 16 programmable I/O lines.

Using the UPI concept, the designer can develop a custom peripheral control processor for his particular I/O problem. The designer simply develops his peripheral control algorithm using the UPI-41A assembly language and programs the EPROM of

Figure 1. Comparison of Mini-Cassette and Floppy Disk Transports and Media.
the 8741A. Voilà! He has a single-chip dedicated controller. Testing may be accomplished using either an ICE-41A or the Single-step mode of the 8741A. UPI-41A peripheral interfaces are being used to control printers, keyboards, displays, custom serial interfaces, and data encryption units. Of course, the UPI family is perfect for developing a dedicated controller for digital mini-cassette transports. To illustrate this application for the UPI family let’s consider the job of controlling the Braemar CM-600 Mini-Dek®.

**THE CM-600 MINI-DEK* **

The Braemar CM-600 is representative of digital mini-cassette transports. It is a single-head, single-motor transport which operates entirely from a single 5-volt power supply. Its power requirements, including the motor, are 200ma for read or write and 700ma for rewind. Tapes speeds are 3 inches per second (IPS) during read or write, 5 IPS fast forward, and 15 IPS rewind. With these speeds and a maximum recording density of 800 bits per inch (BPI), the maximum data rate is 2400 bits per second (BAUD). The data capacity using both sides of a 100-foot tape is 200K bytes. On top of this, the transport occupies only 22.5 cubic inches (3”x3”x2.5”).

All I/O for the CM-600 is TTL-compatible and can be divided into three groups: motor control, data control, and cassette status. The motor group controls are GO/STOP, FAST/SLOW, and FORWARD/REVERSE. The data controls are READ/ WRITE, DATA IN, and DATA OUT. The remaining group of outputs give the transport’s status: CLEAR LEADER, CASSETTE PRESENCE, FILE PROTECT, and SIDE SENSOR. These signals, shown schematically in figure 3 and table 1, give the pin definition of the CM-600 16-pin I/O connector.

**RECORDING FORMAT**

The CM-600 does not provide either encoding or decoding of the recorded data; that task is left for the peripheral interface. A multitude of encoding techniques from which the user may choose are available. In this single-chip dedicated controller application, a “self-clocking” phase encoding scheme similar to that used in floppy discs was chosen. This scheme specifies that a logic “0” is a bit cell with no transition; a cell with a transition is a logic “1.”
### APPLICATIONS

#### Table 1. CM-600™ I/O Pin Definition

<table>
<thead>
<tr>
<th>Pin</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>Index pin—not used</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>Signal ground</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Cassette side (0—side B, 1—side A)</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>Data input (0—space, 1—mark)</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>Cassette presence (0—cassette, 1—no cassette)</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>Read/Write (0—read, 1—write)</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>File protect (0—tab present, 1—tab removed)</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>+5v motor power</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>Power ground</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>Chassis ground</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>Direction (0—forward, 1—rewind)</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>Speed (0—fast, 1—slow)</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>Data output (0—space, 1—mark)</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>Clear leader (0—clear leader, 1—off clear leader)</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>Motion (0—go, 1—stop)</td>
</tr>
<tr>
<td>16</td>
<td>-</td>
<td>+5v logic power</td>
</tr>
</tbody>
</table>

---

**Figure 3. Braemar CM-600™ Block Diagram**

Figure 4 illustrates the encoding of the character 3AH assuming the previous data ended with the data line high. (The least significant bit is sent first.) Notice that there is always a “clocking” transition at the beginning of each cell. Decoding is simply a matter of triggering on this “clocking” transition, waiting 3/4 of a bit cell time, and determining whether a mid-cell transition has occurred. Cells with no mid-cell transitions are data 0’s; cells with transitions are data 1’s. This encoding technique has all the benefits of Manchester encoding with the added advantage that the encoded data may be “decoded by eyeball:” long cells are always 0’s, short cells are always 1’s.

Besides the encoding scheme, the data format is also up to the user. This controller uses a variable byte length, checksum protected block format. Every block starts and ends with a SYNC character (AAH), and the character immediately preceding the last SYNC is the checksum. The checksum is capable of catching 2 bit errors. The number of data characters within a block is limited to 64K bytes. Blocks are separated by an Inter-Record Gap (IRG). The IRG is of such a length that the transport can stop and start within an IRG, as illustrated in the data block timing, figure 5. Braemar specifies a maximum start or stop time of 150ms for the transport, thus the controller uses 450ms for the IRG. This gives plenty of margin for controlling the transport and also for detecting IRGs while skipping blocks.

**THE UPI-41A™ CONTROLLER**

The goal of the UPI software design for this application was to make the UPI-41A microcomputer into an intelligent cassette control processor. The host processor (8086, 8088, 8085A, etc.) simply issues a high-level command such as READ-a-block or WRITE-a-block. The 8741A accepts the command, performs the requested operation, and returns to the host system a result code telling the outcome of the operation, e.g., Good-Completion, Sync Error, etc. Table 2 shows the command and result code repertoire. The 8741A completely manages all the data transfers for reading and writing.

As an example, consider the WRITE-a-block command. When this command is issued, the UPI-41A expects a 16-bit number from the host telling how many data bytes to write (up to 64K bytes per block). Once this number is supplied in the form of two bytes, the host is free to perform other tasks; a bit in the UPI’s STATUS register or an interrupt output will notify the host when a data transfer is required. The 8741A then checks the transport’s status to be sure that a cassette is present and not file protected. If either is false, a result code is...
Table 2. Controller Command/Result Code Set

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read (01H)</td>
<td>Good-Completion (00H)</td>
</tr>
<tr>
<td></td>
<td>Buffer Overrun Error (41H)</td>
</tr>
<tr>
<td></td>
<td>Bad Synch1 Error (42H)</td>
</tr>
<tr>
<td></td>
<td>Bad Synch2 Error (43H)</td>
</tr>
<tr>
<td></td>
<td>Checksum Error (44H)</td>
</tr>
<tr>
<td></td>
<td>Command Error (45H)</td>
</tr>
<tr>
<td></td>
<td>End of Tape Error (46H)</td>
</tr>
<tr>
<td>Rewind (04H)</td>
<td>Good-Completion (00H)</td>
</tr>
<tr>
<td>Skip (03H)</td>
<td>Good-Completion (00H)</td>
</tr>
<tr>
<td></td>
<td>End of Tape Error (47H)</td>
</tr>
<tr>
<td></td>
<td>Beginning of Tape Error (48H)</td>
</tr>
<tr>
<td>Write (02H)</td>
<td>Good-Completion (00H)</td>
</tr>
<tr>
<td></td>
<td>Buffer Underrun Error (81H)</td>
</tr>
<tr>
<td></td>
<td>Command Error (82H)</td>
</tr>
<tr>
<td></td>
<td>End of Tape Error (83H)</td>
</tr>
</tbody>
</table>

An IRG Write operation is started. After the peripheral controller checks to make sure that the tape is off the clear leader and past the hole in the tape, it writes a 450ms IRG, a SYNC character, the block of data, the checksum, and the final SYNC character. (The tape has a clear leader at both ends and a small hole 6 inches from the end of each leader.) The data transfers from the host to the UPI-41A slave microcomputer are double buffered. The controller requests only the desired number of data bytes by keeping track of the count internally.

If nothing unusual happened, such as finding a clear leader while writing, it returns a Good-Completion result code to the host. If clear leader was encountered, the transport is stopped immediately and an End-of-Tape result code is returned to the host. Another possible error would be if the host is late in supplying data. If this occurs, the controller writes an IRG, stops the drive, and returns the appropriate Data-Underrun result code.

The READ-a-block command also provides error checking. Once this command is issued by the host, the controller checks for cassette presence. If present, it starts the transport. The data output from the transport is then examined and decoded continuously. If the first character is not a SYNC, that's an error and the controller returns a Bad-First-SYNC result code (42H) after advancing to the next IRG. If the SYNC is good, the succeeding characters are read into an on-chip 30 character circular buffer. This continues until an IRG is encountered. When this occurs, the transport is stopped. The controller then tests that the last character. If it is a SYNC, the controller then compares the accumulated internal checksum to the block's checksum, the second to the last character of the block. If they match, a Good-Completion result code (00H) is returned to the host. If either test is bad, the appropriate error result code is returned. The READ command also checks for the End-of-Tape (EoT) clear leader and returns the appropriate error result code if it is found before the read operation is complete.

The 30 character circular buffer allows the host up to 30 character times of response time before the host must collect the data. All data transfers take place thru the UPI-41A Data Bus Buffer Output register (DBBOUT). The controller continually monitors the status of this register and moves characters from the circular buffer to the register whenever it is empty.

The SKIP-n-blocks command allows the host to skip the transport forward or backward up to 127 blocks. Once the command is issued, the controller expects one data byte specifying the number of
APPLICATIONS

blocks to skip. The most significant bit of this byte selects the direction of the skip (0=forward, 1=reverse). SKIP is a dual-speed operation in the forward direction. If the number of blocks to skip is greater than 8, the controller uses fast-forward (5 IPS) until it is within 8 blocks of the desired location. Once within 8 blocks, the controller switches to the normal read speed (3 IPS) to allow accurate placement of the tape. The reverse skip uses only the rewind speed (15 IPS). Like the READ and WRITE commands, SKIP also checks for EOT and beginning-of-tape (BOT) depending upon the tape's direction. An error result code is returned if either is encountered before the number of blocks skipped is complete.

The REWIND command simply rewinds the tape to the BOT clear leader. The ABORT command allows the termination of any operation in progress, except a REWIND. All commands, including ABORT, always leave the tape positioned on an IRG.

THE HARDWARE INTERFACE

There's hardly any hardware design effort required for the controller and transport interface in figure 6. Since the CM-600 is TTL compatible, it connects directly to the I/O ports of the UPI controller. If the two are separated (i.e. on different PC cards), it is recommended that TTL buffers be provided.) The only external circuitry needed is an LED driver for the DRIVE ACTIVE status indicator.

The 8741A-to-host interface is equally straightforward. It has a standard asynchronous peripheral interface: 8 data lines (D0-D7), read (RD), write (WR), register select (AO), and chip select (CS). Thus it connects directly to an 8086, 8088, 8085A, 8080, or 8048 bus structure. Two interrupt outputs are provided for data transfer requests if the particular system is interrupt-driven. DMA transfer capability is also available. The clock input can be driven from a crystal directly or with the system clock (6MHz max). The UPI-41A clock may be asynchronous with respect to other clocks within the system.

This application was developed on an Intel iSBC 80/30 single board computer. The iSBC 80/30 is controlled by an 8085A microprocessor, contains 16K bytes of dual-ported dynamic RAM and up to 8K bytes of either EPROM or ROM. Its I/O complement consists of an 8255A Programmable Parallel Interface, an 8251A Programmable Communica-

Figure 6. Controller/Transport System Schematic
APPLICATIONS

The iSBC 80/30 is especially convenient for UPI development since it contains an uncommitted socket dedicated to either an 804IA or 874IA complete with buffering for its I/O ports. The iSBC 80/30 to 874IA interface is reflected in figure 8. (Optionally, an iSBC 569 Digital Controller board could be used. The iSBC 569 board contains three uncommitted UPI sockets with an interface similar to that in figure 8.)

Looking at the host-to-controller interface, the host sees the 8741A as three registers in the host's I/O address space: the data register, the command register, and the status register. The decoding of these registers is shown in figure 7. All data and commands for the controller are written into the Data Bus Buffer Input register (DBBIN). The state of the register select input, AO, determines whether a command or data is written. (Writes with AO set to 1 are commands by convention.) All data and results from the controller are read by the host from the Data Bus Buffer Output register (DBBOUT).

The Status register contains flags which give the host the status of various operations within the controller. Its format is given in figure 8. The Input Buffer Full (IBF) and Output Buffer Full (OBF) flags show the Status of the DBBIN and DBBOUT registers respectively. IBF indicates when the DBBIN register contains data written by the host. The host may write to DBBIN only when IBF is 0. Likewise, the host may read DBBOUT only when OBF is set to a 1. These bits are handled automatically by the UPI-4IA internal hardware. FLAG 0 (F0) and FLAG 1 (F1) are general purpose flags used internally by the controller which have no meaning externally.

The remaining four bits are user-definable. For this application they are DRIVE ACTIVE, FILE PROTECT, CASSETTE PRESENCE, and BUSY flags. The FILE PROTECT and CASSETTE PRESENCE flags reflect the state of the corresponding I/O lines from the transport. DRIVE ACTIVE is set whenever the transport motor is on and the controller is performing an operation. The BUSY flag indicates whether the contents of the DBBOUT register is data or a result code. The BUSY flag is set whenever a command is issued by the host and accepted by the controller. As long as BUSY is set, any character found in DBBOUT is a result code. Thus whenever the host finds OBF set, it should test the BUSY flag to determine whether the character is data or a result code.

Notice the OBF and IBF are available as interrupt outputs to the host processor. These outputs are self-clearing, that is, OBF is set automatically upon the controller loading DBBOUT and cleared automatically by the host reading DBBOUT. Likewise IBF is cleared to a 0 by the host writing into DBBIN: set to a 1 when the controller reads DBBIN into the accumulator.

The flow charts of figure 9 show the flow of sample host software assuming a polling software interface between the host and the controller. The WRITE command requires two additional count bytes which form the 16-bit byte count. These extra bytes are "handshaked" into the controller using the IBF flag in the STATUS register. Once these bytes are written, the host writes data in response to IBF being cleared. This continues until the host finds OBF set indicating that the operation is complete and reads the result code from DBBOUT. No testing of BUSY is needed since only the result code appears in the DBBOUT register.

The READ command does require that BUSY be tested. Once the READ command is written into the
controller, the host must test BUSY whenever OBF is set to determine whether the contents of DBBOUT is data from the tape or the result code.

THE CONTROLLER SOFTWARE

The UPI-41A software to control the cassette can be divided up into various commands such as WRITE, READ and ABORT. In a previous version of this application note (May 1980), software was described that implemented these commands. This code however did not adequately compensate for speed variations of the motor during record and playback nor for data distortion caused by the magnetic media. Since then, new code has been written to include these effects. This revised software is now available through the INTEL User's Library, INSITE. For more information on this software or INSITE, contact your local INTEL Sales Office.
The Intel® 8041A/8741A is a general purpose, programmable interface device designed for use with a variety of 8-bit microprocessor systems. It contains a low cost microcomputer with program memory, data memory, 8-bit CPU, I/O ports, timer/counter, and clock in a single 40-pin package. Interface registers are included to enable the UPI device to function as a peripheral controller in MCS-48™, MCS-80™, MCS-85™, MCS-86™, and other 8-bit systems.

The UPI-41A™ has 1K words of program memory and 64 words of data memory on-chip. To allow full user flexibility the program memory is available as ROM in the 8041A version or as UV-erasable EPROM in the 8741A version. The 8741A and the 8041A are fully pin compatible for easy transition from prototype to production level designs. The 8641A is a one-time programmable (at the factory) 8741A which can be ordered as the first 25 pieces of a new 8041A order. The substitution of 8641A's for 8041A's allows for very fast turnaround for initial code verification and evaluation results.

The device has two 8-bit, TTL compatible I/O ports and two test inputs. Individual port lines can function as either inputs or outputs under software control. I/O can be expanded with the 8243 device which is directly compatible and has 16 I/O lines. An 8-bit programmable timer/counter is included in the UPI device for generating timing sequences or counting external inputs. Additional UPI features include: single 5V supply, low power standby mode (in the 8041A), single-step mode for debug (in the 8741A), and dual working register banks.

Because it's a complete microcomputer, the UPI provides more flexibility for the designer than conventional LSI interface devices. It is designed to be an efficient controller as well as an arithmetic processor. Applications include keyboard scanning, printer control, display multiplexing and similar functions which involve interfacing peripheral devices to microprocessor systems.
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0–D7 (BUS)</td>
<td>Three-state, bidirectional DATA BUS BUFFER lines used to interface the UPI-41A to an 8-bit master system data bus.</td>
</tr>
<tr>
<td>P10–P17</td>
<td>8-bit, PORT 1 quasi-bidirectional I/O lines.</td>
</tr>
<tr>
<td>P20–P27</td>
<td>8-bit, PORT 2 quasi-bidirectional I/O lines.</td>
</tr>
<tr>
<td>WR</td>
<td>I/O write input which enables the master CPU to write data and command words to the UPI-41A INPUT DATA BUS BUFFER.</td>
</tr>
<tr>
<td>RD</td>
<td>I/O read input which enables the master CPU to read data and status words from the OUTPUT DATA BUS BUFFER or status register.</td>
</tr>
<tr>
<td>CS</td>
<td>Chip select input used to select one UPI-41A out of several connected to a common data bus.</td>
</tr>
<tr>
<td>A0</td>
<td>Address input used by the master processor to indicate whether byte transfer is data or command. During a write operation flag F1 is set to the status of the A0 input.</td>
</tr>
<tr>
<td>TEST 0, TEST 1</td>
<td>Input pins which can be directly tested using conditional branch instructions.</td>
</tr>
<tr>
<td></td>
<td>T1 also functions as the event timer input (under software control). T0 is used during PROM programming and verification in the 8741A.</td>
</tr>
<tr>
<td>X1</td>
<td>Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.</td>
</tr>
<tr>
<td>SYNC</td>
<td>Output signal which occurs once per UPI-41A instruction cycle. SYNC can be used as a strobe for external circuitry; it is also used to synchronize single step operation.</td>
</tr>
<tr>
<td>EA</td>
<td>External access input which allows emulation, testing and PROM/ROM verification.</td>
</tr>
<tr>
<td>PROG</td>
<td>Multifunction pin used as the program pulse input during PROM programming. During I/O expander access the PROG pin acts as an address/data strobe to the 8243.</td>
</tr>
<tr>
<td>RESET</td>
<td>Input used to reset status flip-flops and to set the program counter to zero. RESET is also used during PROM programming and verification. RESET should be held low for a minimum of 8 instruction cycles after power-up.</td>
</tr>
<tr>
<td>SS</td>
<td>Single step input used in the 8741A in conjunction with the SYNC output to step the program through each instruction.</td>
</tr>
<tr>
<td>VCC</td>
<td>+5V main power supply pin.</td>
</tr>
<tr>
<td>VDD</td>
<td>+5V during normal operation. +25V during programming operation. Low power standby pin in ROM version.</td>
</tr>
<tr>
<td>VSS</td>
<td>Circuit ground potential.</td>
</tr>
</tbody>
</table>
PROGRAMMING, VERIFYING, AND ERASING THE 8741A EPROM

Programming Verification

In brief, the programming process consists of: activating the program mode; applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAL 1</td>
<td>Clock Input (1 to 6MHz)</td>
</tr>
<tr>
<td>Reset</td>
<td>Initialization and Address Latching</td>
</tr>
<tr>
<td>Test 0</td>
<td>Selection of Program or Verify Mode</td>
</tr>
<tr>
<td>EA</td>
<td>Activation of Program/Verify Modes</td>
</tr>
<tr>
<td>BUS</td>
<td>Address and Data Input</td>
</tr>
<tr>
<td>P20-1</td>
<td>Address Input</td>
</tr>
<tr>
<td>VDD</td>
<td>Programming Power Supply</td>
</tr>
<tr>
<td>PROG</td>
<td>Program Pulse Input</td>
</tr>
</tbody>
</table>

WARNING.

An attempt to program a missocketed 8741A will result in severe damage to the part. An indication of a properly socketed part is the appearance of the SYNC clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. \( A_0 = 0V, C8 = 5V, EA = 5V, \text{RESET} = 0V, \text{TEST0} = 5V, V_{DD} = 5V \), clock applied or internal oscillator operating, BUS and PROG floating.
2. Insert 8741A in programming socket
3. TEST 0 = 0V (select program mode)
4. EA = 23V (activate program mode)
5. Address applied to BUS and P20-1

6. \( \text{RESET} = 5V \) (latch address)
7. Data applied to BUS
8. \( V_{DD} = 25V \) (programming power)
9. PROG = 0V followed by one 50ms pulse to 23V
10. \( V_{DD} = 5V \)
11. \( \text{TEST} 0 = 5V \) (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0V
14. \( \text{RESET} = 0V \) and repeat from step 5
15. Programmer should be at conditions of step 1 when 8741A is removed from socket.

8741A Erasure Characteristics

The erasure characteristics of the 8741A are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8741A in approximately 3 years while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 8741A is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8741A window to prevent unintentional erasure.

The recommended erasure procedure for the 8741A is exposure to shortwave ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity x exposure time) for erasure should be a minimum of 15 \( \text{w-sec/cm}^2 \). The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 \( \mu \text{W/cm}^2 \) power rating. The 8741A should be placed within one inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.
UPI-41A™ FEATURES AND ENHANCEMENTS

1. Two Data Bus Buffers, one for input and one for output. This allows a much cleaner Master/Slave protocol.

```
INPUT DATA BUS BUFFER (8)
```

```
OUTPUT DATA BUS BUFFER (8)
```

2. 8 Bits of Status

<table>
<thead>
<tr>
<th>ST</th>
<th>ST</th>
<th>ST</th>
<th>F</th>
<th>F</th>
<th>IBF</th>
<th>OBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
<td>D5</td>
<td>D4</td>
<td>D3</td>
<td>D2</td>
<td>D1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

ST4-ST7 are user definable status bits. These bits are defined by the “MOV STS, A” single byte, single cycle instruction. Bits 4-7 of the accumulator are moved to bits 4-7 of the status register. Bits 0-3 of the status register are not affected.

```
MOV STS, A Op Code: 90H
```

If “EN FLAGS” has been executed, P25 becomes the OBF (Input Buffer Full) pin. A “1” written to P25 enables the OBF pin (the pin outputs the inverse of the OBF Status Bit). A “0” written to P25 disables the OBF pin (the pin remains low). This pin can be used to indicate that the UPI-41A is ready for data.

```
DATA BUS BUFFER INTERRUPT CAPABILITY
```

```
EN FLAGS Op Code: 0F5H
```

3. RD and WR are edge triggered. IBF, OBF, F1, and INT change internally after the trailing edge of RD or WR.

```
FLAGS AFFECTED
```

4. P24 and P25 are port pins or Buffer Flag pins which can be used to interrupt a master processor. These pins default to port pins on Reset.

If the “EN FLAGS” instruction has been executed, P24 becomes the OBF (Output Buffer Full) pin. A “1” written to P24 enables the OBF pin (the pin outputs the OBF Status Bit). A “0” written to P24 disables the OBF pin (the pin remains low). This pin can be used to indicate that valid data is available from the UPI-41A (in Output Data Bus Buffer).

```
DMA HANDSHAKE CAPABILITY
```

```
EN DMA Op Code: 0E5H
```

5. P26 and P27 are port pins or DMA handshake pins for use with a DMA controller. These pins default to port pins on Reset.

If the “EN DMA” instruction has been executed, P26 becomes the DRQ (DMA ReQuest) pin. A “1” written to P26 causes a DMA request (DRQ is activated). DRQ is deactivated by DACK-RD, DACK-WR, or execution of the “EN DMA” instruction.

If “EN DMA” has been executed, P27 becomes the DACK (DMA ACKnowledge) pin. This pin acts as a chip select input for the Data Bus Buffer registers during DMA transfers.
APPLICATIONS

Figure 1. 8085A-8041A Interface

Figure 2. 8048-8041A Interface

Figure 3. 8041A-8243 Keyboard Scanner

Figure 4. 8041A Matrix Printer Interface
**ABSOLUTE MAXIMUM RATINGS**

*COMMENT: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

Ambient Temperature Under Bias ............. 0°C to 70°C
Storage Temperature ..................... –65°C to +150°C
Voltage on Any Pin With Respect to Ground ..................... 0.5V to +7V
Power Dissipation ......................... 1.5 Watt

**D.C. AND OPERATING CHARACTERISTICS**

$T_A=0°C$ to $70°C$, $V_{SS}=0V$, $V_{CC}=V_{DD}=+5V$ ±10% *

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IL}$</td>
<td>Input Low Voltage (Except XTAL1, XTAL2, RESET)</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IL1}$</td>
<td>Input Low Voltage (XTAL1, XTAL2, RESET)</td>
<td>-0.5</td>
<td>0.6</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IH}$</td>
<td>Input High Voltage (Except XTAL1, XTAL2, RESET)</td>
<td>2.2</td>
<td>$V_{CC}$</td>
<td></td>
</tr>
<tr>
<td>$V_{IH1}$</td>
<td>Input High Voltage (XTAL1, XTAL2, RESET)</td>
<td>3.8</td>
<td>$V_{CC}$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}$</td>
<td>Output Low Voltage ($D_0$-$D_7$)</td>
<td>0.45</td>
<td>V</td>
<td>$I_{OL}=2.0$ mA</td>
</tr>
<tr>
<td>$V_{OL1}$</td>
<td>Output Low Voltage ($P_{10}$-$P_{17}$, $P_{20}$-$P_{27}$, Sync)</td>
<td>0.45</td>
<td>V</td>
<td>$I_{OL}=1.6$ mA</td>
</tr>
<tr>
<td>$V_{OL2}$</td>
<td>Output Low Voltage (Prog)</td>
<td>0.45</td>
<td>V</td>
<td>$I_{OL}=1.0$ mA</td>
</tr>
<tr>
<td>$V_{OH}$</td>
<td>Output High Voltage ($D_0$-$D_7$)</td>
<td>2.4</td>
<td>V</td>
<td>$I_{OH}=-400$ $\mu$A</td>
</tr>
<tr>
<td>$V_{OH1}$</td>
<td>Output High Voltage (All Other Outputs)</td>
<td>2.4</td>
<td>V</td>
<td>$I_{OH}=-50$ $\mu$A</td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>Input Leakage Current ($T_0$, $T_1$, $RD$, $WR$, $CS$, $A_0$, EA)</td>
<td>$\pm 10$</td>
<td>$\mu$A</td>
<td>$V_{SS} \leq V_{IN} \leq V_{CC}$</td>
</tr>
<tr>
<td>$I_{IZ}$</td>
<td>Output Leakage Current ($D_0$-$D_7$, High Z State)</td>
<td>$\pm 10$</td>
<td>$\mu$A</td>
<td>$V_{SS}+0.45 \leq V_{IN} \leq V_{CC}$</td>
</tr>
<tr>
<td>$I_{IL1}$</td>
<td>Low Input Load Current ($P_{10}$-$P_{17}$, $P_{20}$-$P_{27}$)</td>
<td>0.5</td>
<td>mA</td>
<td>$V_{IL}=0.8$V</td>
</tr>
<tr>
<td>$I_{DD}$</td>
<td>Low Input Load Current (RESET, SS)</td>
<td>0.2</td>
<td>mA</td>
<td>$V_{IL}=0.8$V</td>
</tr>
<tr>
<td>$I_{DD}$</td>
<td>$V_{DD}$ Supply Current</td>
<td>15</td>
<td>mA</td>
<td>Typical = 5 mA</td>
</tr>
<tr>
<td>$I_{CC}+I_{DD}$</td>
<td>Total Supply Current</td>
<td>125</td>
<td>mA</td>
<td>Typical = 60 mA</td>
</tr>
</tbody>
</table>

**A.C. CHARACTERISTICS**

$T_A=0°C$ to $70°C$, $V_{SS}=0V$, $V_{CC}=V_{DD}=+5V$ ±10% *

**DBB READ**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{AR}$</td>
<td>$CS, A_0$ Setup to $RD_i$</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{RA}$</td>
<td>$CS, A_0$ Hold After $RD_i$</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{RR}$</td>
<td>$RD$ Pulse Width</td>
<td>250</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{AD}$</td>
<td>$CS, A_0$ to Data Out Delay</td>
<td>225</td>
<td>ns</td>
<td>$C_L=150$ pF</td>
<td></td>
</tr>
<tr>
<td>$t_{RD}$</td>
<td>$RD_i$ to Data Out Delay</td>
<td>225</td>
<td>ns</td>
<td>$C_L=150$ pF</td>
<td></td>
</tr>
<tr>
<td>$t_{DF}$</td>
<td>$RD_i$ to Data Float Delay</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{CY}$</td>
<td>Cycle Time (Except 8741A-8)</td>
<td>2.5</td>
<td>15</td>
<td>$\mu$s</td>
<td>6.0 MHz XTAL</td>
</tr>
</tbody>
</table>

**DBB WRITE**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{AW}$</td>
<td>$CS, A_0$ Setup to $WR_i$</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{WA}$</td>
<td>$CS, A_0$ Hold After $WR_i$</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{WW}$</td>
<td>$WR$ Pulse Width</td>
<td>250</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{DW}$</td>
<td>Data Setup to $WR_i$</td>
<td>150</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{WD}$</td>
<td>Data Hold After $WR_i$</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### A.C. TIMING SPECIFICATION FOR PROGRAMMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>taw</td>
<td>Address Setup Time to RESET 1</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>twa</td>
<td>Address Hold Time After RESET 1</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tdw</td>
<td>Data in Setup Time to PROG 1</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>twd</td>
<td>Data in Hold Time After PROG 1</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tph</td>
<td>RESET Hold Time to Verify</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tVDDW</td>
<td>VDD Setup Time to PROG 1</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tVDDH</td>
<td>VDD Hold Time After PROG 1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tpw</td>
<td>Program Pulse Width</td>
<td>50</td>
<td>60</td>
<td>mS</td>
<td></td>
</tr>
<tr>
<td>ttw</td>
<td>Test 0 Setup Time for Program Mode</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tW</td>
<td>Test 0 Hold Time After Program Mode</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tdo</td>
<td>Test 0 to Data Out Delay</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tww</td>
<td>RESET Pulse Width to Latch Address</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tr, tf</td>
<td>VDD and PROG Rise and Fall Times</td>
<td>0.5</td>
<td>2.0</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>tcY</td>
<td>CPU Operation Cycle Time</td>
<td>5.0</td>
<td></td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>tRE</td>
<td>RESET Setup Time Before EA 1.</td>
<td>4icy</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** If TEST 0 is high, tdo can be triggered by RESET 1.

*For Extended Temperature EXPRESS, use M8741A electrical parameters.

### D.C. SPECIFICATION FOR PROGRAMMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDDH</td>
<td>VDD Program Voltage High Level</td>
<td>24.0</td>
<td>26.0</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VDDL</td>
<td>VDD Voltage Low Level</td>
<td>4.75</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VPH</td>
<td>PROG Program Voltage High Level</td>
<td>21.5</td>
<td>24.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VPL</td>
<td>PROG Voltage Low Level</td>
<td>0.2</td>
<td></td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VEAH</td>
<td>EA Program or Verify Voltage High Level</td>
<td>21.5</td>
<td>24.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VEAL</td>
<td>EA Voltage Low Level</td>
<td>5.25</td>
<td></td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>IDD</td>
<td>Vpp High Voltage Supply Current</td>
<td>30.0</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>IPPROG</td>
<td>PROG High Voltage Supply Current</td>
<td>16.0</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>IEA</td>
<td>EA High Voltage Supply Current</td>
<td>1.0</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
</tbody>
</table>

### A.C. CHARACTERISTICS—PORT 2

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>Port Control Setup Before Falling Edge of PROG</td>
<td>110</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tcp</td>
<td>Port Control Hold After Falling Edge of PROG</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tpr</td>
<td>PROG to Time P2 Input Must Be Valid</td>
<td>810</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tpF</td>
<td>Input Data Hold Time</td>
<td>0</td>
<td>150</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tdp</td>
<td>Output Data Setup Time</td>
<td>250</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tpd</td>
<td>Output Data Hold Time</td>
<td>65</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tpp</td>
<td>PROG Pulse Width</td>
<td>1200</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>
A.C. CHARACTERISTICS—DMA

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{\text{ACC}} )</td>
<td>DACK to WR or RD</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{\text{CAC}} )</td>
<td>RD or WR to DACK</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{\text{ACD}} )</td>
<td>DACK to Data Valid</td>
<td>225</td>
<td></td>
<td>ns</td>
<td>( C_L = 150 , \text{pF} )</td>
</tr>
<tr>
<td>( t_{\text{CRQ}} )</td>
<td>RD or WR to DRQ Cleared</td>
<td>200</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

CRYSTAL OSCILLATOR MODE

-XTAL1: 1.6 MHz
  -15 pF
  -15 - 25 pF

CRYSTAL SERIES RESISTANCE SHOULD BE
<75\Omega AT 6 MHz, <180\Omega AT 3.6 MHz

LC OSCILLATOR MODE

\[
L = \begin{cases} 
45 \, \mu\text{H} & \text{5.2 MHz} \\
120 \, \mu\text{H} & \text{3.2 MHz} 
\end{cases}
\]

\[
C = \begin{cases} 
20 \, \text{pF} & \text{5.2 MHz} \\
20 \, \text{pF} & \text{3.2 MHz} 
\end{cases}
\]

\[
C' = \frac{C + 3 C_{\text{pp}}}{2}
\]

\[
C_{\text{pp}} > 5 - 10 \, \text{pF PIN-TO-PIN CAPACITANCE}
\]

EACH C SHOULD BE APPROXIMATELY 20 pF, INCLUDING STRAY CAPACITANCE

DRIVING FROM EXTERNAL SOURCE

BOTH XTAL1 AND XTAL2 SHOULD BE DRIVEN
RESISTORS TO \( V_{\text{CC}} \) ARE NEEDED TO ENSURE \( V_{\text{IH}} = 3 \, \text{V} \)
IF TTL CIRCUITRY IS USED

TYPICAL 8041/8741A CURRENT

A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

\[ C_L = 150 \, \text{pF} \]

TEMP (°C)
WAVEFORMS

READ OPERATION—DATA BUS BUFFER REGISTER.

WRITE OPERATION—DATA BUS BUFFER REGISTER.

PORT 2 TIMING
The 8741A EPROM can be programmed by either of two Intel products:
1. PROMPT-48 Microcomputer Design Aid, or
### Table 2. UPI™ Instruction Set

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A,Rr</td>
<td>Add register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A,@Rr</td>
<td>Add data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A,#data</td>
<td>Add immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ADDC A,Rr</td>
<td>Add register to A with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A,@Rr</td>
<td>Add data memory to A with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, #data</td>
<td>Add immed. to A with carry</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL A,Rr</td>
<td>AND register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A,@Rr</td>
<td>AND data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A,#data</td>
<td>AND immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL A,Rr</td>
<td>OR register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A,@Rr</td>
<td>OR data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A,#data</td>
<td>OR immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XRL A,Rr</td>
<td>Exclusive OR register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A,@Rr</td>
<td>Exclusive OR data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A,#data</td>
<td>Exclusive OR immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INC A</td>
<td>Increment A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal Adjust A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles of A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate A left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RCL A</td>
<td>Rotate A left through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate A right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A right through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Mnemonic/Output</td>
<td>Description</td>
<td>Bytes</td>
<td>Cycles</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>In A,Pp</td>
<td>Input port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>OUTL Pp,A</td>
<td>Output A to port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL Pp,#data</td>
<td>AND immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL Pp,#data</td>
<td>OR immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>In A, DBB</td>
<td>Input DBB to A, clear IBF</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OUT DBB,A</td>
<td>Output A to DBB, set OBF</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV STS,A</td>
<td>A4-A7 to Bits 4-7 of Status</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOVD A,Pp</td>
<td>Input Expander port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVDPd Pp,A</td>
<td>Output A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANLD Pp,A</td>
<td>AND A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ORLD Pp,A</td>
<td>OR A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Moves</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,Rr</td>
<td>Move register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,#data</td>
<td>Move data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,#data</td>
<td>Move to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV A,Rr</td>
<td>Move to A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV @Rr,A</td>
<td>Move to data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV @Rr,#data</td>
<td>Move to data memory</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV PSW, A</td>
<td>Move PSW to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV PSW, A</td>
<td>Move A to PSW</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A,Rr</td>
<td>Exchange A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A,#data</td>
<td>Exchange A and data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCHD A,#data</td>
<td>Exchange digit of A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOPV A,A</td>
<td>Move to A from current page</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOPV3, A,A</td>
<td>Move to A from page 3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timer/Counter</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,T</td>
<td>Read Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV T,A</td>
<td>Load Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>START T</td>
<td>Start Timer</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STRT CNT</td>
<td>Start Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STOP TCNT</td>
<td>Stop Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EN TCNTI</td>
<td>Enable Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS TCNTI</td>
<td>Disable Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic/Control</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN DMA</td>
<td>Enable DMA Handshake Lines</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EN I</td>
<td>Enable IBF Interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS I</td>
<td>Disable IBF Interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EN FLAGS</td>
<td>Enable Master Interrupts</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL RB0</td>
<td>Select register bank 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL RB1</td>
<td>Select register bank 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Registers</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC Rr</td>
<td>Increment register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC @Rr</td>
<td>Increment data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC Rr</td>
<td>Decrement register</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL addr</td>
<td>Jump to subroutine</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>RET</td>
<td>Return</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RETR</td>
<td>Return and restore status</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flags</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR C</td>
<td>Clear Carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL C</td>
<td>Complement Carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F0</td>
<td>Clear Flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F0</td>
<td>Complement Flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F1</td>
<td>Clear F1 Flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F1</td>
<td>Complement F1 Flag</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Branch</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP addr</td>
<td>Jump unconditional</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JMP @A</td>
<td>Jump indirect</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DJNZ R,#addr</td>
<td>Decrement register and jump</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JC addr</td>
<td>Jump on Carry=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNC addr</td>
<td>Jump on Carry=0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JZ addr</td>
<td>Jump on A Zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNZ addr</td>
<td>Jump on A not Zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTO addr</td>
<td>Jump on T0=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNTO addr</td>
<td>Jump on T0=0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JT1 addr</td>
<td>Jump on T1=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNT1 addr</td>
<td>Jump on T1=0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF0 addr</td>
<td>Jump on F0 Flag=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF1 addr</td>
<td>Jump on F1 Flag=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTF addr</td>
<td>Jump on Timer Flag=1, Clear Flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JN1BF addr</td>
<td>Jump on IBF Flag=0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JOBF addr</td>
<td>Jump on OBF Flag=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JBB addr</td>
<td>Jump on Accumulator Bit</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
8042/8742
UNIVERSAL PERIPHERAL INTERFACE
8-BIT MICROCOMPUTER

- 8042/8742: 12 MHz
- Pin, Software and Architecturally Compatible with 8041A/8741A
- 8-Bit CPU plus ROM, RAM, I/O, Timer and Clock in a Single Package
- 2048 x 8 ROM/EPROM, 128 x 8 RAM, 8-Bit Timer/Counter, 18 Programmable I/O Pins
- One 8-Bit Status and Two Data Registers for Asynchronous Slave-to-Master Interface
- DMA, Interrupt, or Polled Operation Supported
- Fully Compatible with all Intel and Most Other Microprocessor Families
- Interchangeable ROM and EPROM Versions
- Expandable I/O
- RAM Power-Down Capability
- Over 90 Instructions: 70% Single Byte
- Available in EXPRESS —Standard Temperature Range

The Intel 8042/8742 is a general-purpose Universal Peripheral Interface that allows the designer to grow his own customized solution for peripheral device control. It contains a low-cost microcomputer with 2K of program memory, 128 bytes of data memory, 8-bit CPU, I/O ports, 8-bit timer/counter, and clock generator in a single 40-pin package. Interface registers are included to enable the UPI device to function as a peripheral controller in the MCS-48™, MCS-51™, MCS-80™, MCS-85™, iAPX-88, iAPX-86 and other 8-, 16-bit systems.

The 8042/8742 is software, pin, and architecturally compatible with the 8041A, 8741A. The 8042/8742 doubles the on-chip memory space to allow for additional features and performance to be incorporated in upgraded 8041A/8741A designs. For new designs, the additional memory and performance of the 8042/8742 extends the UPI concept to more complex motor control tasks, 80-column printers and process control applications as examples.

To allow full user flexibility, the program memory is available as ROM in the 8042 version or as UV-erasable EPROM in the 8742 version. The 8742 and the 8042 are fully pin compatible for easy transition from prototype to production level designs.

Figure 1. Block Diagram

Figure 2. Pin Configuration
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST 0,</td>
<td>1</td>
<td>I</td>
<td>Test Inputs: Input pins which can be directly tested using conditional branch instructions.</td>
</tr>
<tr>
<td>TEST 1</td>
<td>39</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>XTAL 1,</td>
<td>2</td>
<td>I</td>
<td>Inputs: Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.</td>
</tr>
<tr>
<td>XTAL 2</td>
<td>3</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>4</td>
<td>I</td>
<td>Reset: Input used to reset status flip-flops and to set the program counter to zero.</td>
</tr>
<tr>
<td>SS</td>
<td>5</td>
<td>I</td>
<td>Single Step: Single step input used in conjunction with the SYNC output to step the program through each instruction. (8742 only)</td>
</tr>
<tr>
<td>CS</td>
<td>6</td>
<td>I</td>
<td>Chip Select: Chip select input used to select one UPI microcomputer out of several connected to a common data bus.</td>
</tr>
<tr>
<td>EA</td>
<td>7</td>
<td>I</td>
<td>External Access: External access input which allows emulation, testing and PROM/ROM verification. This pin should be tied low if unused.</td>
</tr>
<tr>
<td>RD</td>
<td>8</td>
<td>I</td>
<td>Read: I/O read input which enables the master CPU to read data and status words from the OUTPUT DATA BUS BUFFER or status register.</td>
</tr>
<tr>
<td>A0</td>
<td>9</td>
<td>I</td>
<td>Command/Data Select: Address input used by the master processor to indicate whether byte transfer is data (A0=0, F1 is reset) or command (A0=1, F1 is set).</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>I</td>
<td>Write: I/O write input which enables the master CPU to write data and command words to the UPI INPUT DATA BUS BUFFER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNC</td>
<td>11</td>
<td>O</td>
<td>Output Clock: Output signal which occurs once per UPI-42 instruction cycle. SYNC can be used as a strobe for external circuitry; it is also used to synchronize single step operation.</td>
</tr>
<tr>
<td>D0-D7</td>
<td>12-19</td>
<td>I/O</td>
<td>Data Bus: Three-state, bidirectional DATA BUS BUFFER lines used to interface the UPI-42 microcomputer to an 8-bit master system data bus.</td>
</tr>
<tr>
<td>P10-P17</td>
<td>27-34</td>
<td>I/O</td>
<td>Port 1: 8-bit, PORT 1 quasi-bidirectional I/O lines.</td>
</tr>
<tr>
<td>P20-P27</td>
<td>21-24</td>
<td>I/O</td>
<td>Port 2: 8-bit, PORT 2 quasi-bidirectional I/O lines. The lower 4 bits (P20-P23) interface directly to the 8243 I/O expander device and contain address and data information during PORT 4-7 access. The upper 4 bits (P24-P27) can be programmed to provide interrupt Request and DMA Handshake capability. Software control can configure P24 as Output Buffer Full (OBF) interrupt, P25 as Input Buffer Full (IBF) interrupt, P26 as DMA Request (DRQ), and P27 as DMA Acknowledge (DACK).</td>
</tr>
<tr>
<td>PROG</td>
<td>25</td>
<td>I/O</td>
<td>Program: Multifunction pin used as the program pulse input during PROM programming. During I/O expander access the PROG pin acts as an address/data strobe to the 8243. This pin should be tied high if unused.</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td></td>
<td>Power: +5V main power supply pin.</td>
</tr>
<tr>
<td>VDD</td>
<td>26</td>
<td></td>
<td>Power: +5V during normal operation. +21V during programming operation. Low power standby pin in ROM version.</td>
</tr>
<tr>
<td>VSS</td>
<td>20</td>
<td></td>
<td>Ground: Circuit ground potential.</td>
</tr>
</tbody>
</table>
UPI-42 FEATURES

1. Two Data Bus Buffers, one for input and one for output. This allows a much cleaner Master/Slave protocol.

2. 8 Bits of Status

   ST7 ST6 ST5 ST4 F1 F0 IBF OBF
   D7 D6 D5 D4 D3 D2 D1 D0

   ST7-ST7 are user-definable status bits. These bits are defined by the "MOV STS, A" single byte, single cycle instruction. Bits 4-7 of the accumulator are moved to bits 4-7 of the status register. Bits 0-3 of the status register are not affected.

   MOV STS, A  Op Code 90H
   1 0 0 1 0 0 0 0

3. RD and WR are edge triggered IBF, OBF, F, and INT change internally after the trailing edge of RD or WR.

   RD or WR

   During the time that the host CPU is reading the status register, the 8042/8742 is prevented from updating this register or is 'locked out.'

4. P24 and P25 are port pins or Buffer Flag pins which can be used to interrupt a master processor. These pins default to port pins on Reset.

   If the "EN FLAGS" instruction has been executed, P24 becomes the OBF (Output Buffer Full) pin. A "1" written to P24 enables the OBF pin (the pin outputs the OBF Status Bit). A "0" written to P24 disables the OBF pin (the pin remains low). This pin can be used to indicate that the UPI-42 is ready for data.

   P24 OBF [interrupt request]

   P25 IBF [interrupt request]

5. P26 and P27 are port pins or DMA handshake pins for use with a DMA controller. These pins default to port pins on Reset.

   If the "EN DMA" instruction has been executed, P26 becomes the DRQ (DMA Request) pin. A "1" written to P26 causes a DMA request (DRQ is activated). DRQ is deactivated by DACK-RD, DACK-WR, or execution of the "EN DMA" instruction.

   If "EN DMA" has been executed, P27 becomes the DACK (DMA Acknowledge) pin. This pin acts as a chip select input for the Data Bus Buffer registers during DMA transfers.

   DRAK DMA HANDSHAKE CAPABILITY

   DRAK

   EN DMA Op Code 0E5H
   1 1 1 0 0 1 0 1

6. The RESET input on the 8042/8742 includes a 2-stage synchronizer to support reliable reset operation for 12 MHz operation.

7. When EA is enabled on the 8042/8742, the program counter is placed on Port 1 and the lower three bits of Port 2 (MSB = P22, LSB = P10). On the 8042/8742 this information is multiplexed with PORT DATA (see port timing diagrams at end of this data sheet).
APPLICATIONS

Figure 3. 8088-8042/8742 Interface

Figure 4. 8048H-8042/8742 Interface

Figure 5. 8042/8742-8243 Keyboard Scanner

Figure 6. 8042/8742 80-Column Matrix Printer Interface

PROGRAMMING, VERIFYING, AND ERASING THE 8742 EPROM

Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAL 1</td>
<td>Clock Input</td>
</tr>
<tr>
<td>Reset</td>
<td>Initialization and Address Latching</td>
</tr>
<tr>
<td>Test 0</td>
<td>Selection of Program or Verify Mode</td>
</tr>
<tr>
<td>EA</td>
<td>Activation of Program/Verify Modes</td>
</tr>
<tr>
<td>BUS</td>
<td>Address and Data Input</td>
</tr>
<tr>
<td>P20-12</td>
<td>Address Input</td>
</tr>
<tr>
<td>VDD</td>
<td>Programming Power Supply</td>
</tr>
<tr>
<td>PROG</td>
<td>Program Pulse Input</td>
</tr>
</tbody>
</table>

WARNING

An attempt to program a missocketed 8742 will result in severe damage to the part. An indication of a properly socketed part is the appearance of the SYNC clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. A0 = 0V, CS = 5V, EA = 5V, RESET = 0V, TEST0 = 5V, VDD = 5V, clock applied or internal oscillator operating, BUS floating, PROG = 5V
2. Insert 8742 in programming socket
3. TEST 0 = 0v (select program mode)
4. EA = 18V (active program mode)*
5. Address applied to BUS and P20-22
6. RESET = 5v (latch address)
7. Data applied to BUS**
8. VDD = 21V (programming power)**
9. PROG = VCC followed by one 50 ms pulse to 18V**
10. VDD = 5v
11. TEST 0 = 0v (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0v
14. RESET = 0v and repeat from step 5
15. Programmer should be at conditions of step 1 when 8742 is removed from socket

*When verifying ROM, EA = 12V.
**Not used in verifying ROM procedure.

8742 Erasure Characteristics

The erasure characteristics of the 8742 are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8742 in approximately 3 years while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 8742 is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8742 window to prevent unintentional erasure.

The recommended erasure procedure for the 8742 is exposure to shortwave ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity x exposure time) for erasure should be a minimum of 15 w-sec/cm². The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 µW/cm² power rating. The 8742 should be placed within one inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.
**ABSOLUTE MAXIMUM RATINGS**

Ambient Temperature Under Bias ................................ 0°C to 70°C
Storage Temperature .................................................. −65°C to +150°C
Voltage on Any Pin With Respect to Ground .................... −0.5V to +7V
Power Dissipation ...................................................... 1.5 Watt

*NOTICE* Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. CHARACTERISTICS**  \( (T_A = 0^\circ \text{C} \text{ to } +70^\circ \text{C}, V_{CC} = V_{DD} = +5V \pm 10\%) \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8042/8742</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>V_{IL}</td>
<td>Input Low Voltage (Except XTAL1, XTAL2, RESET)</td>
<td>−0.5</td>
</tr>
<tr>
<td>V_{IL1}</td>
<td>Input Low Voltage (XTAL1, XTAL2, RESET)</td>
<td>−0.5</td>
</tr>
<tr>
<td>V_{IH}</td>
<td>Input High Voltage (Except XTAL1, XTAL2, RESET)</td>
<td>2.0</td>
</tr>
<tr>
<td>V_{IH1}</td>
<td>Input High Voltage (XTAL1, RESET)</td>
<td>3.5</td>
</tr>
<tr>
<td>V_{IH2}</td>
<td>Input High Voltage (XTAL2)</td>
<td>2.2</td>
</tr>
<tr>
<td>V_{OL}</td>
<td>Output Low Voltage (D₀–D₇)</td>
<td>0.45</td>
</tr>
<tr>
<td>V_{OL1}</td>
<td>Output Low Voltage (P₁₀P₁₇, P₂₀P₂₇, Sync)</td>
<td>0.45</td>
</tr>
<tr>
<td>V_{OL2}</td>
<td>Output Low Voltage (PROG)</td>
<td>0.45</td>
</tr>
<tr>
<td>V_{OH}</td>
<td>Output High Voltage (D₀–D₇)</td>
<td>2.4</td>
</tr>
<tr>
<td>V_{OH1}</td>
<td>Output High Voltage (All Other Outputs)</td>
<td>2.4</td>
</tr>
<tr>
<td>I_{IL}</td>
<td>Input Leakage Current (T₀, T₁, RD, WR, CS, A₀, EA)</td>
<td>±10</td>
</tr>
<tr>
<td>I_{OFL}</td>
<td>Output Leakage Current (D₀–D₇, High Z State)</td>
<td>±10</td>
</tr>
<tr>
<td>I_{LI}</td>
<td>Low Input Load Current (P₁₀P₁₇, P₂₀P₂₇)</td>
<td>0.3</td>
</tr>
<tr>
<td>I_{LI1}</td>
<td>Low Input Load Current (RESET, SS)</td>
<td>0.2</td>
</tr>
<tr>
<td>I_{DD}</td>
<td>V_{DD} Supply Current</td>
<td>20</td>
</tr>
<tr>
<td>I_{CC} + I_{DD}</td>
<td>Total Supply Current</td>
<td>135</td>
</tr>
<tr>
<td>I_{IH}</td>
<td>Input Leakage Current (P₁₀P₁₇, P₂₀P₂₇)</td>
<td>100</td>
</tr>
<tr>
<td>C_{IN}</td>
<td>Input Capacitance</td>
<td>10</td>
</tr>
<tr>
<td>C_{IO}</td>
<td>I/O Capacitance</td>
<td>20</td>
</tr>
</tbody>
</table>

**D.C. CHARACTERISTICS—PROGRAMMING**  \( (T_A = 25^\circ \text{C} \pm 5^\circ \text{C}, V_{CC} = 5V \pm 5\%, V_{DD} = 21V \pm 0.5V) \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_{DOH}</td>
<td>V_{DD} Program Voltage High Level</td>
<td>20.5</td>
<td>21.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{DDL}</td>
<td>V_{DD} Voltage Low Level</td>
<td>4.75</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{PH}</td>
<td>PROG Program Voltage High Level</td>
<td>17.5</td>
<td>18.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{PL}</td>
<td>PROG Voltage Low Level</td>
<td>V_{CC}−0.5</td>
<td>V_{CC}</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{EAH}</td>
<td>EA Program or Verify Voltage High Level</td>
<td>17.5</td>
<td>18.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{EAL}</td>
<td>EA Voltage Low Level</td>
<td>5.25</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_{DD}</td>
<td>V_{DD} High Voltage Supply Current</td>
<td>30.0</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_{PROG}</td>
<td>PROG High Voltage Supply Current</td>
<td>1.0</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_{EA}</td>
<td>EA High Voltage Supply Current</td>
<td>1.0</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A.C. CHARACTERISTICS (T_a=0°C to +70°C, V_{SS}=0V, V_{CC}=V_{DD}=+5V±10%)

DBB READ

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8042</th>
<th>8742</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{AR}</td>
<td>CS, A0 Setup to RD↓</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t_{RA}</td>
<td>CS, A0 Hold After RD↑</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t_{RR}</td>
<td>RD Pulse Width</td>
<td>160</td>
<td>160</td>
</tr>
<tr>
<td>t_{AD}</td>
<td>CS, A0 to Data Out Delay</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>t_{RD}</td>
<td>RD↓ to Data Out Delay</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>t_{DF}</td>
<td>RD↑ to Data Float Delay</td>
<td>85</td>
<td>85</td>
</tr>
</tbody>
</table>

DBB WRITE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8042</th>
<th>8742</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{AW}</td>
<td>CS, A0 Setup to WR↓</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t_{WA}</td>
<td>CS, A0 Hold After WR↑</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t_{WW}</td>
<td>WR Pulse Width</td>
<td>160</td>
<td>160</td>
</tr>
<tr>
<td>t_{DW}</td>
<td>Data Setup to WR↑</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>t_{WD}</td>
<td>Data Hold After WR↑</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

CLOCK

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8042</th>
<th>8742</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{CY}</td>
<td>Cycle Time</td>
<td>1.25</td>
<td>9.20</td>
</tr>
<tr>
<td>t_{CYC}</td>
<td>Clock Period</td>
<td>83.3</td>
<td>613</td>
</tr>
<tr>
<td>t_{PWH}</td>
<td>Clock High Time</td>
<td>33</td>
<td>38</td>
</tr>
<tr>
<td>t_{PWL}</td>
<td>Clock Low Time</td>
<td>33</td>
<td>38</td>
</tr>
<tr>
<td>t_{R}</td>
<td>Clock Rise Time</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>t_{F}</td>
<td>Clock Fall Time</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

NOTE:
1. t_{CY} = 15/f(XTAL)
# A.C. CHARACTERISTICS (T_A=25°C±5°C, V_CC=5V±5%, V_DD=21V±0.5V)

## PROGRAMMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tAW</td>
<td>Address Setup Time to RESET↑</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWA</td>
<td>Address Hold Time After RESET↑</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tDW</td>
<td>Data in Setup Time to PROG↑</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWD</td>
<td>Data in Hold Time After PROG↓</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tPH</td>
<td>RESET Hold Time to Verify</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tVDDW</td>
<td>V_DD Setup Time to PROG↑</td>
<td>0</td>
<td>1.0</td>
<td>mS</td>
<td></td>
</tr>
<tr>
<td>tVDDH</td>
<td>V_DD Hold Time After PROG↑</td>
<td>0</td>
<td>1.0</td>
<td>mS</td>
<td></td>
</tr>
<tr>
<td>tPW</td>
<td>Program Pulse Width</td>
<td>50</td>
<td>60</td>
<td>mS</td>
<td></td>
</tr>
<tr>
<td>tTW</td>
<td>Test 0 Setup Time for Program Mode</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWT</td>
<td>Test 0 Hold Time After Program Mode</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tDO</td>
<td>Test 0 to Data Out Delay</td>
<td></td>
<td></td>
<td></td>
<td>4tCY</td>
</tr>
<tr>
<td>tWW</td>
<td>RESET Pulse Width to Latch Address</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_r, t_f</td>
<td>V_DD and PROG Rise and Fall Times</td>
<td>0.5</td>
<td>100</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>tCY</td>
<td>CPU Operation Cycle Time</td>
<td>4.0</td>
<td></td>
<td></td>
<td>μs</td>
</tr>
<tr>
<td>tRE</td>
<td>RESET Setup Time Before EA↑</td>
<td></td>
<td></td>
<td></td>
<td>4tCY</td>
</tr>
</tbody>
</table>

**NOTE:**

If TEST 0 is high, tDO can be triggered by RESET↑.

## A.C. CHARACTERISTICS DMA

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8042</th>
<th>8742</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>tACC</td>
<td>DACK to WR or RD</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>tCAC</td>
<td>RD or WR to DACK</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>tACD</td>
<td>DACK to Data Valid</td>
<td>130</td>
<td>130</td>
<td>ns</td>
</tr>
<tr>
<td>tCRQ</td>
<td>RD or WR to DRQ Cleared</td>
<td>110</td>
<td>130</td>
<td>ns</td>
</tr>
</tbody>
</table>

**NOTE:**

1. C_L = 150 pF.

## A.C. CHARACTERISTICS PORT 2 (T_A=0°C to +70°C, V_CC= +5V ± 10%)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>f(tCY)</th>
<th>8042/8742 [3]</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCP</td>
<td>Port Control Setup Before Falling Edge of PROG</td>
<td>1/15 tCY−28</td>
<td>55</td>
<td>ns</td>
</tr>
<tr>
<td>tPC</td>
<td>Port Control Hold After Falling Edge of PROG</td>
<td>1/10 tCY</td>
<td>125</td>
<td>ns</td>
</tr>
<tr>
<td>tPR</td>
<td>PROG to Time P2 Input Must Be Valid</td>
<td>18/15 tCY−16</td>
<td>650</td>
<td>ns</td>
</tr>
<tr>
<td>tPF</td>
<td>Input Data Hold Time</td>
<td>0</td>
<td>150</td>
<td>ns</td>
</tr>
<tr>
<td>tDP</td>
<td>Output Data Setup Time</td>
<td>2/10 T_CY</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>tPD</td>
<td>Output Data Hold Time</td>
<td>1/10 tCY−80</td>
<td>45</td>
<td>ns</td>
</tr>
<tr>
<td>tPP</td>
<td>PROG Pulse Width</td>
<td>6/10 tCY</td>
<td>750</td>
<td>ns</td>
</tr>
</tbody>
</table>

**NOTES:**

1. C_L = 80 pF.
2. C_L = 20 pF.
3. t_CY = 1.25 μs
A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING LOAD CIRCUIT

DRIVING FROM EXTERNAL SOURCE-TWO OPTIONS

LC OSCILLATOR MODE

CRYSTAL OSCILLATOR MODE

EACH C SHOULD BE APPROXIMATELY 20 pF INCLUDING STRAY CAPACITANCE
WAVEFORMS

READ OPERATION—DATA BUS BUFFER REGISTER

WRITE OPERATION—DATA BUS BUFFER REGISTER

CLOCK TIMING

5-835
**WAVEFORMS (Continued)**

**COMBINATION PROGRAM/VERIFY MODE (EPROM’S ONLY)**

- **18V**
- **EA**
- **5V**
- **TEST0**
- **RESET**
- **DB0-DB7**
- **P16-P22**
- **VDD**
- **18V**
- **PROG**

---

**VERIFY MODE (ROM/EPROM)**

- **18V**
- **EA**
- **5V**
- **TEST0**
- **RESET**
- **DB0-DB7**
- **P16-P22**

---

**NOTES:**

1. PROG MUST FLOAT IF EA IS LOW OR IF TEST0 = 5V FOR THE 8742. FOR THE 8042 PROG MUST ALWAYS FLOAT.
2. A0 MUST BE HELD LOW (0 V) DURING PROGRAM/VERIFY MODES.
3. TEST 0 MUST BE HELD HIGH.

---

The 8742 EPROM can be programmed by the following Intel products:

1. Universal PROM Programmer (UPP 103) peripheral of the Intellec® Development System with a UPP-549 Personality Card.
2. IUP-200/IUP-201 PROM Programmer with the IUP-F87/44 Personality Module.
WAVEFORMS (Continued)

DMA

PORT 2

EXPANDER PORT

OUTPUT

PORT 20-3 DATA

PORT CONTROL

OUTPUT DATA

SYNC

EXPANDER PORT

INPUT

PORT 20-3 DATA

PORT CONTROL

INPUT DATA

PROG

PORT TIMING DURING EA

SYNC

P10-17 PORT DATA

PC

P20-22 PORT DATA

PC

ON THE RISING EDGE OF SYNC AND EA IS ENABLED, PORT DATA IS VALID AND CAN BE STROBED. ON THE TRAILING EDGE OF SYNC THE PROGRAM COUNTER CONTENTS ARE AVAILABLE.

5-837

210393-001
Table 2. UPI™ Instruction Set

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A, Rr</td>
<td>Add register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, @Rr</td>
<td>Add data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, #data</td>
<td>Add immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ADDC A, Rr</td>
<td>Add register to A with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, @Rr</td>
<td>Add data memory to A with carry</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL A, Rr</td>
<td>AND register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, @Rr</td>
<td>AND data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, Rr</td>
<td>OR register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, @Rr</td>
<td>OR data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate A left through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A right through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IN A, Pp</td>
<td>Input port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>OUTL Pp, A</td>
<td>Output A to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>OLR Pp, #data</td>
<td>OR immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>IN A, DBB</td>
<td>Input DBB to A, clear IBF</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OUT DBB, A</td>
<td>Output A to DBB, set OBF</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV STS, A</td>
<td>A8–A7 to Bits 4–7 of Status</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOVD A, Pp</td>
<td>Input Expander port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVDP Pp, A</td>
<td>Output A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANLD Pp, A</td>
<td>AND A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ORLD Pp, A</td>
<td>OR A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, Rr</td>
<td>Move register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, @Rr</td>
<td>Move data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, #data</td>
<td>Move immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV A, PSW</td>
<td>Move PSW to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOVPSW, A</td>
<td>Move to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A, Rr</td>
<td>Exchange A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCHR A, @Rr</td>
<td>Exchange A and data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, Rr</td>
<td>Exclusive OR register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, #data</td>
<td>Exclusive OR immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INC A</td>
<td>Increment A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal Adjust A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles of A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate A left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate A left through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate A right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A right through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IN A, DBB</td>
<td>Input DBB to A, clear IBF</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, PP</td>
<td>Input Expander port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVDP Pp, A</td>
<td>Output A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANLD Pp, A</td>
<td>AND A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ORLD Pp, A</td>
<td>OR A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, Rr</td>
<td>Move register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, @Rr</td>
<td>Move data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, #data</td>
<td>Move immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV A, PSW</td>
<td>Move PSW to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOVPSW, A</td>
<td>Move to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A, Rr</td>
<td>Exchange A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCHR A, @Rr</td>
<td>Exchange A and data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, Rr</td>
<td>Exclusive OR register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, #data</td>
<td>Exclusive OR immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INC A</td>
<td>Increment A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal Adjust A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles of A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate A left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate A left through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate A right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A right through carry</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, T</td>
<td>Read Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV T, A</td>
<td>Load Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STRT T</td>
<td>Start Timer</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STRT CNT</td>
<td>Start Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STOP TCNT</td>
<td>Stop Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EN TCNTI</td>
<td>Enable Timer/Counter Interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS TCNTI</td>
<td>Disable Timer/Counter Interrupt</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN DMA</td>
<td>Enable DMA Handshake Lines</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EN I</td>
<td>Enable IBF Interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS I</td>
<td>Disable IBF Interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EN FLAGS</td>
<td>Enable Master Interrupts</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL RB0</td>
<td>Select register bank 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL RB1</td>
<td>Select register bank 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC Rr</td>
<td>Increment register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC @Rr</td>
<td>Increment data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC Rr</td>
<td>Decrement register</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL addr</td>
<td>Jump to subroutine</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>RET</td>
<td>Return</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RETR</td>
<td>Return and restore status</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
## Table 2. UPI™ Instruction Set (Continued)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FLAGS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR C</td>
<td>Clear Carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL C</td>
<td>Complement Carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F0</td>
<td>Clear Flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F0</td>
<td>Complement Flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F1</td>
<td>Clear F1 Flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F1</td>
<td>Complement F1 Flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>BRANCH</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP addr</td>
<td>Jump unconditional</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JMPP @A</td>
<td>Jump indirect</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DJNZ Rr, addr</td>
<td>Decrement register and jump</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JC addr</td>
<td>Jump on Carry=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNC addr</td>
<td>Jump on Carry=0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JZ addr</td>
<td>Jump on A Zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNZ addr</td>
<td>Jump on A not Zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTO addr</td>
<td>Jump on T0=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNT0 addr</td>
<td>Jump on T0=0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JT1 addr</td>
<td>Jump on T1=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNT1 addr</td>
<td>Jump on T1=0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF0 addr</td>
<td>Jump on F0 Flag=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF1 addr</td>
<td>Jump on F1 Flag=1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTF addr</td>
<td>Jump on Timer Flag =1, Clear Flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNIBF addr</td>
<td>Jump on IBF Flag =0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JOBF addr</td>
<td>Jump on OBF Flag =1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JBb addr</td>
<td>Jump on Accumulator Bit</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
The Intel® 8243 is an input/output expander designed specifically to provide a low cost means of I/O expansion for the MCS-48® family of single chip microcomputers. Fabricated in 5 volts NMOS, the 8243 combines low cost, single supply voltage and high drive current capability.

The 8243 consists of four 4-bit bidirectional static I/O ports and one 4-bit port which serves as an interface to the MCS-48 microcomputers. The 4-bit interface requires that only 4 I/O lines of the 8048 be used for I/O expansion, and also allows multiple 8243's to be added to the same bus.

The I/O ports of the 8243 serve as a direct extension of the resident I/O facilities of the MCS-48 microcomputers and are accessed by their own MOV, ANL, and ORL instructions.
**FUNCTIONAL DESCRIPTION**

### General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports 4-7. The following operations may be performed on these ports:

- Transfer Accumulator to Port.
- Transfer Port to Accumulator.
- AND Accumulator to Port.
- OR Accumulator to Port.

All communication between the 8048 and the 8243 occurs over Port 2 (P20-P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles:

The first containing the “op code” and port address and the second containing the actual 4-bits of data. A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the 4-bit bus and chip selected using additional output lines from the 8048/8748/8035.

### Power On Initialization

Initial application of power to the device forces input/output ports 4, 5, 6, and 7 to the tri-state and port 2 to the input mode. The PROG pin may be either high or low when power is applied. The first high to low transition of PROG causes device to exit power on mode. The power on sequence is initiated if VCC drops below 1V.

#### Write Modes

The device has three write modes. MOVD Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data, OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port.

Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputed. The old data remains latched until new valid outputs are entered.

### Read Mode

The device has one read mode. The operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-state mode while port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.

---

**Table 1. Pin Description**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROG</td>
<td>7</td>
<td>Clock Input. A high to low transition on PROG signifies that address and control are available on P20-P23, and a low to high transition signifies that data is available on P20-P23.</td>
</tr>
<tr>
<td>CS</td>
<td>6</td>
<td>Chip Select Input. A high on CS inhibits any change of output or internal status.</td>
</tr>
<tr>
<td>P20-P23</td>
<td>11-8</td>
<td>Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation.</td>
</tr>
<tr>
<td>GND</td>
<td>12</td>
<td>0 volt supply</td>
</tr>
<tr>
<td>P40-P43</td>
<td>2-5</td>
<td>Four (4) bit bi-directional I/O ports. May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read). Data on pins P20-P23 may be directly written, ANDed or ORed with previous data.</td>
</tr>
<tr>
<td>P50-P53</td>
<td>1, 23-21</td>
<td>May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read).</td>
</tr>
<tr>
<td>P60-P63</td>
<td>20-17</td>
<td>May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read).</td>
</tr>
<tr>
<td>P70-P73</td>
<td>13-16</td>
<td>May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read).</td>
</tr>
<tr>
<td>VCC</td>
<td>24</td>
<td>+5 volt supply</td>
</tr>
</tbody>
</table>

---

**Write Modes**

The device has three write modes. MOVD Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data, OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputed. The old data remains latched until new valid outputs are entered.
**ABSOLUTE MAXIMUM RATINGS***

Ambient Temperature Under Bias .................. 0°C to 70°C
Storage Temperature .......................... -65°C to +150°C
Voltage on Any Pin With Respect to Ground ........... -0.5 V to +7V
Power Dissipation .................................. 1 Watt

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. CHARACTERISTICS**  \( T_A = 0°C \text{ to } 70°C, \ V_{CC} = 5V \ 10\%

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td></td>
<td>VCC+0.5 V</td>
<td></td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage Ports 4-7</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 4.5 mA*</td>
<td></td>
</tr>
<tr>
<td>VOL2</td>
<td>Output Low Voltage Port 7</td>
<td>1</td>
<td>V</td>
<td>IOL = 20 mA</td>
<td></td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage Ports 4-7</td>
<td>2.4</td>
<td>V</td>
<td>IOH = 240\mu A</td>
<td></td>
</tr>
<tr>
<td>IIL1</td>
<td>Input Leakage Ports 4-7</td>
<td>-10</td>
<td>20 \mu A</td>
<td>Vin = VCC to OV</td>
<td></td>
</tr>
<tr>
<td>IIL2</td>
<td>Input Leakage Port 2, CS, PROG</td>
<td>-10</td>
<td>10 \mu A</td>
<td>Vin = VCC to OV</td>
<td></td>
</tr>
<tr>
<td>VOL3</td>
<td>Output Low Voltage Port 2</td>
<td>.45</td>
<td>V</td>
<td>IOL = 0.6 mA</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>VCC Supply Current</td>
<td>10</td>
<td>20 mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOH2</td>
<td>Output Voltage Port 2</td>
<td>2.4</td>
<td></td>
<td>IOH = 100\mu A</td>
<td></td>
</tr>
<tr>
<td>IOL</td>
<td>Sum of all IOL from 16 Outputs</td>
<td>72</td>
<td>mA</td>
<td>4.5 mA Each Pin</td>
<td></td>
</tr>
</tbody>
</table>

*See following graph for additional sink current capability

**A.C. CHARACTERISTICS**  \( T_A = 0°C \text{ to } 70°C, \ V_{CC} = 5V \ 10\%

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tA</td>
<td>Code Valid Before PROG</td>
<td>100</td>
<td>ns</td>
<td>80 pF Load</td>
<td></td>
</tr>
<tr>
<td>tB</td>
<td>Code Valid After PROG</td>
<td>60</td>
<td>ns</td>
<td>20 pF Load</td>
<td></td>
</tr>
<tr>
<td>tC</td>
<td>Data Valid Before PROG</td>
<td>200</td>
<td>ns</td>
<td>80 pF Load</td>
<td></td>
</tr>
<tr>
<td>tD</td>
<td>Data Valid After PROG</td>
<td>20</td>
<td>ns</td>
<td>20 pF Load</td>
<td></td>
</tr>
<tr>
<td>tH</td>
<td>Floating After PROG</td>
<td>0</td>
<td>150</td>
<td>ns</td>
<td>20 pF Load</td>
</tr>
<tr>
<td>tK</td>
<td>PROG Negative Pulse Width</td>
<td>700</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tCS</td>
<td>CS Valid Before/After PROG</td>
<td>50</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tPO</td>
<td>Ports 4-7 Valid After PROG</td>
<td>700</td>
<td>ns</td>
<td>100 pF Load</td>
<td></td>
</tr>
<tr>
<td>tLP1</td>
<td>Ports 4-7 Valid Before/After PROG</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tACC</td>
<td>Port 2 Valid After PROG</td>
<td>650</td>
<td>ns</td>
<td>80 pF Load</td>
<td></td>
</tr>
</tbody>
</table>

24

2.0

0.8

TEST POINTS

0.8

2.0
WAVEFORMS

PROG

PORT 2

INSTRUCTION  FLOAT  DATA  FLOAT

PORT 2

OUTPUT VALID

PORTS 4-7

PREVIOUS OUTPUT VALID  OUTPUT VALID

PORTS 4-7

INPUT VALID

CS
Sink Capability

The 8243 can sink 5 mA @ .45V on each of its 16 I/O lines simultaneously. If, however, all lines are not sinking simultaneously or all lines are not fully loaded, the drive capability of any individual line increases as is shown by the accompanying curve.

For example, if only 5 of the 16 lines are to sink current at one time, the curve shows that each of those 5 lines is capable of sinking 9 mA @ .45V (if any lines are to sink 9 mA the total IOL must not exceed 45 mA or five 9 mA loads).

Example: How many pins can drive 5 TTL loads (1.6 mA) assuming remaining pins are unloaded?

IOL = 5 x 1.6 mA = 8 mA
εIOL = 60 mA from curve
# pins = 60 mA / 8 mA/pin = 7 5 = 7

In this case, 7 lines can sink 8 mA for a total of 56mA. This leaves 4 mA sink current capability which can be divided in any way among the remaining 8 I/O lines of the 8243.

Example: This example shows how the use of the 20 mA sink capability of Port 7 affects the sinking capability of the other I/O lines.

An 8243 will drive the following loads simultaneously.

2 loads—20 mA @ 1V (port 7 only)
8 loads—4 mA @ .45V
6 loads—3.2 mA @ .45V
Is this within the specified limits?

εIOL = (2 x 20) + (8 x 4) + (6 x 3.2) = 91.2 mA.
From the curve: for IOL = 4 mA, εIOL = 93 mA.
Since 91.2 mA < 93 mA the loads are within specified limits.

Although the 20 mA @ 1V loads are used in calculating εIOL, it is the largest current required @ .45V which determines the maximum allowable εIOL.

NOTE: A 10 to 50KΩ pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.
Figure 4. Expander Interface

Figure 5. Output Expander Timing

Figure 6. Using Multiple 8243's
Multimodule™ Winchester Controller Using the 82062

J. SLEEZER
TECHNICAL MARKETING
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>ST506 Winchester Drive Overview</td>
<td>1</td>
</tr>
<tr>
<td><strong>82062 WINCHESTER DISK CONTROLLER</strong></td>
<td>2</td>
</tr>
<tr>
<td>Clock Inputs</td>
<td>2</td>
</tr>
<tr>
<td>Microprocessor Interface</td>
<td>2</td>
</tr>
<tr>
<td>Sector Buffer Control</td>
<td>4</td>
</tr>
<tr>
<td>Data Transfer Logic</td>
<td>4</td>
</tr>
<tr>
<td>Drive Interface</td>
<td>6</td>
</tr>
<tr>
<td>Microprocessor Interfaces</td>
<td>8</td>
</tr>
<tr>
<td>PIN DESCRIPTIONS</td>
<td>10</td>
</tr>
<tr>
<td>TASK REGISTER FILE</td>
<td>12</td>
</tr>
<tr>
<td>Error Register</td>
<td>12</td>
</tr>
<tr>
<td>Reduce Write Current Register</td>
<td>18</td>
</tr>
<tr>
<td>Sector Count Register</td>
<td>13</td>
</tr>
<tr>
<td>Sector Number</td>
<td>13</td>
</tr>
<tr>
<td>Cylinder Number Low Register</td>
<td>13</td>
</tr>
<tr>
<td>Cylinder Number High Register</td>
<td>13</td>
</tr>
<tr>
<td>Sector/Drive/Head Register</td>
<td>14</td>
</tr>
<tr>
<td>Status Register</td>
<td>14</td>
</tr>
<tr>
<td>Command Register</td>
<td>15</td>
</tr>
<tr>
<td><strong>PROGRAMMING THE 82062</strong></td>
<td>16</td>
</tr>
<tr>
<td>Commands</td>
<td>16</td>
</tr>
<tr>
<td>Software Section: General Programming</td>
<td>23</td>
</tr>
<tr>
<td><strong>APPLICATION EXAMPLE</strong></td>
<td>29</td>
</tr>
<tr>
<td>iSBX Bus Multimodule Boards</td>
<td>29</td>
</tr>
<tr>
<td>The SBX82062 Design Example</td>
<td>31</td>
</tr>
<tr>
<td>Software Driver Overview</td>
<td>34</td>
</tr>
</tbody>
</table>
CONTENTS
APPENDIX A
ST506 INTERFACE ......................... A-1
THE ST506 INTERFACE ................... A-1
Data Transfer Rate ..................... A-1
ID Fields ............................. A-2
Sector Interleaving .................... A-2
Electrical Interface .................... A-3
ST412 HP (High Performance) Interface ................... A-4

CONTENTS
APPENDIX B
SOFTWARE DRIVER ...................... B-1
APPENDIX C
SCHEMATICS ........................... C-1
APPENDIX D
PAL SCHEMATICS ...................... D-1
INTRODUCTION

The 82062 Winchester Disk Controller (WDC) was developed to ease the complex task of interfacing Winchester disk drives to microprocessor systems. Specifically, the 82062 WDC interfaces to drives that conform to the ST506 specification, which is the dominant interface for 5¼ inch drives. This Application Note provides some background on the 82062 WDC, the drive interfaces and general software routines. It concludes with a design example using the 82062 WDC interfaced to the SBX™ bus. Appendix B contains the listing of the software necessary to operate this controller board.

ST506 Winchester Drive Overview

Since the 82062 WDC interfaces only to drives conforming to the ST506 specification, this overview will limit itself to those drives. A summary of the ST506 specification is shown in Appendix A for those who are not familiar with it. The ST506 Winchester Disk contains from 1 to 8 hard disks (or platters) with the average being 2 to 3 disks. These disks are made from aluminum (hence the term hard disk) and are coated with some type of recording media. The recording media is typically made of magnetic-oxide, which is similar to the material used on floppy disks and cassette tapes. Each side of a hard disk is coated with recording media and each side can store data. Each surface of a disk has its own read/write head.

Hard disk drives are sealed units because the R/W heads actually fly above the disk surface at about 8 to 20 microinches. A piece of dust or dirt, which appears as a boulder to the gap between the heads and the disk surface, will wreak havoc upon the disk media.

The R/W heads are mechanically connected together and move as a single unit across the surface of the disk. There are 2 basic methods for positioning the heads. The first is with stepper motors, which is the most common method and is also used on most floppy disk drives. These positioners are used mainly because of their low cost.

The second method of positioning the heads is to use a voice-coil mechanism. These units do not move in steps but swing across the disk. These mechanisms generally permit greater track density than steppers, but also require complex feedback electronics which increases the cost of the drive. Generally, voice-coil head positioners use closed loop servo positioning, as compared to the open loop positioning used with stepper motors.

The surface of a disk is divided logically into concentric circles radiating from the center as shown in Figure 1. Each concentric circle is called a track.

The group of same tracks on all cylinders is collectively called a cylinder. The number of tracks on a surface (which affects storage density) is determined by the head positioners. Typically, stepper head positioners have fewer tracks than drives that use a voice coil positioner. Which type of positioner is used is irrelevant to the 82062 as positioners are part of the drive electronics. The 82062 can access up to 1024 tracks per surface.

Once the surface is divided into cylinders it is further divided radially (as with a pie). The area between the radial spokes is referred to as a sector. The number of sectors per track is determined by many variables, but is basically determined by the number of data bytes and the length of the ID field (which locates a sector). Figure 2 shows one manufacturer's specifications for their drive. The manufacturer formats the drive with 32–256 byte sectors per track. Alternatively, the drive could be reformatted to contain 17–512. byte sectors per track. This second option has fewer sectors per track but stores more data. Determining how many bytes each sector contains is done by extensive analysis of the hardware and operating system. The 82062 WDC is programmable for sector size during formatting.

The order in which sectors are logically numbered on the track is called interleaving. An interleave factor of four would have three sectors separating logically se-

![Figure 1](https://via.placeholder.com/150)
### Table 1: A Typical Drive Specification

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Unformatted</th>
<th>Per Drive</th>
<th>6.38 Megabytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Per Surface</td>
<td>1.59 Megabytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Per Track</td>
<td>10416 Bytes</td>
<td></td>
</tr>
<tr>
<td>Formatted</td>
<td>Per Drive</td>
<td>5.0 Megabytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Per Surface</td>
<td>1.25 Megabytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Per Track</td>
<td>8192 Bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Per Sector</td>
<td>256 Bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sectors per Track</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Transfer Rate</td>
<td>5.0 Megabits</td>
<td>per second</td>
<td></td>
</tr>
<tr>
<td>Access Time</td>
<td>Track to Track</td>
<td>3 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average (Inc. Settle)</td>
<td>170 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maximum (Inc. Settle)</td>
<td>500 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Settling Time</td>
<td>15 ms</td>
<td></td>
</tr>
<tr>
<td>Functional Specifications</td>
<td>Average Latency</td>
<td>8.33 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rotational speed</td>
<td>3600 rpm ± 1%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Recording density</td>
<td>7690 bpi max</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Flux density</td>
<td>7690 fci</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Track density</td>
<td>255 tpi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cylinders</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tracks</td>
<td>612</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R/W Heads</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Disks</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2. A Typical Drive Specification**

Sequential sectors. Starting at the index pulse, an example of four way interleaving is:

Sector 1, Sector X, Sector Y, Sector Z, Sector 2, Sector

Interleaving is used primarily because one sector at a time is transferred from disk to sector buffer to system RAM. Without interleaving, the delay in transferring data would result in sectors on the disk rotating past the heads. The operating system would then have to wait one disk revolution to get to the next sector (a 16.7 msec delay). With interleaved sectors, the next logical sector would be positioned beneath the heads after the previous sector of data had been transferred to the system RAM. Interleaving unfortunately slows down the overall transfer rate from the disk. A 5 Mbit/second transfer rate averages out to a 1.25 Mbit/second transfer rate when many sectors are transferred with four way interleaving. Again, how much interleaving to use is determined by extensive hardware/software benchmarking.

Whenever data is stored on a multiple platter disk drive, the same track on all surfaces should be used before repositioning the heads to another track. Repositioning the heads generates a longer delay due to the mechanical delay of moving the heads. Switching to another head incurs no mechanical positioning delay. Only one head can be selected at a time.

Hard disk drives tend to be faster than floppies for two reasons. The speed at which the disk spins is about 10 times faster than the floppy (a floppy spins at 360 rpm). This yields an immediate one-tenth reduction in access times for the same size drive. While both ST506 drives and floppies use stepper motors, the steppers utilized by the hard disk drives are approximately twice as fast as those used by floppies.

### 82062 Winchester Disk Controller

The 82062 WDC provides most of the functions necessary to interface between a microprocessor and an ST506 compatible disk drive. The 82062 converts the high level commands and parallel data of a microprocessor bus into ST506 compatible disk control signals and serial MFM encoded data. This section presents a detailed description of the 82062 and a discussion of various techniques which can be used to interface the 82062 to a microprocessor.

The internal structure of the 82062 is divided into several sections as shown in Figure 3. They are:

1. the microprocessor interface which includes the status and task registers;
2. sector buffer control;
3. the drive interface;
4. and the data transfer section, which includes the CRC logic and the conversion and MFM encoding/decoding of microprocessor data.

### Clock Inputs

The 82062 has two clock inputs: read clock (RD CLOCK) and write clock (WR CLOCK). The PLA controller, the processor interface, buffer control and MFM encoding sections operate off the WR CLOCK input. The RD CLOCK input is used only for decoding the MFM data stream. The clocks may be asynchronous to one another. Both clocks have non-TTL compatible inputs. The easiest method to interface to TTL requires a pull-up resistor to satisfy their input voltage needs. The resistor’s value must be compatible with the VIL specification of these pins. See the Pin Descriptions Section for more specific information.

### Microprocessor Interface

The microprocessor interface of the 82062 contains the control logic which permits commands and data to be
transferred between the host and the 82062. The interface consists of an 8 bit, tri-state, bidirectional data bus; the task registers; a 3 to 8 address decoder for selecting one of the seven registers; and the general read, write, and chip select logic. Externally, the 82062 expects a buffer equal in size to a sector on the disk, and tri-state transceivers between the sector buffer and the microprocessors data bus in order to isolate itself from the microprocessor during disk data transfers.

**A0-A2, Data Bus**

These three address lines are active high signals and select one of the seven register locations in the 82062. They are not latched internally. If the three addresses are equal to 0 and the 82062 is selected, the data bus is kept tri-stated to ease interfacing to a sector buffer. The 82062's data bus is controlled by both the microprocessor and the 82062. The microprocessor has control for loading the registers and command. During disk reads or writes, control switches to the 82062 so that it may access the local sector buffer when transferring data between the disk and the buffer.

**RD, WR, CS**

The chip select (CS) is typically decoded from the higher order address lines. CS only permits data to be placed into, or read from, the 82062's task registers. Once a disk operation starts, CS no longer effects the 82062. RD and WR are bidirectional lines and are used to read or write the 82062's registers by the host microprocessor and are valid only if CS is present. The 82062 will drive RD and WR when transferring data between the sector buffer and the disk. A signal is provided to tri-state the RD and WR lines from the host during a buffer access. This is covered in the Sector Buffer Control Section.

**Interrupts**

An interrupt is issued at the end of all commands, and the interrupt is cleared by reading any register. For the Read Sector command only, the 82062 allows the user the option of an interrupt either at the termination of the command, as is the case with all other commands,
or when data needs to be transferred to the host from the sector buffer. This is discussed further in the Interrupt Mode Section. When selecting the data transfer option, the interrupt line will go active at the same time as the BDRQ line and the interrupt will be removed only when the proper handshake occurs with the sector buffer.

**Task Registers**

The Task Register File contains the command, status, track number, sector number, and other information necessary to properly execute a command. These registers are accessed with A0-A2, RD (or WR), and CS being valid and are not cleared by a reset. The registers are covered in detail in the Task Register File Section.

**Sector Buffer Control**

The 82062 was designed to operate with an external buffer equal in size to one sector. To ease the design-in of this buffer, the 82062 provides all of the control signals it needs to operate the buffer. This buffer must be isolated from the system bus, using tri-state buffers, during disk transfers to prevent contention during the period that the 82062 is accessing the buffer. A sector buffer is generally used to ease interfacing to the system due to the high disk data rates (625 kbytes/sec), though it is not required.

**BCS**

The Buffer Chip Select (BCS) line goes active whenever the 82062 is accessing the sector buffer. This signal should remove the microprocessors ability to access the 82062 and sector buffer and must enable the sector buffer for use by the 82062.

At a 5 Mbit/sec disk data rate, the 82062 will access the buffer every 1.6 microseconds (8 bits × 200 ns/bit). BCS will remain low the entire time the 82062 is accessing the buffer. The 82062 will pulse the appropriate RD or WR line for each byte transferred.

**BCR**

Buffer Counter Reset (BCR) goes active each time that BCS changes state. Its purpose is to reset the address counter of the sector buffer back to zero before and after the 82062 uses the sector buffer. Its function is optimized for single sector transfers. Multiple sector transfers should use a software controlled buffer counter reset and not use BCR as the sector buffer will be reset to the beginning after each sector is transferred.

**BDRQ, BRDY**

Buffer Data Request (BDRQ) and Buffer Ready (BRDY) provide the handshake needed to transfer data between the sector buffer and the host. BDRQ signals that data must be moved to/from the sector buffer and the host. BRDY has two functions. Once the transfer signaled by BDRQ is finished, asserting BRDY will inform the 82062 that the transfer is completed and that it may finish executing the command. BRDY is also used in multiple sector commands. BRDY going high during a multiple sector transfer indicates that the buffer is full (or empty—depending upon the command) and the transfer should wait until the buffer is serviced. The sector that was being transferred will finish and the 82062 will deactivate BCS and activate BDRQ. The host microprocessor must then transfer the data between the buffer and system memory. When this transfer is finished, asserting BRDY will cause the 82062 to resume the command.

The handshaking between BDRQ and BRDY occurs only in full sector increments—not on a byte basis. A high on BDRQ indicates a full sector's worth of data is required; BRDY going high indicates a full sector of data is available to the 82062 without interruption.

Only the rising edge of BRDY is valid. A falling edge may occur at any time without effect. BCR will pulse and BCS will go active eight byte times (8 bytes × 8 bits/byte × 200 ns/bit = 12.8 microseconds) before the first data byte is transferred from the sector buffer to the disk.

![Figure 4. BRDY Generation Logic](image)

**Data Transfer Logic**

This section of the 82062 is responsible for conversion of serial disk data to parallel data (and vice versa); encoding/decoding of the disk's MFM serial bit stream; detecting the address mark; and verifying data integrity through its CRC generation and checking logic.

5-852
MFM Encoding/Decoding

The MFM encoding section will receive 8 bit parallel data when a valid command has been recognized and BRDY has gone high. The parallel data is first serialized and converted to an intermediate, NRZ encoded, data stream. The serial NRZ data is sent to the MFM encoding section and then transferred to the disk. Decoding of the MFM bit stream (during disk reads) happens in reverse order.

The control logic operates off the write clock (WR CLOCK) running at a frequency of the desired transfer rate. The MFM decoding portion operates off of the read clock (RD CLOCK) input, which is supplied by an external phase lock loop. The two clocks need not be synchronized to each other. Data is written (and hence read) with the most significant bit first.

Address Mark Detector

The address mark is a unique 2 byte code written at the beginning of each ID field and data field. This address mark serves two purposes. It tells the controller what type of data is about to be received so that internal computations can be performed, and to ensure that ID fields are not sent to the host. The second purpose is to align the serial data back to the original 8 bit boundaries that existed when data was written (there are no byte boundaries on a disk).

An address mark is always preceded by the all zeros synchronization field. The 82062 starts comparing the incoming data stream when the synchronization field ends. A high speed comparator is used since the 82062 does not yet know where the proper byte boundaries are. When a proper comparison of the address mark is made the controller starts assembling bytes, starting with the second byte of the address mark.

The first byte of the address mark is an “A1” Hex, but purposely violates the MFM encoding rules by removing a clock pulse. In Figure 5, the first example is of a normal MFM encoded A1H. The second example is of the address mark and shows the missing clock pulse. The non-MFM compatible A1 is to prevent the host from issuing a similar data byte and possibly confusing detection logic.

The second byte specifies either an ID or data field and is encoded according to normal MFM rules. It is either an “F8” Hex for a data field, or “FC” through “FF” for an ID field. The different values correspond to a range of cylinders on the drive in increments of 256 tracks. The 82062 makes no use of this information, but writes it for compatibility with the ST506 specification during formatting.

CRC Generation/Checking

The CRC generator computes and checks the cyclic redundancy check bytes that are appended to the ID and data fields. CRC generation/checking is always done on ID fields. Data fields have a choice between 82062 CRC or externally supplied ECC. Read Sector commands with a CRC error will still have transferred the data into the sector buffer. When bit 7 in the SDH register is low (enabling CRC for data fields) the CRC bytes are not transferred to the sector buffer or host.

The generator polynomial for the CRC-CCITT (CRC-16) code is:

\[ x^{16} + x^{12} + x^5 + 1 = (x + 1)(x^{15} + x^{14} + x^{13} + x^{12} + x^4 + x^3 + x^2 + x + 1) \]

The code's capability is as follows:

a) Detects all occurrences of an odd number of bits in error.

b) Detects all single, double, and triple bit errors if the record length (including check bits) is less than 32,767 bits.

c) Detects all single-burst errors of sixteen bits or less.

d) Detects 99.99695% of all possible 17 bit burst errors, and 99.99847% of all possible longer burst, assuming all errors are possible and equally probable.

The CRC code has some double-burst capability when used with short records (sectors). For a 256 byte sector the code will detect double-bursts as long as the total number of bits in error does not exceed 7.
**PLA Control**

The PLA Controller interprets command sent by the microprocessor. Its operation is synchronized to the WR CLOCK input. The PLA controller is started when a command is written into the command register. It generates control signals and operates in a handshake mode when communicating with the MFM decoding block.

**Magnitude Comparator**

A 10 bit magnitude comparator is used to calculate the direction and number of step pulses needed to move the head from the present cylinder position to the desired position. A separate high speed equivalence comparator is used to compare ID field bytes when searching for a sector ID field.

**Drive Interface**

The drive interface of the 82062 contains the logic that makes possible the storage and reliable recovery of data. This interface consists of the drive and head select logic, the disk control signals, and read and write data logic as shown in Figure 6. This section describes the external circuitry which is required to complete the 82062's drive interface.

![Figure 6. Drive Interface](image-url)

---

**Figure 6. Drive Interface**
Drive/Head Select

The 82062 has no outputs for selecting the head or drive. Therefore these signals must be generated by the user as shown in Figure 6. Data bits 0–4 should be latched whenever the SDH register is written. Bits 0–2 would then be driven onto the drive cable with open collector buffers. Bits 3 and 4 would be decoded after being latched, then buffered for the cable. The head information written to the 82062's SDH register is used to write the proper ID fields during formatting. Changing the drive bits in the SDH register will cause a Scan ID to be performed by the 82062 to update non user accessible registers.

Drive Control

The drive control (STEP, DIR, WR FAULT, TRACK 000, INDEX, SC, RWC, and WR GATE) signals are merely conditioned for transmission over the drive cable. The purpose of each pin can be found in the section on Pin Descriptions and their use in the Command Section.

WR DATA, EARLY, LATE

Figure 7 is a diagram of the interface required on the write data line. The final stage of the MFM encoding requires applying the WR DATA to an external flip-flop clocked at 10 MHz. The 82062 monitors the serial write data output for particular bit patterns that require precompensation to prevent bit shifting. EARLY and LATE are active on all cylinders and will normally require that RWC be factored into them to activate the data precompensation on the proper cylinder.

A delay line is required to generate the delayed data for precompensation since the actual delay varies between drive manufacturers. EARLY and LATE go active in the same clock period that generates the data bit to be shifted.
RD Data, DRUN, RD Gate

The read data interface is shown in Figure 8, and consists of the data run (DRUN) signal and a phase lock loop to generate the RD CLOCK input to decode the serial data. DRUN is generated from a retriggerable one-shot with a period just exceeding one bit cell. A sync field consisting of a string of clock pulses will continually retrigger the one-shot producing a steady high level on DRUN. The 82062 counts off 16 clock pulses internally, and if DRUN is still active, will make RD GATE active. Any byte other than an address mark will deactivate RD GATE and the sequence starts over.

The phase lock loop generates RD CLOCK which is used to decode the incoming serial data. Until RD GATE is activated by the 82062, the phase lock loop (PLL) should be locked onto a local 10 MHz clock to minimize PLL lock-up times. When RD GATE is activated, the PLL starts locking onto the incoming data stream, which should consist of the all zeros sync field. Once the PLL locks onto this synch field, the 82062 will start examining the serial data for a non-zero byte. A non-zero byte will be indicated by DRUN dropping since the address mark follows the sync field and is an “A1” Hex. This sequence is shown in Figure 9. If the address mark is detected, and if it was preceded by at least 9 bytes of zeros, RD GATE will stay active. The 82062 will then assemble bytes of data, and ensure the proper ID field is found. If a non-zero or non-address mark byte was detected, RD GATE will go inactive for a minimum of 2 byte times. If a data field or the wrong ID field was detected, or the ID field was not preceded by 8 bytes of zeros, then RD GATE goes inactive and the sequence starts over with the 82062 examining the DRUN input.

Microprocessor Interfaces

This section shows the general 82062 interfaces to a microprocessor system. There are essentially four interfaces which consist of a combination of polled, DMA, and interrupts. While the 82062 was designed to interface directly to one type, it accommodates all with minor additional logic.

DMA Interface

The 82062 is designed to use a DMA controller for data transfer between its sector buffer and the host system, and to interrupt the host when the command has finished. This interface is shown in Figure 10.

When the 82062 determines that a transfer is needed between the sector buffer and the host (either at the beginning of a command or through BRDY going active in a multiple sector transfer), it will assert BDRQ. BDRQ will initiate a DMA transfer via the DMA request input. The DMA controller will generate reads or writes which will increment an address counter. BRDY indicates that the data transfer has finished and is issued off the carry-out line (or high order address line) of the counter. The 82062 will assert BDRQ at this point and activate BCS to prevent the host from interfering with disk/buffer transfers. There can be no polling for a data transfer or a register read without an interrupt in this scheme.

Figure 9. PLL Control Sequence
Figure 10. 82062 DMA Interface

Figure 11. 82062 Polled Interface

Figure 12. 82062 Interrupt Interface
Polled Interface

Since the 82062 isolates itself from the host during several commands, the host cannot read the status register during some periods to determine what course should be taken. In Figure 10, trying to read the status register when BCS is active will return indeterminate data. To prevent the microprocessor from reading this indeterminate data, a hardware generated “Busy” pattern should be driven onto the data bus if BCS is active. This is shown in Figure 11. The status register contains a data request (DRQ) bit whose timing is equal to the BDRQ output signal, thus making a polled operation possible. DRQ will stay set in the status register until a BRDY is generated.

One design issue with the polled interface occurs when the microprocessor is polling the status and the 82062 deactivates BCS. The microprocessor would normally read the hardware busy pattern. If BCS is deasserted, the hardware pattern is disabled and the microprocessor will start to read the real status register. The read cycle may almost be finished, and the read access period of the 82062 will not be satisfied. The data returned to the microprocessor will be invalid.

Interrupt Interface

There are cases where the designer does not want to tie up the microprocessor with polling. The typical 82062 design will need two interrupts per command. One for a data transfer and one for the completion of the command. The 82062 has an output to issue an interrupt when the command has finished. However for data transfers an interrupt must be generated from the BDRQ line as shown in Figure 12 (whether a DMA controller is used or not). When a data transfer is needed, the 82062 will activate the BDRQ line. The microprocessor will be interrupted and do the data transfer function. BDRQ will stay active until BRDY is generated, so the system must either use edge-triggered interrupts or must not write the end-of-interrupt byte until BDRQ is removed (this is true of Intel's 8259A).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin. No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCS</td>
<td>1</td>
<td>O</td>
<td>Buffer Chip Select: Output used to enable reading or writing of the external sector buffer by the 82062. When low, the host should not be able to drive the 82062 data bus, RD, or WR lines.</td>
</tr>
<tr>
<td>BCR</td>
<td>2</td>
<td>O</td>
<td>Buffer Counter Reset: Output that is strobed by the 82062 prior to read/write operation. This pin is strobed whenever BCS changes state. Used to reset the address counter of the buffer memory.</td>
</tr>
<tr>
<td>INTRQ</td>
<td>3</td>
<td>O</td>
<td>Interrupt Request: Interrupt generated by the 82062 upon command termination. It is reset when any register is read. Optionally signifies when a data transfer is required on Read Sector commands.</td>
</tr>
<tr>
<td>N/C</td>
<td>4</td>
<td></td>
<td>No connection. Reserved for future use.</td>
</tr>
<tr>
<td>RESET</td>
<td>5</td>
<td>I</td>
<td>Reset: Initializes the controller and clears all status flags. Does not clear the Task Registers.</td>
</tr>
<tr>
<td>RD</td>
<td>6</td>
<td>I/O</td>
<td>Read: As an input, RD controls the transfer of information from the 82062 registers to the host. RD is an output when the 82062 is reading data from the sector buffer (BCS low).</td>
</tr>
<tr>
<td>WR</td>
<td>7</td>
<td>I/O</td>
<td>Write: As an input, WR controls the transfer of command or task information into the 82062 registers. WR is an output when the 82062 is writing data to the sector buffer (BCS low).</td>
</tr>
<tr>
<td>CS</td>
<td>8</td>
<td>I</td>
<td>Chip Select: Enables RD and WR as inputs for access to the Task Registers. It has no effect once a disk command starts.</td>
</tr>
<tr>
<td>A0-A2</td>
<td>9-11</td>
<td>I</td>
<td>Address: Used to select a register from the task register file.</td>
</tr>
<tr>
<td>DB0-DB7</td>
<td>12-19</td>
<td>I/O</td>
<td>Data Bus: Bidirectional 8-bit Data Bus with control determined by BCS. When BCS is high the microprocessor has full control of the data bus for reading and writing the Task Registers. When BCS is low the 82062 controls the data bus to transfer data to or from the buffer.</td>
</tr>
</tbody>
</table>
## Pin Descriptions (continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin. No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>20</td>
<td></td>
<td>Ground.</td>
</tr>
<tr>
<td>WR DATA</td>
<td>21</td>
<td>O</td>
<td>Write Data: Open drain output that shifts out MFM data at a rate determined by Write Clock. Final stage requires an external flip-flop clock at 10 MHz. See note 1.</td>
</tr>
<tr>
<td>LATE</td>
<td>22</td>
<td>O</td>
<td>Late: Open drain output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders. See note 1.</td>
</tr>
<tr>
<td>EARLY</td>
<td>23</td>
<td>O</td>
<td>Early: Open drain output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders. See note 1.</td>
</tr>
<tr>
<td>WR GATE</td>
<td>24</td>
<td>O</td>
<td>Write Gate: High when write data is valid. WR GATE goes low if the WR FAULT input is active. This output is used by the drive to enable head write current.</td>
</tr>
<tr>
<td>WR CLOCK</td>
<td>25</td>
<td>I</td>
<td>Write Clock: Clock input used to derive the write data rate. Frequency — 5 MHz for the ST506 interface, 4.34 MHz for the SA 1000 interface. See Note 2.</td>
</tr>
<tr>
<td>DIR</td>
<td>26</td>
<td>O</td>
<td>Direction: High level on this output tells the drive to move the head inward (increasing cylinder number). The state of this signal is determined by the 82062’s internal comparison of actual cylinder location vs desired cylinder.</td>
</tr>
<tr>
<td>STEP</td>
<td>27</td>
<td>O</td>
<td>Step: Provides 8.4 microsecond pulses to move the drive head to another cylinder at a programmable frequency.</td>
</tr>
<tr>
<td>DRDY</td>
<td>28</td>
<td>I</td>
<td>Drive Ready: If DRDY from the drive goes low, the command will be terminated.</td>
</tr>
<tr>
<td>INDEX</td>
<td>29</td>
<td>I</td>
<td>Index: Signal from the drive indicating the beginning of a track. It is used by the 82062 during formatting, and for counting retries. Index is edge triggered. Only the rising edge is valid.</td>
</tr>
<tr>
<td>WR FAULT</td>
<td>30</td>
<td>I</td>
<td>Write Fault: An error input to the 82062 which indicates a fault condition at the drive. If WR FAULT from the drive goes high, the command will be terminated.</td>
</tr>
<tr>
<td>TRACK 000</td>
<td>31</td>
<td>I</td>
<td>Track Zero: Signal from the drive which indicates that the head is at the outermost cylinder. Used by the Restore command.</td>
</tr>
<tr>
<td>SC</td>
<td>32</td>
<td>I</td>
<td>Seek Complete: Signal from the drive indicating to the 82062 that the drive head has settled and that reads or writes can be made. SC is edge triggered. Only the rising edge is valid.</td>
</tr>
<tr>
<td>RWC</td>
<td>33</td>
<td>O</td>
<td>Reduced Write Current: Signal goes high for all cylinder numbers above the value programmed in the Write Precomp Cylinder register. It is used by the precompensation logic and by the drive to reduce the effects of bit shifting.</td>
</tr>
<tr>
<td>DRUN</td>
<td>34</td>
<td>I</td>
<td>Data Run: This signal informs the 82062 when a field of ones or zeros has been detected by an external one-shot. This indicates the beginning of an ID field. RD GATE is brought high when DRUN is sampled high for 16 clock periods. See Note 2.</td>
</tr>
<tr>
<td>BRDY</td>
<td>35</td>
<td>I</td>
<td>Buffer Ready: Input used to signal the controller that the buffer is ready for reading (full), or writing (empty), by the host μP. Only the rising edge indicates the condition.</td>
</tr>
</tbody>
</table>
Pin Descriptions (continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin. No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDRQ</td>
<td>36</td>
<td>O</td>
<td><strong>Buffer Data Request:</strong> Activated during Read or Write commands when a data transfer between the host and the 82062’s sector buffer is required. Typically used as a DMA request line, or to generate an interrupt.</td>
</tr>
<tr>
<td>RD DATA</td>
<td>37</td>
<td>I</td>
<td><strong>Read Data:</strong> Single ended input that accepts MFM data from the drive. See note 2.</td>
</tr>
<tr>
<td>RD GATE</td>
<td>38</td>
<td>O</td>
<td><strong>Read Gate:</strong> Output that is high for data and ID fields. Goes active when DRUN has been high for 16 WR CLOCK periods to permit the external phase lock loop to lock onto the incoming disk data stream.</td>
</tr>
<tr>
<td>RD CLOCK</td>
<td>39</td>
<td>I</td>
<td><strong>Read Clock:</strong> Clock input derived from the external data recovery circuits. See note 2.</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td>I</td>
<td><strong>D.C. Power:</strong> +5V</td>
</tr>
</tbody>
</table>

NOTES:
1. This pin requires a pull-up resistor to function properly. A value of 1000 ohms will work satisfactorily.
2. This pin requires input levels that are not TTL compatible. These lines can be interfaced to TTL with a pull-up resistor. Too small of a resistor will produce a $V_{IL}$ level that is too high. Too large of a resistor will degrade the signal’s rise time. A minimum value for the resistor is determined as follows:

$$\frac{(V_{CC} \text{ max}) - (82062 \ V_{IL} \text{ max})}{(TTL \ I_{OL} \text{ min.}) - (82062 \ I_{IL} \text{ max})} = \text{Resistor}$$

**Task Register File**

The Task Register File is a bank of registers used to hold parameter information pertaining to each command. These registers and their addresses are:

<table>
<thead>
<tr>
<th>A2</th>
<th>A1</th>
<th>A0</th>
<th>READ</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(Bus Tri-Stated)</td>
<td>(Bus Tri-Stated)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Error Flags</td>
<td>Reduce Write Current</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Sector Count</td>
<td>Sector Count</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Sector Number</td>
<td>Sector Number</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Cylinder Low</td>
<td>Cylinder Low</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Cylinder High</td>
<td>Cylinder High</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>SDH</td>
<td>SDH</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Status Register</td>
<td>Command Register</td>
</tr>
</tbody>
</table>

**Error Register**

This read-only register contains specific error status after the completion of a command. If any bit in this register is set, then the Error bit in the Status Register will also be set. The bits are defined as follows:

7 6 5 4 3 2 1 0

BBD CRC ID - AC TK000 DM

Bit 7 - Bad Block Detect
This bit is set when an ID field has been encountered that contains a bad block mark. The bad block bit is set only during formatting. The 82062 will terminate a command if an attempt is made to read a sector that contains this bit.

Bit 6 - CRC Data Field
This bit is set when a data field CRC error has occurred. The sector buffer may still be read but will contain errors.

Bit 5 - Reserved.
Not used. Set to zero.

Bit 4 - ID Not Found
This bit is set when the desired cylinder, head, sector or size parameter cannot be found after 8 revolutions of the disk, or if an ID field CRC error has occurred.

Bit 3 - Reserved.
Not used. Set to zero.
Bit 2 - Aborted Command

This bit is set if a command was issued or in progress while DRDY (Pin 28) was deasserted or WR FAULT (Pin 30) was asserted. The Aborted Command bit will also be set if an undefined command is written into the COMMAND register, but an implied seek will be executed.

Bit 1 - TRACK 000

This bit is set only by the RESTORE command. It indicates that TRACK 000 (Pin 31) has not gone active after the issuance of 1024 stepping pulses.

Bit 0 - Data Address Mark

This bit is set during a READ SECTOR command if the Data Address Mark is not found after the proper Sector ID is read.

Reduce Write Current Register

This register is used to define the cylinder number where RWC (Pin 33) is asserted:

```
7 6 5 4 3 2 1 0
```

The value (0-255) written into this register is internally multiplied by 4 to specify the actual cylinder where RWC is asserted. Thus a value of 01H will cause RWC to activate on cylinder 4, 02H on cylinder 8 and so on. RWC will be asserted when the present cylinder is greater than or equal to the cylinder indicated by this register. For example, one ST506 compatible drive requires precompensation on cylinder 128 (80H) and above. Therefore the REDUCE WRITE CURRENT register should be loaded with 32 (20H). A value of FFH will keep the RWC output inactive regardless of the actual cylinder number.

Sector Number

This register holds the sector number of the desired sector:

```
7 6 5 4 3 2 1 0
```

For a multiple sector command it specifies the first sector to be transferred. It is decremented after each sector is transferred to/from the sector buffer. The SECTOR NUMBER register may contain any value from 0 to 255. The ID Not Found bit will be set if the desired sector cannot be located on the track.

The SECTOR NUMBER register is also used to program the Gap 1 and Gap 3 lengths to be used when formatting a disk. See the WRITE FORMAT command description for further explanation.

Cylinder Number Low Register

This register holds the lower byte of the desired cylinder number:

```
7 6 5 4 3 2 1 0
```

It is used in conjunction with the CYLINDER NUMBER HIGH register to specify a range of 0 to 1024 tracks.

Cylinder Number High Register

This register holds the two most significant bits of the desired cylinder number:

```
7 6 5 4 3 2 1 0
```

```
(x) (x)
```

x = ignored

The 82062 contains a pair of registers that store the actual position where the R/W head are located. The CYLINDER NUMBER HIGH and LOW registers are considered the cylinder destination registers for seeks and other commands. The 82062 compares its internal registers to the destination registers and issues the number of steps in the right direction to make both sets of registers equal. After a command is executed, the internal cylinder position registers' contents are equal to the cylinder high/low registers. If a drive number change is detected on a new command, the 82062 automatically reads an ID field to update its internal cylinder position registers. This affects all commands except a RESTORE.
Sector/Drive/Head Register

The SDH register contains the desired sector size, drive number, and head number parameters. The format is shown below.

```
7 6 5 4 3 2 1 0
EXT SECT SIZE DRIVE HEAD #
```

Both head number and sector size are compared against the disk's ID field. Head select and drive select lines are not available as outputs from the 82062 and must be generated externally.

Bit 7, the extension bit (EXT), is used to extend the data field by seven bytes when using ECC codes for READ/WRITE SECTOR commands. When EXT = 1, the CRC is not appended to the end of the data field and the data field becomes "sector size + 7" bytes long. The CRC is checked on the ID field regardless of the state of EXT. The SDH byte written into the ID field is different than the SDH Register contents. The recorded SDH byte does not have the drive number (DRIVE) written but does have the BAD BLOCK mark written. The EXT bit must not be set during the Format command.

Note that use of the extension bit requires the gap lengths to be modified as described in the WRITE FORMAT command description.

Status Register

The status register is a read-only register which informs the host of certain events. This register is a flow-through latch until the microprocessor reads it at which point the drive status lines are latched. The INTRQ line will be reset when this register is read. The format is:

```
7 6 5 4 3 2 1 0
BUSY READY WF SC DRQ - CIP ERROR
```

Bit 7 - Busy

This bit is set whenever the 82062 is transferring data between its sector buffer and the disk and reflects the state of the BCS pin. When BCS is active, the host should not access the sector buffer or any 82062 register. The 82062 will be generating a RD or WR pulse every 1.6 μsec and the host must not interfere with these data transfers. Busy is cleared when the data transfer operation is completed.

During other non-data transfer commands, Busy should be ignored as it will go active for short periods.

Bit 6 - Ready

This bit reflects the state of the DRDY (Pin 28) line at the time the microprocessor reads the status register. Transitions on the DRDY line will abort a command and set the aborted command bit in the error register.

Bit 5 - Write Fault

This bit reflects the state of the WR FAULT (Pin 30) line. Transitions on this line will abort a command and set the aborted command bit in the error register.

Short transitions on DRDY and WR FAULT may not show up in the status register. These pins are not latched until the microprocessor reads the status and by that time the error condition may have disappeared. However the aborted command bit will be set to notify the host of an error. To hold short transitions on these pins it is recommended that they be latched.

Bit 4 - Seek Complete

This bit reflects the state of the SC (Pin 32) line. Commands which initiate a seek will pause until Seek Complete is set.

Bit 3 - Data Request

The Data request bit (DRQ) reflects the state of the BDRQ (Pin 36) line. It is set when the sector buffer should be loaded with data or read by the host processor, depending upon the command. The DRQ bit and the BDRQ line remain high until BRDY is sampled, indicating the operation has completed.

Bit 2 - Reserved

Not used. Set to zero.

Bit 1 - Command in Progress

When this bit is set, a command is being executed and a new command should not be loaded until it is cleared. Although a command may be executing, the sector buffer is still available for access by the host processor. If CIP is set, only the status register can be read regardless of which register is selected.

Bit 0 - Error

This bit is an OR of the contents of the error register. Any bit being set in the error register sets this bit. This bit is cleared when a new command is loaded.
Command Register

This write-only register is loaded with the desired command:

```
7 6 5 4 3 2 1 0
```

The 82062 begins to execute immediately upon loading any value into this register. This register should not be written while the Busy or Command in Progress bits are set in the STATUS register. The INTRQ line (Pin 3) if set, will be cleared by a write to the COMMAND register.

Instruction Set

The 82062 WDC instruction set contains six commands. Prior to loading the command register, the host processor must first set up the Task Register File with the information needed for the command. Except for the COMMAND register, the registers may be loaded in any order. If a command is in progress, a subsequent write to the COMMAND register will be ignored. A command is finished when the command in progress (CIP) bit in the STATUS register is cleared. See the Command Section for an explanation of each command.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTORE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
</tr>
<tr>
<td>SEEK</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
</tr>
<tr>
<td>READ SECTOR</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>WRITE SECTOR</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>SCAN ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WRITE FORMAT</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

R 3 – 0 = Rate Field

For 5 MHz WR Clock:

- 0000 — ≈ 35 μs
- 0001 — 0.5 ms
- 0010 — 1.0 ms
- 0011 — 1.5 ms
- 0100 — 2.0 ms
- 0101 — 2.5 ms
- 0110 — 3.0 ms
- 0111 — 3.5 ms
- 1000 — 4.0 ms
- 1001 — 4.5 ms
- 1010 — 5.0 ms
- 1011 — 5.5 ms
- 1100 — 6.0 ms
- 1101 — 6.5 ms
- 1110 — 7.0 ms
- 1111 — 7.5 ms

Programming the 82062

This section consists of two parts. The first part gives an explanation of each command, a flowchart showing the 82062's sequence of events, and the commands' sequence of events as seen by the host microprocessor. The second section shows flowcharts of general software routines and their PLM equivalent, for both polled and interrupt driven software.

The designer must remember that the 82062 expects a full sector buffer that can be isolated from the host during data transfers between the 82062 and the disk. Since the 82062 assumes a full sector buffer is available, it does not check for data overrun or underrun error conditions. If such a condition occurs, corruption of data will happen and the host will have no indication of an error. The design must guarantee against over-run and under-run conditions when not using the sector buffer approach.

Commands

A command is placed into the command register only after the Task Registers have been written with proper values. The Task Registers may be loaded in any order. A command, once started, can only be terminated by a hardware reset to the 82062. This may corrupt data on the disk by removing necessary control signals out of sequence.

The general sequence of a command is as follows:

- The host loads the Task Registers
- The host loads the Command Register
- The 82062 locates the correct cylinder
- Data transfer takes place
- The 82062 issues an interrupt
The Restore command is used to position the heads to cylinder 0. This command must be issued to the 82062 on power-up to initialize internal registers. The user specified rate field (R3-R0) is stored internally for FUTURE use in commands with implied seeks. The step rate value is not used with this command. The actual stepping rate used is dependent upon the handshake delay between the 82062 issuing a step pulse and the drive returning a seek complete for each track (roughly 20 ms). After each step pulse is issued, the 82062 waits for a rising edge on the Seek Complete (SC) line before issuing the next pulse. If 8 index pulses are received without a rising edge on SC, the 82062 will switch to sampling the level of the SC line. If after 1,024 step pulses the Track 00 signal has not gone active, the 82062 will terminate the command and set the TRACK 000 bit in the Error Register. The command will terminate if WR Fault goes active or DRDY goes inactive at any time. Figure 13 is a flow chart of the command.

This command should precede the format command. The format command will be aborted if an ID field is not present (because the disk was never formatted) and
a new drive is selected. Recall the 82062 will do a Scan ID to update internal registers when the drive is changed. This information is used to calculate the number of steps required to get to the destination cylinder. When the heads are positioned to track zero the 82062 will not try to read an ID field, but will issue the correct number of steps.

Seek Command – 0 1 1 1 R3 R2 R1 R0

The Seek command positions the heads to the cylinder specified in the Task Registers. The direction and number of step pulses issued is calculated by comparing the cylinder high/low registers to an internal “present position” cylinder register. The present position register is updated after all step pulses are issued and the command is terminated. The Seek Complete input is not checked.

The actual stepping rate is taken from the rate field bits (R3–R0) and stored for future use. The command terminates at once if WR FAULT goes active or DRDY goes inactive at any time. Figure 14 is a flowchart of the command.

Since the data transfer commands feature implied seeks, this command is of use mainly to those using multiple drives and software that can take advantage of overlapped seeks.

Scan ID Command – 0 1 0 0 0 0 0 T

The Scan ID command is used by both the 82062 and the host to update the SDH, the Sector Number, Cylinder and internal present position registers. Once the command is issued, the Seek Complete line is sampled until valid. The first ID field found, as indicated by the address mark, is loaded into the previously mentioned registers. The Bad Block bit will be set if detected, and the command will terminate. ID CRC errors will start the search sequence over for a maximum of 10 index pulses, but the registers will be loaded with whatever data the 82062 had perceived as ID information. Improper states on WR Fault on DRDY will terminate the command. Figure 15 is the flow chart of the command.

The main use for this command is to determine where the heads are currently located and what size the sectors are (i.e. 256, 512 etc.). Without this command, it would be necessary to recall the heads to track zero and then step out to the desired cylinder each time a drive was changed. Specifying the wrong sector size would yield an ID not found error. This command enables the system to read the disk drive to determine what size sectors were recorded.

Read Sector Command – 0 0 1 0 1 M 0 T

The READ SECTOR command is used to transfer one or more sectors of data from the disk to the sector buffer. Upon receipt of the READ SECTOR command, the 82062 checks the CYLINDER NUMBER LOW/HIGH register pair against an internal cylinder position register to see if they are equal. If not, the direction and number of steps are calculated and a seek takes place. If an implied seek is performed, the 82062

![Figure 15. Scan ID Command Flow](image-url)
will search until a rising edge of SC is received. The WR FAULT and DRDY lines are monitored throughout the command.

Once the Seek Complete (SC) line is high (with or without an implied seek having occurred), the search for an ID field begins. If \( T = 0 \) (retries enabled), the 82062 must find an ID with the correct cylinder number, head, sector size, and CRC within 10 revolutions, or a Scan ID and re-Seek will be performed. The search for the proper ID will again be tried for up to 10 revolutions. If the correct sector is still not found, the appropriate error bits will be set and the command terminated. Data CRC errors will also be retried for up to 10 revolutions (if \( T = 0 \)).

If \( T = 1 \) (retries disabled), the ID search must find the correct sector within 2 revolutions or the appropriate error bits will be set and the command terminated.

Both the READ SECTOR and WRITE SECTOR commands feature a “simulated completion” to ease programming. DRQ/BDRQ will be generated upon detecting an error condition. This allows the same program flow for successful or unsuccessful completion of a command.

In summary then, READ SECTOR operation is as follows:

When \( M = 0 \) (READ SECTOR)

1. Host: Sets up parameters; issues READ SECTOR command.
2. 82062: Strobes BCR; sets BCS = 0.
3. 82062: Finds sector specified; transfers data to buffer.
4. 82062: Strobes BCR; sets BCS = 1.
5. 82062: Sets BDRQ = 1; DRQ = 1.
6. 82062: If \( I = 1 \) go to (9).
8. 82062: Waits for BRDY, then sets INTRQ = 1; END.
10. Host: Reads out contents of buffer; END.
11. 82062: If \( I = 1 \) wait for BRDY, then clear BDRQ; END.

When \( M = 1 \) (READ MULTIPLE SECTOR)

1. Host: Sets up parameters; issues READ SECTOR command.
2. 82062: Strobes BCR; sets BCS = 0.
3. 82062: Finds sector specified; transfers data to buffer.
4. 82062: Decrement SECTOR COUNT register; increments SECTOR NUMBER register.
5. 82062: Strobes BCR; sets BCS = 0.
6. 82062: Sets BDRQ = 1; DRQ = 1.
8. 82062: Waits for BRDY.
9. 82062: When BRDY = 1, if Sector Count = 0 then go to (11).
10. 82062: Go to (2).
11. 82062: Set INTRQ = 1; End.

A flowchart of the READ SECTOR command is shown in Figures 16A and 16B.
Write Sector Command – 01110M0T

The WRITE SECTOR command is used to write one or more sectors of data to the disk from the sector buffer. Upon receipt of a WRITE SECTOR command the 82062 checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation is performed and a seek takes place. The WR FAULT and DRDY lines are checked throughout the command.

When the Seek Complete (SC) line is found to be true (with or without an implied seek having occurred), the BDRQ signal is made active and the host proceeds to load the buffer. Once BRDY goes high, the ID field with the specified cylinder number, head, and sector size is searched for. Once found, WR GATE is made active and the data is written to the disk. If retries are enabled (T = 0), and if the ID field cannot be found within 10 revolutions, a Scan ID and re-Seek are performed. If the correct ID field is not found within 10 additional revolutions, the ID Not Found error bit is set and the command is terminated. If retries are disabled, (T = 1) and if the ID field cannot be found within 2 revolutions, the ID Not Found error bit is set and the command is terminated.

During a WRITE MULTIPLE SECTOR command (M = 1), the SECTOR NUMBER register is decremented and the SECTOR COUNT register is incremented after the transfer to the disk takes place. During multiple sector transfers if BRDY is asserted after the first sector is transferred from the buffer, the 82062 will transfer the next sector before issuing BDRQ. The 82062 will set BDRQ and wait for the host processor to place more data in the buffer.

In summary then, the WRITE SECTOR operation is as follows:

When M = 0, 1 (WRITE SECTOR)

1) Host: Sets up parameters; issues WRITE SECTOR command.
2) 82062: Sets BDRQ = 1, DRQ = 1.
3) Host: Loads sector buffer with data.
4) 82062: Waits for BRDY = 0 to 1.
5) 82062: Finds specified ID field; writes sector to disk.
6) 82062: If M = 0, then set INTRQ = 1; END.
7) 82062: Increment SECTOR NUMBER register; decrement SECTOR COUNT register.
8) 82062: If SECTOR = 0, then set INTRQ = 1; END.
9) 82062: Go to (2).

A flowchart of the WRITE SECTOR command is shown in Figure 17.

Write Format Command 01010000

The WRITE FORMAT command is used to format one track using the Task Register File and the sector buffer. During execution of this command, the sector buffer is used for additional parameter information instead of sector data. Shown in Figure 18 is the contents of the sector buffer for a 32 sector/track format with an interleave factor of two. Each sector requires a two byte sequence. The first byte designates whether a bad block mark is to be recorded in the sector's ID field. A 00 Hex is normal; an 80H indicates a bad block mark for the sector. In the example of Figure 18, sector 04 will get a back block mark recorded. Any attempt to access sector 4 in the future will terminate the command.

The second byte indicates the logical sector number to be recorded. This allows sectors to be recorded with any interleave factor desired. The remaining memory in the sector buffer may contain any value. Its only purpose is to generate a BRDY to tell the 82062 to begin formatting the track. An implied seek is in effect on this command. As for other commands, if the drive number has been changed an ID field will be scanned for cylinder position information before the implied seek is performed. If no ID field can be read (because the track had been erased or because an incomplete format had been used), an ID Not Found error will result and the WRITE FORMAT command will be aborted. This can be avoided by issuing a RESTORE command before formatting.

The SECTOR COUNT register is used to hold the total number of sectors to be formatted (01H = 1 sector; 00H = 256 sectors), while the SECTOR NUMBER register holds the number of bytes (minus three) to be used for Gap 1 and Gap 3. For instance, if the SECTOR COUNT register value is 02H and the SECTOR NUMBER register value is 00H, then 2 sectors are written on a track and 3 bytes of 4EH are written for Gap 1 and Gap 3. The data fields are filled with FFH and the CRC is automatically generated and appended. All gaps are filled with 4EH. After the last sector is written, the track is filled with 4EH until the index pulse terminates the write. The Gap 3 value is deter-
IF \( T = 1 \) THEN DASHED PATH IS TAKEN AFTER 2 INDEX PULSES.

Figure 16A. Read Sector Command Flow
Figure 16B. Read Sector Command Flow
Figure 17. Write Sector Command Flow

AP-182

WRITE SECTOR
RESET INTRO ERRORS DRO
SET BUSY, CIP

YES

DRIVE # CHANGED

YES

SCAN ID GET CYL #

NO

CYL
REGISTERS & INTERNAL CYL REGISTER
SAME

NO

PERFORM SEEK COMMAND

SEEK COMPLETE?

YES

ACTIVATE BORQ

BRDY
ACTIVE?

NO

YES

RESET BORQ

2

ACTIVATE BCS PULSE BCR

WR FAULT?

YES

NO

SEARCH FOR ID FIELD

1

CORRECT ID FOUND?

NO

10 INDEXES?

NO

SCAN ID GET CYL #

YES

NOTE?

SELECT WR GATE

RESEEK TO CYL #

RESEEK MADE?

NO

WRITE DATA TO SECTOR THEN DEASSERT WR GATE

M = 0?

YES

INCREMENT SECTOR NUMBER DECREMENT SECTOR COUNT

SECTOR COUNT = 0?

YES

DEACTIVATE BCS PULSE BCR ACTIVATE BORQ

BRDY
ACTIVE?

NO

YES

2

SET ABORTED COMMAND

PULSE BCR, SET INTRO
RESET BUSY, CIP
DEACTIVATE BCS

"IF RETRIES ARE DISABLED THE DASHED PATH IS TAKEN AFTER 2 INDEX PULSES"
mined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 (no) interleave is used. The formula for determining the minimum Gap 3 length value is:

\[
\text{Gap 3} = (2 \times M \times S) + K + E
\]

- **M** = motor speed variation (e.g., 0.03 for ±3%)
- **S** = sector length in bytes
- **K** = 25 for interleave factor of 1
- **K** = 0 for any other interleave factor
- **E** = 7 if the sector is to be extended

As with all commands, a WR FAULT or drive not ready condition, will terminate execution of the WRITE FORMAT command. Figure 19 shows the format that the 82062 will write on the disk. The extend bit in the SDH register must not be set during the Format command.

A flowchart of the WRITE FORMAT command is shown in Figure 20.

### SOFTWARE SECTION: GENERAL PROGRAMMING

This section describes the software in a general manner and Appendix B contains the actual implementation used to exercise the 82062 SBX board.

#### Polled Mode

As discussed in the Polled Interface Section, the 82062 does not directly support polled operation for data transfers without the addition of hardware. This section is based upon the polled interface as described in the Polled Interface Section.

The six 82062 commands can be divided into two groups, those with data transfers and those without. The commands that do not use the sector buffer are: Restore, Seek and Scan ID. The functions of each command are explained in the Commands Section. Figure 21 is a flowchart of a polled operation and a PLM example.

The last status that was read will contain any error conditions that might have occurred during the command.

For commands that do make use of the sector buffer, the size of the sector buffer will affect the software. If the sector buffer is equal in size to one sector, then a carry out of an address counter (for the sector buffer) as the buffer is being filled will indicate to the 82062 that the command should continue.
Figure 20. Write Format Command Flow
size is equal to two or more disk sectors, and only one sector is being transferred, then the carry out signal would not go active, and the 82062 will be forever waiting for BRDY. In this case an I/O port would have to be used to generate this signal for the 82062 so that command execution can finish. Figure 22 is a flowchart of the READ SECTOR command, and its PLM representation. The WRITE SECTOR and FORMAT TRACK commands are equivalent in terms of software interfacing. Their flowcharts and their PLM equivalents are shown in Figure 23.

Once the command register is written the 82062 requests a data transfer before locating the proper track. Once the buffer is filled and BRDY is asserted, the 82062 will locate the target track and sector. If the ID is not located before the selected number of retries have occurred, the 82062 will terminate the command. The data transferred to the sector buffer will not have been used. Once the command has finished (i.e., $CIP = 0$), the status and error registers will inform the host of an error.

Figure 24 is the PLM routine that allows for all six of the commands. It differs from the READ and WRITE routines in that the direction that data is to be transferred is determined by the command.

Figure 24 also works for multiple sector transfers. However, the BRDY signal must be generated in hardware (the carry-out of an address counter).

**Interrupt Mode**

Interrupt driven software is chosen when the microprocessor must execute other tasks and cannot sit waiting for the disk to reposition its heads, as in a polled environment. The delay in repositioning heads can be anything from a couple of milliseconds to a second or more.

The 82062's interrupt (INTRQ) pin goes active to indicate that the command has finished. The READ SECTOR command provides the programmable choice of having the interrupt occur at the end of the data transfer or the normal end of the command. The reason for this option is that when the 82062 signals that a data transfer is required (via BDRQ, DRQ) the disk has been read and the data has been placed in the buffer. The host would remove the data and issue BRDY. The 82062 would then issue an interrupt indicating that the command has finished. The interrupt procedure would only have to read the status register. If the interrupt is issued at BDRQ the host would remove the buffer data.

---

**Disk$Operation: Procedure:**

Call Write$82062$Task$Reg's; /* Write Task Registers */
Output (Command$Reg) = Command;
Status = Input (Status$Reg); /* Read Status Reg */
Do while Status and CIP = CIP; /* Wait until command finishes */
    Status = Input (Status$Reg);
End;
End Disk$Operation;

---

---

**Figure 21. Polling Status**

5-873
Disk$Operation: Procedure;
    Call Write$82062$Task$Reg$ = Command;
    Output (Command $ Reg) = Command;
    Status = Input (Status$Reg);
    Do while Status and CIP = CIP;
        If Status and DRQ = DRQ then Do;
            Call Read$Data$From$Buffer;
            Output (BRDY$PORT) = 01;
        End;
        Status = Input (Status$Port)
    End;
End Disk$Operation;

Figure 22. Polling For Read Data
Disk$Operation: Procedure;
   Call Write$82062$Task$Regs;
   Output (Command$Reg) = Command;
   Status = Input (Status$Reg);
   Do while status and CIP = CIP;
      If status and DRQ = DRW then do;
         Call Write$Data$to$Buffer;
         Output (BRDY$Port) = 01; /* Make BRDY go active */
      End;
   Status = Input (Status$Reg)
   End;
End Disk$Operation;

Figure 23. Polling For Write Data
and generate BRDY. At this point the status and error registers contain valid information. Generating an interrupt at BDRQ time may save some systems some software overhead.

The WRITE SECTOR and FORMAT commands do not have this option because the sector buffer is filled before the track and sector are located. Hence, there can be significant delays between asking for data and the command terminating.

In an interrupt driven environment, the 82062 can interface to a DMA controller for data transfers between the sector buffer and the host’s RAM. If a DMA controller is not available an interrupt must be generated via the BDRQ line. However, BDRQ can stay active for long periods of time (until BRDY is generated). The interrupt sensing logic must take this into account to avoid being retriggered constantly. Intel’s 8259A Interrupt Controller 8259A provides that capability. It should be programmed for edge triggered interrupts or the end of interrupt byte must not be issued until BDRQ is removed to prevent retriggering.

Figure 25 is a PLM example of starting a disk operation in an interrupt driven environment. The command starts, and some indefinite amount of time later an interrupt would be generated, indicating service is required.

If a DMA controller is used, it would have to be programmed and initialized before the command is issued to the 82062. Recall that once a data transfer between the microprocessor and 82062 has finished, BRDY must be set high. As long as BRDY is generated from hardware, no microprocessor intervention is needed. If BRDY is generated by an I/O port the microprocessor will have to perform this function (this will be the case with any system that has a sector buffer larger than one sector). (One option could be to generate an interrupt from the terminal count pin of the DMA controller. The microprocessor would then issue a BRDY.) Data transfers between host RAM and the sector buffer would be handled without microprocessor intervention. The interrupt would then signal that the command has finished as shown in Figure 26. The only operation the host processor would perform is to check the status register of the 82062 for any error conditions.

If BDRQ is used to generate an interrupt in addition to the normal interrupt, then the routines shown in Figure 27 will check the status register to see if a data transfer should be executed or if the command is finished. If DRQ is not set, the command has finished and any error conditions would be in the status register.

Another possibility would be to have separate interrupt routines for the two possible sources of interrupts.
End of Transfer: Procedure Interrupt;
   Status = Input (Status Register);
   Output (8259A PIC) = End of Interrupt;
End

Figure 26. Checking Status via Interrupt

Service Disk Controller: Procedure Interrupt;
   Status = Input (Status Port);
   If Status and DRQ = DRQ then
      Call Transfer Data To/From Buffer;/* Enable DMAC */
   Output (8259A PIC) = End of Interrupt;
End Service Disk Controller;

Figure 27. Complete Interrupt Procedure

(INTRQ, BRDQ). There would then be no need to test the status to see which interrupt had occurred.

APPLICATION EXAMPLE

This section shows an application using the 82062 interfaced to the SBX bus. A quick overview of the SBX bus is provided (pin descriptions, general wave forms) as a background for the application. Designing the 82062 onto an SBX Multimodule board was chosen to highlight the size and complexity differences between earlier TTL, MSI, LSI-based disk controller boards and what is possible using the 82062. Both the hardware and software sections will be applicable to most other designs using the 82062. This design example is called SBX82062 and does not represent a real product offered by Intel Corporation. Appendix C contains the schematic of the SBX board.

The advantage of the SBX Multimodule is that it permits the system to be tailored for specific needs with a minimum of effort. The advantage of an SBX based disk controller is that a current system can make use of the capacity, reliability and speed of a hard disk with no (or minimal) hardware redesign.

iSBX Bus Multimodule Boards

The iSBX Multimodule boards are small, specialized, I/O mapped boards which plug onto base boards. The iSBX boards connect to the iSBX bus connector and convert the iSBX bus signals to a defined I/O interface.

Base Boards

The base board decodes I/O addresses and generates the chip selects for the iSBX Multimodule boards. In 8-bit systems, the base board decodes all but the lower three addresses in generating the iSBX Multimodule board chip selects. In 16-bit systems, the base board decodes all but the lower order four addresses in generating the iSBX Multimodule board chip selects. Thus, a base board would normally reserve two blocks of 8 I/O ports for each iSBX socket it provides.

There are two classes of base boards, those with Direct Memory Access (DMA) support and those without. Base boards with DMA support are boards with DMA controllers on them. These boards, in conjunction with an iSBX Multimodule board (with DMA capability), can perform direct I/O to memory or memory to I/O operations.

iSBX Bus Interface

The iSBX bus interface can be grouped into six functional classes:
1. Control Lines
2. Address and Chip Select Lines
3. Data Lines
4. Interrupt Lines
5. Option Lines
6. Power Lines
Control Lines
The following signals are classified as control lines:

COMMANDS:
- IORD (I/O Read)
- IOWRT (I/O Write)

DMA:
- MDRQT (DMA Request)
- MDACK (DMA Acknowledge)
- TDMA (Terminate DMA)

INITIALIZE:
- RESET

CLOCK:
- MCLK (iSBX Multimodule Clock)

SYSTEM CONTROL:
- MWAIT
- MPST (iSBX Multimodule Board Present)

Command Lines (IORD, IOWRT)
The command lines are active low signals which provide the communication link between the base board and the iSBX Multimodule board. An active command line, conditioned by chip select, indicates to the iSBX Multimodule board that the address lines are valid and the iSBX Multimodule board should perform the specified operation.

DMA Lines (MDRQT, MDACK, TDMA)
The DMA lines are the communication link between the DMA controller device on the base board and the iSBX Multimodule board. MDRQT is an active high output signal from the iSBX Multimodule board to the base board's DMA device requesting a DMA cycle. MDACK is an active low input signal to the iSBX Multimodule board from the base board DMA device acknowledging that the requested DMA cycle has been granted. TDMA is an active high output signal from the iSBX Multimodule board to the base board. TDMA is used by the iSBX Multimodule board to terminate DMA activity. The use of the DMA lines is optional as not all base boards will provide DMA channels and not all iSBX Multimodule boards will be capable of supporting a DMA channel.

Initialize Lines (Reset)
This input line to the iSBX Multimodule board is generated by the base board to put the iSBX Multimodule board into a known internal state.

Clock Lines (MCLK)
This input to the iSBX Multimodule board is a timing signal. The 10 MHz (+0%, 10%) frequency can vary from base board to base board. This clock is asynchronous from all other iSBX bus signals.

System Control Lines (MWAIT, MPST)
These output signals from the iSBX Multimodule board control the state of the system.

An active MWAIT (Active Low) will put the CPU on the board into wait states providing additional time for the iSBX Multimodule board to perform the requested operation. MWAIT must be generated from address 5-878.
(address plus chip select) information only. If \( \text{MWAIT} \) is driven active due to a glitch on the \( \text{CS} \) line during address transitions, \( \text{MWAIT} \) must be driven inactive in less than 75 ns.

The iSBX Multimodule board present (MPST) is an active low signal (tied to signal ground) that informs the base board I/O decode logic that an iSBX Multimodule board has been installed.

**Address and Chip Select Lines**

The address and chip select lines are made up of two groups of signals.

- **Address Lines:** MA0–MA2
- **Chip Select Lines:** MCS0–MCS1

The base board decodes I/O addresses and generates the chip selects for the iSBX Multimodule boards. The base board decodes all but the lower order three addresses in generating the iSBX Multimodule board chip selects.

**Address Lines (MA0–MA2)**

These positive true input lines to the iSBX Multimodule boards are generally the least three significant bits of the I/O address. In conjunction with the command and chip select lines, they establish the I/O port address being accessed. In 16-bit systems, MA0–MA2 may be connected to ADR1–ADR3 of the base board address lines.

**Chip Select Lines (MCS0–MCS1/)**

In an 8-bit system, these input lines to the iSBX Multimodule board are the result of the base board I/O decode logic. MCS is an active low signal which conditions the I/O command signals and thus enables communication with the iSBX Multimodule boards.

**The SBX82062 Design Example**

The SBX82062 Multimodule board will interface an ST506 compatible drive to any host board having an SBX connector. Two restrictions on the disk drive are that there is a maximum of 1024 cylinders and/or 8 heads. The SBX connector cannot supply the power-up current requirements of the drive. The drive must be connected directly to the power supply. The SBX82062 in Appendix C does not support DMA transfers. The version in Appendix D does support DMA transfers. Since this multimodule has a 2 kbyte sector buffer, the host microprocessor must generate a BRDY by accessing an I/O port during data transfers.

The software for communicating to the SBX board is intended to be interrupt driven. Polling for data transfers is not supported. Reading the status without an interrupt is not recommended. During the times the 82062 is accessing the sector buffer, the SBX82062 will isolate itself from the host. To support polling, a hardware generated busy pattern should be driven onto the host's data bus as is shown in the Polled Interface section. The sector buffer stores up to 2 kbytes of disk data, for multiple sector transfers. The SBX board only interfaces to one drive (for space reasons), but four drives could be used with the addition of a read data multiplexor (one IC) and the drive data cables.

**Microprocessor Interface**

Figure 29 is a block diagram of the SBX82062's microprocessor interface. The I/O port assignments are listed in Table 1. The functional blocks of the interface are:

- Sector Buffer Isolation Logic
- Wait State Logic
- Sector Buffer
- Sector/Drive/Head Register Logic

<table>
<thead>
<tr>
<th>Port Address</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>80H</td>
<td>Sector Buffer</td>
<td>Sector Buffer</td>
</tr>
<tr>
<td>82H</td>
<td>Error Reg</td>
<td>RWC Reg</td>
</tr>
<tr>
<td>84H</td>
<td>Sector Count</td>
<td>Sector Count</td>
</tr>
<tr>
<td>86H</td>
<td>Sector Number</td>
<td>Sector Number</td>
</tr>
<tr>
<td>88H</td>
<td>Cylinder Low</td>
<td>Cylinder Low</td>
</tr>
<tr>
<td>8AH</td>
<td>Cylinder High</td>
<td>Cylinder High</td>
</tr>
<tr>
<td>8CH</td>
<td>SDH Reg</td>
<td>SDH Reg</td>
</tr>
<tr>
<td>8EH</td>
<td>Status Reg</td>
<td>Command Reg</td>
</tr>
<tr>
<td>90H</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>92H</td>
<td>None</td>
<td>Asserts BCR</td>
</tr>
<tr>
<td>94H</td>
<td>None</td>
<td>Asserts BRDY</td>
</tr>
</tbody>
</table>

**NOTE:**

Address assignments are determined by the host board.

**Sector Buffer Isolation Logic**

The host will be isolated from the SBX board whenever the 82062 is accessing its sector buffer which is enabled by BCS. The host's control signals, RD, WR, MCS0, and MCS1 and data bus are also disabled at the same time to prevent any data in the sector buffer from being corrupted. The host should wait for an interrupt before reading the 82062's Status register. Attempting to read the SBX board while BCS is active will return invalid data, since the SBX board will have the data bus tristated.
Figure 29
**Wait State Logic**

The wait state logic drives the 'not ready' line, MWAIT, active whenever the host reads the SBX board. MWAIT does not go active for buffer or 82062 register writes. This logic was required for two reasons. First, a delayed read is generated, because the address setup to RD margin of the SBX bus is less than the 82062's needs (50 ns vs 100 ns). Second, the RD to data valid access period of the 82062 (375 ns), is greater than the SBX bus' full speed read cycle (275 ns) permits. MWAIT is deactivated after allowing for the delayed RD and the access period of the 82062. This delay is accomplished with a 500 ns delay line. The first tap at 100 ns generates the read request to allow for the address setup margin. The next tap 400 ns later removes MWAIT to allow the host to continue.

**Sector Buffer**

The sector buffer consists of an address counter (using '1s393's) and a 2 kbyte static RAM. The address counter is incremented on the trailing edge of a valid RD or WR cycle, either host microprocessor or 82062 initiated. The counter is reset by a hardware reset, the 82062 buffer reset BCR, or by accessing an I/O port to provide software control. The 82062 will issue BCR each time BCS changes state (i.e. twice per sector). Resetting the buffer counter can be put under software control for multiple sector transfers. BRDY going high tells the 82062 that the buffer is available for its use. BRDY is generated by the address counter, by filling or emptying the entire buffer in multiple sector transfers, or from an I/O port when single sector transfers are done (since single sectors won't use all 2 kbytes of the buffer, the hardware signal will not be generated). When the 82062 is using the buffer, BCS will be low, and the RD or WR line will be pulsed every 1.6 microseconds.

When the 82062 is using the buffer it prevents access by the host by tri-stating the read, write, select and data lines with a low on BCS.

**SDH Register Logic**

The drive and head select bits must be latched externally to the 82062, since these outputs are not provided. An 8 bit latch is strobed on the trailing edge of the WR pulse when the SDH register is selected. The two drive select bits are then demultiplexed to provide a one of four drive select line. If multiple drives are used then these outputs would also be used to select which disk's read data line would be gated into the PLL.

**Interrupts**

While the interrupt line is programmable (to notify of an end of command or data transfer request for the Read Sector command only), software will ensure that the interrupt from the 82062 signifies command termination. The BDRQ line is OR'ed with the 82062's INTRQ line or BDRQ can generate its own interrupt. BDRQ is also gated off-board for a DMA controller.

**Disk Interface**

Figure 30 is a block diagram of the interface between the 82062 and the disk drive. The functional blocks are:
- Write Data Logic
- Read Data Logic (PLL)
- Drive Control

**Write Data Logic**

The WR DATA output requires a D flip-flop clocked at 10 MHz to complete the conversion of data to MFM. The output of this D flip-flop is true MFM and is sent for a delay line. A delay line determines the amount of delay for precompensation. No delay corresponds to shifting the data bit early; the first tap is approximately 12 ns of delay and is the "normal", or no delay and the second tap provides 12 ns of delay, referenced to the "normal" write data. Which output is selected is determined by the states on RWC, Early and Late. This function was generated with a 74s151 multiplexer. When RWC is inactive EARLY and LATE only select "normal" data since they are always active. The pre-compensated write data is then driven onto the data cable by an RS-422 driver.

**Read Data Logic**

The PLL generates the RD CLOCK that is used to decode the serial MFM data from the drive. A selected drive issues read data, unless WR GATE is active. A one-shot generates a pulse of 220–270 ns to provide the DRUN input. Only during an all zero's or one's field will the DRUN input stay high, as it will be retriggered every 200 ns (the minimum distance that separates continuous clock and data bits). As soon as DRUN is determined to be valid, the RD GATE output will go active, switching the PLL from the 10 MHz local clock input to disk data. The PLL will synchronize to the incoming serial data and generate a Read Clock of the proper timing and phase. The 82062 will then start to search for the address mark which is indicated by DRUN going low at the address mark.
Software Driver Overview

Presented in Appendix B is a listing of the software used to exercise the SBX 82062 board. Communication between the host software and the SBX driver routine is done through a structure located in system RAM. The host routine fills in required parameters, then passes the address of this communication block to the SBX driver routine. The driver routine pulls necessary values from this command block (CBL), executes a disk operation, then fills the CBL with the 82062’s register contents, plus status and error information. The command block structure is shown in Figure 31.

The host board did not have a DMA controller available, so an interrupt is issued from the BDRQ line and OR’ed with the 82062’s interrupt line as interrupt sources were limited by the host. When an interrupt occurs, the interrupt procedure checks for either a data transfer, and executes it, or the completion of the command. If the interrupt signifies command completion, the interrupt procedure fills the command block with the 82062’s task, status and error registers.
In this example, the host software examines one byte in
the command block and until this byte is changed to a
00, no other command blocks will be passed to the disk
driver routine. An alternative would be to issue a soft­
ware interrupt to notify the microprocessor that the
disk operation has finished and the command block
contains parameters from the last operation and that a
new disk command could start.

The driver for this example allows polling for non-data
transfer commands, and must use interrupts for data
transfers. As mentioned earlier, microprocessor inter­
vention is required since the sector buffer is much larg­
er than one sector and will not generate a BRDY. The
microprocessor must write to an I/O port, which sets
BRDY, after each host to sector buffer transfer. An
actual software implementation would not include the
polling and interrupt routines together, as only one
method would generally be used.

The calling routine, which would normally be a direc­
tory program, places the values for which sector, num­
ber of sectors, etc., in the CBL. The disk routine is
called and the address of this structure is passed on the
stack. The disk driver places these parameters in the
82062's Task registers and initiates a command.

If the interrupt driven method was chosen, the disk
driver routine returns to the calling routine. This per­
mits other processing to be performed while the disk is
executing a command. At some point, an interrupt will
be generated, either from BRDY or INTRQ. Control
will pass to the driver and the status register will be
checked. If a data transfer is needed, either the micro­
processor can transfer data or a DMA controller can
perform the function. Once the transfer of data to the
buffer is finished, the microprocessor must set BRDY
through an I/O port.
APPENDIX A
ST506 INTERFACE

THE ST506 INTERFACE

The ST506 interface is a modified version of Shugarts floppy disk drive interface and has been promoted by Seagate Technology. This interface is intended to be easy and low in cost to implement, yet provide a medium level of performance. The interface rigidly defines several areas: the hardware interconnects, the data transfer rate, the data encoding method, and how the disk is formatted.

Data Transfer Rate

The data transfer rate depends upon the linear bit density of the disk media and the speed at which the disk spins. ST506 specifies a 5 Mbit/second transfer rate. The typical ST506 drive has a nominal linear density of 10,416 bytes and a disk speed of 3600 rpm, which yields a 5 Mbit/second data transfer rate. No deviation from 5 M/bits second is allowed.

Increasing the linear density to increase storage capacity would require a decrease in disk speed. Otherwise, the data rate would increase. This decrease in disk speed would cause access times to increase, which many would deem unacceptable. To increase storage capacity, and remain ST506 compatible, either the number of cylinders and/or the number of platters can increase.

Data Encoding

ST506 requires that the serial data, sent between the drive and the controller, be encoded according to MFM rules. The basic unit of storage is a bit cell, which stores one bit infromation. This bit cell is divided into two halves, consisting of a clock bit and a data bit (see Figure A-1).

The encoding rules for MFM are fairly simple:

1. A clock bit is written when the previous and the current bit cell does not contain a data bit.

2. A data bit is written whenever there is a “one” from the user.

Sync fields are composed of zeroes which generates a series of clock bits in the bit cell’s. A phase lock loop locks on to the data stream during this period and generates a signal of the proper phase and frequency which is used to decode the combined clock and data serial data stream.

Disk Format

All disk media must be written with a specified format so that data may be reliably stored and retrieved. The smallest unit of controller accessible data is the sector which typically contains sync fields, ID fields, and a data field, and buffer fields.

The format of the disk required by ST506 is shown in Figure A-2. It should be noted that this format is fixed in the 82062. The user has options only for GAP1 and 3 length (when changing sector size or ECC) and whether to have 82062 CRC checking or user supplied ECC syndrome bits.

Gap 1 – Index Gap

Gap 1 serves two purposes. The first is to allow for variations in the index pulse timing due to motor speed variations. The second purpose is to allow a small delay to permit a different head to be selected without missing a sector. This is more of a data transfer optimization function and requires the disk controller to know which head is to be selected, when the last sector of a track has been read, and the next logical sector in the file exists on another platter. The 82062 does not switch heads automatically. Whether this scheme can be used or not depends upon the μP being able to alter one register in the 82062, before the next sector passes beneath the heads.

This gap is typically 12 bytes long and is written by the 82062 as 4E Hex.

Gap 2 – Write Splice Gap

This gap follows the CRC bytes of the ID field and continues up to the data field address mark. When up-
dating a previously written sector, motor speed variations could turn on the write coil, as the head was passing over the ID field. This gap prevents this from occurring. The value written is OOH and also serves as the PLL sync field for the data field. The minimum value is determined by the "lock up" performance of the PLL. The 82062 writes sixteen bytes for this field once WG is activated. The user has no control over this field.

**Gap 3 - Post Data Field Gap**

Gap 3 is very similar to Gap 2 as it is used as a speed tolerance buffer also. Without this gap, and with the motor speed varying slightly, it would be possible for the upcoming sector's sync field and ID field to be overwritten. This value is '4E' H and is typically 15 bytes long. The 82062's Gap 3 length is programmable. The exact value is dependent upon several factors. Refer to 82062 Format command, Software Section: General Programming Section.

**Gap 4 - Track Buffer Gap**

This gap follows the last sector on a track and is written until an index pulse is received. Its purpose is to prevent the last sector from overflowing past the index gap, and absorb track length variations when ECC is used (ECC uses more bytes than CRC). The value is '4E' H and is about 320 bytes when CRC and 256 byte sectors are used. The 82062 writes this field only during formatting. The user has no control over the number of bytes written with the 82062.

**ID Fields**

The controller uses ID fields to locate any individual sector. An address mark of two bytes precedes the ID field and the data field in a sector. An address mark tells the controller the nature of the upcoming information. ID fields are used by the disk controller and are not passed to the host.

**Sector Interleaving**

Sector interleaving occurs when logical sectors are in a non-sequential order, which is determined during formatting. An advantage is that there is a delay between logically sequential sectors. This delay can be used for data processing and then deciding if the next sector should be read. Without interleaving, the next sector could slip by, imposing a one revolution delay (approx. 16.7 ms). An additional benefit to this delay is that bus utilization is reduced by spreading the data transfer over a greater amount of time. The delay between sectors can be determined as follows:

\[ \text{1 Revolution Period} \times (\text{Interleave factor} - 1) = \text{Delay/Sectors/Track} \]

For the typical ST506 drive with four-way interleaving this yields 1.57 ms of delay.

![Figure A-2. Format Field](image)
The disadvantage to interleaving is that file transfers take longer, which may slow down the overall system. A four-way interleaved disk will have the transfer rate reduced to an average of 1.25 Mbit/sec.

The 82062 leaves the logical sector sequence to the user.

**ELECTRICAL INTERFACE**

The interface to the ST506 drive is divided into three categories and they are:

1. control signals,
2. data signals,
3. power.

**Control Signals**

The functions of the control signals are not covered in detail here. Their purpose can be found in the pin descriptions section. All control lines are digital in nature and either provide signals to the drive or inform the host of certain conditions. A diagram of the 34 pin control connector is shown in Figure A-3.

The driver/receiver logic diagram is shown in Figure A-4 and the electrical characteristics are:

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>0.0 VDC to 0.4 VDC</td>
</tr>
<tr>
<td>False</td>
<td>2.5 VDC to 5.25 VDC</td>
</tr>
</tbody>
</table>

![Figure A-4](image-url)
Data Signals

The lines associated with the transfer of read/write data between the host and the drive are differential in nature and may not be multiplexed between drives. There is one pair of balanced lines for each read and write data line per drive and must conform to the RS-422 specification. Figure A-5 shows the receiver/transmitter combination.

![Diagram of E1A RS22 Driver/Receiver Pair Flat Ribbon or Twisted Pair](231133-29)

Figure A-5. E1A RS22 Driver/Receiver Pair Flat Ribbon or Twisted Pair
APPENDIX B
SOFTWARE DRIVER

SERIES-III PLM-86 V2 3 COMPIILATION OF MODULE DISK IO_MODULE
OBJECT MODULE PLACED IN F2 DISKIO OBJ
COMPILER INVOKED BY PLMB6 B6 F2 DISKIO PB6

$TITLE('B2062/SBX DISK CONTROLLER')

1
DISK IO_MODULE:DO,

/* CBL_PTR IS A POINTER TO A COMMAND BLOCK- HENCE CBL
   THIS COMMAND BLOCK RESIDES IN RAM AND CONTAINS ALL
   VALUES REQUIRED BY THIS PROGRAM TO OPERATE THE B2062
   DISK CONTROLLER ONCE THIS PROCEDURE IS CALLED. THE
   CBL IS REMAIN UNTOUCHED UNTIL THE COMMAND BYTE IS
   SET TO A 00 VALUE THIS ROUTINE WILL CALL THE CALLING
   PROGRAM WHEN A COMMAND IS COMPLETED

REV  DATE  NAME  DESCRIPTION
1  0  1/JUL/84  J SLEEZER  INITIAL
*/

/** PROGRAM CONSTANTS */
2 1
DECLARE LIT LITERALLY 'LITERALLY',
TRUE  LIT 'OFFH',
FALSE  LIT 'OOH',
FOREVER LIT 'WHILE TRUE',

/* BOARD ADDRESSING FOR THE 66/05 */
3 1
DECLARE BASE_ADDR LIT 'BOH',
SCTR_BFFR  LIT 'BASE_ADDR',
ERR_REG  LIT 'BASE_ADDR + 02H', /* READ ONLY */
SEC_CNT_REQ  LIT 'BASE_ADDR + 04H',
SEC_NUM_REQ  LIT 'BASE_ADDR + 06H',
CYL_LOW_REG  LIT 'BASE_ADDR + 08H',
CYL_HI_REG  LIT 'BASE_ADDR + 0AH',
S_DR_HD_REG  LIT 'BASE_ADDR + 0CH',
STATUS_REQ  LIT 'BASE_ADDR + 0EH', /* READ ONLY */
COMMAND_REQ  LIT 'BASE_ADDR + 0FH', /* WRITE ONLY */
WR_PCMP_REQ  LIT 'BASE_ADDR + 00H', /* WRITE ONLY */
BFRR_RESET  LIT '92H',
BFRR_RDY  LIT '94H',
SEC_BUF  LIT '2048',

/******** B2062 COMMANDS ********/
4 1
DECLARE RESTORE LIT '1FH',
SEEK  LIT '7FH',
FORMAT  LIT '50H',
SCAN_ID  LIT '40H',
READ_SEC  LIT '20H',
WRITE_SEC  LIT '30H', /* TO BE OR'D WITH VALUE IN SDH REG */
ECC_EN  LIT '80H',
NO_INTERPT  LIT '0OH',
INTR_ON_CMD  LIT '08H',
MULT_SCTR  LIT '04H',

231133-36
INPUT (STATUS_REG), COUNT = COUNT - 1;
21 3 END;
22 2 END POLL.

/***************************~******************************************
胥FER_DATA PROCEDURE,
23 1 DECLARE CNT BYTE, INDEX WORD, SI BYTE, SECTR_SZ WORD.

 APPENDIX A

// STATUS REGISTER BITS */

5 1 DECLARE ERR LIT '01H', CIP LIT '02H', DRG LIT '08H', SC LIT '10H', WRF LIT '20H', DDRY LIT '40H', BUFSY LIT '80H', /* USER WILL NEVER SEE THIS BIT SET */

/* PROGRAM VARIABLES */

6 1 DECLARE CMD_BLOCK_PTR POINTER, CBL BASED CMD_BLOCK_PTR STRUCTURE ( COMMAND BYTE, PRECMP BYTE, S_CNT BYTE, SCTR BYTE, LOW_CYL BYTE, HI_CYL BYTE, SDH BYTE, STATUS BYTE, ERRS BYTE, INTERRUPT BYTE, RET_PROC POINTER, BUFF_PTR POINTER),

7 1 DECLARE BUFFER_PTR POINTER, BUFF BASED BUFFER_PTR (1) BYTE, STATUS BYTE, ERRS BYTE, COMMAND BYTE;

$EJECT

/***************************~******************************************/
SZI = (SHR(CBL SDH, 3) AND 03H),  /* OBTAIN SECTOR SIZE BITS FROM SDH */
IF SZI = 00 THEN SECTR_SZ = 256,  /* REGISTER */
ELSE IF SZI = 01 THEN SECTR_SZ = 512,
ELSE IF SZI = 02 THEN SECTR_SZ = 1024,
ELSE IF SZI = 03 THEN SECTR_SZ = 128.

IF CBL SDH AND ECC_EN = ECC_EN THEN
SECTR_SZ = SECTR_SZ + 7,
IF (((CBL COMMAND AND OFOH = READ_SEC) OR (CBL COMMAND AND OFOH = WRITE_SEC))
AND (CBL COMMAND AND OFH = MULT_SCTR)) THEN DO,  /* VARIOUS SECTOR SIZES */
CNT = (SEC_BUF/CBL S_CNT),  /* ARE POSSIBLE THIS FIGURES */
DO WHILE (CNT * SECTR_SZ) > SEC_BUF,  /* HOW MANY SECTORS WILL FIT */
CNT = CNT - 1,  /* INTO THE BOARD'S SECTOR BFFR */
END.
SECTR_SZ = SECTR_SZ * CNT,
END.

OUTPUT(BFFR_RESET) = 00.  /* ACTIVATES 062'S BFRD LINE */
END XFER_DATA.

UPDATE COMMAND BLOCK
UPDATE_CBL
CBL S_CNT = INPUT(SEC_CNT_REG);
CBL SCTR = INPUT(SEC_NUM_REG);
CBL LOW_CYL = INPUT(CYL_LOW_REG);
CBL HI_CYL = INPUT(CYL_HI_REG);
CBL SDH = INPUT(S_DR_HD_REG);
CBL STATUS = STATUS,
CBL ERRS = INPUT(ERR_REG);
END UPDATE_CBL.

WRITE THE CBL TO 82062
WR_CBL
OUTPUT(WR_CMP_REG) = CBL PRECMP;
OUTPUT(SEC_CNT_REG) = CBL S_CNT;
OUTPUT(SEC_NUM_REG) = CBL SCTR;
OUTPUT(CYL_LOW_REG) = CBL LOW_CYL;
OUTPUT(CYL_HI_REG) = CBL HI_CYL;
OUTPUT(S_DR_HD_REG) = CBL SDH;
END WR_CBL.

*EJECT
DISK PROCEDURE(CBL_PTR) PUBLIC.
DECLARE CBL_PTR POINTER.
CMD_BLOCK_PTR = CBL_PTR. /* ADDRESS OF STRUCTURE */
BUFFER_PTR = CBL BUFF_PTR. /* THAT CONTAINS B2062 */
CALL WR_CBL:
IF CBL COMMAND = 99H THEN DO. /* A DUMMY COMMAND TO READ*/
CBL COMMAND = 00.
CBL STATUS = INPUT STATUS_REG).
RETURN.
END.
IF (INPUT STATUS_REG AND DRDY) <> DRDY THEN DO. /* NO COMMAND IS ISSUED */
CBL COMMAND = 00.
CBL STATUS = INPUT STATUS_REG).
RETURN.
END.
OUTPUT(BFFR_RESET) = OOH.
IF (CBL COMMAND AND OF0H) = READ_SEC THEN /* FOR PROGRAM CONSISTENCY */
CBL COMMAND = CBL COMMAND OR INTR_ON_CMD. /* SET INTERRUPT FOR COMMAND*/
/* TERMINATION */
OUTPUT COMMAND_REG) = CBL COMMAND; /* A DELAY IS NEEDED BECAUSE FAST*/
CALL TIME(100); /* UP'S CAN READ THE STATUS REG */
IF CBL INTERRUPT = NO INTERRUPT THEN DO. /* BEFORE A VALID STATUS IS READY*/
CALL POLL.
CALL UPDATE_CBL.
CBL COMMAND = 00.
RETURN.
END.
END DISK.
**EJECT**

DISK_SERVICE PROCEDURE PUBLIC.
CALL TIME(500)
STATUS = INPUT STATUS_REG).
IF (STATUS AND CIP) = 00 THEN DO:
CALL UPDATE_CBL.
CBL COMMAND = 00.
OUTPUT(BFFR_RESET) = OOH.
RETURN.
END.
ELSE CALL XFER_DATA.
END DISK_SERVICE.

MODULE INFORMATION
CODE AREA SIZE = 02EEH 750D
CONSTANT AREA SIZE = 00OH 0D
VARIABLE AREA SIZE = 0011H 17D
MAXIMUM AREA SIZE = 000AH 10D
272 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS

5-892
231133-002
DICTIONARY SUMMARY

31KB MEMORY AVAILABLE
5KB MEMORY USED (16%)  
OKB DISK SPACE USED

END OF PL/M-86 COMPILATION

SERIES-III PL/M-86 V2.3 COMPILATION OF MODULE HOST_MODULE
OBJECT MODULE PLACED IN 'F2:DSKHST OBJ'
COMPILER INVOKED BY  PLMB8 8b o F2:DSKHST P8b

$TITLE('DEMO PROGRAM FOR SBX062')

HOST_MODULE DO:

/* PROGRAM TO EXERCISE THE B2062/SBX BOARD USING THE 957 MONITOR
ON AN SBC 86/05 THIS PROGRAM DEMONSTRATES HOW THE DISKIO MODULE
IS USED. THE CASE STATEMENTS IN THE MAIN SECTION SHOW THE VARIOUS
ROUTINES THE TYPICAL ROUTINES LIKE HEX TO ASCII, ETC., WERE
NOT INCLUDED IN THIS LISTING SEVERAL OF THE ROUTINES USE
STATEMENTS THAT COULD BE REDUCED CONSIDERABLY BUT WERE LEFT
SIMPLIFIED SO THAT ALL WOULD UNDERSTAND

REV DATE NAME DESCRIPTION
1 0 20/JUL/84 J SLEEZER INITIAL

*/

/* EXTERNAL ROUTINES */

2 1 CO PROCEDURE(CHAR) EXTERNAL,
3 2 DECLARE CHAR BYTE,
4 2 END CO.

5 1 CI PROCEDURE BYTE EXTERNAL,
6 2 END CI.

7 1 DISK:PROCEDURE(CMD_BLK_PTR) EXTERNAL,/ THIS ROUTINE STARTS A DISK OPERATION */
8 2 DECLARE CMD_BLK_PTR POINTER,
9 2 END DISK.

10 1 DISK_SERVICE PROCEDURE EXTERNAL,/ THIS ROUTINE SERVICES THE B2062 INTERRUPTS*/
11 2 END DISK_SERVICE.

/* PROGRAM CONSTANTS */

12 1 DECLARE LIT LITERALLY 'LITERALLY',
    TRUE LIT 'OFFH',
    FALSE LIT 'OOFH',
    FOREVER LIT 'WHILE TRUE',
    SPACE LIT '20H',
    CR LIT '0DH',
    LF LIT 'OAH',
    RUB LIT '7FH',
    BACKSP LIT '08H',
    ESC LIT '1BH',

/* B2062 COMMANDS */
AP-182

13 1 DECLARE RESTORE LIT '10H',
    SEEK LIT '70H',
    FORMAT LIT '50H',
    SCAN_ID LIT '40H',
    READ_SECT LIT '28H', /* INTR ONLY ON COMMAND TERMINATION */
    WRITE_SCT LIT '30H',
    MULT_SCTR LIT '04H', /* TO BE OR'D WITH COMMAND */
    NO_RETRIES LIT '01H', /* TO BE OR'D WITH COMMAND */
    NO_CRC LIT '80H', /* TO BE OR'D WITH VALUE IN SDH REG */
    P_COMP LIT '0', /* INDEXING INTO DISK_REG ARRAY */
    SEC_CNT LIT '1',
    SECTOR LIT '2',
    CYL_LB LIT '3',
    CYL_MB LIT '4',
    SDH LIT '5'.

    /* STATUS REGISTER BITS */

14 1 DECLARE ERR LIT '01H',
    CIP LIT '02H',
    DRQ LIT '80H',
    SC LIT '10H',
    WRF LIT '20H',
    DRDY LIT '40H',
    BUFBSY LIT '80H', /* USER WILL NEVER SEE THIS BIT SET */

    /* ERROR REGISTER BITS */

15 1 DECLARE VALID_BITS LIT '0D7H',
    AM_NT_FND LIT '001H',
    TKOO_ERR LIT '002H',
    ABRTD_CMD LIT '004H',
    ID_NT_FND LIT '010H',
    DATA_ERR LIT '040H',
    BAD_BLK LIT '080H'.

    /***** PROGRAM VARIABLES *****/

16 1 DECLARE CMD_BLK(1) STRUCTURE { COMMAND BYTE,
    PRECMP BYTE,
    S_CNT BYTE,
    SCTR BYTE,
    LOWB_CYL BYTE,
    MSH_CYL BYTE,
    SDHD BYTE,
    STATUS BYTE,
    ERRS BYTE,
    INTERRUPT BYTE,
    RET_PROC POINTER,
    BUFF_PTR POINTER),

17 1 DECLARE COUNT WORD,
    CHAR BYTE,
    ERRORS BYTE,
    COMMAND BYTE,
    STEP_RATE BYTE,
    I WORD,
    I2 BYTE,
    BUFFER(100) BYTE,
    INDEX WORD,
    DISK_IS_NOT_BUSY BYTE,
    TRACKS BYTE,
    PLATTERS BYTE,
    PLAT_CNT BYTE,
    TRACK_CNT BYTE,
    I_FACTOR BYTE,
    FRMT_BFFR_SIZE BYTE,
    LOG_SECT_NUM BYTE,
    MAKING_TABLE BYTE,
    AA BYTE,
    INDEX BYTE

18 1 DECLARE DISK_REGS(6) BYTE.

*EJECT

5-894 231133-41

231133-002
337 1 MAIN PROGRAM

338 2 STEP_RATE = OFFH,
339 2 PLAT_CNT = OFFH,
340 2 TRACK_CNT = OFFH,
341 2 PLATTERS = 00,
342 2 TRACKS = 00,
343 2 DO I = 0 TO 5,
344 3 DISK_REGS(I) = 00,
345 3 END;
346 2 DISK_REGS(P_COMP) = OFFH,
347 2 CALL UPDATE_CMD_BLK,
348 2 CMD_BLK(INDX) INTERRUPT = 00,
349 2 CALL WRITEA(@LF,LF,LF,LF,LF,00»,
350 2 CALL WRITEA(®SIGN_ON),
351 2 OUTPUT(OCC2) = OFDH; /* PERMITS AN INTERRUPT */
352 2 INDEX = 0,
353 2 CMD_BLK(INDX) BUFF_PTR = ®BUFFER,
354 2 CALL SET_INTERRUPT(21H, CHECK_DISK),
355 2 ENABLE,
356 2 DISK_IS_NOT_BUSY = TRUE.
357 2 DO FOREVER.
358 3 IF DISK_IS_NOT_BUSY THEN DO,
359 4 CMD_BLK(INDX) COMMAND = OFFH,
360 4 CALL WRITEA(®CR,LF,COMMAND >',00»),
361 4 CHAR = CI,
362 4 CALL CO(CHAR);
363 4 CALL CO(LF),
364 4 CALL CO(00),
365 4 CMD_BLK(INDX) COMMAND = FALSE.
366 4 I2 = 0,
367 4 DO WHILE (COMMAND = FALSE).
368 5 IF I2 > LENGTH(VALID_CMDS) THEN DO,
369 6 CALL WRITEA(®'INVALID COMMAND',CR,LF,00»),
370 6 I2 = 0.
371 6 CHAR = CI,
372 6 END;
373 5 IF CHAR = VALID_CMDS(I2) THEN
374 5 COMMAND = TRUE,
375 5 I2 = I2 + 1,
376 5 END;
377 4 DO CASE (I2 - 1),
378 4 /* CASE 0 - READ SECTOR */
379 6 CALL WRITEA(®'READ SECTOR COMMAND',CR,LF,LF,00»),
380 6 CALL WRITE_REGS,
381 6 DISK_IS_NOT_BUSY = FALSE,
382 6 CMD_BLK(INDX) COMMAND = READ_SECT,
383 6 CALL WRITEA(®'MULTIPLE SECTOR'S >',00»),
384 6 CHAR = CI,
385 6 IF CHAR = 'Y' THEN DO,
386 7 CALL WRITEA(®'YES',CR,LF,00»),
387 7 CMD_BLK(INDX) COMMAND = CMD_BLK(INDX).COMMAND OR MULT_SCTR,
388 7 CALL WRITEA(®'DO NOT EXCEED BUFFER LIMIT ',CR,LF,00»),
389 7 END;
390 6 ELSE CALL WRITEA(®'NO',CR,LF,00»),
391 6 CALL WRITEA(®'AUTOMATIC RETRIES >',00»),
392 6 CHAR = CI,
393 6 IF CHAR = 'N' THEN DO,
394 7 CALL WRITEA(®'NO',CR,LF,00»),
395 7 CMD_BLK(INDX) COMMAND = CMD_BLK(INDX).COMMAND OR NO_RETRYS,
396 7 END;
397 6 ELSE CALL WRITEA(®'YES',CR,LF,00»),
398 6 CALL DISK(®CMD_BLK(INDX))
399 6 END.
/* CASE 1 - WRITE SECTOR */

DO.

CALL WRITEA('WRITE SECTOR COMMAND',CR,LF,LF,00));

CALL WRITE_REGS;

DISK_IS_NOT_BUSY = FALSE;

CMD_BLK(INDX) COMMAND = WRITE_SCT;

CALL WRITEA('MULTIPLE SECTOR'S? ',00));

CHAR = 'C;

IF CHAR = 'Y' THEN DO,

CALL WRITEA('YES ',00));

CMD_BLK(INDX) COMMAND =

CMD_BLK(INDX) COMMAND OR MULT_SCTR,

CALL WRITEA('DO NOT EXCEED BUFFER LIMIT ',CR,LF,00));

END.

ELSE CALL WRITEA('NO',CR,LF,00));

CALL WRITEA('ENABLE RETRIES? ',00));

CHAR = 'C;

IF CHAR = 'N' THEN DO,

CALL WRITEA('NO',CR,LF,00));

CMD_BLK(INDX) COMMAND =

CMD_BLK(INDX) COMMAND OR NO_RETRYS;

END.

ELSE CALL WRITEA('YES',CR,LF,00));

CALL DISK(CMD_BLK(INDX));

END.

/* CASE 2 - FORMAT TRACK */

DO.

CALL WRITEA('FORMAT TRACK',CR,LF,LF,00));

CALL WRITE_REGS;

DISK_IS_NOT_BUSY = FALSE;

CMD_BLK(INDX) COMMAND = FORMAT;

CALL WRITEA('INTERLEAVE FACTOR? (1 TO ?)',00));

I_FACTOR = 'C - '0';

CALL CO(I_FACTOR + '0'),

CALL CO(CR),

CALL CO(LF),

FRMT_BFFR_SIZE = (2 * (CMD_BLK(INDX) S_CNT) + 1);

DO I = 0 TO FRMT_BFFR_SIZE,

BUFFER(I) = 00,

END,

LOG_SECT_NUM = 0;

I = 1;

MAKING_TABLE = TRUE,

DO WHILE MAKING_TABLE,

DO WHILE I < FRMT_BFFR_SIZE,

BUFFER(I) = LOG_SECT_NUM;

LOG_SECT_NUM = LOG_SECT_NUM + 1;

I = I + I_FACTOR * 2),

END.

IF LOG_SECT_NUM < CMD_BLK(INDX) S_CNT THEN DO,

I = I - (FRMT_BFFR_SIZE + 1);

IF (I = 1) OR (BUFFER(I) <> 00) THEN

I = I + 2,

END.

ELSE MAKING_TABLE = FALSE.

END.

CALL WRITEA('256 TRACKS IS THE LIMIT',CR,LF,00));

CALL WRITEA('HOW MANY TRACKS? IN HEX ',00));

TRACKS = HEXIN(TRACKS);

CALL CO(CR),

CALL CO(LF),

PLATTERS = HEXIN(PLATTERS);

CALL CO(CR),

CALL CO(LF),

TRACK_CNT = 1,
DO WHILE TRACK_CNT <= TRACKS.
  PLAT_CNT = 1.
  DO WHILE PLAT_CNT <= PLATTERS.
    CALL UPDATE_CMD_BLK.
    CALL CO(CR).
    CALL WRITEA('@(TRACK = ',00)
    CALL DISP_HEX(@TRACK_CNT,I).
    CALL WRITEA('@(', HEAD = ',00).
    AA = DISK_REGS(SDH) AND 07H.
    CMD_BLK(INDX) COMMAND = FORMAT.
    CALL DISK(@CMD_BLK(INDX)).
  END.
  CALL UPDATE_CMD_BLK.
  PLAT_CNT = PLAT_CNT + 1.
  CALL CO(CR).
END.
/* CASE 3 - SCAN ID */
DO,
  CALL WRITEA('@(SCAN ID',CR,LF,LF,00).
  CALL WRITE_REGS.
  DISK_REGS(SDH) = DISK_REGS(SDH) - (PLATTERS).
  DISK_REGS(CYL_LB) = DISK_REGS(CYL_LB) + 1.
  IF DISK_REGS(CYL_LB) = 00 THEN
    DISK_REGS(CYL_LB) = DISK_REGS(CYL_LB) + 1.
  END.
  CALL DISPLAY HEX(@TRACK_CNT,I).
END.
/* CASE 4 - SEEK TRACK */
DO,
  CALL WRITEA('@(SEEK TRACK',CR,LF,LF,00).
  CALL WRITE_REGS.
  CMD_BLK(INDX) COMMAND = SEEK OR STEP RATE.
  DISK_REGS(SDH) = DISK_REGS(SDH) + 1.
END.
/* CASE 5 - RESTORE */
DO,
  CALL WRITEA('@(RESTORE COMMAND',CR,LF,LF,00).
  CALL WRITE_REGS.
  CMD_BLK(INDX) COMMAND = RESTORE OR STEP RATE.
  DISK_REGS(SDH) = DISK_REGS(SDH) - (PLATTERS).
END.
/* CASE 6 - READ DISK REGISTER FILE */
DO,
  CALL WRITEA('@(READ DISK REGISTERS',CR,LF,LF,00).
  CALL DISK(@CMD_BLK(INDX)).
  CALL DISP_CMD_BLK.
  CMD_BLK(INDX) COMMAND = 99H.
END.
/* CASE 7 - HELP TABLE */
DO,
  CALL WRITEA('@HELP).
END.
/* CASE 8 - EXAMINE COMMAND BLOCK */
DO,
  CALL DISP_CMD_BLK.
END.
CASE 9 - DISPLAY BUFFER DATA */

DO CASE 9
   CALL WRITEA(CHR('DISPLAY ASCII<A) OR HEX:(H:) ? 'OX) ».',
CHAR = CI.
   CALL CO(CHAR).
   CALL CO(LF).
   IF CHAR = 'A' THEN DO,
      INDEX = 0.
      DO WHILE CHAR <> ESC,
         DO I = 0 TO 255,
            CALL CO(BUFFER(INDEX + I)).
      END.
      INDEX = INDEX + I.
      CHAR = CI.
   END.
   IF CHAR = 'H' THEN DO,
      INDEX = 0.
      DO WHILE CHAR <> ESC,
         CALL DISP_HEX(BUFFER(INDEX), 256).
      INDEX = INDEX + 256.
      CHAR = CI.
   END.
   END.
   END. /* DO CASE */
ELSE DO,
   CALL WRITEA(CHR('*** DISK IS BUSY *** CR.00 »).
   DISK_IS_NOT_BUSY = TRUE.
   CALL DISP_STATUS.
   END.
   END. /* FOREVER */

END MAIN;

MODULE INFORMATION

CODE AREA SIZE  = OFDBH  40560
CONSTANT AREA SIZE  = 093CH  23640
VARIABLE AREA SIZE  = 0480H  11520
MAXIMUM STACK SIZE  = 00E5H  620
86B LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS

DICTIONARY SUMMARY

31KB MEMORY AVAILABLE
12KB MEMORY USED  (38%)
0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
APPENDIX D

This appendix contains a schematic of the previous design using PAL's to replace the random logic. The previous design could not do DMA transfers and inserted a large delay when transferring data from buffer RAM to the system. The PAL version does do DMA transfers and buffer reads happen at full SBX bus speed. One other minor change was to replace the 500 ns delay line with a 74LS164, which is a more cost effective solution.

This schematic is only a paper design since only random logic was replaced with the PAL's.

**PAL Equation's**

**PAL - Page 1:**

\[
\begin{align*}
\text{BDRD/} & = (IORD/ \cdot \text{MDACK/}) + (IORD/ \cdot \text{MCSO/} \cdot \text{MAO} \cdot \text{MA1} \cdot \text{MA2}) + \\
& \quad \text{(DELAYED-READ/} \cdot \text{CLK) IF BCS} \\
\text{LTCHSDH/} & = (\text{MCSO/} \cdot \text{MAO} \cdot \text{MA1} \cdot \text{MA2} \cdot \text{IOWR/}) \\
\text{RAMSEL/} & = (\text{MCSO} \cdot \text{MAO} \cdot \text{MA1} \cdot \text{MA2}) + (\text{BCS/}) + (\text{MDACK/}) \\
\text{IOBRDY/} & = (\text{MCS1/} \cdot \text{MAO} \cdot \text{MA1} \cdot \text{MA2} / \cdot \text{IOWR/}) \\
\text{IOBCR/} & = (\text{MCS1/} \cdot \text{MAO} \cdot \text{MA1} / \cdot \text{MA2} / \cdot \text{IOWR/}) \\
\text{BDWR/} & = (\text{IOWR/}) \text{ IF BCS} \\
\text{CS/} & = (\text{MCSO/}) \text{ IF BCS} \\
\text{CLK} & = (\text{MCSO/} \cdot \text{MAO} \cdot \text{MA1} / \cdot \text{MA2/}) + (\text{MCSO/} \cdot \text{MAO} / \cdot \text{MA1} \cdot \text{MA2}) + (\text{MCSO/} \cdot \text{MAO} \cdot \text{MA1} \cdot \text{MA2}) + (\text{MCSO/} \cdot \text{MAO} / \cdot \text{MA1} \cdot \text{MA2}) + (\text{MCSO/} \cdot \text{MAO} \cdot \text{MA1} \cdot \text{MA2}) + (\text{MCSO/} \cdot \text{MAO} \cdot \text{MA1} \cdot \text{MA2}) + (\text{MCSO/} \cdot \text{MAO} \cdot \text{MA1} \cdot \text{MA2})
\end{align*}
\]

**PAL - Page 2:**

\[
\begin{align*}
\text{MINTR1/MDRQT} & = (\text{PIN1}) \\
\text{MINTR0} & = (\text{PIN2}) + (\text{INTRQ}) \\
\text{COUNT} & = (\text{BDWR/} + \text{BDRD/}) \cdot (\text{RAMSEL/}) \\
\text{RSTCOUNT} & = (\text{IOBCR/}) + (\text{BCR/}) \\
\text{OE/} & = (\text{MDACK/}) + (\text{CS/}) \\
\text{CLR/} & = (\text{IOBCR/}) + (\text{BCR/})
\end{align*}
\]

5-903 231133-002
ICE™-42
8042 IN-CIRCUIT EMULATOR

- Precise, full-speed, real-time emulation
  - Load, drive, timing characteristics
  - Full-speed program RAM
  - Parallel ports
  - Data Bus
- User-specified breakpoints
- Execution trace
  - User-specified qualifier registers
  - Conditional trigger
  - Symbolic groupings and display
  - Instruction and frame modes
- Emulation timer
- Full symbolic debugging
- Single-line assembly and disassembly for program instruction changes
- Macro commands and conditional block constructs for automated debugging sessions
- HELP facility: ICE™-42 command syntax reference at the console
- User confidence test of ICE™-42 hardware

The ICE™-42 module resides in the Intellec Microcomputer Development System and interfaces to any user-designed 8042 or 8041A system through a cable terminating in an 8042 emulator microprocessor and a pin-compatible plug. The emulator processor, together with 2K bytes of user program RAM located in the ICE-42 buffer box, replaces the 8042 device in the user system while maintaining the 8042 electrical and timing characteristics. Powerful Intellec debugging functions are thus extended into the user system. Using the ICE-42 module, the designer can emulate the system's 8042 chip in real-time or single-step mode. Breakpoints allow the user to stop emulation on user-specified conditions, and a trace qualifier feature allows the conditional collection of 1000 frames of trace data. Using the single-line 8042 assembler the user may alter program memory using the 8042 assembler mnemonics and symbolic references, without leaving the emulator environment. Frequently used command sequences can be combined into compound commands and identified as macros with user-defined names.
FUNCTIONAL DESCRIPTION

Integrated Hardware and Software Development

The ICE-42 emulator allows hardware and software development to proceed interactively. This approach is more effective than the traditional method of independent hardware and software development followed by system integration. With the ICE-42 module, prototype hardware can be added to the system as it is designed. Software and hardware integration occurs while the product is being developed. Figure 1 shows the ICE-42 emulator connected to a user prototype.

The ICE-42 emulator assists four stages of development:

SOFTWARE DEBUGGING

This emulator operates without being connected to the user’s system before any of the user’s hardware is available. In this stage ICE-42 debugging capabilities can be used in conjunction with the Intellec text editor and 8042 macro-assembler to facilitate program development.

HARDWARE DEVELOPMENT

The ICE-42 module’s precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware.

SYSTEM INTEGRATION

Integration of software and hardware begins when any functional element of the user system hardware is connected to the 8042 socket. As each section of the user’s hardware is completed, it is added to the prototype. Thus, each section of the hardware and software is “system” tested in real-time operation as it becomes available.

SYSTEM TEST

When the user’s prototype is complete, it is tested with the final version of the user system software. The ICE-42 module is then used for real-time emulation of the 8042 chip to debug the system as a completed unit.

The final product verification test may be performed using the 8742 EPROM version of the 8042 microcomputer. Thus, the ICE-42 module provides the ability to debug a prototype or production system at any stage in its development without introducing extraneous hardware or software test tools.

Symbolic Debugging

The ICE-42 emulator permits the user to define and to use symbolic, rather than absolute, references to program and data memory addresses. Thus, there is no need to recall or look up the addresses of key locations in the program, or to become involved with machine code.

When a symbol is used for memory reference in an ICE-42 emulator command, the emulator supplies the corresponding location as stored in the ICE-42 emulator symbol table. This table can be loaded with the symbol table produced by the assembler during application program assembly. The user obtains the symbol table during software preparation simply by using the “DEBUG” switch in the 8042 macroassembler. Furthermore, the user interactively modifies the emulator symbol table by adding new symbols or changing or deleting old ones. This feature provides great flexibility in debugging and minimizes the need to work with hexadecimal values.

Through symbolic references in combination with other features of the emulator, the user can easily:

- Interpret the results of emulation activity collected during trace.
- Disassemble program memory to mnemonics, or assemble mnemonic instructions to executable code.
- Reference labels or addresses defined in a user program.

Automated Debugging and Testing

MACRO COMMAND

A macro is a set of commands given a name. A group of commands executed frequently can be defined as a macro. The user executes the group of commands by typing a colon followed by the macro name. Up to ten parameters may be passed to the macro.

Macro commands can be defined at the beginning of a debug session and then used throughout the whole session. One or more macro definitions can be saved on diskette for later use. The Intellec text editor may be used to edit the macro file. The macro definitions are easy to include in any later emulation session.
The power of the development system can be applied to manufacturing testing as well as development by writing test sequences as macros. The macros are stored on diskettes for use during system test.

**COMPOUND COMMAND**

Compound commands provide conditional execution of commands (IF command) and execution of commands repeatedly until certain conditions are met (COUNT, REPEAT commands).

Compound commands may be nested any number of times, and may be used in macro commands.

**Example:**

```
*DEFINE .I=0  ; Define symbol .I to 0
*COUNT 100H  ; Repeat the following commands 100H times.
..'IF .I AND 1 THEN ; Check if .I is odd
  ..*CBYTE .I= .I ; Fill the memory at location .I to value .I
..*END
..*I,.I+1 ; Increment .I by 1.
..*END ; Command executes upon carriage-return after END
```

(The asterisks are system prompts; the dots indicate the nesting level of compound commands.)

**Operating Modes**

The ICE-42 software is an Intelic RAM-based program that provides easy-to-use commands for initiating emulation, defining breakpoints, controlling trace data collection, and displaying and controlling system parameters. ICE-42 commands are configured with a broad range of modifiers that provide maximum flexibility in describing the operation to be performed.

**EMULATION**

The ICE-42 module can emulate the operation of prototype 8042 system, at real-time speed (up to 12MHz) or in single steps. Emulation commands to the ICE-42 module control the process of setting up, running, and halting an emulation of the user's 8042-based system. Breakpoints and tracepoints enable the ICE-42 emulator to halt emulation and provide a detailed trace of execution in any part of the user's program. A summary of the emulation commands is shown in Table 1.

**Table 1 Major Emulation Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO</td>
<td>Begins real-time emulation and optionally specifies break conditions.</td>
</tr>
<tr>
<td>BR0, BR1, BR</td>
<td>Sets or displays either or both Breakpoint Registers used for stopping real-time emulation.</td>
</tr>
<tr>
<td>STEP</td>
<td>Performs single-step emulation.</td>
</tr>
<tr>
<td>QR0, QR1</td>
<td>Specifies match conditions for qualified trace.</td>
</tr>
<tr>
<td>TR</td>
<td>Specifies or displays trace-data collection conditions and optionally sets Qualifier Register (QR0, QR1).</td>
</tr>
<tr>
<td>Synchronization Line Commands</td>
<td>Sets and displays status of synchronization line outputs or latched inputs. Used to allow real-time emulation or trace to start and stop synchronously with external events.</td>
</tr>
</tbody>
</table>

**Breakpoints**

The ICE-42 hardware includes two breakpoint registers that allow halting of emulation when specified conditions are met. The emulator continuously compares the values stored in the breakpoint registers with the status of specified address, opcode, operand, or port values, and halts emulation when this comparison is satisfied. When an instruction initiates a break, that instruction is executed completely before the break takes place. The ICE-42 emulator then regains control of the console and enters the interrogation mode. With the breakpoint feature, the user can request an emulation break when the program:

- Executes an instruction at a specific address or within a range of addresses.
ICE™-42 IN-CIRCUIT EMULATOR

- Executes a particular opcode.
- Receives a specific signal on a port pin.
- Fetches a particular operand from the user program memory.
- Fetches an operand from a specific address in program memory.

**Trace and Tracepoints**

Tracing is used with real-time and single-step emulation to record diagnostic information in the trace buffer as a program is executed. The information collected includes opcodes executed, port values, and memory addresses. The ICE-42 emulator collects 1000 frames of trace data.

If desired this information can be displayed as assembler instruction mnemonics for analysis during interrogation or single-step mode. The trace-collection facility may be set to run conditionally or unconditionally. Two unique trace qualifier registers, specified in the same way as breakpoint registers, govern conditional trace activity. The qualifiers can be used to condition trace data collection to take place as follows:

- Under all conditions (forever).
- Only while the trace qualifier is satisfied.
- For the frames or instructions preceding the time when a trace qualifier is first satisfied (pre-trigger trace).
- For the frames or instructions after a trace qualifier is first satisfied (post-triggered trace).

Table 2 shows an example of trace display.

**INTERROGATION AND UTILITY**

Interrogation and utility commands give convenient access to detailed information about the

<table>
<thead>
<tr>
<th>FRAME LOC OBJ</th>
<th>INSTRUCTION</th>
<th>P1 P2 TO T1</th>
<th>DBYIN</th>
<th>YOUT</th>
<th>YSTS</th>
<th>T0VF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000: 100H 2355</td>
<td>MOV A,#55H</td>
<td>FFH F FH 0 0</td>
<td>bFH DFH 02H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0004: 102H 39</td>
<td>OUTL Pl,A</td>
<td>F FH F FH 0 0</td>
<td>bFH DFH 02H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0008: 103H 3A</td>
<td>OUTL P2,A</td>
<td>55H F FH 0 0</td>
<td>bFH DFH 02H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0012: 104H 22</td>
<td>IN A,DbB</td>
<td>55H 55H 0 0</td>
<td>bFH 02H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0014: 105H 37</td>
<td>CPL A</td>
<td>55H 55H 0 0</td>
<td>DFH 02H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0016: 106H 02</td>
<td>OUT DDB,A</td>
<td>55H 55H 0 0</td>
<td>bFH 00H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0018: 107H bA03</td>
<td>MOV R2,#03H</td>
<td>55H 55H 0 0</td>
<td>bFH 99H 00H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0022: 109H b460</td>
<td>MOV R0,#.TABLE0</td>
<td>55H 55H 0 0</td>
<td>bFH 99H 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0026: 10BH b9b0</td>
<td>MOV R1,#.TABLE1</td>
<td>55H 55H 0 0</td>
<td>bFH 99H 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.LOOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0030: 10DH F0</td>
<td>MOV A,@RO</td>
<td>55H 55H 0 0</td>
<td>99H 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0032: 10EH A1</td>
<td>MOV @R1,A</td>
<td>55H 55H 0 0</td>
<td>bFH 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0034: 10FH 18</td>
<td>INC RO</td>
<td>55H 55H 0 0</td>
<td>bFH 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0036: 110H 19</td>
<td>INC R1</td>
<td>55H 55H 0 0</td>
<td>bFH 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0038: 111H EADD</td>
<td>DJNZ R2,.LOOP</td>
<td>55H 55H 0 0</td>
<td>bFH 99H 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.LOOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0042: 10DH F0</td>
<td>MOV A,@RO</td>
<td>55H 55H 0 0</td>
<td>99H 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0044: 10EH A1</td>
<td>MOV @R1,A</td>
<td>55H 55H 0 0</td>
<td>bFH 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0046: 10FH 18</td>
<td>INC RO</td>
<td>55H 55H 0 0</td>
<td>bFH 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0048: 110H 19</td>
<td>INC R1</td>
<td>55H 55H 0 0</td>
<td>bFH 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0050: 111H EADD</td>
<td>DJNZ R2,.LOOP</td>
<td>55H 55H 0 0</td>
<td>bFH 99H 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.LOOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0054: 10DH F0</td>
<td>MOV A,@RO</td>
<td>55H 55H 0 0</td>
<td>99H 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0056: 10EH A1</td>
<td>MOV @R1,A</td>
<td>55H 55H 0 0</td>
<td>bFH 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0058: 10FH 18</td>
<td>INC RO</td>
<td>55H 55H 0 0</td>
<td>bFH 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0060: 110H 19</td>
<td>INC R1</td>
<td>55H 55H 0 0</td>
<td>bFH 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0062: 111H EADD</td>
<td>DJNZ R2,.LOOP</td>
<td>55H 55H 0 0</td>
<td>bFH 99H 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0066: 113H 00</td>
<td>NOP</td>
<td>55H 55H 0 0</td>
<td>99H 01H 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
user program and the state of the 8042 that is useful in debugging hardware and software. Changes can be made in memory and in the 8042 registers, flags, and port values. Commands are also provided for various utility operations such as loading and saving program files, defining symbols, displaying trace data, controlling system synchronization and returning control to ISIS-II. A summary of the basic interrogation and utility commands is shown in Table 3. Two additional time-saving emulator features are discussed below.

**Single-Line Assembler/Disassembler**

The single-line assembler/disassembler (ASM and DASM commands) permits the designer to examine and alter program memory using assembly language mnemonics, without leaving the emulator environment or requiring time-consuming program reassembly. When assembling new mnemonic instructions into program memory, previously defined symbolic references (from the original program assembly, or subsequently defined during the emulation session)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELP</td>
<td>Displays help messages for ICE-42 emulator command-entry assistance.</td>
</tr>
<tr>
<td>LOAD</td>
<td>Loads user object program (8042 code) into user-program memory, and user symbols into ICE-42 emulator symbol table.</td>
</tr>
<tr>
<td>SAVE</td>
<td>Saves ICE-42 emulator symbol table and/or user object program in ISIS-II hexadecimal file.</td>
</tr>
<tr>
<td>LIST</td>
<td>Copies all emulator console input and output to ISIS-II file.</td>
</tr>
<tr>
<td>EXIT</td>
<td>Terminates ICE-42 emulator operation.</td>
</tr>
<tr>
<td>DEFINE</td>
<td>Defines ICE-42 emulator symbol or macro.</td>
</tr>
<tr>
<td>REMOVE</td>
<td>Removes ICE-42 emulator symbol or macro.</td>
</tr>
<tr>
<td>ASM</td>
<td>Assembles mnemonic instructions into user-program memory.</td>
</tr>
<tr>
<td>DASM</td>
<td>Disassembles and displays user-program memory contents.</td>
</tr>
<tr>
<td>Change/Display</td>
<td>Change or display value of symbolic reference in ICE-42 emulator symbol table, contents of key-word references (including registers, I/O ports, and status flags), or memory references.</td>
</tr>
<tr>
<td>EVALUATE</td>
<td>Evaluates expression and displays resulting value.</td>
</tr>
<tr>
<td>MACRO</td>
<td>Displays ICE-42 macro or macros.</td>
</tr>
<tr>
<td>INTERRUPT</td>
<td>Displays contents for the Data Bus and timer interrupt registers.</td>
</tr>
<tr>
<td>SECONDS</td>
<td>Displays contents of emulation timer, in microseconds.</td>
</tr>
<tr>
<td>PRINT</td>
<td>Displays trace data pointed to by trace buffer pointer.</td>
</tr>
<tr>
<td>MODE</td>
<td>Sets or displays the emulation mode, 8041A or 8042.</td>
</tr>
</tbody>
</table>
Table 4 HELP Command

*HELP
Help is available for the following items. Type HELP followed by the item name. The help items cannot be abbreviated. (For more information, type HELP HELP or HELP INFO.)

<table>
<thead>
<tr>
<th>Emulation</th>
<th>Trace Collection</th>
<th>Misc:</th>
<th>&lt;address&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO GR SYO</td>
<td>TR QR ORRO OR1 SY1</td>
<td>BASE</td>
<td>&lt;CPU#keyword&gt;</td>
</tr>
<tr>
<td>BR BROBR1</td>
<td>Step</td>
<td>DISABLE</td>
<td>&lt;expr&gt;</td>
</tr>
<tr>
<td>Trace Display</td>
<td>TRACE MOVE PRINT</td>
<td>ENABLE</td>
<td>&lt;ICE42#keyword&gt;</td>
</tr>
<tr>
<td>OLDEST NEWEST</td>
<td>Evaluate</td>
<td>ERROR</td>
<td>&lt;identifier&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EVALUATE</td>
<td>&lt;instruction&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HELP</td>
<td>&lt;masked#constant&gt;</td>
</tr>
<tr>
<td>Change/</td>
<td>Display/ Define/ Remove: INFO</td>
<td></td>
<td>&lt;match#cond&gt;</td>
</tr>
<tr>
<td>&lt;CHANGE&gt;</td>
<td>REMOVE CBYTE</td>
<td>&lt;LIGHTS&gt;</td>
<td>&lt;numeric#constant&gt;</td>
</tr>
<tr>
<td>&lt;DISPLAY&gt;</td>
<td>SYMBOL DBYTE DASM</td>
<td>LIST</td>
<td>&lt;partition&gt;</td>
</tr>
<tr>
<td>REGISTER</td>
<td>RESET</td>
<td>LOAD</td>
<td>&lt;string&gt;</td>
</tr>
<tr>
<td>SECONDS</td>
<td>WRITE</td>
<td>SAVE</td>
<td>&lt;string#constant&gt;</td>
</tr>
<tr>
<td>DEFINE</td>
<td>STACK</td>
<td>SY</td>
<td>SYMIMICIC</td>
</tr>
<tr>
<td>DEFINE</td>
<td></td>
<td></td>
<td>&lt;symbolic#constant&gt;</td>
</tr>
<tr>
<td>Macro:</td>
<td>Compound</td>
<td></td>
<td>&lt;trace#reference&gt;</td>
</tr>
<tr>
<td>DEFINE</td>
<td>DIR Commands:</td>
<td></td>
<td>&lt;unlimited#match#cond&gt;</td>
</tr>
<tr>
<td>DISABLE</td>
<td>ENABLE COUNT</td>
<td></td>
<td>&lt;user#symbols&gt;</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>PUT IF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;MACRO#DISPLAY&gt;</td>
<td>REPEAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;MACRO#INVOCATION&gt;</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*HELP IF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF &lt;expr&gt; &quot;THEN&quot;</td>
<td>&lt;true#list&gt; :=&quot;&lt;command&gt; &lt;cr&gt; @</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORIF &lt;expr&gt; &quot;OR&quot;</td>
<td>&lt;false#list&gt; :=&quot;&lt;command&gt; &lt;cr&gt; @</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELSE &lt;cr&gt;</td>
<td>&lt;command&gt; :=&quot;An ICE-42 command.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The <expr>s are evaluated in order as 16-bit unsigned integers. If one is reached whose value has low-order bit 1 (TRUE), all commands in the <true#list> following that <expr> are then executed and all commands in the other <true#list>s and in the <false#list> are skipped. If all <expr>s have value with low-order bit 0 (FALSE), then all commands in all <true#list>s are skipped and, if ELSE is present, all commands in the <false#list> are executed.

(EX: IF .LOOP=5 THEN
  STEP
  ELSE
  GO
END)

* * *

*EXIT
may be used in the instruction operand field. The emulator supplies the absolute address or data values as stored in the emulator symbol table. These features eliminate user time spent translating to and from machine code and searching for absolute addresses, with a corresponding reduction in transcription errors.

HELP

The HELP file allows display of ICE-42 command syntax information at the Intellec console. By typing "HELP", a listing of all items for which help messages are available is displayed. Typing "HELP <Item>" then displays relevant information about the item requested, including typical usage examples. Table 4 shows some sample HELP messages.

EMULATION ACCURACY

The speed and interface demands of a high-performance single-chip microcomputer require extremely accurate emulation, including full-speed, real-time operation with the full function of the microcomputer. The ICE-42 module achieves accurate emulation with an 8042 emulator chip, a special configuration of the 8042 microcomputer family, as its emulation processor.

Each of the 40 pins on the user plug is connected directly to the corresponding 8042 pin on the emulator chip. Thus the user system sees the emulator as an 8042 microcomputer at the 8042 socket. The resulting characteristics provide extremely accurate emulation of the 8042 including speed, timing characteristics, load and drive values, and crystal operation. However, the emulator may draw more power from the user system than a standard 8042 family device.

Additional emulator processor pins provide signals such as internal address, data, clock, and control lines to the emulator buffer box. These signals let static RAM in the buffer box substitute for on-chip program ROM or EPROM. The emulator chip also gives the ICE module "back-door" access to internal chip operation, allowing the emulator to break and trace execution without interfering with the values on the user-system pins.

Figure 1  A Typical 8042 Development Configuration. The host system is an Intellec Series IV. The ICE-42 module is connected to a user prototype system.

SPECIFICATIONS

ICE™-42 Operating Requirements

Intellec Model 800, Series II, Series III, or Series IV Microcomputer Development SYstem (64K RAM required)

System console (Model 800 only)

Intellec Diskette Operating System: ISIS (Version 3.4 or later).

Equipment Supplied

- Printed circuit boards (2)
- Emulation buffer box, Intellec interface cables, and user-interface cable with 8042 emulation processor

- Crystal power accessory
- Operating instructions manuals
- Diskette-based ICE-42 software (single and double density)

Emulation Clock

User's system clock (up to 12MHz) or ICE-42 crystal power accessory (12 MHz)

Environmental Characteristics

Operating Temperature — 0° to 40°C

Operating Humidity — Up to 95% relative humidity without condensation.
Physical Characteristics

Printed Circuit Boards
Width: 12.00 in. (30.48 cm)
Height: 6.75 in. (17.15 cm)
Depth: 0.50 in. (1.27 cm)

Buffer Box
Width: 8.00 in. (20.32 cm)
Length: 12.00 in. (30.48 cm)
Depth: 1.75 in. (4.44 cm)
Weight: 4.0 lb. (1.81 kg)

Electrical Characteristics

DC Power Requirements
(from Intellec® system)

- $V_{CC} = +5V$, ± 5%
- $I_{CC} = 13.2A$ max; $11.0A$ typical
- $V_{DD} = +12V$, ±5%
- $I_{DD} = 0.1A$ max; $0.05A$ typical
- $V_{BB} = -10V$, ±5%
- $I_{BB} = 0.05A$ max; $0.01A$ typical

User plug characteristics at 8042 socket —
Same as 8042 or 8742 except that the user system sees an added load of 25 pF capacitance and 50μA leakage from the ICE-42 emulator user plug at ports 1, 2, T0, and T1.

ORDERING INFORMATION

Part Number Description

ICE-42  8042 Microcontroller In-Circuit Emulator, cable assembly and interactive diskette software
MCS®-48
DISKETTE-BASED SOFTWARE
SUPPORT PACKAGE

- Extends Intellec microcomputer development system to support MCS-48 development
- MCS-48 assembler provides conditional assembly and macro capability
- Takes advantage of powerful ISIS-II file handling and storage capabilities
- Provides assembler output in standard Intel hex format

The MCS-48 assembler translates symbolic 8048 assembly language instructions into the appropriate machine operation codes, and provides both conditional and macroassembler programming. Output may be loaded either to an ICE-49 module for debugging or into the iUP Universal PROM Programmer for 8748 PROM programming. The MCS-48 assembler operates under the ISIS-II operating system on Intel Development systems.
FUNCTIONAL DESCRIPTION

The MCS-48 assembler translates symbolic 8048 assembly language instructions into the appropriate machine operation codes. The ability to refer to program addresses with symbolic names eliminates the errors of hand translation and makes it easier to modify programs when adding or deleting instructions. Conditional assembly permits the programmer to specify which portions of the master source document should be included or deleted in variations on a basic system design, such as the code required to handle optional external devices. Macro capability allows the programmer use of a single label to define a routine. The MCS-48 assembler will assemble the code required by the reserved routine whenever the macro label is inserted in the text. Output from the assembler is in standard Intel hex format. It may be either loaded directly to an in-circuit emulator (ICE-49) module for integrated hardware/software debugging, or loaded into the iUP Universal PROM Programmer for 8748 PROM programming. A sample assembly listing is shown in Table 1.

The MCS-48 assembler supports the 8048, 8049, 8050, 8020, 8021, 8022, 8041 and 8042. The MCS-48 assembler can also support CMOS versions of the 8048 family.

SPECIFICATIONS

Operating Environment
(All) Intel Microcomputer Development Systems
      (Series II, Series III/Series IV)
      Intel Personal Development System

Documentation Package

Titles of: User Guides
          Operating Instructions
          Reference Manuals

Ordering Information

Part Number   Description
MDS-D48*    MCS-48 Disk Based Assembler
            Requires Software License

*SMD is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.
iUP-200A/iUP-201A UNIVERSAL PROM PROGRAMMERS

MAJOR iUP-200A/iUP-201A FEATURES:

- Support for all Intel PROM families through multiple-device personality modules, which may also be used with the Intel personal development system (iPDS™).
- Serial interface to all Intellec® development systems.
- Powerful PROM programming software (iPPS).
- iUP system self-tests plus device integrity checks.

- Support for new personality modules that provide state of the art fast programming algorithms, the intelligen Identifier™, and a security bit.

ADDITIONAL iUP-201A FEATURES:

- Off-line editing, device duplication, and PROM memory locking.
- 32K-byte iUP RAM.
- 24-character alphanumeric display.
- Full hexadecimal plus 12-function keypad.

The iUP-200A and iUP-201A universal programmers program and verify data in all the Intel programmable ROMs (PROMs). They can also program the PROM memory portions of Intel's single-chip microcomputer and peripheral devices. When used with any Intellec® development system, the iUP-200A and iUP-201A universal programmers provide on-line programming and verification using the Intel PROM programming software (iPPS). In addition, the iUP-201A universal programmer supports off-line, stand-alone program editing, PROM duplication, and PROM memory locking. The iUP-200A universal programmer is expandable to an iUP-201A model.
FUNCTIONAL DESCRIPTION

The iUP-200A universal programmer operates in on-line mode. The iUP-201A universal programmer operates in both on-line and off-line mode.

On-line System Hardware

The iUP-200A and iUP-201A universal programmers are free-standing units that, when connected to any Intel development system having at least 64K bytes of host memory, provide on-line PROM programming and verification of Intel programmable devices. In addition, the universal programmer can read the contents of the ROM versions of these devices.

The universal programmer communicates with the host through a standard RS-232C serial data link. A serial converter is needed when using the MDS 800 as a host system. (Serial converters are available from other manufacturers.)

Each universal programmer contains an 8085 CPU, selectable power supply, 4.3K bytes of static RAM, a programmable timer, an interface for personality modules, an interface for the host system, and 12K bytes of programmed EPROM. The iUP-201A also has a keyboard and display. The programmed EPROM contains the firmware needed for all universal programmer editing and control functions.

A personality module adapts the universal programmer to a family of PROM devices; it contains all the hardware and firmware necessary to program either a family of Intel PROMs or a single Intel device. The user inserts the personality module into the universal programmer front panel. The personality module comes ready to use; no additional sockets or adapters are required.

Figure 1 shows the iUP-200A on-line system configuration, and Figure 2 shows the on-line system data flow.

On-line System Software

The Intel PROM programming software (iPPS) is included with both the iUP-200A and iUP-201A models of the universal programmer. Created to run on any Intellec development system, the iPPS software provides user control through an easy-to-use interactive interface. The iPPS software performs the following functions to make PROM programming quick and easy:

- Reads PROMs and ROMs
- Programs PROMs directly or from a file
Figure 2 On-Line System Data Flow

- Verifies PROM data with buffer data
- Locks EPROM memory from unauthorized access (on devices which support this feature)
- Prints PROM contents on the network or development system printer
- Performs interactive formatting operations such as interleaving, nibble swapping, bit reversal, and block moves
- Programs multiple PROMs from the source file, prompting the user to insert new PROMs
- Uses a buffer to change PROM contents

All iPPS commands, as well as program address and data information, are entered through the development system ASCII keyboard and displayed on the system CRT. Table 1 summarizes the iPPS commands.

The iPPS software lets the user load programs into a PROM from Intellec system memory or directly from a disk file. Access to the disk lets the user create and manipulate data in a virtual buffer with an address range up to 16M. This large block of data can be formatted into different PROM word sizes for program storage into several different PROM types. In addition, a program stored in the target PROM, the Intellec system memory, or a system disk file can be interleaved with a second program and entered into a specific target PROM or PROMs.

The iPPS software supports data manipulation in the following Intel formats: 8080 hexadecimal ASCII, 8080 absolute object, 8086 hexadecimal ASCII, 8086 absolute object, and 80286 absolute object. Addresses and data can be displayed in binary, octal, decimal, or hexadecimal. The user can easily change default data formats as well as number bases.

The user invokes the iPPS software from the ISIS operating system (Intellec 800, Series II, and Series III, versions V3.4 and later; Series IV, versions V1.0 and later). The software can be run under control of ISIS submit files, thereby freeing the user from repetitious command entry.
<table>
<thead>
<tr>
<th>Command Group</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND CONTROL GROUP</td>
<td>PROGRAM</td>
<td>CONTROLS EXECUTION OF THE IPPS SOFTWARE.</td>
</tr>
<tr>
<td></td>
<td>EXIT</td>
<td>Exits the IPPS software and returns control to the ISIS operating system.</td>
</tr>
<tr>
<td></td>
<td>&lt;ESC&gt;</td>
<td>Terminates the current command.</td>
</tr>
<tr>
<td></td>
<td>REPEAT</td>
<td>Repeats the previous command.</td>
</tr>
<tr>
<td></td>
<td>ALTER</td>
<td>Edits and re-executes the previous command.</td>
</tr>
<tr>
<td>UTILITY GROUP</td>
<td>DISPLAY</td>
<td>DISPLAYS USER INFORMATION AND STATUS AND SETS DEFAULT VALUES.</td>
</tr>
<tr>
<td></td>
<td>PRINT</td>
<td>Displays PROM, buffer, or file data on the console.</td>
</tr>
<tr>
<td></td>
<td>QUEUE</td>
<td>Prints PROM, buffer, or file data on the local printer.</td>
</tr>
<tr>
<td></td>
<td>HELP</td>
<td>Prints PROM, buffer, or file data on the network spooled printer.</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>Displays user assistance information.</td>
</tr>
<tr>
<td></td>
<td>BLANKCHECK</td>
<td>Displays buffer structure and status.</td>
</tr>
<tr>
<td></td>
<td>OVERLAY</td>
<td>Checks for unprogrammed PROMs.</td>
</tr>
<tr>
<td></td>
<td>TYPE</td>
<td>Checks whether non-blank PROMs can be programmed.</td>
</tr>
<tr>
<td></td>
<td>INITIALIZE</td>
<td>Selects the PROM type.</td>
</tr>
<tr>
<td></td>
<td>WORKFILES</td>
<td>Initializes the default number base and file type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies the drive device for temporary work files.</td>
</tr>
<tr>
<td>BUFFER GROUP</td>
<td>SUBSTITUTE</td>
<td>EDITS, MODIFIES, AND VERIFIES DATA IN THE BUFFER.</td>
</tr>
<tr>
<td></td>
<td>LOADDATA</td>
<td>Examines and modifies buffer data.</td>
</tr>
<tr>
<td></td>
<td>VERIFY</td>
<td>Loads a section of the buffer with a constant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Verifies data in the PROM with buffer data.</td>
</tr>
<tr>
<td>FORMATTING GROUP</td>
<td>FORMAT&quot;</td>
<td>REARRANGES DATA FROM THE PROM, BUFFER, OR FILE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Formats and interleaves buffer, PROM, or file data.</td>
</tr>
<tr>
<td>COPY GROUP</td>
<td>COPY (file to PROM)</td>
<td>COPIES DATA FROM ONE DEVICE TO ANOTHER.</td>
</tr>
<tr>
<td></td>
<td>COPY (PROM to file)</td>
<td>Programs the PROM with data in a file on disk.</td>
</tr>
<tr>
<td></td>
<td>COPY (buffer to PROM)</td>
<td>Saves PROM data in a file on disk.</td>
</tr>
<tr>
<td></td>
<td>COPY (PROM to buffer)</td>
<td>Programs the PROM with data from the buffer.</td>
</tr>
<tr>
<td></td>
<td>COPY (buffer to file)</td>
<td>Loads the buffer with data in the PROM.</td>
</tr>
<tr>
<td></td>
<td>COPY (file to buffer)</td>
<td>Saves the contents of the buffer in a file on disk.</td>
</tr>
<tr>
<td></td>
<td>COPY (file to URAM)</td>
<td>Loads the buffer from a file on disk.</td>
</tr>
<tr>
<td></td>
<td>COPY (URAM to file)</td>
<td>Loads file data into the iUP RAM (iUP-201A model only).</td>
</tr>
<tr>
<td></td>
<td>COPY (buffer to URAM)</td>
<td>Saves iUP URAM data in a file on disk (iUP-201A model only).</td>
</tr>
<tr>
<td></td>
<td>COPY (URAM to buffer)</td>
<td>Loads the buffer into the iUP URAM (iUP-201A model only).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Loads iUP URAM data into the buffer (iUP-201A model only).</td>
</tr>
<tr>
<td>SECURITY GROUP</td>
<td>KEYLOCK</td>
<td>LOCKS SELECTED DEVICES TO PREVENT UNAUTHORIZED ACCESS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Locks the PROM from unauthorized access.</td>
</tr>
</tbody>
</table>
System Expansion

The iUP-200A universal programmer can be easily upgraded (by the user) to an iUP-201A universal programmer for off-line operation. The upgrade kit (iUP-PAK-A) is available from Intel or your local Intel distributor.

Off-line System

The iUP-201A universal programmer has all the on-line features of the iUP-200A universal programmer plus off-line editing, PROM duplication, program verification, and locking of PROM memory independent of the host system. The iUP-201A universal programmer also accepts Intel hexadecimal programs developed on non-Intel development systems. Just a few key-strokes download the program into the iUP RAM for editing and loading into a PROM.

Off-line commands are entered using the off-line command keys summarized in Table 2.

In addition to the hardware components included as part of the iUP-200A, the iUP-201A contains a 24-character alphanumeric display, full hexadecimal 12-function keypad, and 32K bytes of iUP RAM. Figure 3 illustrates the iUP-201A keyboard and display.

The two logical devices accessible during off-line operation are the PROM device and the iUP RAM. A typical operation is copying the data from a PROM (or ROM) into the iUP RAM, modifying this data in iUP RAM, and programming the modified data back into a PROM device. The address range of the iUP RAM is automatically determined by the universal programmer when PROM type selection is made. Figure 4 shows the off-line system data flow.

Figure 3 iUP-201A Keyboard and Display
### Table 2 Off-Line Command Keys Summary

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ON LINE</strong></td>
<td>Selects either on-line or off-line operation. When on-line, all other function keys are disabled.</td>
</tr>
<tr>
<td><strong>DEVICE SELECT</strong></td>
<td>Selects the PROM type when using a personality module able to program multiple PROM devices.</td>
</tr>
<tr>
<td><strong>VER</strong></td>
<td>Verifies the contents of the installed PROM device with the contents of the iUP RAM. The universal programmer display indicates the address and the XOR of any mismatches.</td>
</tr>
<tr>
<td><strong>PROG</strong></td>
<td>Performs a device blank check and then programs the target PROM with data from the iUP RAM. If the blank check fails, pressing PROG again performs an overlay check to verify that non-blank PROMs can be programmed.</td>
</tr>
<tr>
<td><strong>ROM TO RAM</strong></td>
<td>Loads the iUP RAM with the data from the PROM device installed in the personality module.</td>
</tr>
<tr>
<td><strong>CLEAR</strong></td>
<td>Terminates the current off-line function, clears a user entry, or restores the display after an error.</td>
</tr>
<tr>
<td><strong>ENTER</strong></td>
<td>Transfers information from the universal programmer display (addresses, data, or baud rate) into the iUP RAM.</td>
</tr>
<tr>
<td><strong>SHIFT</strong></td>
<td>Selects an address field for display.</td>
</tr>
<tr>
<td><strong>ADDR 0</strong></td>
<td>Selects a data field for keypad editing and entry.</td>
</tr>
<tr>
<td><strong>SHIFT</strong></td>
<td>Loads a contiguous section of iUP RAM locations with a constant.</td>
</tr>
<tr>
<td><strong>DATA 1</strong></td>
<td>Downloads Intel hexadecimal data from any development system which has an RS-232C port.</td>
</tr>
<tr>
<td><strong>SHIFT</strong></td>
<td>Locks a PROM from unauthorized access.</td>
</tr>
</tbody>
</table>

5-923  
Order Number: 210319-003
SYSTEM DIAGNOSTICS

Both the iUP-200A and iUP-201A universal programmers include self-contained system diagnostics that verify system operation and aid the user in fault isolation. Diagnostics are performed on the power supply, CPU internal firmware ROM, internal RAM, timer, the iUP-201A keyboard, and the iUP RAM. In addition, tests are made on any personality module installed in the programmer the first time the module is accessed. The personality module tests include the power select circuitry and up to 4K of module firmware. Straight-forward messages are provided on the development system display in on-line mode and on the iUP-201A display in off-line mode.

PERSONALITY MODULES

A personality module is the interface between the iUP-200A/iUP-201A universal programmer (or an iPDS system) and a selected PROM (or ROM). Personality modules contain all the hardware and firmware for reading and programming a family of Intel devices. Each personality module is a single molded unit inserted into the front panel of the universal programmer. No additional adapters or sockets are needed. Table 3 lists the available personality modules.

Each personality module connects to the universal programmer through a 41-pin connector. Module firmware is uploaded into the iUP RAM and executed by the internal 8085A processor.

Table 3 IUP Personality Modules

<table>
<thead>
<tr>
<th>Personality Module</th>
<th>PROM Type</th>
<th>PROMs and ROMs Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>IUP-Fast 27/K</td>
<td>EPROM</td>
<td>2764, 2764A, 27128, 27256</td>
</tr>
<tr>
<td>IUP-F27/128</td>
<td>E²/EPROM</td>
<td>2716, 2732, 2732A, 2764, 27128, 2815, 2816</td>
</tr>
<tr>
<td>IUP-F87/51A</td>
<td>Microcontroller</td>
<td>8748, 8748H, 8048, 8749H, 8049H, 8049H,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8050H, 8751, 8751H, 8051</td>
</tr>
<tr>
<td>IUP-F87/44A</td>
<td>Peripheral</td>
<td>8741A, 8041A, 8742, 8042, 8744H, 8044AH, 8755A</td>
</tr>
</tbody>
</table>

Order Number. 210319-003
The personality module firmware contains routines necessary to read and program a family of PROMs. In addition, the personality module sends specific information about the selected PROM to the universal programmer to help perform PROM device integrity checks.

LEDs on each personality module indicate operational status. On some personality modules a column of LEDs indicate which PROM device type the user has selected. On some personality modules an LED below the socket indicates which socket is to be used. A red indicator light tells the user when power is being supplied to the selected device. Figure 5 shows the personality modules supported on the universal programmer.

In addition to the testing done by the iUP system self-tests, each personality module contains diagnostic firmware that performs selected PROM tests and indicates status. These tests are performed in both on-line and off-line modes. The PROM installation test verifies that the device is installed in the module correctly and that the ZIF socket is closed. The PROM blank check determines whether a device is blank. The universal programmer automatically determines whether the blank state is all zeros or all ones. The overlay check (performed when a PROM is not blank) determines which bits are programmed, compares those bits against the program to be loaded, and allows programming to continue if they match. As with the system self-tests, straightforward messages are provided. The user can invoke all of the PROM device integrity checks except the installation test (which occurs automatically any time an operation is selected).

Figure 6 illustrates a typical testing sequence.
Figure 6 PROM Testing Sequence
iUP-200A/iUP-201A SPECIFICATIONS

Control Processor
Intel 8085A microprocessor
6.144 MHz clock rate

Memory
RAM — 4.3 bytes static
ROM — 12K bytes EPROM

Interfaces
Keyboard — 16-character hexadecimal and 12-function keypad (iUP-201A model only)
Display — 24-character alphanumeric (iUP-201A model only)

Software
Monitor — system controller in pre-programmed EPROM
iPPS — Intel PROM programming software on supplied diskette

Physical Characteristics
Depth — 15 inches (38.1 cm)
Width — 15 inches (38.1 cm)
Height — 6 inches (15.2 cm)
Weight — 15 pounds (6.9 kg)

Electrical Characteristics
Selectable 100, 120, 200, or 240 Vac ± 10%; 50-60 Hz
Maximum power consumption — 80 watts

Environmental Characteristics
Reading temperature — 10°C to 40°C
Programming temperature — 25°C ± 5°
Operating humidity — 10% to 85% relative humidity

Reference Material

PERSONALITY MODULE SPECIFICATIONS

Memory
EPROM — up to 4K bytes

Physical Characteristics
Width — 5.5 inches (1.4 cm)
Height — 1.6 inches (4.1 cm)
Depth — 7.0 inches (17.8 cm)
Weight — 1 pound (.45 kg)

Electrical Characteristics
Maximum power consumption (module) — 7.5 watts
Maximum power consumption (device) — 2.5 watts
Maximum power consumption (total from iUP) — 10 watts

Environmental Characteristics
Reading temperature — 10°C to 40°C
Programming temperature — 25°C ± 5°
Operating humidity — 10% to 85% relative humidity

Reference Material
Appropriate personality module user’s guide:
164855 — IUP-F87/51A Personality Module User’s Guide.
164853 — IUP-F87/44A Personality Module User’s Guide.

ORDERING INFORMATION

<table>
<thead>
<tr>
<th>Part number</th>
<th>Description</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>iUP-200A</td>
<td>Intel on-line universal programmer</td>
<td>5-927</td>
</tr>
<tr>
<td>iUP-201A</td>
<td>Intel on-line/off-line universal programmer</td>
<td>210319-003</td>
</tr>
</tbody>
</table>

iUP-F87/51A  Microcontroller personality module

iUP-F87/44A  Peripheral personality module

iUP-200/201 U1 Upgrade Kit  Upgrades an iUP-200/201 universal programmer to an iUP-200A/201A universal programmer

iUP-PAK-A Upgrade Kit  Upgrades an iUP-200A universal programmer to an iUP-201A universal programmer

*The iUP-Fast 27/K personality module can be used only with an iUP-200A/201A universal programmer or an iUP-200/iUP-201 universal programmer upgraded to an A with the iUP-200/201 U1 upgrade kit. If used in an iPDS, this personality module requires version 1.4 or later of the IPPS-iPDS software. All iPDS-140 units shipped after June 1984 will contain this software.
Peripherals
Section
INTEL DATA COMMUNICATIONS
FAMILY OVERVIEW

Data Communications has become an increasingly important factor in computer system design with the evolution of distributed processing and remote, networked peripherals. Intel's data communications product line provides a range of components to satisfy the broad spectrum of speed, protocol support and protocol flexibility needs (Figure 1).

GLOBAL DATA COMMUNICATIONS:
ASYCHRONOUS AND SYNCHRONOUS PROTOCOLS

Dedicated data communications controllers

For low-to-medium speed (up to 19.2 Kbps), the 8251A USART (Universal Synchronous Asynchronous Receiver/Transmitter) is the industry standard for asynchronous communications. It can be used in such applications as personal computers, workstations, word processors, CRT terminals point-of-sale terminals, banking terminals, printers, communications processors, data concentrators, industrial control networks, etc.

The 8256 UART (Multi-function Universal Asynchronous Receiver/Transmitter) is an highly competent asynchronous communications controller. It considerably minimizes the number of LSI required in a system with an asynchronous interface. The 8256 integrates the four more common peripheral functions of a microprocessor based system as well as a full-duplex, double buffered serial asynchronous receiver/transmitter with an on-chip baud rate generator.

The 8273 is a dedicated high level peripheral controller for SDLC/HDLC protocol support. It provides an high level of Data Link Control support for IBM-SNA or CCITT X.25 compatible microcomputer systems. This device minimizes CPU overhead by supporting a comprehensive frame level operation. The 8273 is compatible with every telephone network-based communication system due to its speed (up to 64 Kbps) and flexible modem interface.

Multiprotocol controllers

Multi-protocol controllers bridge the gap between byte oriented and bit oriented protocols (HDLC/SDLC). They provide an easy migration path for the user through a single software reconfiguration. Design of high-level protocols like X.25 are considerably simplified when they are coupled with the power of high performance processors such as the iAPX 86/88/186, or 188. They are also used to implement custom high-level protocols on top of standard bit-synchronous protocols.

The dual-channel 8274 MPSC (Multi-Protocol Serial

![Figure 1: A Spectrum of Data Communications Solutions](image)
Controller) provide a solution for Asynchronous, Byte Synchronous (IBM Bisync) and Bit Synchronous (HDLC/SDLC) protocols support. It is optimized for high-speed applications requiring the flexibility of the protocol support and the integration of multiple communications channels.

The 82530 SCC (Serial Communications Controller) is another dual channel multiprotocol controller. It contains new functions including on-chip baud rate generators, digital phase locked loops, various data encoding/decoding schemes and extensive diagnostic capabilities. All these added features reduce the need for external logic and greatly improve the reliability and maintainability of the system.

Distributed Intelligence Systems

The 8044/8744 is a microcontroller with an on chip serial communication processor. It simplifies control of remote subsystems (subsystems that are physically separated from the host CPU and communicate over a serial link).

The 8044 and 8051 CPUs are identical. The serial communication is handled by an additional processor called the Serial Interface Unit (SIU). The SIU operates concurrently with the CPU and offers a high level of intelligence and performance for HDLC/SDLC based communications. The SIU can handle 2.4 Mbps in Half-Duplex mode.

In addition to controlling communications with the host CPU, the 8044 provides significant peripheral control. Examples include local keyboard, CRT and printer control as well as design of network for Distributed Intelligence Systems (Medical instrumentation, CATV, PABX, etc. . . .)

Detailed 8044/8744 information is contained in the Intel Microcontroller Handbook.

Instrumentation

The 8291A, 8292, and 8293 family of components provide complete, high-performance support for IEEE-488 (GPIB) standard interface. GPIB is used in instrumentation applications.

The 8291A implements the Talker/Listener functions of the GPIB.

The 8292 provides the controller functions. Operating in tandem with the 8291A, it complements its interface functions to provide a full-capability GPIB interface.

The 8293 is a low-power, high-current, HMOS 8-line transceiver. It provides the electrical interface to the GPIB.

Local Area Networks

Intel has developed the first complete VLSI solution for Local Area Networks (LANs) and Ethernet in particular: the 82586 Local Area Network Coprocessor and the 82501 ESI (Ethernet Serial Interface).

Four on chip DMA channels allow the 82586 to operate as a bus master. The 82586 manages the entire process of transmitting and receiving frames, thereby relieving the host processor of the tasks of managing the communication interface to the network.

An extensive set of diagnostic capabilities, implemented in silicon, simplifies the design of more reliable local networks and facilitates their maintenance. In order to take full advantage of the LAN concept and CSMA/CD access method, the 82586 architecture is software configurable. This allows the 82586 to be "customized" for other applications including serial backplanes (serial peripheral interconnection), low cost short distance LANs, broadband networks and medium speed (1-2 Mbps) LANs.

The 82501 is designed to work directly with the 82586 in Ethernet applications. The major functions of the ESI are to generate the 10 MHz transmit clock for the 82586, to perform Manchester encoding/decoding of transmitted/received frames, and to provide the electrical interface to the Ethernet transceiver cable.

The Intel Data Communications product family provides a wide range of solutions for the needs of data communications systems.
8251A PROGRAMMABLE COMMUNICATION INTERFACE

- Synchronous and Asynchronous Operation
- Synchronous 5–8 Bit Characters; Internal or External Character Synchronization; Automatic Sync Insertion
- Asynchronous 5–8 Bit Characters; Clock Rate—1, 16 or 64 Times Baud Rate; Break Character Generation; 1, 1½, or 2 Stop Bits; False Start Bit Detection; Automatic Break Detect and Handling
- Synchronous Baud Rate—DC to 64K Baud

- Asynchronous Baud Rate—DC to 19.2K Baud
- Full-Duplex, Double-Buffered Transmitter and Receiver
- Error Detection—Parity, Overrun and Framing
- Compatible with an Extended Range of Intel Microprocessors
- 28-Pin DIP Package
- All Inputs and Outputs are TTL Compatible
- Available in EXPRESS
  —Standard Temperature Range
  —Extended Temperature Range

The Intel® 8251A is the enhanced version of the industry standard, Intel 8251 Universal Synchronous/Asynchronous Receiver/Transmitter (USART), designed for data communications with Intel’s microprocessor families such as MCS-48, 80, 85, and iAPX-86, 88. The 8251A is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique presently in use (including IBM “bi-sync”). The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU. The CPU can read the complete status of the USART at any time. These include data transmission errors and control signals such as SYNDET, TxEMPTY. The chip is fabricated using N-channel silicon gate technology.

Figure 1. Block Diagram  Figure 2. Pin Configuration
FEATURES AND ENHANCEMENTS

The 8251A is an advanced design of the industry standard UART, the Intel® 8251. The 8251A operates with an extended range of Intel microprocessors and maintains compatibility with the 8251. Familiarization time is minimal because of compatibility and involves only knowing the additional features and enhancements, and reviewing the AC and DC specifications of the 8251A.

The 8251A incorporates all the key features of the 8251 and has the following additional features and enhancements:

- 8251A has double-buffered data paths with separate I/O registers for control, status, Data In, and Data Out, which considerably simplifies control programming and minimizes CPU overhead.
- In asynchronous operations, the Receiver detects and handles "break" automatically, relieving the CPU of this task.
- A refined Rx initialization prevents the Receiver from starting when in "break" state, preventing unwanted interrupts from a disconnected USART.
- At the conclusion of a transmission, TxD line will always return to the marking state unless SBRK is programmed.
- Tx Enable logic enhancement prevents a Tx Disable command from halting transmission until all data previously written has been transmitted. The logic also prevents the transmitter from turning off in the middle of a word.
- When External Sync Detect is programmed, Internal Sync Detect is disabled, and an External Sync Detect status is provided via a flip-flop which clears itself upon a status read.
- Possibility of false sync detect is minimized by ensuring that if double character sync is programmed, the characters be contiguously detected and also by clearing the Rx register to all ones whenever Enter Hunt command is issued in Sync mode.
- As long as the 8251A is not selected, the RD and WR do not affect the internal operation of the device.
- The 8251A Status can be read at any time but the status update will be inhibited during status read.
- The 8251A is free from extraneous glitches and has enhanced AC and DC characteristics, providing higher speed and better operating margins.
- Synchronous Baud rate from DC to 64K.

FUNCTIONAL DESCRIPTION

General

The 8251A is a Universal Synchronous/Asynchronous Receiver/Transmitter designed for a wide range of Intel microcomputers such as 8048, 8080, 8085, 8086 and 8088. Like other I/O devices in a microcomputer system, its functional configuration is programmed by the system’s software for maximum flexibility. The 8251A can support most serial data techniques in use, including IBM “bi-sync.”

In a communication environment an interface device must convert parallel format system data into serial format for transmission and convert incoming serial format data into parallel system data for reception. The interface device must also delete or insert bits or characters that are functionally unique to the communication technique. In essence, the interface should appear “transparent” to the CPU, a simple input or output of byte-oriented system data.

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8251A to the system Data Bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions of the CPU. Control words, Command words and Status information are also transferred through the Data Bus Buffer. The Command Status, Data-In and Data-Out registers are separate, 8-bit registers communicating with the system bus through the Data Bus Buffer.

This functional block accepts inputs from the system Control bus and generates control signals for overall device operation. It contains the Control Word Register and Command Word Register that store the various control formats for the device functional definition.

RESET (Reset)

A “high” on this input forces the 8251A into an “Idle” mode. The device will remain at “Idle” until a new set of control words is written into the 8251A to program its functional definition. Minimum RESET pulse width is 6 tCY (clock must be running).

A command reset operation also puts the device into the “Idle” state.
CLK (Clock)

The CLK input is used to generate internal device timing and is normally connected to the Phase 2 (TTL) output of the Clock Generator. No external inputs or outputs are referenced to CLK but the frequency of CLK must be greater than 30 times the Receiver or Transmitter data bit rates.

WR (Write)

A “low” on this input informs the 8251A that the CPU is writing data or control words to the 8251A.

RD (Read)

A “low” on this input informs the 8251A that the CPU is reading data or status information from the 8251A.

C/D (Control/Data)

This input, in conjunction with the WR and RD inputs, informs the 8251A that the word on the Data Bus is either a data character, control word or status information.

1 = CONTROL/STATUS; 0 = DATA.

CS (Chip Select)

A “low” on this input selects the 8251A. No reading or writing will occur unless the device is selected. When CS is high, the Data Bus is in the float state and RD and WR have no effect on the chip.

Modem Control

The 8251A has a set of control inputs and outputs that can be used to simplify the interface to almost any modem. The modem control signals are general purpose in nature and can be used for functions other than modem control, if necessary.

DSR (Data Set Ready)

The DSR input signal is a general-purpose, 1-bit inverting input port. Its condition can be tested by the CPU using a Status Read operation. The DSR input is normally used to test modem conditions such as Data Set Ready.

DTR (Data Terminal Ready)

The DTR output signal is a general-purpose, 1-bit inverting output port. It can be set “low” by programming the appropriate bit in the Command Instruction word. The DTR output signal is normally used for modem control such as Data Terminal Ready.

RTS (Request to Send)

The RTS output signal is a general-purpose, 1-bit inverting output port. It can be set “low” by programming the appropriate bit in the Command Instruction word. The RTS output signal is normally used for modem control such as Request to Send.

CTS (Clear to Send)

A “low” on this input enables the 8251A to transmit serial data if the Tx Enable bit in the Command byte is set to a “one.” If either a Tx Enable off or CTS off condition occurs while the Tx is in operation, the Tx will transmit all the data in the USART, written prior to Tx Disable command before shutting down.
Transmitter Buffer

The Transmitter Buffer accepts parallel data from the Data Bus Buffer, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin on the falling edge of TxC. The transmitter will begin transmission upon being enabled if CTS = 0. The TxD line will be held in the marking state immediately upon a master Reset or when Tx Enable or CTS is off or the transmitter is empty.

Transmitter Control

The Transmitter Control manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

TxRDY (Transmitter Ready)

This output signals the CPU that the transmitter is ready to accept a data character. The TxRDY output pin can be used as an interrupt to the system, since it is masked by TxEnable; or, for Polled operation, the CPU can check TxRDY using a Status Read operation. TxRDY is automatically reset by the leading edge of WR when a data character is loaded from the CPU.

Note that when using the Polled operation, the TxRDY status bit is not masked by TxEnable, but will only indicate the Empty/Full Status of the Tx Data Input Register.

TxEMPTY (Transmitter Empty)

When the 8251A has no characters to send, the TxEMPTY output will go "high." It resets upon receiving a character from CPU if the transmitter is enabled. TxEMPTY remains high when the transmitter is disabled. TxEMPTY can be used to indicate the end of a transmission mode, so that the CPU "knows" when to "turn the line around" in the half-duplex operational mode.

In the Synchronous mode, a "high" on this output indicates that a character has not been loaded and the SYNC character or characters are about to be or are being transmitted automatically as "fillers." TxEMPTY does not go low when the SYNC characters are being shifted out.

Receiver Buffer

The Receiver accepts serial data, converts this serial input to parallel format, checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU. Serial data is input to RxD pin, and is clocked in on the rising edge of RxC.

TxC (Transmitter Clock)

The Transmitter Clock controls the rate at which the character is to be transmitted. In the Synchronous transmission mode, the Baud Rate (1x) is equal to the TxC frequency. In Asynchronous transmission mode, the baud rate is a fraction of the actual TxC frequency. A portion of the mode instruction selects this factor; it can be 1, 1/16 or 1/64 the TxC.

For Example:

If Baud Rate equals 110 Baud,
TxC equals 110 Hz in the 1x mode.
TxC equals 1.72 kHz in the 16x mode.
TxC equals 7.04 kHz in the 64x mode.

The falling edge of TxC shifts the serial data out of the 8251A.
Receiver Control

This functional block manages all receiver-related activities which consists of the following features.

The RxD initialization circuit prevents the 8251A from mistaking an unused input line for an active low data line in the "break condition." Before starting to receive serial characters on the RxD line, a valid "1" must first be detected after a chip master Reset. Once this has been determined, a search for a valid low (Start bit) is enabled. This feature is only active in the asynchronous mode, and is only done once for each master Reset.

The False Start bit detection circuit prevents false starts due to a transient noise spike by first detecting the falling edge and then strobing the nominal center of the Start bit (RxD = low).

Parity error detection sets the corresponding status bit.

The Framing Error status bit is set if the Stop bit is absent at the end of the data byte (asynchronous mode).

RxRDY (Receiver Ready)

This output indicates that the 8251A contains a character that is ready to be input to the CPU. RxRDY can be connected to the interrupt structure of the CPU or, for polled operation, the CPU can check the condition of RxRDY using a Status Read operation.

RxEnable, when off, holds RxRDY in the Reset Condition. For Asynchronous mode, to set RxRDY, the Receiver must be enabled to sense a Start Bit and a complete character must be assembled and transferred to the Data Output Register. For Synchronous mode, to set RxRDY, the Receiver must be enabled and a character must finish assembly and be transferred to the Data Output Register.

Failure to read the received character from the Rx Data Output Register prior to the assembly of the next Rx Data character will set overrun condition error and the previous character will be written over and lost. If the Rx Data is being read by the CPU when the internal transfer is occurring, overrun error will be set and the old character will be lost.

RxC (Receiver Clock)

The Receiver Clock controls the rate at which the character is to be received. In Synchronous Mode, the Baud Rate (1x) is equal to the actual frequency of RxC. In Asynchronous Mode, the Baud Rate is a fraction of the actual RxC frequency. A portion of the mode instruction selects this factor: 1, 1/16 or 1/64 the RxC.

For example:

- Baud Rate equals 300 Baud, if RxC equals 300 Hz in the 1x mode;
- RxC equals 4800 Hz in the 16x mode;
- RxC equals 19.2 kHz in the 64x mode.

- Baud Rate equals 2400 Baud, if RxC equals 2400 Hz in the 1x mode;
- RxC equals 38.4 kHz in the 16x mode;
- RxC equals 153.6 kHz in the 64x mode.

Data is sampled into the 8251A on the rising edge of RxC.

NOTE: In most communications systems, the 8251A will be handling both the transmission and reception operations of a single link. Consequently, the Receive and Transmit Baud Rates will be the same. Both TxC and RxC will require identical frequencies for this operation and can be tied together and connected to a single frequency source (Baud Rate Generator) to simplify the interface.

![Figure 5. 8251A Block Diagram Showing Receiver Buffer and Control Functions](image-url)
SYNDET (SYNC Detect/BRKDET Break Detect)

This pin is used in Synchronous Mode for SYNDET and may be used as either input or output, programmable through the Control Word. It is reset to output mode low upon RESET. When used as an output (Internal Sync mode), the SYNDET pin will go "high" to indicate that the 8251A has located the SYNC character in the Receive mode. If the 8251A is programmed to use double Sync characters (bisync), then SYNDET will go "high" in the middle of the last bit of the second Sync character. SYNDET is automatically reset upon a Status Read operation.

When used as an input (external SYNC detect mode), a positive going signal will cause the 8251A to start assembling data characters on the rising edge of the next RxC. Once in SYNC, the "high" input signal can be removed. When External SYNC Detect is programmed, Internal SYNC Detect is disabled.

BREAK (Async Mode Only)

This output will go high whenever the receiver remains low through two consecutive stop bit sequences (including the start bits, data bits, and parity bits). Break Detect may also be read as a Status bit. It is reset only upon a master chip Reset or Rx Data returning to a "one" state.

DETAILED OPERATION DESCRIPTION

General

The complete functional definition of the 8251A is programmed by the system’s software. A set of control words must be sent out by the CPU to initialize the 8251A to support the desired communications format. These control words will program the: BAUD RATE, CHARACTER LENGTH, NUMBER OF STOP BITS, SYNCHRONOUS or ASYNCHRONOUS OPERATION, EVEN/ODD/OFF PARITY, etc. In the Synchronous Mode, options are also provided to select either internal or external character synchronization.

Once programmed, the 8251A is ready to perform its communication functions. The TxRDY output is raised "high" to signal the CPU that the 8251A is ready to receive a data character from the CPU. This output (TxRDY) is reset automatically when the CPU writes a character into the 8251A. On the other hand, the 8251A receives serial data from the MODEM or I/O device. Upon receiving an entire character, the RxRDY output is raised "high" to signal the CPU that the 8251A has a complete character ready for the CPU to fetch. RxRDY is reset automatically upon the CPU data read operation.

The 8251A cannot begin transmission until the Tx Enable (Transmitter Enable) bit is set in the Command Instruction and it has received a Clear To Send (CTS) input. The TxD output will be held in the marking state upon Reset.
Programming the 8251A

Prior to starting data transmission or reception, the 8251A must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251A and must immediately follow a Reset operation (internal or external).

The control words are split into two formats:
1. Mode Instruction
2. Command Instruction

Mode Instruction

This instruction defines the general operational characteristics of the 8251A. It must follow a Reset operation (internal or external). Once the Mode Instruction has been written into the 8251A by the CPU, SYNC characters or Command Instructions may be written.

Command Instruction

This instruction defines a word that is used to control the actual operation of the 8251A.

Both the Mode and Command Instructions must conform to a specified sequence for proper device operation (see Figure 7). The Mode Instruction must be written immediately following a Reset operation, prior to using the 8251A for data communication.

All control words written into the 8251A after the Mode Instruction will load the Command Instruction. Command Instructions can be written into the 8251A at any time in the data block during the operation of the 8251A. To return to the Mode Instruction format, the master Reset bit in the Command Instruction word can be set to initiate an internal Reset operation which automatically places the 8251A back into the Mode Instruction format. Command Instructions must follow the Mode Instructions or Sync characters.

Mode Instruction Definition

The 8251A can be used for either Asynchronous or Synchronous data communication. To understand how the Mode Instruction defines the functional operation of the 8251A, the designer can best view the device as two separate components, one Asynchronous and the other Synchronous, sharing the same package. The format definition can be changed only after a master chip Reset. For explanation purposes the two formats will be isolated.

NOTE: When parity is enabled it is not considered as one of the data bits for the purpose of programming the word length. The actual parity bit received on the Rx Data line cannot be read on the Data Bus. In the case of a programmed character length of less than 8 bits, the least significant Data Bus bits will hold the data; unused bits are "don't care" when writing data to the 8251A, and will be "zeros" when reading the data from the 8251A.

Asynchronous Mode (Transmission)

Whenever a data character is sent by the CPU the 8251A automatically adds a Start bit (low level) followed by the data bits (least significant bit first), and the programmed number of Stop bits to each character. Also, an even or odd Parity bit is inserted prior to the Stop bit(s), as defined by the Mode Instruction. The character is then transmitted as a serial data stream on the TxD output. The serial data is shifted out on the falling edge of TxC at a rate equal to 1, 1/16, or 1/64 that of the TxC, as defined by the Mode Instruction. BREAK characters can be continuously sent to the TxD if commanded to do so.

When no data characters have been loaded into the 8251A the TxD output remains "high" (marking) unless a Break (continuously low) has been programmed.

![Mode Instruction Format, Asynchronous Mode](image-url)
Asynchronous Mode (Receive)

The RxD line is normally high. A falling edge on this line triggers the beginning of a START bit. The validity of this START bit is checked by again strobing this bit at its nominal center (16X or 64X mode only). If a low is detected again, it is a valid START bit, and the bit counter will start counting. The bit counter thus locates the center of the data bits, the parity bit (if it exists) and the stop bits. If parity error occurs, the parity error flag is set. Data and parity bits are sampled on the RxD pin with the rising edge of RxC. If a low level is detected as the STOP bit, the Framing Error flag will be set. The STOP bit signals the end of a character. Note that the receiver requires only one stop bit, regardless of the number of stop bits programmed. This character is then loaded into the parallel I/O buffer of the 8251A. The RxRDY pin is raised to signal the CPU that a character is ready to be fetched. If a previous character has not been fetched by the CPU, the present character replaces it in the I/O buffer, and the OVERRUN Error flag is raised (thus the previous character is lost). All of the error flags can be reset by an Error Reset Instruction. The occurrence of any of these errors will not affect the operation of the 8251A.

Synchronous Mode (Transmission)

The TxD output is continuously high until the CPU sends its first character to the 8251A which usually is a SYNC character. When the CTS line goes low, the first character is serially transmitted out. All characters are shifted out on the falling edge of TxC. Data is shifted out at the same rate as the TxC.

Once transmission has started, the data stream at the TxD output must continue at the TxC rate. If the CPU does not provide the 8251A with a data character before the 8251A Transmitter Buffers become empty, the SYNC characters (or character if in single SYNC character mode) will be automatically inserted in the TxD data stream. In this case, the TxEMPTY pin is raised high to signal that the 8251A is empty and SYNC characters are being sent out. TxEMPTY does not go low when the SYNC is being shifted out (see figure below). The TxEMPTY pin is internally reset by a data character being written into the 8251A.

**Figure 9. Asynchronous Mode**

Synchronous Mode (Receive)

In this mode, character synchronization can be internally or externally achieved. If the SYNC mode has been programmed, ENTER HUNT command should be included in the first command instruction word written. Data on the RxD pin is then sampled on the rising edge of RxC. The content of the Rx buffer is compared at every bit boundary with the first SYNC character until a match occurs. If the 8251A has been programmed for two SYNC characters, the subsequent received character is also compared; when both SYNC characters have been detected, the USART ends the HUNT mode and is in character synchronization. The SYNDET pin is then set high, and is reset automatically by a STATUS READ. If parity is programmed, SYNDET will not be set until the middle of the parity bit instead of the middle of the last data bit.

In the external SYNC mode, synchronization is achieved by applying a high level on the SYNDET pin, thus forcing the 8251A out of the HUNT mode. The high level can be removed after one RXC cycle. An ENTER HUNT command has no effect in the asynchronous mode of operation.
Parity error and overrun error are both checked in the same way as in the Asynchronous Rx mode. Parity is checked when not in Hunt, regardless of whether the Receiver is enabled or not.

The CPU can command the receiver to enter the HUNT mode if synchronization is lost. This will also set all the used character bits in the buffer to a "one," thus preventing a possible false SYNDET caused by data that happens to be in the Rx Buffer at ENTER HUNT time. Note that the SYNDET F/F is reset at each Status Read, regardless of whether internal or external SYNC has been programmed. This does not cause the 8251A to return to the HUNT mode. When in SYNC mode, but not in HUNT, Sync Detection is still functional, but only occurs at the "known" word boundaries. Thus, if one Status Read indicates SYNDET and a second Status Read also indicates SYNDET, then the programmed SYNDET characters have been received since the previous Status Read. (If double character sync has been programmed, then both sync characters have been continguously received to gate a SYNDET indication.) When external SYNDET mode is selected, internal Sync Detect is disabled, and the SYNDET F/F may be set at any bit boundary.

The CPU can command the receiver to enter the HUNT mode if synchronization is lost. This will also set all the used character bits in the buffer to a "one," thus preventing a possible false SYNDET caused by data that happens to be in the Rx Buffer at ENTER HUNT time. Note that the SYNDET F/F is reset at each Status Read, regardless of whether internal or external SYNC has been programmed. This does not cause the 8251A to return to the HUNT mode. When in SYNC mode, but not in HUNT, Sync Detection is still functional, but only occurs at the "known" word boundaries. Thus, if one Status Read indicates SYNDET and a second Status Read also indicates SYNDET, then the programmed SYNDET characters have been received since the previous Status Read. (If double character sync has been programmed, then both sync characters have been continguously received to gate a SYNDET indication.) When external SYNDET mode is selected, internal Sync Detect is disabled, and the SYNDET F/F may be set at any bit boundary.

**Figure 10. Mode Instruction Format, Synchronous Mode**

**COMMAND INSTRUCTION DEFINITION**

Once the functional definition of the 8251A has been programmed by the Mode Instruction and the sync characters are loaded (if in Sync Mode) then the device is ready to be used for data communication. The Command Instruction controls the actual operation of the selected format. Functions such as: Enable Transmit/Receive, Error Reset and Modem Controls are provided by the Command Instruction.

Once the Mode Instruction has been written into the 8251A and Sync characters inserted, if necessary, then all further "control writes" (C/D = 1) will load a Command Instruction. A Reset Operation (internal or external) will return the 8251A to the Mode Instruction format.

**Note:** Internal Reset on Power-up

When power is first applied, the 8251A may come up in the Mode, Sync character or Command format. To guarantee that the device is in the Command Instruction format before the Reset command is issued, it is safest to execute the worst-case initialization sequence (sync mode with two sync characters). Loading three 00Hs consecutively into the device with C/D = 1 configures sync operation and writes two dummy 00H sync characters. An Internal Reset command (40H) may then be issued to return the device to the "Idle" state.
**STATUS READ DEFINITION**

In data communication systems it is often necessary to examine the "status" of the active device to ascertain if errors have occurred or other conditions that require the processor's attention. The 8251A has facilities that allow the programmer to "read" the status of the device at any time during the functional operation. (Status update is inhibited during status read.)

A normal "read" command is issued by the CPU with C/D = 1 to accomplish this function.

Some of the bits in the Status Read Format have identical meanings to external output pins so that the 8251A can be used in a completely polled or interrupt-driven environment. TxD RDY is an exception.

Note that status update can have a maximum delay of 28 clock periods from the actual event affecting the status.

**APPLICATIONS OF THE 8251A**

**Figure 12. Command Instruction Format**

**Figure 13. Status Read Format**

**Figure 14. Asynchronous Serial Interface to CRT Terminal, DC—9600 Baud**
Figure 15. Synchronous Interface to Terminal or Peripheral Device

Figure 16. Asynchronous Interface to Telephone Lines

Figure 17. Synchronous Interface to Telephone Lines
**ABSOLUTE MAXIMUM RATINGS**

Ambient Temperature Under Bias ................... 0°C to 70°C
Storage Temperature ............................... -65°C to +150°C
Voltage On Any Pin With Respect To Ground ........ -0.5V to +7V
Power Dissipation .................................. 1 Watt

*NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = 5.0V ± 5%, GND = 0V)**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>VCC</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 2.2 mA</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td>V</td>
<td>IOL = -400 µA</td>
<td></td>
</tr>
<tr>
<td>IOL</td>
<td>Output Float Leakage</td>
<td>±10</td>
<td>µA</td>
<td>VOUT = VCC TO 0.45V</td>
<td></td>
</tr>
<tr>
<td>IIL</td>
<td>Input Leakage</td>
<td>±10</td>
<td>µA</td>
<td>VIN = VCC TO 0.45V</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>Power Supply Current</td>
<td>100</td>
<td>mA</td>
<td>All Outputs = High</td>
<td></td>
</tr>
</tbody>
</table>

**CAPACITANCE (TA = 25°C, VCC = GND = 0V)**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cin</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td></td>
<td>fc = 1MHz</td>
</tr>
<tr>
<td>CIN/O</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
<td></td>
<td>Unmeasured pins returned to GND</td>
</tr>
</tbody>
</table>

**A.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = 5.0V ±10%, GND = 0V)**

**Bus Parameters** (Note 1)

**READ CYCLE**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tAR</td>
<td>Address Stable Before READ (CS, C/D)</td>
<td>0</td>
<td>ns</td>
<td></td>
<td>Note 2</td>
</tr>
<tr>
<td>tRA</td>
<td>Address Hold Time for READ (CS, C/D)</td>
<td>0</td>
<td>ns</td>
<td></td>
<td>Note 2</td>
</tr>
<tr>
<td>tRR</td>
<td>READ Pulse Width</td>
<td>250</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tRD</td>
<td>Data Delay from READ</td>
<td>200</td>
<td>ns</td>
<td>3, CL = 150 pF</td>
<td></td>
</tr>
<tr>
<td>tDF</td>
<td>READ to Data Floating</td>
<td>10</td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

**WRITE CYCLE**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tAW</td>
<td>Address Stable Before WRITE</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWA</td>
<td>Address Hold Time for WRITE</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWW</td>
<td>WRITE Pulse Width</td>
<td>250</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tDW</td>
<td>Data Set-Up Time for WRITE</td>
<td>150</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWD</td>
<td>Data Hold Time for WRITE</td>
<td>20</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tRV</td>
<td>Recovery Time Between WRITEES</td>
<td>6</td>
<td>tCY</td>
<td>Note 4</td>
<td></td>
</tr>
</tbody>
</table>

6-14
### OTHER TIMINGS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{CY}$</td>
<td>Clock Period</td>
<td>320</td>
<td>1350</td>
<td>ns</td>
<td>Notes 5, 6</td>
</tr>
<tr>
<td>$t_{CY}$</td>
<td>Clock High Pulse Width</td>
<td>120</td>
<td>$t_{CY} - 90$</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>$t_{CY}$</td>
<td>Clock Low Pulse Width</td>
<td>90</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>$t_R, t_F$</td>
<td>Clock Rise and Fall Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>$t_{DTx}$</td>
<td>TxD Delay from Falling Edge of TxC</td>
<td>1</td>
<td>µs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{TX}$</td>
<td>Transmitter Input Clock Frequency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1x Baud Rate</td>
<td>DC</td>
<td>64</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16x Baud Rate</td>
<td>DC</td>
<td>310</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64x Baud Rate</td>
<td>DC</td>
<td>615</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td>$t_{TPW}$</td>
<td>Transmitter Input Clock Pulse Width</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1x Baud Rate</td>
<td>12</td>
<td></td>
<td></td>
<td>$t_{CY}$</td>
</tr>
<tr>
<td></td>
<td>16x and 64x Baud Rate</td>
<td>1</td>
<td></td>
<td></td>
<td>$t_{CY}$</td>
</tr>
<tr>
<td>$t_{TPD}$</td>
<td>Transmitter Input Clock Pulse Delay</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1x Baud Rate</td>
<td>15</td>
<td></td>
<td></td>
<td>$t_{CY}$</td>
</tr>
<tr>
<td></td>
<td>16x and 64x Baud Rate</td>
<td>3</td>
<td></td>
<td></td>
<td>$t_{CY}$</td>
</tr>
<tr>
<td>$t_{RX}$</td>
<td>Receiver Input Clock Frequency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1x Baud Rate</td>
<td>DC</td>
<td>64</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16x Baud Rate</td>
<td>DC</td>
<td>310</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64x Baud Rate</td>
<td>DC</td>
<td>615</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td>$t_{RPW}$</td>
<td>Receiver Input Clock Pulse Width</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1x Baud Rate</td>
<td>12</td>
<td></td>
<td></td>
<td>$t_{CY}$</td>
</tr>
<tr>
<td></td>
<td>16x and 64x Baud Rate</td>
<td>1</td>
<td></td>
<td></td>
<td>$t_{CY}$</td>
</tr>
<tr>
<td>$t_{RPD}$</td>
<td>Receiver Input Clock Pulse Delay</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1x Baud Rate</td>
<td>15</td>
<td></td>
<td></td>
<td>$t_{CY}$</td>
</tr>
<tr>
<td></td>
<td>16x and 64x Baud Rate</td>
<td>3</td>
<td></td>
<td></td>
<td>$t_{CY}$</td>
</tr>
<tr>
<td>$t_{TXRDY}$</td>
<td>TxRDY Pin Delay from Center of Last Bit</td>
<td>14</td>
<td></td>
<td></td>
<td>$t_{CY}$ Note 7</td>
</tr>
<tr>
<td>$t_{TXRDY}$</td>
<td>TxRDY ↓ from Leading Edge of WR</td>
<td>400</td>
<td></td>
<td>ns</td>
<td>Note 7</td>
</tr>
<tr>
<td>$t_{RXRDY}$</td>
<td>RxRDY Pin Delay from Center of Last Bit</td>
<td>26</td>
<td></td>
<td></td>
<td>$t_{CY}$ Note 7</td>
</tr>
<tr>
<td>$t_{RXRDY}$</td>
<td>RxRDY ↓ from Leading Edge of RD</td>
<td>400</td>
<td></td>
<td>ns</td>
<td>Note 7</td>
</tr>
<tr>
<td>$t_{IS}$</td>
<td>Internal SYNDET Delay from Rising Edge of RxC</td>
<td>26</td>
<td></td>
<td></td>
<td>$t_{CY}$ Note 7</td>
</tr>
<tr>
<td>$t_{ES}$</td>
<td>External SYNDET Set-Up Time After Rising Edge of RxC</td>
<td>18</td>
<td></td>
<td></td>
<td>$t_{CY}$ Note 7</td>
</tr>
<tr>
<td>$t_{TXEMPTY}$</td>
<td>TxEMPTY Delay from Center of Last Bit</td>
<td>20</td>
<td></td>
<td></td>
<td>$t_{CY}$ Note 7</td>
</tr>
<tr>
<td>$t_{WC}$</td>
<td>Control Delay from Rising Edge of WRITE (TxEn, DTR, RTS)</td>
<td>8</td>
<td></td>
<td></td>
<td>$t_{CY}$ Note 7</td>
</tr>
<tr>
<td>$t_{CR}$</td>
<td>Control to READ Set-Up Time (DSR, CTS)</td>
<td>20</td>
<td></td>
<td></td>
<td>$t_{CY}$ Note 7</td>
</tr>
</tbody>
</table>

*NOTE:*

1. For Extended Temperature EXPRESS, use M8251A electrical parameters.
**A.C. CHARACTERISTICS (Continued)**

**NOTES:**
1. AC timings measured \( V_{OH} = 2.0\, V_{OL} = 2.0,\, V_{OL} = 0.8 \), and with load circuit of Figure 1.
2. Chip Select (CS) and Command/Data (C/D) are considered as Addresses.
3. Assumes that Address is valid before \( R_{D} \).
4. This recovery time is for Mode Initialization only. Write Data is allowed only when TxRDY = 1. Recovery Time between Writes for Asynchronous Mode is \( 8\, t_{CY} \) and for Synchronous Mode is \( 16\, t_{CY} \).
5. The TxC and RxC frequencies have the following limitations with respect to CLK: For 1x Baud Rate, \( f_{TX} \) or \( f_{RX} \leq 1/(30\, t_{CY}) \):
   - For 16x and 64x Baud Rate, \( f_{TX} \) or \( f_{RX} \leq 1/(4.5\, t_{CY}) \).
6. Reset Pulse Width = \( 6\, t_{CY} \) minimum; System Clock must be running during Reset.
7. Status update can have a maximum delay of 28 clock periods from the event affecting the status.

**TYPICAL Δ OUTPUT DELAY VS. Δ CAPACITANCE (pF)**

**A.C. TESTING INPUT, OUTPUT WAVEFORM**

![A.C. Testing Input, Output Waveform](image1)

**A.C. TESTING LOAD CIRCUIT**

![A.C. Testing Load Circuit](image2)
WAVEFORMS

SYSTEM CLOCK INPUT

TRANSMITTER CLOCK AND DATA

CLOCK ~

TRANSMITTER CLOCK AND DATA

CLOCK ~

RECEIVER CLOCK AND DATA

CLOCK ~

WRITE DATA CYCLE (CPU → USART)

WRITE DATA CYCLE (CPU → USART)

READ DATA CYCLE (CPU ← USART)
WAVEFORMS (Continued)

WRITE CONTROL OR OUTPUT PORT CYCLE (CPU → USART)

READ CONTROL OR INPUT PORT (CPU ← USART)

NOTE 1: \( T_{\text{INC}} \) includes the response timing of a control byte
NOTE 2: \( T_{\text{CR}} \) includes the effect of CTS on the TXENABLE circuitry

TRANSMITTER CONTROL AND FLAG TIMING (ASYNC MODE)

EXAMPLE FORMAT = 7 BIT CHARACTER WITH PARITY & 2 STOP BITS
8273, 8273-4
PROGRAMMABLE HDLC/SDLC PROTOCOL CONTROLLER

- CCITT X.25 Compatible
- HDLC/SDLC Compatible
- Full Duplex, Half Duplex, or Loop SDLT Operation
- Up to 64K Baud Synchronous Transfers (56K Baud with 8273-4)
- Automatic FCS (CRC) Generation and Checking
- Up to 9.6K Baud with On-Board Phase Locked Loop
- Programmable NRZI Encode/Decode
- Two User Programmable Modem Control Ports
- Digital Phase Locked Loop Clock Recovery
- Minimum CPU Overhead
- Fully Compatible with 8048/8080/8085/8088/8086/80188/80186 CPUs
- Single +5V Supply

The Intel® 8273 Programmable HDLC/SDLC Protocol Controller is a dedicated device designed to support the ISO/CCITT's HDLC and IBM's SDLC communication line protocols. It is fully compatible with Intel's new high performance microcomputer systems such as the MCS1 88/186™. A frame level command set is achieved by a unique microprogrammed dual processor chip architecture. The processing capability supported by the 8273 relieves the system CPU of the low level real-time tasks normally associated with controllers.

![Figure 1. Block Diagram](image1.png)

![Figure 2. Pin Configuration](image2.png)
A BRIEF DESCRIPTION OF HDLC/SDLC PROTOCOLS

General
The High Level Data Link Control (HDLC) is a standard communication link protocol established by International Standards Organization (ISO). HDLC is the discipline used to implement ISO X.25 packet switching systems. The Synchronous Data Link Control (SDLC) is an IBM communication link protocol used to implement the System Network Architecture (SNA). Both the protocols are bit oriented, code independent, and ideal for full duplex communication. Some common applications include terminal to terminal, terminal to CPU, CPU to CPU, satellite communication, packet switching and other high speed data links. In systems which require expensive cabling and interconnect hardware, any of the two protocols could be used to simplify interfacing (by going serial), thereby reducing interconnect hardware costs. Since both the protocols are speed independent, reducing interconnect hardware could become an important application.

Network
In both the HDLC and SDLC line protocols, according to a pre-assigned hierarchy, a PRIMARY (Control) STATION controls the overall network (data link) and issues commands to the SECONDARY (Slave) STATIONS. The latter comply with instructions and respond by sending appropriate RESPONSES. Whenever a transmitting station must end transmission prematurely it sends an ABORT character. Upon detecting an abort character, a receiving station ignores the transmission block called a FRAME. Time fill between frames can be accomplished by transmitting either continuous frame preambles called FLAGS or an abort character. A time fill within a frame is not permitted. Whenever a station receives a string of more that fifteen consecutive ones, the station goes into an IDLE state.

Frames
A single communication element is called a FRAME which can be used for both Link Control and data transfer purposes. The elements of a frame are the beginning eight bit FLAG (F) consisting of one zero, six ones, and a zero, an eight bit ADDRESS FIELD (A), an eight bit CONTROL FIELD (C), a variable (N-bit) INFORMATION FIELD (I), a sixteen bit FRAME CHECK SEQUENCE (FCS), and an eight bit end FLAG (F), having the same bit pattern as the beginning flag. In HDLC the Address (A) and Control (C) bytes are extendable. The HDLC and the SDLC use three types of frames; an Information Frame is used to transfer data, a Supervisory Frame is used for control purposes, and a Non-sequenced Frame is used for initialization and control of the secondary stations.

Frame Characteristics
An important characteristic of a frame is that its contents are made code transparent by use of a zero bit insertion and deletion technique. Thus, the user can adopt any format or code suitable for his system — it may even be a computer word length or a "memory dump". The frame is bit oriented that is, bits, not characters in each field, have specific meanings. The Frame Check Sequence (FCS) is an error detection scheme similar to the Cyclic Redundancy Checkword (CRC) widely used in magnetic disk storage devices. The Command and Response information frames contain sequence numbers in the control fields identifying the sent and received frames. The sequence numbers are used in Error Recovery Procedures (ERP) and as implicit acknowledgement of frame communication, enhancing the true full-duplex nature of the HDLC/SDLC protocols.

In contrast, BISYNC is basically half-duplex (two way alternate) because of necessity to transmit immediate acknowledgement frames. HDLC/SDLC therefore saves propagation delay times and have a potential of twice the throughput rate of BISYNC.

It is possible to use HDLC or SDLC over half duplex lines but there is a corresponding loss in throughput because both are primarily designed for full-duplex communication. As in any synchronous system, the bit rate is determined by the clock bits supplied by the modem, protocols themselves are speed independent.

A byproduct of the use of zero-bit insertion-deletion technique is the non-return-to-zero invert (NRZI) data transmission/reception compatibility. The latter allows HDLC/SDLC protocols to be used with asynchronous data communication hardware in which the clocks are derived from the NRZI encoded data.

References
IBM Synchronous Data Link Control General Information, IBM, GA 27-3093-1
IBM 3650 Retail Store System Loop Interface OEN Information, IBM, GA 27-3098-0
Guidebook to Data Communications, Training Manual, Hewlett-Packard 5955-1715
IBM Introduction to Teleprocessing, IBM, GC 20-8095-02
System Network Architecture, Technical Overview, IBM, GA 27-3102
System Network Architecture Format and Protocol, IBM GA 27-3112

<table>
<thead>
<tr>
<th>OPENING</th>
<th>ADDRESS</th>
<th>CONTROL</th>
<th>INFORMATION</th>
<th>FRAME CHECK</th>
<th>CLOSING</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLAG (F)</td>
<td>FIELD (A)</td>
<td>FIELD (C)</td>
<td>FIELD (I)</td>
<td>SEQUENCE (FCS)</td>
<td>FLAG (F)</td>
</tr>
<tr>
<td>01111110</td>
<td>8 BITS</td>
<td>8 BITS</td>
<td>VARIABLE LENGTH (ONLY IN I FRAMES)</td>
<td>16 BITS</td>
<td>01111110</td>
</tr>
</tbody>
</table>

Figure 3. Frame Format
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>40</td>
<td>I</td>
<td>Power Supply: +5V Supply.</td>
</tr>
<tr>
<td>GND</td>
<td>20</td>
<td>I</td>
<td>Ground: Ground.</td>
</tr>
<tr>
<td>RESET</td>
<td>4</td>
<td>I</td>
<td>Reset: A high signal on this pin will force the 8273 to an idle state. The 8273 will remain idle until a command is issued by the CPU. The modem interface output signals are forced high. Reset must be true for a minimum of 10 TCY.</td>
</tr>
<tr>
<td>CS</td>
<td>24</td>
<td>I</td>
<td>Chip Select: The RD and WR inputs are enabled by the chip select input.</td>
</tr>
<tr>
<td>DB7-DB0</td>
<td>19-12</td>
<td>I/O</td>
<td>Data Bus: The Data Bus lines are bi-directional three-state lines which interface with the system Data Bus.</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>I</td>
<td>Write Input: The Write signal is used to control the transfer of either a command or data from CPU to the 8273.</td>
</tr>
<tr>
<td>RD</td>
<td>9</td>
<td>I</td>
<td>Read Input: The Read signal is used to control the transfer of either a data byte or a status word from the 8273 to the CPU.</td>
</tr>
<tr>
<td>TxINT</td>
<td>2</td>
<td>O</td>
<td>Transmitter Interrupt: The Transmitter interrupt signal indicates that the transmitter logic requires service.</td>
</tr>
<tr>
<td>RxINT</td>
<td>11</td>
<td>O</td>
<td>Receiver Interrupt: The Receiver interrupt signal indicates that the Receiver logic requires service.</td>
</tr>
<tr>
<td>TxDRQ</td>
<td>6</td>
<td>O</td>
<td>Transmitter Data Request: Requests a transfer of data between memory and the 8273 for a transmit operation.</td>
</tr>
<tr>
<td>RxRDQ</td>
<td>8</td>
<td>O</td>
<td>Receiver DMA Request: Requests a transfer of data between the 8273 and memory for a receive operation.</td>
</tr>
<tr>
<td>TxDACK</td>
<td>5</td>
<td>I</td>
<td>Transmitter DMA Acknowledge: The Transmitter DMA acknowledge signal notifies the 8273 that the TxDMA cycle has been granted.</td>
</tr>
<tr>
<td>RxDACK</td>
<td>7</td>
<td>I</td>
<td>Receiver DMA Acknowledge: The Receiver DMA acknowledge signal notifies the 8273 that the RxDMA cycle has been granted.</td>
</tr>
<tr>
<td>A1-A0</td>
<td>22-21</td>
<td>I</td>
<td>Address: These two lines are CPU Interface Register Select lines.</td>
</tr>
<tr>
<td>TxD</td>
<td>29</td>
<td>O</td>
<td>Transmitter Data: This line transmits the serial data to the communication channel.</td>
</tr>
<tr>
<td>TxC</td>
<td>28</td>
<td>I</td>
<td>Transmitter Clock: The transmitter clock is used to synchronize the transmit data.</td>
</tr>
<tr>
<td>RxD</td>
<td>26</td>
<td>I</td>
<td>Receiver Data: This line receives serial data from the communication channel.</td>
</tr>
<tr>
<td>RxC</td>
<td>27</td>
<td>I</td>
<td>Receiver Clock: The Receiver Clock is used to synchronize the receive data.</td>
</tr>
</tbody>
</table>

FUNCTIONAL DESCRIPTION

General

The Intel® 8273 HDLC/SDLC controller is a microcomputer peripheral device which supports the International Standards Organization (ISO) High Level Data Link Control (HDLC), and IBM Synchronous Data Link Control (SDLC) communications protocols. This controller minimizes CPU software by supporting a comprehensive frame-level instruction set and by hardware implementation of the low level tasks associated with frame assembly/disassembly and data integrity. The 8273 can be used in either synchronous or asynchronous applications.

In asynchronous applications the data can be programmed to be encoded/decoded in NRZI code. The clock is derived from the NRZI data using a digital phase locked loop. The data transparency is achieved by using a zero-bit insertion/deletion technique. The frames are automatically checked for errors during reception by verifying the Frame Check Sequence (FCS); the FCS is automatically generated and appended before the final flag in transmit.
The 8273 recognizes and can generate flags (01111110): Abort, Idle, and GA (EOP) characters. The 8273 can assume either a primary (control) or a secondary (slave) role. It can therefore be readily implemented in an SDLC loop configuration as typified by the IBM 3650 Retail Store System by programming the 8273 into a one-bit delay mode. In such a configuration, a two wire pair can be effectively used for data transfer between controllers and loop stations. The digital phase locked loop output pin can be used by the loop station without the presence of an accurate Tx clock.

CPU Interface
The CPU interface is optimized for the MCS-80/85™ bus with an 8257 DMA controller. However, the interface is flexible, and allows either DMA or non-DMA data transfers, interrupt or non-interrupt driven. It further allows maximum line utilization by providing early interrupt mechanism for buffered (only the information field can be transferred to memory) Tx command overlapping. It also provides separate Rx and Tx interrupt output channels for efficient operation. The 8273 keeps the interrupt request active until all the associated interrupt results have been read. The CPU utilizes the CPU interface to specify commands and transfer data. It consists of seven registers addressed via CS, A1, Ao, RD and WR signals and two independent data registers for receive data and transmit data. A1, Ao are generally derived from two low order bits of the address bus. If an 8080 based CPU is utilized, the RD and WR signals may be driven by the 8228 I/OR and I/OW.

Register Description
Command
Operations are initiated by writing an appropriate command in the Command Register.

Parameter
Parameters of commands that require additional information are written to this register.

Result
Contains an immediate result describing an outcome of an executed command.

Transmit Interrupt Result
Contains the outcome of 8273 transmit operation (good/bad completion).

Receive Interrupt Result
Contains the outcome of 8273 receive operation (good/bad completion), followed by additional results which detail the reason for interrupt.

Status
The status register reflects the state of the 8273 CPU Interface.

DMA Data Transfers
The 8273 CPU interface supports two independent data interfaces: receive data and transmit data. At high data transmission speeds the data transfer rate of the 8273 is great enough to justify the use of direct memory access (DMA) for the data transfers. When the 8273 is configured in DMA mode, the elements of the DMA interfaces are

<table>
<thead>
<tr>
<th>Register</th>
<th>A1</th>
<th>A0</th>
<th>TxDACK</th>
<th>RxDACK</th>
<th>CS</th>
<th>RD</th>
<th>WR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Status</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parameter</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Result</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Reset</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TxINT Result</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RxDINT Result</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Transmit Data</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4. 8273 Block Diagram Showing CPU Interface Functions
**RxDACK:** Receive DMA Acknowledge

The RxDACK signal notifies the 8273 that a receive DMA cycle has been granted. It is also used with RD to read data from the 8273 in non-DMA mode. Note: WR must not be asserted while RxDACK is active.

**RD, WR:** Read, Write

The RD and WR signals are used to specify the direction of the data transfer.

DMA transfers require the use of a DMA controller such as the Intel 8257. The function of the DMA controller is to provide sequential addresses and timing for the transfer, at a starting address determined by the CPU. Counting of data block lengths is performed by the 8273.

To request a DMA transfer the 8273 raises the appropriate DMA REQUEST. DMA ACKNOWLEDGE and READ enable DMA data onto the bus (independently of CHIP SELECT). DMA ACKNOWLEDGE and WRITE transfers DMA data to the 8273 (independent of CHIP SELECT).

It is also possible to configure the 8273 in the non-DMA data transfer mode. In this mode the CPU module must pass data to the 8273 in response to non-DMA data requests indicated by the status word.

**Modem Interface**

The 8273 Modem interface provides both dedicated and user defined modem control functions. All the control signals are active low so that EIA RS-232C inverting drivers (MC 1488) and inverting receivers (MC 1489) may be used to interface to standard modems. For asynchronous operation, this interface supports programmable NRZI data encode/decode, a digital phase locked loop for efficient clock extraction from NRZI data, and modem control ports with automatic CF, CD monitoring and RTS generation. This interface also allows the 8273 to operate in PRE-FRAME SYNC mode in which the 8273 prefixes 16 transitions to a frame to synchronize idle lines before transmission of the first flag.

It should be noted that all the 8273 port operations deal with logical values, for instance, bit D0 of Port A will be a one when CTS (Pin 30) is a physical zero (logical one).

**Port A — Input Port**

During operation, the 8273 interrogates input pins CTS (Clear to Send) and CD (Carrier Detect). CTS is used to condition the start of a transmission. If during transmission CTS is lost the 8273 generates an interrupt. During reception, if CD is lost, the 8273 generates an interrupt. The user defined input bits correspond to the 8273 PA4, PA3, and PA2 pins. The 8273 does not interrogate or manipulate these bits.

<table>
<thead>
<tr>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>CTS — CLEAR TO SEND</th>
<th>CD — CARRIER DETECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The user defined input bits correspond to the 8273 PA4, PA3, and PA2 pins. The 8273 does not interrogate or manipulate these bits.

**Port B — Output Port**

During normal operation, if the CPU sets RTS active, the 8273 will not change this pin; however, if the CPU sets RTS inactive, the 8273 will activate it before each transmission and deactivate it one byte time after transmission. While the receiver is active the flag detect pin is pulsed each time a flag sequence is detected in the receive data stream. Following an 8273 reset, all pins of Port B are set to a high, inactive level.

The user defined output bits correspond to the state of PB4-PB1 pins. The 8273 does not interrogate or manipulate these bits.
Serial Data Logic

The Serial data is synchronized by the user transmit (TxC) and receive (RxC) clocks. The leading edge of TxC generates new transmit data and the trailing edge of RxC is used to capture receive data. The NRZI encoding/decoding of the receive and transmit data is programmable.

The diagnostic features included in the Serial Data logic are programmable loop back of data and selectable clock for the receiver. In the loop-back mode, the data presented to the TxD pin is internally routed to the receive data input circuitry in place of the RxD pin, thus allowing a CPU to send a message to itself to verify operation of the 8273.

In the selectable clock diagnostic feature, when the data is looped back, the receiver may be presented incorrect sample timing by the external circuitry. The user may select to substitute the TxC pin for the RxC input on-chip so that the clock used to generate the loop back data is used to sample it. Since TxD is generated off the leading edge of TxC and RxD is sampled on the trailing edge, the selected clock allows bit synchronism.

![Figure 6. Transmit/Receive Timing](image)

Asynchronous Mode Interface

Although the 8273 is fully compatible with the HDLC/SDLC communication line protocols, which are primarily designed for synchronous communication, the 8273 can also be used in asynchronous applications by using this interface. The interface employs a digital phase locked loop (DPLL) for clock recovery from a receive data stream and programmable NRZI encoding and decoding of data. The use of NRZI coding with SDLC transmission guarantees that within a frame, data transitions will occur at least every five bit times — the longest sequence of ones which may be transmitted without zero-bit insertion. The DPLL should be used only when NRZI coding is used since the NRZI coding will transmit zero sequence as line transitions. The digital phase locked loop also facilitates full-duplex and half-duplex asynchronous implementation with, or without modems.
Digital Phase Locked Loop

In asynchronous applications, the clock is derived from the receiver data stream by the use of the digital phase locked loop (DPLL). The DPLL requires a clock input at 32 times the required baud rate. The receive data (RxO) is sampled with this $32X$ CLK and the 8273 DPLL supplies a sample pulse nominally centered on the RxO bit cells. The DPLL has a built-in "stiffness" which reduces sensitivity to line noise and bit distortion. This is accomplished by making phase error adjustments in discrete increments. Since the nominal pulse is made to occur at 32 counts of the $32X$ CLK, these counts are subtracted or added to the nominal, depending upon which quadrant of the four error quadrants the data edge occurs in. For example if an RxO edge is detected in quadrant A1, it is apparent that the DPLL sample "A" was placed too close to the trailing edge of the data cell; sample "B" will then be placed at $T = (T_{nominal} - 2\text{ counts}) = 30$ counts of the $32X$ CLK to move the sample pulse "B" toward the nominal center of the next bit cell. A data edge occurring in quadrant B1 would cause a smaller adjustment of phase with $T = 31$ counts of the $32X$ CLK. Using this technique the DPLL pulse will converge to nominal bit center within 12 data bit times, worst case, with constant incoming RxO edges.

A method of attaining bit synchronism following a line idle is to use PRE-FRAME SYNC mode of transmission.

![Figure 7. DPLL Sample Timing](image-url)
Synchronous Modem — Duplex or Half Duplex Operation

Asynchronous Modems — Duplex or Half Duplex Operation

Asynchronous — No Modems — Duplex or Half Duplex
**SDLC Loop**

The DPLL simplifies the SDLC loop station implementation. In this application, each secondary station on a loop data link is a repeater set in one-bit delay mode. The signals sent out on the loop by the loop controller (primary station) are relayed from station to station then, back to the controller. Any secondary station finding its address in the A field captures the frame for action at that station. All received frames are relayed to the next station on the loop.

Loop stations are required to derive bit timing from the incoming NRZI data stream. The DPLL generates sample Rx clock timing for reception and uses the same clock to implement Tx clock timing.

**Figure 8. SDLC Loop Application**
PRINCIPLES OF OPERATION

The 8273 is an intelligent peripheral controller which relieves the CPU of many of the rote tasks associated with constructing and receiving frames. It is fully compatible with the MCS-80/85™ system bus. As a peripheral device, it accepts commands from a CPU, executes these commands and provides an Interrupt and Result back to the CPU at the end of the execution. The communication with the CPU is done by activation of CS, RD, WR pins, while the A1, A0 select the appropriate registers on the chip as described in the Hardware Description Section.

The 8273 operation is composed of the following sequence of events:

1. **Command Phase**: CPU writes command and parameters into the 8273 command and parameter registers.
2. **Execution Phase**: The 8273 is on its own to carry out the command.
3. **Result Phase**: The 8273 signals the CPU that the execution has finished. The CPU must perform a read operation of one or more of the registers.

*Figure 9. Command Phase Flowchart*

The Command Phase

During the command phase, the software writes a command to the command register. The command bytes provide a general description of the type of operation requested. Many commands require more detailed information about the command. In such a case up to four parameters are written into the parameter register. The flowchart of the command phase indicates that a command may not be issued if the Status Register indicates that the device is busy. Similarly, if a parameter is issued when the Parameter Buffer shows full, incorrect operation will occur.

The 8273 is a duplex device and both transmitter and receiver may each be executing a command or passing results at any given time. For this reason separate interrupt pins are provided. However, the command register must be used for one command sequence at a time.

**Status Register**

The status register contains the status of the 8273 activity. The description is as follows.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>CBSY (Command Busy)</td>
</tr>
<tr>
<td>6</td>
<td>CBF (Command Buffer Full)</td>
</tr>
<tr>
<td>5</td>
<td>CPBF (Command Parameter Buffer Full)</td>
</tr>
<tr>
<td>4</td>
<td>CRBF (Command Result Buffer Full)</td>
</tr>
</tbody>
</table>

**Bit 7 CBSY (Command Busy)**

Indicates in-progress command, set for CPU poll when Command Register is full, reset upon command phase completion. It is improper to write a command when CBSY is set; it results in incorrect operation.

**Bit 6 CBF (Command Buffer Full)**

Indicates that the command register is full, it is reset when the 8273 accepts the command byte but does not imply that execution has begun.

**Bit 5 CPBF (Command Parameter Buffer Full)**

CPBF is set when the parameter buffer is full, and is reset by the 8273 when it accepts the parameter. The CPU may poll CPBF to determine when additional parameters may be written.

**Bit 4 CRBF (Command Result Buffer Full)**

Indicates that an executed command immediate result is present in the Result Register. It is set by 8273 and reset when CPU reads the result.
Bit 3 RxINT (Receiver Interrupt)
RxINT indicates that the receiver requires CPU attention. It is identical to RxINT (pin 11) and is set by the 8273 either upon good/bad completion of a specified command or by Non-DMA data transfer. It is reset only after the CPU has read the result byte or has received a data byte from the 8273 in a Non-DMA data transfer.

Bit 2 TxlNT (Transmitter Interrupt)
The TxlNT indicates that the transmitter requires CPU attention. It is identical to TxlNT (pin 2). It is set by 8273 either upon good/bad completion of a specified command or by Non-DMA data transfer. It is reset only after the CPU has read the result byte or has transferred transmit data byte to the 8273 in a Non-DMA transfer.

Bit 1 RxlRA (Receiver Interrupt Result Available)
The RxlRA is set by the 8273 when an interrupt result byte is placed in the RxlNT register. It is reset after the CPU has read the RxlNT register.

Bit 0 TxlRA (Transmitter Interrupt Result Available)
The TxlRA is set by the 8273 when an interrupt result byte is placed in the TxlNT register. It is reset when the CPU has read the TxlNT register.

The Execution Phase
Upon accepting the last parameter, the 8273 enters into the Execution Phase. The execution phase may consist of a DMA or other activity, and may or may not require CPU intervention. The CPU intervention is eliminated in this phase if the system utilizes DMA for the data transfers, otherwise, for non-DMA data transfers, the CPU is interrupted by the 8273 via TxlNT and RxINT pins, for each data byte request.

The Result Phase
During the result phase, the 8273 notifies the CPU of the execution outcome of a command. This phase is initiated by:
1. The successful completion of an operation
2. An error detected during an operation.

To facilitate quick network software decisions, two types of execution results are provided:
1. An Immediate Result
2. A Non-Immediate Result

---

**Figure 10. Rx Interrupt Result Byte Format**

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1</td>
<td>All 8 bits received</td>
<td>1 0 0</td>
<td>D0 received</td>
<td>1 0 0</td>
<td>D1-D2 received</td>
<td>1 1 0</td>
<td>D3-D4 received</td>
</tr>
<tr>
<td>0 0 0</td>
<td>A1 match or general receive</td>
<td>0 0 0</td>
<td>A2 match</td>
<td>0 0 0</td>
<td>CRC error</td>
<td>0 0 0</td>
<td>Idle detect</td>
</tr>
<tr>
<td>1 1 0</td>
<td>Abort detected</td>
<td>1 1 1</td>
<td>EOP detected</td>
<td>1 1 1</td>
<td>Frame less than 32 bits</td>
<td>1 0 0</td>
<td>DMA overrun detected</td>
</tr>
<tr>
<td>0 0 0</td>
<td>Memory buffer overflow</td>
<td>0 0 0</td>
<td>Carrier detect failure</td>
<td>0 0 0</td>
<td>Receive interrupt overrun</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Partial Byte Received

**Figure 11. Tx Interrupt Result Byte Format**

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>Early transmit interrupt</td>
<td>0 1 1</td>
<td>Frame transmit complete</td>
<td>1 1 1</td>
<td>DMA underrun</td>
<td>1 1 1</td>
<td>Clear to Send (CTS) error</td>
</tr>
<tr>
<td>1 0 0</td>
<td>Abort complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Immediate result is provided by the 8273 for commands such as Read Port A and Read Port B which have information (CTS, CD, RTS, etc.) that the network software needs to make quick operational decisions.

A command which cannot provide an immediate result will generate an interrupt to signal the beginning of the Result phase. The immediate results are provided in the Result Register; all non-immediate results are available upon device interrupt, through Tx Interrupt Result Register Txl/R or Rx Interrupt Result Register Rxl/R. The result may consist of a one-byte interrupt code indicating the condition for the interrupt and, if required, one or more bytes which detail the condition.

**Tx and Rx Interrupt Result Registers**

The Result Registers have a result code, the three high order bits D7-D5 of which are set to zero for all but the receive command. This command result contains a count that indicates the number of bits received in the last byte. If a partial byte is received, the high order bits of the last data byte are indeterminate.

All results indicated in the command summary must be read during the result phase.

---

**Figure 12. Result Phase Flowchart—Interrupt Results**
IMMEDIATE RESULTS

AFTER COMMAND PHASE COMPLETION (READ PORT A, PORT B)

START

READ STATUS REGISTER

CRBF >

YES

READ RESULT REGISTER

NO

END

Figure 13. (Rx Interrupt Service)
DETAILED COMMAND DESCRIPTION

General
The 8273 HDLC/SDLC controller supports a comprehensive set of high level commands which allows the 8273 to be readily used in full-duplex, half-duplex, synchronous, asynchronous and SDLC loop configuration, with or without modems. These frame-level commands minimize CPU and software overhead. The 8273 has address and control byte buffers which allow the receive and transmit commands to be used in buffered or non-buffered modes.

In buffered transmit mode, the 8273 transmits a flag automatically, reads the Address and Control buffer registers and transmits the fields, then via DMA, it fetches the information field. The 8273, having transmitted the information field, automatically appends the Frame Check Sequence (FCS) and the end flag. Correspondingly, in buffered read mode, the Address and Control fields are stored in their respective buffer registers and only Information Field is transferred to memory.

In non-buffered transmit mode, the 8273 transmits the beginning flag automatically, then fetches and transmits the Address, Control and Information fields from the memory, appends the FCS character and an end flag. In the non-buffered receive mode the entire contents of a frame are sent to memory with the exception of the flags and FCS.

HDLC Implementation
HDLC Address and Control field are extendable. The extension is selected by setting the low order bit of the field to be extended to a one, a zero in the low order bit indicates the last byte of the respective field.

Since Address/Control field extension is normally done with software to maximize extension flexibility, the 8273 does not create or operate upon contents of the extended HDLC Address/Control fields. Extended fields are transparently passed by the 8273 to user as either interrupt results or data transfer requests. Software must assemble the fields for transmission and interrogate them upon reception.

However, the user can take advantage of the powerful 8273 commands to minimize CPU/Software overhead and simplify buffer management in handling extended fields. For instance buffered mode can be used to separate the first two bytes, then interrogate the others from buffer. Buffered mode is perfect for a two byte address field.

The 8273, when programmed, recognizes protocol characters unique to HDLC such as Abort, which is a string of seven or more ones (0111111). Since Abort character is the same as the GA (EOP) character used in SDLC Loop applications, Loop Transmit and Receive commands are not recommended to be used in HDLC. HDLC does not support Loop mode.

Initialization Set/Reset Commands
These commands are used to manipulate data within the 8273 registers. The Set commands have a single parameter which is a mask that corresponds to the bits to be set. (They perform a logical-OR of the specified register with the mask provided as a parameter). The Register commands have a single parameter which is a mask that has a zero in the bit positions that are to be reset. (They perform a logical-AND of the specified register with the mask).

Set One-Bit Delay (CMD Code 64)

<table>
<thead>
<tr>
<th>CMD</th>
<th>PAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

When one bit delay is set, 8273 retransmits the received data stream one bit delayed. This mode is entered at a receiver character boundary, and should only be used by Loop Stations.

Reset One-Bit Delay (CMD Code 57)

<table>
<thead>
<tr>
<th>CMD</th>
<th>PAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The 8273 stops the one bit delayed retransmission mode.

Set Data Transfer Mode (CMD Code 97)

<table>
<thead>
<tr>
<th>CMD</th>
<th>PAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

When the data transfer mode is set, the 8273 will interrupt when data bytes are required for transmission or are available from a receive. If a transmit interrupt occurs and the status indicates that there is no Transmit Result (TxIRA = 0), the interrupt is a transmit data request. If a receive interrupt occurs and the status indicates that there is no receive result (RxIRA = 0), the interrupt is a receive data request.

Reset Data Transfer Mode (CMD Code 57)

<table>
<thead>
<tr>
<th>CMD</th>
<th>PAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

If the Data Transfer Mode is reset, the 8273 data transfers are performed through the DMA requests without interrupting the CPU.

6-33
Set Operating Mode (CMD Code 91)

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAR:</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

1 = FLAG STREAM MODE
1 = PREFRAME SYNC MODE
1 = BUFFERED MODE
1 = EARLY INTERRUPT MODE
1 = EOP INTERRUPT MODE
1 = HDLC MODE

Reset Operating Mode (CMD Code 51)

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAR:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Any mode switches set in CMD code 91 can be reset using this command by placing zeros in the appropriate positions.

(D5) HDLC Mode

In HDLC mode, a bit sequence of seven ones (0111111) is interpreted as an abort character. Otherwise, eight ones (01111111) signal an abort.

(D4) EOP Interrupt Mode

In EOP interrupt mode, an interrupt is generated whenever an EOP character (01111111) is detected by an active receiver. This mode is useful for the implementation of an SDLC loop controller in detecting the end of a message stream after a loop poll.

(D3) Transmitter Early Interrupt Mode (Tx)

The early interrupt mode is specified to indicate when the 8273 should generate an end of frame interrupt. When set, an early interrupt is generated when the last data character has been passed to the 8273. If the user software responds with another transmit command before the final flag is sent, the final flag interrupt will not be generated and a new frame will immediately begin when the current frame is complete. This permits frames to be separated by a single flag. If no additional Tx commands are provided, a final interrupt will follow.

Note: In buffered mode, if a supervisory frame (no Information) Transmit command is sent in response to an early Transmit interrupt, the 8273 will repeatedly transmit the same supervisory frame with one flag in between, until a non-supervisory transmit is issued.

Early transmitter interrupt can be used in buffered mode by waiting for a transmit complete interrupt instead of early Transmit Interrupt before issuing a transmit frame command for a supervisory frame. See Figure 14.

Figure 14.

If this bit is zero, the interrupt will be generated only after the final flag has been transmitted.

(D2) Buffered Mode

If the buffered mode bit is set to a one, the first two bytes (normally the address (A) and control (C) fields) of a frame are buffered by the 8273. If this bit is a zero the address and control fields are passed to and from memory.

(D1) Preframe Sync Mode

If this bit is set to a one the 8273 will transmit two characters before the first flag of a frame. To guarantee sixteen line transitions, the 8273 sends two bytes of data (00)\textsubscript{n} if NRZI is set or data (55)\textsubscript{n} if NRZI is not set.

(D0) Flag Stream Mode

If this bit is set to a one, the following table outlines the operation of the transmitter.

<table>
<thead>
<tr>
<th>TRANSMITTER STATE</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Send Flags immediately.</td>
</tr>
<tr>
<td>Transmit or Transmit</td>
<td>Send Flags after the transmission complete</td>
</tr>
<tr>
<td>Transparent Active</td>
<td>Ignore command.</td>
</tr>
<tr>
<td>Loop Transmit Active</td>
<td>Ignore command.</td>
</tr>
<tr>
<td>1 Bit Delay Active</td>
<td></td>
</tr>
</tbody>
</table>
If this bit is reset to zero the following table outlines the operation of the transmitter.

<table>
<thead>
<tr>
<th>TRANSMITTER STATE</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>Send Idles on next character boundary.</td>
</tr>
<tr>
<td>Transmit or Transmit-</td>
<td>Send Idles after the transmission is complete.</td>
</tr>
<tr>
<td>Transparent Active</td>
<td></td>
</tr>
<tr>
<td>Loop Transmit Active</td>
<td>Ignore command.</td>
</tr>
<tr>
<td>1 Bit Delay Active</td>
<td>Ignore command.</td>
</tr>
</tbody>
</table>

Set Serial I/O Mode (CMD Code A0)

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

CMD: (01)H
PAR: (00)H

Reset Serial I/O Mode (CMD Code 60)

This command allows bits set in CMD code A0 to be reset by placing zeros in the appropriate positions.

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

CMD: (02)H
PAR: (00)H

(D2) Loop Back
If this bit is set to a one, the transmit data is internally routed to the receive data circuitry.

(D1) TxC → RxC
If this bit is set to a one, the transmit clock is internally routed to the receive clock circuitry. It is normally used with the loop back bit (D2).

(D0) NRZI Mode
If this bit is set to a one, NRZI encoding and decoding of transmit and receive data is provided. If this bit is a zero, the transmit and receive data is treated as a normal positive logic bit stream.

NRZI encoding specifies that a zero causes a change in the polarity of the transmitted signal and a one causes no polarity change. NRZI is used in all asynchronous operations. Refer to IBM document GA27-3093 for details.

Reset Device Command

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

An 8273 reset command is executed by outputting a (01)H followed by (00)H to the reset register (TMR). See 8273 AC timing characteristics for Reset pulse specifications.

The reset command emulates the action of the reset pin.
1. The modem control signals are forced high (inactive level).
2. The 8273 status register flags are cleared.
3. Any commands in progress are terminated immediately.
4. The 8273 enters an idle state until the next command is issued.
5. The Serial I/O and Operating Mode registers are set to zero and DMA data register transfer mode is selected.
6. The device assumes a non-loop SDLC terminal role.

Receive Commands

The 8273 supports three receive commands: General Receive, Selective Receive, and Selective Loop Receive.

General Receive (CMD Code C0)

General receive is a receive mode in which frames are received regardless of the contents of the address field.

Selective Receive (CMD Code C1)

The intervening character is not preceded by a flag and is ignored. If this bit is set to a zero, the receive clock is internally routed to the receive data circuitry. If this bit is set to a one, the transmit data is internally routed to the receive data circuitry.
Selective receive is a receive mode in which frames are ignored unless the address field matches any one of two address fields given to the 8273 as parameters. When selective receive is used in HDLC the 8273 looks at the first character, if extended, software must then decide if the message is for this unit.

Selective Loop Receive (CMD Code C2)

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Receive Disable (CMD Code C5)

Terminates an active receive command immediately.

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Transmit Commands

The 8273 supports three transmit commands: Transmit Frame, Loop Transmit, Transmit Transparent.

Transmit Frame (CMD Code C8)

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Loop Transmit (CMD Code CA)

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Receive Disable (CMD Code C5)

Terminates an active receive command immediately.

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Transmit Commands

The 8273 supports three transmit commands: Transmit Frame, Loop Transmit, Transmit Transparent.

Transmit Frame (CMD Code C8)

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Transmits one frame including: initial flag, frame check sequence, and the final flag.

If the buffered mode is specified, the L0, L1, frame length provided as a parameter is the length of the information field and the address and control fields must be input.

In unbuffered mode the frame length provided must be the length of the information field plus two and the address and control fields must be the first two bytes of data. Thus only the frame length bytes are required as parameters.

The 8273 will transmit a block of raw data without protocol, i.e., no zero bit insertion, flags, or frame check sequences.

Abort Transmit Commands

An abort command is supported for each type of transmit command. The abort commands are ignored if a transmit command is not in progress.

Abort Transmit Frame (CMD Code CC)

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

After an abort character (eight contiguous ones) is transmitted, the transmitter reverts to sending flags or idles as a function of the flag stream mode specified.

Abort Loop Transmit (CMD Code CE)

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

After a flag is transmitted the transmitter reverts to one bit delay mode.

Abort Transmit Transparent (CMD Code CD)

<table>
<thead>
<tr>
<th>CMD:</th>
<th>A1</th>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The transmitter reverts to sending flags or idles as a function of the flag stream mode specified.
Modem Control Commands

The modem control commands are used to manipulate the modem control ports.

When read Port A or Port B commands are executed the result of the command is returned in the result register. The Bit Set Port B command requires a parameter that is a mask that corresponds to the bits to be set. The Bit Reset Port B command requires a mask that has a zero in the bit positions that are to be reset.

Read Port A (CMD Code 22)

Command Description: Read Port A
Command (HEX): 22
Parameter: None
Results: None
Result Port: None
Completion Interrupt: No

(D5) Flag Detect
This bit can be used to set the flag detect pin. However, it will be reset when the next flag is detected.

(D4-D1) User Defined Outputs
These bits correspond to the state of the PB4-PB1 output pins.

(D0) Request to Send
This is a dedicated 8273 modem control signal, and reflects the same logical state of RTS pin.

Read Port B (CMD Code 23)

Command Description: Read Port B
Command (HEX): 23
Parameter: None
Results: None
Result Port: None
Completion Interrupt: No

Set Port B Bits (CMD Code A3)

This command allows user defined Port B pins to be set.

Command Description: Set Port B Bits
Command (HEX): A3
Parameter: Set Mask
Results: None
Result Port: None
Completion Interrupt: No

(D5) Flag Detect
This bit can be used to set the flag detect pin. However, it will be reset when the next flag is detected.

(D4-D1) User Defined Outputs
These bits correspond to the state of the PB4-PB1 output pins.

(D0) Request to Send
This is a dedicated 8273 modem control signal, and reflects the same logical state of RTS pin.

8273 Command Summary

<table>
<thead>
<tr>
<th>Command Description</th>
<th>Command (HEX)</th>
<th>Parameter</th>
<th>Results</th>
<th>Result Port</th>
<th>Completion Interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set One Bit Delay</td>
<td>A4</td>
<td>Set Mask</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>Reset One Bit Delay</td>
<td>64</td>
<td>Reset Mask</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>Set Data Transfer Mode</td>
<td>97</td>
<td>Set Mask</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>Reset Data Transfer Mode</td>
<td>57</td>
<td>Reset Mask</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>Set Operating Mode</td>
<td>91</td>
<td>Set Mask</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>Reset Operating Mode</td>
<td>51</td>
<td>Reset Mask</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>Set Serial I/O Mode</td>
<td>A0</td>
<td>Set Mask</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>Reset Serial I/O Mode</td>
<td>60</td>
<td>Reset Mask</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>General Receive</td>
<td>C0</td>
<td>B0,B1</td>
<td>RIC,R0,R1,(A,C)²</td>
<td>RXI/R</td>
<td>Yes</td>
</tr>
<tr>
<td>Selective Receive</td>
<td>C1</td>
<td>B0,B1,A1,A2</td>
<td>RIC,R0,R1,(A,C)²</td>
<td>RXI/R</td>
<td>Yes</td>
</tr>
<tr>
<td>Selective Loop Receive</td>
<td>C2</td>
<td>B0,B1,A1,A2</td>
<td>RIC,R0,R1,(A,C)²</td>
<td>RXI/R</td>
<td>Yes</td>
</tr>
<tr>
<td>Receive Disable</td>
<td>C5</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>Transmit Frame</td>
<td>C8</td>
<td>L0,L1,(A,C)¹</td>
<td>TIC</td>
<td>TXI/R</td>
<td>Yes</td>
</tr>
<tr>
<td>Loop Transmit</td>
<td>C9</td>
<td>L0,L1,(A,C)¹</td>
<td>TIC</td>
<td>TXI/R</td>
<td>Yes</td>
</tr>
<tr>
<td>Transmit Transparent</td>
<td>CC</td>
<td>None</td>
<td>TIC</td>
<td>TXI/R</td>
<td>Yes</td>
</tr>
<tr>
<td>Abort Transmit Frame</td>
<td>CE</td>
<td>None</td>
<td>TIC</td>
<td>TXI/R</td>
<td>Yes</td>
</tr>
<tr>
<td>Abort Loop Transmit</td>
<td>CD</td>
<td>None</td>
<td>TIC</td>
<td>TXI/R</td>
<td>Yes</td>
</tr>
<tr>
<td>Read Port A</td>
<td>22</td>
<td>None</td>
<td>Port Value</td>
<td>Result</td>
<td>No</td>
</tr>
<tr>
<td>Read Port B</td>
<td>23</td>
<td>None</td>
<td>Port Value</td>
<td>Result</td>
<td>No</td>
</tr>
<tr>
<td>Set Port B Bit</td>
<td>A3</td>
<td>Set Mask</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>Reset Port B Bit</td>
<td>63</td>
<td>Reset Mask</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
</tbody>
</table>

NOTES:
1. Issued only when in buffered mode.
2. Read as results only in buffered mode.
8273 Command Summary Key

**B0** — Least significant byte of the receive buffer length.

**B1** — Most significant byte of the receive buffer length.

**L0** — Least significant byte of the Tx frame length.

**L1** — Most significant byte of the Tx frame length.

**A1** — Receive frame address match field one.

**A2** — Receive frame address match field two.

**A** — Address field of received frame. If non-buffered mode is specified, this result is not provided.

**C** — Control field of received frame. If non-buffered mode is specified this result is not provided.

**RXI/R** — Receive interrupt result register.

**TXI/R** — Transmit interrupt result register.

**R0** — Least significant byte of the length of the frame received.

**R1** — Most significant byte of the length of the frame received.

**RIC** — Receiver interrupt result code.

**TIC** — Transmitter interrupt result code.

---

**NOTE:**
In order to ensure proper operation to the maximum baud rate, Receive commands or Read/Write Port commands should be written only when either the transmitter or the receiver is inactive. In full duplex systems, it is recommended that these commands be issued after servicing a transmitter interrupt but before a new transmit command is issued.
Figure 16a. Typical Frame Transmission, Buffered Mode

Figure 16b. Typical Frame Transmission, Non-Buffered Mode

Figure 17. 8273 System Diagram
Table 2. Command Phase Timing (Full Duplex)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Timing Parameter</th>
<th>Buffered</th>
<th></th>
<th>Non-Buffered</th>
<th></th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Between command &amp; first parameter</td>
<td>13 tcy</td>
<td>756 tcy</td>
<td>13 tcy</td>
<td>857 tcy</td>
<td>tcy</td>
</tr>
<tr>
<td>T2</td>
<td>Between consecutive parameters</td>
<td>10 tcy</td>
<td>604 tcy</td>
<td>10 tcy</td>
<td>705 tcy</td>
<td>tcy</td>
</tr>
<tr>
<td>T3</td>
<td>Command Parameter Buffer full bit Reset after Parameter loaded</td>
<td>10 tcy</td>
<td>604 tcy</td>
<td>10 tcy</td>
<td>705 tcy</td>
<td>tcy</td>
</tr>
<tr>
<td>T4</td>
<td>Command busy bit reset after last parameter</td>
<td>128 tcy</td>
<td>702 tcy</td>
<td>128 tcy</td>
<td>803 tcy</td>
<td>tcy</td>
</tr>
<tr>
<td>T5</td>
<td>CPBF bit reset after last parameter</td>
<td>10 tcy</td>
<td>604 tcy</td>
<td>10 tcy</td>
<td>705 tcy</td>
<td>tcy</td>
</tr>
</tbody>
</table>
WAVEFORMS (Continued)

RECEIVER INTERRUPT

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Timing Parameter (clock cycles)</th>
<th>Buffer Min.</th>
<th>Buffer Max.</th>
<th>Non-Buffer Min.</th>
<th>Non-Buffer Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>RxIRA bit set after RIC read</td>
<td>18</td>
<td>29</td>
<td>18</td>
<td>29</td>
<td>tcy</td>
</tr>
<tr>
<td>T2</td>
<td>RxINT goes away after last Int. Result read</td>
<td>16</td>
<td>27</td>
<td>16</td>
<td>27</td>
<td>tcy</td>
</tr>
</tbody>
</table>
WAVEFORMS (Continued)

TRANSMIT INTERRUPT

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Timing (Clock Cycle)</th>
<th>Buffered</th>
<th>Non-Buffered</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
</tr>
<tr>
<td>T1</td>
<td>TxIINT inactive after Int. Results read</td>
<td>13</td>
<td>353</td>
<td>13</td>
</tr>
</tbody>
</table>
**ABSOLUTE MAXIMUM RATINGS**

Ambient Temperature Under Bias .......... 0°C to 70°C
Storage Temperature ...................... -65°C to +150°C
Voltage on Any Pin With
Respect to Ground ......................... -0.5V to +7V
Power Dissipation .......................... 1 Watt

*NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. CHARACTERISTICS (8273, 8273-4)**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>VCC + 0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td></td>
<td>V</td>
<td>IOL = 2.0 mA for Data Bus Pins</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td></td>
<td>V</td>
<td>IOL = 1.0 mA for Output Port Pins</td>
</tr>
<tr>
<td>IL</td>
<td>Input Load Current</td>
<td>±10</td>
<td></td>
<td>μA</td>
<td>VIN = VCC to 0V</td>
</tr>
<tr>
<td>IOFL</td>
<td>Output Leakage Current</td>
<td>±10</td>
<td></td>
<td>μA</td>
<td>VOUT = VCC to 0.45V</td>
</tr>
<tr>
<td>ICC</td>
<td>VCC Supply Current</td>
<td>180</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
</tbody>
</table>

**CAPACITANCE (8273, 8273-4)**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIN</td>
<td>Input Capacitance</td>
<td></td>
<td>10 pF</td>
<td></td>
<td>pF</td>
<td>t_c = 1 MHz</td>
</tr>
<tr>
<td>CII0</td>
<td>I/O Capacitance</td>
<td></td>
<td>20 pF</td>
<td></td>
<td>pF</td>
<td>Unmeasured Pins Returned to GND</td>
</tr>
</tbody>
</table>

**A.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = +5.0V ± 5%)**

**CLOCK TIMING (8273)**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCY</td>
<td>Clock</td>
<td>250</td>
<td></td>
<td>1000 ns</td>
<td></td>
<td>64K Baud Max Operating Rate</td>
</tr>
<tr>
<td>tCL</td>
<td>Clock Low</td>
<td>120</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tCH</td>
<td>Clock High</td>
<td>120</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

**CLOCK TIMING (8273-4)**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCY</td>
<td>Clock</td>
<td>286</td>
<td></td>
<td>1000 ns</td>
<td></td>
<td>56K Baud Max Operating Rate</td>
</tr>
<tr>
<td>tCL</td>
<td>Clock Low</td>
<td>135</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tCH</td>
<td>Clock High</td>
<td>135</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>
# A.C. Characteristics (8273, 8273-4)

\( T_A = 0^\circ C \) to \( 70^\circ C \), \( V_{CC} = +5.0V \pm 5\% \)

## Read Cycle

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{AC} )</td>
<td>Select Setup to RD</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td>Note 2</td>
</tr>
<tr>
<td>( t_{CA} )</td>
<td>Select Hold from RD</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td>Note 2</td>
</tr>
<tr>
<td>( t_{RR} )</td>
<td>RD Pulse Width</td>
<td>250</td>
<td>250</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{AD} )</td>
<td>Data Delay from Address</td>
<td>300</td>
<td></td>
<td>ns</td>
<td>Note 2</td>
</tr>
<tr>
<td>( t_{RD} )</td>
<td>Data Delay from RD</td>
<td>200</td>
<td></td>
<td>ns</td>
<td>( C_L = 150 \text{pF} ), Note 2</td>
</tr>
<tr>
<td>( t_{DF} )</td>
<td>Output Float Delay</td>
<td>20</td>
<td>100</td>
<td>ns</td>
<td>( C_L = 20 \text{pF} ) for Minimum; 150 pF for Maximum</td>
</tr>
<tr>
<td>( t_{DC} )</td>
<td>DACK Setup to RD</td>
<td>25</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{CD} )</td>
<td>DACK Hold from RD</td>
<td>25</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{KD} )</td>
<td>Data Delay from DACK</td>
<td>300</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

## Write Cycle

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{AC} )</td>
<td>Select Setup to WR</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{CA} )</td>
<td>Select Hold from WR</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{WW} )</td>
<td>WR Pulse Width</td>
<td>250</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{DW} )</td>
<td>Data Setup to WR</td>
<td>150</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{WD} )</td>
<td>Data Hold from WR</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{DC} )</td>
<td>DACK Setup to WR</td>
<td>25</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{CD} )</td>
<td>DACK Hold from WR</td>
<td>25</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

## DMA

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{QC} )</td>
<td>Request Hold from WR or RD (for Non-Burst Mode)</td>
<td>200</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

## Other Timing

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{RSTW} )</td>
<td>Reset Pulse Width</td>
<td>10</td>
<td></td>
<td>( t_{CY} )</td>
<td></td>
</tr>
<tr>
<td>( t_{r} )</td>
<td>Input Signal Rise Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{i} )</td>
<td>Input Signal Fall Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{RSTS} )</td>
<td>Reset to First IOWR</td>
<td>2</td>
<td></td>
<td>( t_{CY} )</td>
<td></td>
</tr>
<tr>
<td>( t_{CY32} )</td>
<td>32X Clock Cycle Time</td>
<td>13.02 ( t_{CY} )</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{OL32} )</td>
<td>32X Clock Low Time</td>
<td>4 ( t_{CY} )</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{OH32} )</td>
<td>32X Clock High Time</td>
<td>4 ( t_{CY} )</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{DPLL} )</td>
<td>DPLL Output Low</td>
<td>1 ( t_{CY} ) - 50</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{DCL} )</td>
<td>Data Clock Low</td>
<td>1 ( t_{CY} ) - 50</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{OCH} )</td>
<td>Data Clock High</td>
<td>2 ( t_{CY} )</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{DCY} )</td>
<td>Data Clock</td>
<td>62.5 ( t_{CY} )</td>
<td></td>
<td>ns</td>
<td>Note 3</td>
</tr>
<tr>
<td>( t_{TD} )</td>
<td>Transmit Data Delay</td>
<td>200</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{DS} )</td>
<td>Data Setup Time</td>
<td>200</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{DH} )</td>
<td>Data Hold Time</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{FLD} )</td>
<td>FLAG DET Output Low</td>
<td>8 ( t_{CY} ) ± 50</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**

1. All timing measurements are made at the reference voltages unless otherwise specified: Input "1" at 2.0V, "0" at 0.8V; Output "1" at 2.0V, "0" at 0.8V.
2. \( t_{AD}, t_{RD}, t_{AC}, \) and \( t_{CA} \) are not concurrent specs.
3. If receive commands or Read/Write Port commands are issued while both the transmitter and receiver are active, this specification will be \( 81.5 \text{t}_{CY} \) min.
A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

\[ C_L = 150 \text{ pF} \]

\[ C_L \text{ includes JIG capacitance} \]

WAVEFORMS

READ

WRITE
WAVEFORMS (Continued)

DMA

DRO

DACK

RD OR WR

CHIP CLOCK

32X CLOCK

TRANSMIT

TxC

TxD

RECEIVE

RxC

RxD
WAVEFORMS (Continued)

**DPLL OUTPUT**

![DPLL output waveform diagram]

**FLAG DETECT OUTPUT**

![Flag detect output waveform diagram]
The Intel® 8274 Multi-Protocol Series Controller (MPSC) is designed to interface High Speed Communications Lines using Asynchronous, IBM Bisync, and SDLC/HDLC protocol to Intel microcomputer systems. It can be interfaced with Intel's MCS-48, -85, -51; iAPX-86, -88, -166 and -188 families, the 8237 DMA Controller, or the 8089 I/O Processor in polled, interrupt driven, or DMA driven modes of operation.

The MPSC is a 40 pin device fabricated using Intel's High Performance HMOS Technology.
# Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>1</td>
<td>I</td>
<td>Clock: System clock, TTL compatible.</td>
</tr>
<tr>
<td>RESET</td>
<td>2</td>
<td>I</td>
<td>Reset: A low signal on this pin will force the MPSC to an idle state. TXD and RXD are forced high. The modern interface output signals are forced high. The MPSC will remain idle until the control registers are initialized. Reset must be true for one complete CLK cycle.</td>
</tr>
<tr>
<td>CDa</td>
<td>3</td>
<td>I</td>
<td>Carrier Detect (Channel A): This interface signal is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RXD line. If the auto enable control is set the 8274 will not enable the serial receiver until CDa has been activated.</td>
</tr>
<tr>
<td>RXCn</td>
<td>4</td>
<td>I</td>
<td>Receive Clock (Channel B): The serial data are shifted into the Receive Data input (RXD) on the rising edge of the Receive Clock.</td>
</tr>
<tr>
<td>CDn</td>
<td>5</td>
<td>I</td>
<td>Carrier Detect (Channel B): This interface signal is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RXD line. If the auto enable control is set the 8274 will not enable the serial receiver until CDn has been activated.</td>
</tr>
<tr>
<td>CTSn</td>
<td>6</td>
<td>I</td>
<td>Clear to Send (Channel B): This interface signal is supplied by the modem in response to a request to send a data signal. CTS indicates that the data terminal/computer equipment is permitted to transmit data. In addition, if the auto enable control is set, the 8274 will not transmit data bytes until CTS has been activated.</td>
</tr>
<tr>
<td>TXCn</td>
<td>7</td>
<td>I</td>
<td>Transmit Clock (Channel B): The serial data are shifted out from the Transmit Data output (TXD) on the falling edge of the Transmit Clock.</td>
</tr>
<tr>
<td>TXDn</td>
<td>8</td>
<td>O</td>
<td>Transmit Data (Channel B): This pin transmits serial data to the communications channel (Channel B).</td>
</tr>
<tr>
<td>RXDn</td>
<td>9</td>
<td>I</td>
<td>Receive Data (Channel B): This pin receives serial data from the communications channel (Channel B).</td>
</tr>
<tr>
<td>SYNDET/RSTS</td>
<td>10</td>
<td>I/O</td>
<td>Synchronous Detection (Channel B): This pin is used in byte synchronous mode as either an internal sync detect (output) or as a means to force external synchronization (input) in SDLC mode, this pin is an output indicating flag detection. In asynchronous mode it is a general purpose input (Channel B). Request to Send (Channel B): General purpose output, generally used to send data SYNDET or RSTS selection is done by WR2; D7 (Channel A).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDY8/</td>
<td>11</td>
<td>O</td>
<td>Ready (Channel B)/Transmitter DMA Request (Channel A): In mode 0 this pin is RDY8 and is used to synchronize data transfers between the processor and the MPSC (Channel B). In modes 1 and 2 this pin is TXDRQ8, and is used by the channel A transmitter to request a DMA transfer.</td>
</tr>
<tr>
<td>TXDRQ8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB7</td>
<td>12</td>
<td>I/O</td>
<td>Data Bus: The Data Bus lines are bidirectional three state lines which interface with the system's Data Bus.</td>
</tr>
<tr>
<td>DB6</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB5</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB4</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB3</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB1</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB0</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCC</td>
<td>20</td>
<td></td>
<td>Power: 5V Supply.</td>
</tr>
<tr>
<td>CTSn</td>
<td>39</td>
<td>I</td>
<td>Clear to Send (Channel A): This interface signal is supplied by the modem in response to an active RTS signal. CTS indicates that the data terminal/computer equipment is permitted to transmit data. In addition, if the auto enable control is set, the 8274 will not transmit data bytes until CTS has been activated.</td>
</tr>
<tr>
<td>RTSn</td>
<td>38</td>
<td>O</td>
<td>Request To Send (Channel A): general purpose output commonly used to signal that Channel A is ready to send data.</td>
</tr>
<tr>
<td>TXDA</td>
<td>37</td>
<td>O</td>
<td>Transmit Data (Channel A): This pin transmits serial data to the communications channel (Channel A).</td>
</tr>
<tr>
<td>TXCA</td>
<td>36</td>
<td>I</td>
<td>Transmit Clock (Channel A): The serial data are shifted out from the Transmit Data output (TXD) on the falling edge of the Transmit Clock.</td>
</tr>
<tr>
<td>RXCA</td>
<td>35</td>
<td>I</td>
<td>Receive Clock (Channel A): The serial data are shifted into the Receive Data input (RXD) on the rising edge of the Receive Clock.</td>
</tr>
<tr>
<td>RXDn</td>
<td>34</td>
<td>I</td>
<td>Receive Data (Channel A): This pin receives serial data from the communications channel (Channel A).</td>
</tr>
<tr>
<td>SYNDET</td>
<td>33</td>
<td>I/O</td>
<td>Synchronous Detection (Channel A): This pin is used in byte synchronous mode as either an internal sync detect (output) or as a means to force external synchronization (input) in SDLC mode, this pin is an output indicating flag detection. In asynchronous mode it is a general purpose input (Channel A).</td>
</tr>
<tr>
<td>RDY8/</td>
<td>32</td>
<td>O</td>
<td>Ready: In mode 0 this pin is RDY8 and is used to synchronize data transfers between the processor and the MPSC (Channel A). In modes 1 and 2 this pin is RXDRQ8 and is used by the channel A receiver to request a DMA transfer.</td>
</tr>
<tr>
<td>DTRn</td>
<td>31</td>
<td>O</td>
<td>Data Terminal Ready (Channel A): General purpose output.</td>
</tr>
</tbody>
</table>
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRQ/TxDRQa</td>
<td>30</td>
<td>O</td>
<td>Interrupt Priority Out/Transmitter DMA Request (Channel B): In modes 0 and 1, this pin is Interrupt Priority Out. It is used to establish a hardware interrupt priority scheme with IRQ. It is low only if IRQ is low and the controlling processor is not servicing an interrupt from this MPSC. In mode 2 it is TxDRQa and is used to request a DMA cycle for a transmit operation (Channel B).</td>
</tr>
<tr>
<td>TI/RxDRQb</td>
<td>29</td>
<td>I/O</td>
<td>Interrupt Priority In/Receiver DMA Request (Channel B): In modes 0 and 1, TI is Interrupt Priority In. A low on TI means that no higher priority device is being serviced by the controlling processor's interrupt service routine. In mode 2 this pin is RxDRQb and is used to request a DMA cycle for a receive operation (Channel B). In Interrupt mode, this pin must be tied low.</td>
</tr>
<tr>
<td>INT</td>
<td>28</td>
<td>O</td>
<td>Interrupt: The interrupt signal indicates that the highest priority internal interrupt requires service (open collector). Priority can be resolved via an external interrupt controller or a daisy-chain scheme.</td>
</tr>
</tbody>
</table>

RESET

When the 8274 RESET line is activated, both MPSC channels enter the idle state. The serial output lines are forced to the marking state (high) and the modem interface signals (RTS, DTR) are forced high. In addition, the pointers registers are set to zero.

GENERAL DESCRIPTION

The Intel 8274 Multi-Protocol Serial Controller is a microcomputer peripheral device which supports Asynchronous, Byte Synchronous (Monosync, IBM Bisync), and Bit Synchronous (ISO's HDLC, IBM's SDLV) protocols. This controller's flexible architecture allows easy implementation of many variations of these three protocols with low software and hardware overhead.

The Multi-Protocol Serial controller (MPSC) implements two independent serial receiver/transmitter channels.

The MPSC supports several microprocessor interface options: Polled, Wait, Interrupt driven and DMA driven. The MPSC is designed to support INTEL'S MCS-85 and IAPX 86, 88, 186, 188 families.

FUNCTIONAL DESCRIPTION

Additional information on Asynchronous and Synchronous Communications with the 8274 is available respectively in the Applications Notes AP 134 and AP 145.

Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers for each channel, 2 readable registers for Channel A and 3 readable registers for Channel B).

In the following discussion, the writable registers will be referred to as WRO through WR7 and the readable registers will be referred to as RRO through RR2.

This section of the data sheet describes how the Asynchronous and Synchronous protocols are implemented in the MPSC. It describes general considerations, transmit operation, and receive operation for Asynchronous, Byte Synchronous, and Bit Synchronous protocols.
ASYNCHRONOUS OPERATIONS

TRANSMITTER/RECEIVER INITIALIZATION

(See Detailed Command Description Section for complete information)

In order to operate in asynchronous mode, each MPSC channel must be initialized with the following information:

1. Transmit/Receive Clock Rate. This parameter is specified by bits 6 and 7 of WR4. The clock rate may be set to 1, 16, 32, or 64 times the data-link bit rate. If the X1 clock mode is selected, the bit synchronization must be accomplished externally.

2. Number of Stop Bits. This parameter is specified by bits 2 and 3 of WR4. The number of stop bits may be set to 1, 1 1/2, or 2.

3. Parity Selection. Parity may be set for odd, even, or no parity by bits 0 and 1 of WR4.

4. Receiver Character Length. This parameter sets the length of received characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 6 and 7 of WR3.

5. Receiver Enable. The serial-channel receiver operation may be enabled or disabled by setting or clearing bit 0 of WR3.

6. Transmitter Character Length. This parameter sets the length of transmitted characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 5 and 6 of WR5. Characters of less than 5 bits in length may be transmitted by setting the transmitted length to five bits (set bits 5 and 6 of WR5 to 0).

The MPSC then determines the actual number of bits to be transmitted from the character data byte. The bits to be transmitted must be right justified in the data byte, the next three bits must be set to 0 and all remaining bits must be set to 1. The following table illustrates the data formats for transmission of 1 to 5 bits of data:

<table>
<thead>
<tr>
<th>D7 D6 D5 D4 D3 D2 D1 D0</th>
<th>Number of Bits Transmitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 0 0 0 c</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1 0 0 0 c c</td>
<td>2</td>
</tr>
<tr>
<td>1 1 0 0 0 c c c</td>
<td>3</td>
</tr>
<tr>
<td>1 0 0 0 c c c c</td>
<td>4</td>
</tr>
<tr>
<td>0 0 0 c c c c c</td>
<td>5</td>
</tr>
</tbody>
</table>

( Character Length )

7. Transmitter Enable. The serial channel transmitter operation may be enabled or disabled by setting or clearing bit 3 of WR5

8. Interrupt Mode.

For data transmission via a modem or RS-232-C interface, the following information must also be specified:

1. The Request To Send (RTS) (WR5; D1) and Data Terminal Ready (DTR) (WR5; D7) bits must be set along with the Transmit Enable bit (WR5; D3).

2. Auto Enable may be set to allow the MPSC to automatically enable the channel transmitter when the clear-to-send signal is active and to automatically enable the receiver when the carrier-detect signal is active. However, the Transmit Enable bit (WR3; D3) and Receive Enable bit (WR3; D1) must be set in order to use the Auto Enable mode. Auto Enable is controlled by bit 5 of WR3.

When loading initialization parameters into the MPSC, WR4 information must be written before the WR1, WR3, WR5 parameters commands.

During initialization, it is desirable to guarantee that the external/status latches reflect the latest interface information. Since up to two state changes are internally stored by the MPSC, at least two Reset External/Status Interrupt commands must be issued. This procedure is most easily accomplished by simply issuing this reset command whenever the pointer register is set during initialization.

An MPSC initialization procedure (MPSC$RX$INIT) for asynchronous communication is listed in Intel Application Note AP 134.

TRANSMIT

The transmit function begins when the Transmit Enable bit (WR5; D3) is set. The MPSC automatically adds the start bit, the programmed parity bit (odd, even or no parity) and the programmed number of stop bits (1, 1.5 or 2 bits) to the data character being transmitted. 1.5 stop bits option must be used with X16, X32 or X64 clock options only.
The serial data are shifted out from the Transmit Data (TxD) output on the falling edge of the Transmit Clock (TxC) input at a rate programmable to 1, 1/16th, 1/32nd, or 1/64th of the clock rate supplied to the TxC input.

The TxD output is held high when the transmitter has no data to send, unless, under program control, the Send Break (WR5; D4) command is issued to hold the TxD low.

If the External/Status Interrupt bit (WR1; D0) is set, the status of CTS, CTS and SYNDET are monitored and, if any changes occur for a period of time greater than the minimum specified pulse width, an interrupt is generated. CTS is usually monitored using this interrupt feature (e.g. Auto Enable option).

The Transmit Buffer Empty bit (RRO; D2) is set by the MPSC when the data byte from the buffer is loaded in the transmit shift register. Data should be written to the MPSC only when the Tx buffer becomes empty to prevent overwriting.

**Receive**

The receive function begins when the Receive Enable (WR3; D0) bit is set. If the Auto Enable (WR3: D5) option is selected, then Carrier Detect (CD) must also be low. A valid start bit is detected if a low persists for at least 1/2 bit time on the Receive Data (RxD) input.

The data is sampled at mid-bit time, on the rising edge of RxC, until the entire character is assembled. The receiver inserts 1's when a character is less than 8 bits. If parity (WR4; D0) is enabled and the character is less than 8 bits the parity bit is not stripped from the character.

**Error Reporting**

The receiver also stores error status for each of the 3 data characters in the data buffer. Three error conditions may be encountered during data reception in the asynchronous mode:

1. **Parity.** If parity bits are computed and transmitted with each character and the MPSC is set to check parity (bit 0 in WR4 is set), a parity error will occur whenever the number of “1” bits within the character (including the parity bit) does not match the odd/even setting of the parity check flag (bit 1 in WR4). When a parity error is detected, the parity error flag (RR1; D4) is set and remains set until it is reset by the Error Reset command (WR0; D5, D4, D3).

2. **Framing.** A framing error will occur if a stop bit is not detected immediately following the parity bit (if parity checking is enabled) or immediately following the most-significant data bit (if parity checking is not enabled). When a Framing Error is detected, the Framing Error bit (RR1; D6) is set. The detection of a Framing Error adds an additional 1/2 bit time to the character time so the Framing Error is not interpreted as a new start bit.

3. **Overrun.** If the CPU fails to read a data character while more than three characters have been received, the Receive Overrun bit (RR1; D5) is set. When this occurs, the fourth character assembled replaces the third character in the receive buffers. Only the overwritten character is flagged with the Receive Overrun bit. The Receive Overrun bit (RR1, D5) is reset by the Error Reset command (WR0; D5, D4, D3).

**External/Status Latches**

The MPSC continuously monitors the state of five external/status conditions:

1. **CTS** — clear-to-send input pin.
2. **CD** — carrier-detect input pin.
3. **SYNDET** — sync-detect input pin. This pin may be used as a general-purpose input in the asynchronous communication mode.
4. **BREAK** — a break condition (series of space bits on the receiver input pin).
5. **Tx UNDERRUN/EOM** — Transmitter Underrun/End of Message.

A change of state in any of these monitored conditions will cause the associated status bit in RR0 to be latched (and optionally cause an interrupt).

If the External/Status Interrupt bit (WR1; D0) is enabled, Break Detect (RR0; D7) and Carrier Detect (RR0; D3) will cause an interrupt. Reset External/Status interrupts (WR0; D5, D4, D3) will clear Break Detect and Carrier Detect bits if they are set.
8274

Asynchronous Mode Register Setup

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WR3</td>
<td></td>
<td>AUTO ENABLE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Rx ENABLE</td>
</tr>
<tr>
<td></td>
<td>00 Rx 5 b/char</td>
<td></td>
<td>01 Rx 5 b/char</td>
<td>10 Rx 6 b/char</td>
<td>11 Rx 8 b/char</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| WR4|     | 00 X1 Clock  | 0  | 0  | 0  | 0  | EVEN/ ODD PARITY |
|    | 01 X16 Clock|              | 10 X32 Clock| 11 X64 Clock|     | PARITY ENABLE |

| WR5| DTR | 00 Tx\(\leq\)5 b/char| 01 Tx 7 b/char| 10 Tx 6 b/char| 11 Tx 8 b/char| SEND BREAK| Tx ENABLE| 0| RTS| 0 |

SYNCHRONOUS OPERATION—MONOSYNC, BISYNC

General

The MPSC must be initialized with the following parameters: odd or even parity (WR4; D1, D0), X1 clock mode (WR4; D7, D6), 8- or 16-bit sync character (WR4; D5, D4), CRC polynomial (WR5; D2), Transmitter Enable (WR5; D3), interrupt modes (WR1, WR2), transmit character length (WR5; D6, D5) and receive character length (WR3; D7, D6). WR4 parameters must be written before WR1, WR3, WR5, WR6 and WR7.

The data is transmitted on the falling edge of the Transmit Clock, (TxC) and is received on the rising edge of Receive Clock (RxC). The X1 clock is used for both transmit and receive operations for all three sync modes: Mono, Bi and External.

Transmit Set-Up—Monosync, Bisync

Transmit data is held high after channel reset, or if the transmitter is not enabled. A break may be programmed to generate a spacing line that begins as soon as the Send Break (WR5; D4) bit is set. With the transmitter fully initialized and enabled, the default condition is continuous transmission of the 8- or 16-bit sync character.

Using interrupts for data transfer requires that the Transmit Interrupt/DMA Enable bit (WR1; D1) be set. An interrupt is generated each time the transmit buffer becomes empty. The interrupt can be satisfied

Synchronous Mode Register Setup—Monosync, Bisync

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WR3</td>
<td></td>
<td>AUTO ENABLE</td>
<td>ENTER HUNT MODE</td>
<td>Rx CRC ENABLE</td>
<td>0</td>
<td>SYNC CHAR LOAD INHIBIT</td>
<td>Rx ENABLE</td>
</tr>
<tr>
<td></td>
<td>00 Rx 5 b/char</td>
<td></td>
<td>01 Rx 7 b/char</td>
<td>10 Rx 6 b/char</td>
<td>11 Rx 8 b/char</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| WR4|     | 00 8 bit Sync| 01 16 bit Sync| 11 Ext Sync| 0  | 0  | EVEN/ ODD PARITY |
|    | 0  |              |              |            | 0  | 0  | PARITY ENABLE |

| WR5| DTR | 00 Tx\(\leq\)5 b/char| 01 Tx 7 b/char| 10 Tx 6 b/char| 11 Tx 8 b/char| SEND BREAK| Tx ENABLE| 1 (SELECTS CRC-16)| RTS| Tx CRC ENABLE |
Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers for each channel, 2 readable registers for Channel A and 3 readable registers for Channel B). They are all accessed via the command ports.

An internal pointer register selects which of the command or status registers will be read or written during a command/status access of an MPSC channel.

After reset, the contents of the pointer registers are zero. The first write to a command register causes the data to be loaded into Write Register 0 (WR0). The three least significant bits of WR0 are loaded into the Command/Status Pointer. The next read or write operation accesses the read or write register selected by the pointer. The pointer is reset after the read or write operation is completed.
either by writing another character into the transmitter or by resetting the Transmitter interrupt/DMA Pending latch with a Reset Transmitter interrupt/DMA Pending Command (WR0; D5, D4, D3). If nothing more is written into the transmitter, there can be no further Transmit Buffer Empty interrupt, but this situation does cause a Transmit Underrun condition (RRO; D6).

Data Transfers using the RDY signal are for software controlled data transfers such as block moves. RDY tells the CPU that the MPSC is not ready to accept/provide data and that the CPU must extend the output/input cycle. DMA data transfers use the TxDRQ A/B signals which indicate that the transmit buffer is empty, and that the MPSC is ready to accept the next data character. If the data character is not loaded into the MPSC by the time the transmit shift register is empty, the MPSC enters the Transmit Underrun condition.

The MPSC has two programmable options for solving the transmit underrun condition: it can insert sync characters, or it can send the CRC characters generated so far, followed by sync characters. Following a chip or channel reset, the Transmit Underrun/EOM status bit (RRO; D6) is in a set condition allowing the insertion of sync characters when there is no data to send. The CRC is not calculated on these automatically inserted sync characters. When the CPU detects the end of message, a Reset Transmit Underrun/EOM command can be issued. This allows CRC to be sent when the transmitter has no data to send.

In the case of sync insertion, an interrupt is generated only after the first automatically inserted sync character has been loaded in the Transmit Shift Register. The status register indicates the Transmit Underrun/ EOM bit and the Transmit Buffer Empty bit are set.

In the case of CRC insertion, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset while CRC is being sent. When CRC has been completely sent, the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin (this interrupt occurs because CRC has been sent and sync has been loaded into the Tx Shift Register). If no more messages are to be sent, the program can terminate transmission by resetting RTS, and disabling the transmitter (WR5; D3).

**Bisync CRC Generation.** Setting the Transmit CRC enable bit (WR5; D0) indicates CRC accumulation when the program sends the first data character to the MPSC. Although the MPSC automatically transmits up to two sync characters (16 bit sync), it is wise to send a few more sync characters ahead of the message (before enabling Transmit CRC) to ensure synchronization at the receiving end.

The Transmit CRC Enable bit can be changed on the fly any time in the message to include or exclude a particular data character from CRC accumulation. The Transmit CRC Enable bit should be in the desired state when the data character is loaded into the transmit shift register. To ensure this bit in the proper state, the Transmit CRC Enable bit must be issued before sending the data character to the MPSC.

**Transmit Transparent Mode.** Transparent mode (Bisync protocol) operation is made possible by the ability to change Transmit CRC Enable on the fly and by the additional capability of inserting 16 bit sync characters. Exclusion of DLE characters from CRC calculation can be achieved by disabling CRC calculation immediately preceding the DLE character transfer to the MPSC.

In the transmit mode, the transmitter always sends the programmed number of sync bits (8 or 16) (WR4; D5, D4). When in the Monosync mode, the transmitter sends from WR6 and the receiver compares transfer to the MPSC. In the transmit initialization process, the CRC generator is initialized by setting the Reset Transmit CRC Generator command (WR0; D7, D6).

The External/Status interrupt (WR1; D0) mode can be used to monitor the status of the CTS input as well as the Transmit Underrun/EOM latch. Optionally, the Auto Enable (WR3; D5) feature can be used to enable the transmitter when CTS is active. The first data transfer to the MPSC can begin when the External/Status interrupt occurs (CTS (RRO; D5) status bit set) following the Transmit Enable command (WR5; D3).

**Receive**

After a channel reset, the receiver is in the Hunt phase, during which the MPSC looks for character synchronization. The Hunt begins only when the receiver is enabled and data transfer begins only when character synchronization has been achieved. If character synchronization is lost, the hunt phase can be re-entered by writing the Enter Hunt Phase (WR3; D4) bit. The assembly of received data continues until the MPSC is reset or until the receiver is
Interrupt receive coming data rate, or a and thereafter allows sequence using signal. The MPSC interrupt whenever a character enters the receive interrupt. Parity Errors do not cause interrupts, but transfers to the receive data FIFO. After receiving the first data character, the Sync Character Load Inhibit bit should be reset to zero so that all characters are received, including the sync characters. This is important because the received CRC may look like a sync character and not get received.

Data may be transferred with or without interrupts. Transferring data without interrupts is used for a purely polled operation or for off-line conditions. There are three interrupt modes available for data transfer: Interrupt on First Character Only, Interrupt on Every Character, and Special Receive Conditions Interrupt.

Interrupt on First Character Only mode is normally used to start a polling loop, a block transfer sequence using RDY to synchronize the CPU to the incoming data rate, or a DMA transfer using the RxDRQ signal. The MPSC interrupts on the first character and thereafter only interrupts after a Special Receive Condition is detected. This mode can be reinitialized using the Enable Interrupt On Next Receive Character (WR0; D5, D4, D3) command which allows the next character received to generate an interrupt. Parity Errors do not cause interrupts, but End of Frame (SDLC operation) and Receive Overrun do cause interrupts in this mode. If the external status interrupts (WR1; D0) are enabled an interrupt may be generated any time the CD changes state.

Interrupt On Every Character mode generates an interrupt whenever a character enters the receive buffer. Errors and Special Receive Conditions generate a special vector if the Status Affects Vector (WR1B; D2) is selected. Also the Parity Error may be programmed (WR1; D4, D3) not to generate the special vector while in the Interrupt On Every Character mode.

The Special Receive Condition interrupt can only occur while in the Receive Interrupt On First Character Only or the Interrupt On Every Receive Character modes. The Special Receive Condition interrupt is caused by the Receive Overrun (RR1; D5) error condition. The error status reflects an error in the current word in the receive buffer, in addition to any Parity or Overrun errors since the last Error Reset (WR0; D5, D4, D3). The Receive Overrun and Parity error status bits are latched and can only be reset by the Error Reset (WR0; D5, D4, D3) command.

The CRC check result may be obtained by checking for CRC bit (RR1; D6). This bit gives the valid CRC result 16 bit times after the second CRC byte has been read from the MPSC. After reading the second CRC byte, the user software must read two more characters (may be sync characters) before checking for CRC result in RR1. Also for proper CRC computation by the receiver, the user software must reset the Receive CRC Checker (WR0; D7, D6) after receiving the first valid data character. The receive CRC Enable bit (WR3; D3) may also be enabled at this time.

SYNCHRONOUS OPERATION—SDLC

General

Like the other synchronous operations the SDLC mode must be initialized with the following parameters: SDLC mode (WR4; D5, D4), SDLC polynomial (WR5; D2), Request to Send, Data Terminal Ready,

<table>
<thead>
<tr>
<th>Synchronous Mode Register Setup—SDLC/HDLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>WR3</td>
</tr>
<tr>
<td>WR4</td>
</tr>
<tr>
<td>WR5</td>
</tr>
</tbody>
</table>
transmit character length (WR5; D6, D5), interrupt modes (WR1; WR2), Transmit Enable (WR5; D3), Receive Enable (WR3; D0), Auto Enable (WR3; D5) and External/Status Interrupt (WR1; D0). WR4 parameters must be written before WR1, WR3, WR5, WR6 and WR7.

The Interrupt modes for SDLC operation are similar to those discussed previously in the synchronous operations section.

**Transmit**

After a channel reset, the MPSC begins sending SDLC flags.

Following the flags in an SDLC operation the 8-bit address field, control field and information field may be sent to the MPSC by the microprocessor. The MPSC transmits the Frame Check Sequence using the Transmit Underrun feature. The MPSC automatically inserts a zero after every sequence of 5 consecutive 1’s except when transmitting Flags or Aborts.

SDLC—like protocols do not have provision for fill characters within a message. The MPSC therefore automatically terminates an SDLC frame when the transmit data buffer and output shift register have no more bits to send. It does this by sending the two bytes of CRC and then one or more flags. This allows very high-speed transmissions under DMA or CPU control without requiring the CPU to respond quickly to the end-of-message situation.

After a reset, the Transmit Underrun/EOM status bit is in the set state and prevents the insertion of CRC characters during the time there is no data to send. Flag characters are sent. The MPSC begins to send the frame when data is written into the transmit buffer. Between the time the first data byte is written, and the end of the message, the Reset Transmit Underrun/EOM (WR0; D7, D6) command must be issued. The Transmit Underrun/EOM status bit (RR0; D6) is in the reset state at the end of the message which automatically sends the CRC characters.

The MPSC may be programmed to issue a send Abort command (WR0; D5, D4, D3). This command causes at least eight 1’s but less than fourteen 1’s to be sent before the line reverts to continuous flags.

**Receive**

After initialization, the MPSC enters the Hunt phase, and remains in the Hunt phase until the first Flag is received. The MPSC never again enters the Hunt phase unless the microprocessor writes the Enter Hunt command. The MPSC will also detect flags separated by a single zero. For example, the bit pattern 01110110111110 will be detected as two flags.

The MPSC can be programmed to receive all frames or it can be programmed to the Address Search Mode. In the Address Search Mode, only frames with addresses that match the value in WR6 or the global address (0FFH) are received by the MPSC. Extended address recognition must be done by the microprocessor software.

The control and information fields are received as data.

SDLC/HDLC CRC calculation does not have an 8-bit delay, since all characters are included in the calculation, unlike Byte Synchronous Protocols.

Reception of an abort sequence (7 or more 1’s) will cause the Break/Abort bit (RR0; D7) to be set and will cause an External/Status interrupt, if enabled. After the Reset External/Status Interrupts Command has been issued, a second interrupt will occur at the end of the abort sequence.

**MPSC**

**Detailed Command/Status Description**

**GENERAL**

The MPSC supports an extremely flexible set of serial and system interface modes.

The system interface to the CPU consists of 8 ports or buffers:

<table>
<thead>
<tr>
<th>CS</th>
<th>A1</th>
<th>Ao</th>
<th>Read Operation</th>
<th>Write Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Ch A Data Read</td>
<td>Ch A Data Write</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Ch A Status Read</td>
<td>Ch A Command/Parameter</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Ch B Data Read</td>
<td>Ch B Data Write</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Ch B Status Read</td>
<td>Ch B Command/Parameter</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>High Impedance</td>
<td>High Impedance</td>
</tr>
</tbody>
</table>

Data buffers are addressed by \( A_1 = 0 \), and Command ports are addressed by \( A_1 = 1 \).
COMMAND/STATUS DESCRIPTION

The following command and status bytes are used during initialization and execution phases of operation. All Command/Status operations on the two channels are identical, and independent, except where noted.

Detailed Register Description

Write Register 0 (WR0):

WR0
D2, D1, D0—Command/Status Register Pointer bits determine which write-register the next byte is to be written into, or which read-register the next byte is to be read from. After reset, the first byte written into either channel goes into WR0. Following a read or write to any register (except WR0) the pointer will point to WR0.

D5, D4, D3—Command bits determine which of the basic seven commands are to be performed.
10  Reset Transmit CRC Generator — resets the CRC generator to 0's. If in SDLC mode the CRC generator's initialized to all 1's.

11  Reset Tx Underrun/End of Message Latch.

**Write Register 1 (WR1):**

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>EXT INTERRUPT ENABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>TX INTERRUPT/DMA ENABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STATUS AFFECTS VECTOR (CH B ONLY)</td>
<td>0 = FIXED (NULL CODE CH A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td></td>
<td>RX INT/DMA DISABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td></td>
<td>RX INT ON FIRST CHAR OR SPECIAL CONDITION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td></td>
<td>INT ON ALL Rx CHAR (PARITY AFFECTS VECTOR) OR SPECIAL CONDITION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td>INT ON ALL Rx CHAR (PARITY DOES NOT AFFECT VECTOR) OR SPECIAL CONDITION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**D1**  Transmitter Interrupt/DMA Enable — allows the MPSC to interrupt or request a DMA transfer when the transmitter buffer becomes empty.

**D2**  Status Affects vector — (WR1, D2 active in channel B only.) If this bit is not set, then the fixed vector, programmed in WR2, is returned from an interrupt acknowledge sequence. If the bit is set then the vector returned from an interrupt acknowledge is variable as shown in the Interrupt Vector Table.

**WR1**

| D0  | External/Status Interrupt Enable — allows interrupt to occur as the result of transitions on the CD, CTS or SYNDET inputs. Also allows interrupts as the result of a Break/Abort detection and termination, or at the beginning of CRC, or sync character transmission when the Transmit Underrun/EOM latch becomes set. |
| D4, D3 |
| D4  | Receive Interrupt Mode |
| D3  | Receive Interrupts/DMA Disabled |
| D1  | Receive Interrupt on First Character Only or Special Condition |
| D2  | Interrupt on All Receive Characters or Special Condition (Parity Error is a Special Receive Condition) |
|     | Interrupt on All Receive Characters or Special Condition (Parity Error is not a Special Receive Condition). |
| D5  | Wait on Receive/Transmit — when the following conditions are met the RDY pin is activated, otherwise it is held in the High-Z state. (Conditions: Interrupt Enabled Mode, Wait Enabled, CS = 0, A0 = 0/1, and A1 = 0). The RDY pin is pulled low when the transmitter buffer is full or the receiver buffer is empty and it is driven High when the transmitter buffer is empty or the receiver buffer is full. The RDY_A and RDY_B may be wired OR connected since only one signal is active at any one time while the other is in the High Z state. |
|     | Must be Zero |
| D6  | Wait Enable — enables the wait function. |
**WR2**

<table>
<thead>
<tr>
<th>Channel A</th>
<th>D5, D4, D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1, D0</td>
<td>Channel A and Channel B both use interrupts</td>
</tr>
<tr>
<td>0 0</td>
<td>Channel A and Channel B both use DMA</td>
</tr>
<tr>
<td>0 1</td>
<td>Illegal Code</td>
</tr>
<tr>
<td>1 0</td>
<td>(Highest) RxA, TxA, RxB, TxB ExTA, ExTB (Lowest)</td>
</tr>
<tr>
<td>1 1</td>
<td>(Highest) RxA, RxB, TxA, TxB ExTA, ExTB (Lowest)</td>
</tr>
</tbody>
</table>

**Write Register 2 (WR2): Channel A**

```
<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>D5</td>
<td>D4</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D1</td>
<td>D0</td>
</tr>
</tbody>
</table>
```

- **D7** D6 D5 D4 D3 D2 D1 D0
- 0 0 BOTH INTERRUPT
- 0 1 A DMA, B INT
- 1 0 BOTH DMA
- 1 1 ILLEGAL

- 1 PRIORITY RxA RxB TxA TxB EXTA EXTB
- 0 8085 MODE 1
- 0 1 8085 MODE 2
- 1 0 8086/88 MODE
- 1 1 ILLEGAL

**VECTORED INTERRUPT**

- 0 NON VECTORED INTERRUPT
- MUST BE ZERO

**EXTERNAL STATUS INTERRUPT:**
- ONLY IF EXT INTERRUPT ENABLE (WR1: D0) IS SET
The following table describes the MPSC's response to an interrupt acknowledge sequence:

<table>
<thead>
<tr>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>iP</th>
<th>MODE</th>
<th>INTA</th>
<th>Data Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Non-vectored</td>
<td>Any INTA</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>85 Mode 1</td>
<td>V7 V6 V5 V4' V3' V2' V1 V0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1st INTA</td>
<td>2nd INTA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3rd INTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>85 Mode 1</td>
<td>1st INTA</td>
<td>V7 V6 V5 V4' V3' V2' V1 V0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1st INTA</td>
<td>2nd INTA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3rd INTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>86 Mode</td>
<td>1st INTA</td>
<td>V7 V6 V5 V4' V3' V2' V1 V0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2nd INTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3rd INTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>85 Mode 2</td>
<td>1st INTA</td>
<td>V7 V6 V5 V4' V3' V2' V1 V0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2nd INTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3rd INTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>85 Mode 2</td>
<td>1st INTA</td>
<td>V7 V6 V5 V4' V3' V2' V1 V0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2nd INTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3rd INTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>86 Mode</td>
<td>1st INTA</td>
<td>V7 V6 V5 V4' V3' V2' V1 V0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2nd INTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3rd INTA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*These bits are variable if the "status affects vector" mode has been programmed, (WR1B, D2).

Interrupt/DMA Mode, Pin Functions, and Priority

<table>
<thead>
<tr>
<th>Ch. A WR2</th>
<th>Int/DMA Mode</th>
<th>Pin Functions</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td>CH. A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>INT</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>INT</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>DMA</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>DMA</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DMA</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>DMA</td>
</tr>
</tbody>
</table>

*Special Receive Condition
Interrupt Vector Mode Table

<table>
<thead>
<tr>
<th>8085 Modes</th>
<th>V4</th>
<th>V3</th>
<th>V2</th>
<th>Channel</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>8086/88 Mode</td>
<td>V2</td>
<td>V1</td>
<td>V0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note 1: Special</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>B</td>
<td>Tx Buffer Empty</td>
</tr>
<tr>
<td>Receive Condition =</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>Ext/Status Change</td>
</tr>
<tr>
<td>Parity Error,</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>Rx Char. Available</td>
</tr>
<tr>
<td>Rx Overrun Error,</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td>Special Rx Condition</td>
</tr>
<tr>
<td>Framing Error,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Note 1)</td>
</tr>
<tr>
<td>End of Frame (SDLC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Write Register 2 (WR2): Channel B

D7–D0 Interrupt vector—This register contains the value of the interrupt vector placed on the data bus during interrupt acknowledge sequences.

Write Register 3 (WR3):

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
<td>D5</td>
<td>D4</td>
<td>D3</td>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Rx 5 BITS/CHAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Rx 7 BITS/CHAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Rx 6 BITS/CHAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Rx 8 BITS/CHAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
WR3
D0 Receiver Enable—A one enables the receiver to begin. This bit should be set only after the receiver has been initialized.
D1 Sync Character Load Inhibit—A one prevents the receiver from loading sync characters into the receive buffers. In SDLC, this bit must be zero.
D2 Address Search Mode—If the SDLC mode has been selected, the MPSC will receive all frames unless this bit is a 1. If this bit is a 1, the MPSC will receive only frames with address bytes that match the global address (0FFH) or the value loaded into WR6. This bit must be zero in non-SDLC modes.
D3 Receive CRC Enable—A one in this bit enables (or re-enables) CRC calculation. CRC calculation starts with the last character placed in the Receiver FIFO. A zero in this bit disables, but does not reset, the Receiver CRC generator.
D4 Enter Hunt Phase—After initialization, the MPSC automatically enters the Hunt mode. If synchronization is lost, the Hunt phase can be re-entered by writing a one to this bit.
D5 Auto Enable—A one written to this bit causes CD to be automatic enable signal for the receiver and CTS to be an automatic enable signal for the transmitter. A zero written to this bit limits the effect of CD and CTS signals to setting/resetting their corresponding bits in the status register (RRO).
D7, D6 Receive Character length
0 0 Receive 5 Data bits/character
0 1 Receive 7 Data bits/character
1 0 Receive 6 Data bits/character
1 1 Receive 8 Data bits/character

Write Register 4 (WR4):

D0 Parity—a one in this bit causes a parity bit to be added to the programmed number of data bits per character for both the transmitted and received character. If the MPSC is programmed to receive 8 bits per character, the parity bit is not transferred to the microprocessor. With other receiver character lengths, the parity bit is transferred to the microprocessor.
D1 Even/Odd Parity—if parity is enabled, a one in this bit causes the MPSC to transmit and expect even parity, and a zero causes it to send and expect odd parity.
D3, D2 Stop bits/sync mode
0 0 Selects synchronous modes.
0 1 Async mode, 1 stop bit/character
1 0 Async mode, 1-1/2 stop bits/character
1 1 Async mode, 2 stop bits/character
D5, D4 Sync mode select
0 0 8 bit sync character
0 1 16 bit sync character
1 0 SDLC mode (Flag sync)
1 1 External sync mode
D7, D6 Clock mode—selects the clock/data rate multiplier for both the receiver and the transmitter. 1x mode must be selected for synchronous modes. If the 1x mode is selected, bit synchronization must be done externally.
0 0 Clock rate = Data rate x 1
0 1 Clock rate = Data rate x 16
1 0 Clock rate = Data rate x 32
1 1 Clock rate = Data rate x 64

Write Register 5 (WR5):

D0 Transmit CRC Enable—a one in this bit enables the transmitter CRC generator. The CRC calculation is done when a character is moved from the transmit buffer into the shift register. A zero in this bit disables CRC calculations. If this bit is not set when a transmitter underrun occurs, the CRC will not be sent.
D1 Request to Send—a one in this bit forces the RTS pin active (low) and zero in this bit forces the RTS pin inactive (high).
D2 CRC Select—a one in this bit selects the CRC -16 polynomial \(X^{16} + X^{15} + X^2 + 1\) and a zero in this bit selects the CCITT-CRC polynomial \(X^{16} + X^{12} + X^5 + 1\). In SDLC mode, CCITT-CRC must be selected.
D3 Transmitter Enable—a zero in this bit forces a marking state on the transmitter output. If this bit is set to zero during data or sync character transmission, the marking state is entered after the character has been sent. If this bit is set to zero during transmission of a CRC character, sync or flag bits are substituted for the remainder of the CRC bits.
D4 Send Break—a one in this bit forces the transmit data low. A zero in this bit allows normal transmitter operation.
D6, D5 Transmit Character length
0 0 Transmit 1 - 5 bits/character
0 1 Transmit 7 bits/character
1 0 Transmit 6 bits/character
1 1 Transmit 8 bits/character

Bits to be sent must be right justified least significant bit first, eg:

D7 D6 D5 D4 D3 D2 D1 D0
0 0 B5 B4 B3 B2 B1 B0
Five or less mode allows transmission of one to five bits per character. The microprocessor must format the data in the following way:

D7 D6 D5 D4 D3 D2 D1 D0
1 1 1 1 0 0 0 B0 Sends one data bit
1 1 1 0 0 0 B1 B0 Sends two data bits
1 1 0 0 0 B2 B1 B0 Sends three data bits
1 0 0 0 B3 B2 B1 B0 Sends four data bits
0 0 0 B4 B3 B2 B1 B0 Sends five data bits

D7 Data Terminal Ready—when set, this bit forces the DTR pin active (low). When reset, this bit forces the DTR pin inactive (high).

Write Register 6 (WR6):

Write Register 7 (WR7):

WR6
D7–D0 Sync/Address—this register contains the transmit sync character in Monosync mode, the low order 8 sync bits in Bisync mode, or the Address byte in SDLC mode.

WR7
D7–D0 Sync/Flag—this register contains the receive sync character in Monosync mode, the high order 8 sync bits in Bisync mode, or the Flag character (01111110) in SDLC mode. WR7 is not used in External Sync mode.
Read Register 0 (RR0):

RR0

D0  Receive Character Available—this bit is set when the receive FIFO contains data and is reset when the FIFO is empty.

D1  Interrupt Pending*—This Interrupt-Pending bit is reset when an EOI command is issued and there is no other interrupt request pending at that time.

D2  Transmit Buffer Empty—This bit is set whenever the transmit buffer is empty except when CRC characters are being sent in a synchronous mode. This bit is reset when the transmit buffer is loaded. This bit is set after an MPSC reset.

D3  Carrier Detect—This bit contains the state of the CD pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the CD pin causes the CD bit to be latched and causes an External/Status interrupt. This bit indicates current state of the CD pin immediately following a Reset External/Status Interrupt command.

D4  Sync/Hunt—In asynchronous modes, the operation of this bit is similar to the CD status bit, except that Sync/Hunt shows the state of the SYNDET input. Any High-to-Low transition on the SYNDET pin sets this bit, and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the SYNDET pin at time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the SYNDET input.

In the External Sync mode, the Sync/Hunt bit operates in a fashion similar to the Asynchronous mode, except the Enter Hunt Mode control bit enables the external sync detection logic. When the External Sync Mode and Enter Hunt Mode bits are set (for example, when the receiver is enabled following a reset), the SYNDET input must be held High by the external logic until external character synchronization is achieved. A High at the SYNDET input holds the Sync/Hunt status in the reset condition.

*In vector mode this bit is set at the falling edge of the second INTA in an INTA cycle for an internal interrupt request. In non-vector mode, this bit is set at the falling edge of RD input after pointer 2 is specified. This bit is always zero in Channel B.
When external synchronization is achieved, SYNDET must be driven Low on the second rising edge of RxC after the rising edge of RxC on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the SYNDET input. Once SYNDET is forced Low, it is good practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. The High-to-Low transition of the SYNDET output sets the Sync/Hunt bit, which sets the External/Status interrupt. The CPU must clear the interrupt by issuing the Reset External/Status Interrupt Command.

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt mode bit. The Sync/Hunt bit is reset when the MPSC establishes character synchronization. The High-to-Low transition of the Sync/Hunt bit causes an External/Status interrupt that must be cleared by the CPU issuing the Reset External/Status Interrupt Command. This enables the MPSC to detect the next transition of other External/Status bits.

When the CPU detects the end of message or that character synchronization is lost, it sets the Enter Hunt Mode control bit, which sets the Sync/Hunt bit to 1. The Low-to-High transition of the Sync/Hunt bit sets the External/Status Interrupt, which must also be cleared by the Reset External/Status Interrupt Command. Note that the SYNDET pin acts as an output in this mode, and goes low every time a sync pattern is detected in the data stream.

In the SDLC mode, the Sync/Hunt bit is initially set by the Enter Hunt mode bit, or when the receiver is disabled. In any case, it is reset to 0 when the opening flag of the first frame is detected by the MPSC. The External/Status interrupt is also generated, and should be handled as discussed previously.

Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in the SDLC mode, it does not need to be set when the end of message is detected. The MPSC automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode bit, or by disabling the receiver.

Clear to Send—this bit contains the inverted state of the CTS pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the CTS pin causes the CTS bit to be latched and causes an External/Status interrupt. This bit indicates the inverse of the current state of the CTS pin immediately following a Reset External/Status Interrupt command.

Transmitter Underrun/End of Message—this bit is in a set condition following a reset (internal or external). The only command that can reset this bit is the Reset Transmit Underrun/EOM Latch command (WR0, D5 and D6). When the Transmit Underrun condition occurs, this bit is set, which causes the External/Status interrupt which must be reset by issuing a Reset External/Status command (WR0; command 2).

Break/Abort—in the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, Command 2) to the break detection logic so the Break sequence termination can be recognized.
The Break/Abort bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

In the SDLC Receive mode, this status bit is set by the detection of an Abort sequence (seven or more 1's). The External/Status interrupt is handled the same way as in the case of a Break. The Break/Abort bit is not used in the Synchronous Receive mode.

D0 All sent—this bit is set when all characters have been sent, in asynchronous modes. It is reset when characters are in the transmitter, in asynchronous modes. In synchronous modes, this bit is always set.

D3, D2, D1 Residue Codes—bit synchronous protocols allow I-fields that are not an integral number of characters. Since transfers from the MPSC to the CPU are character oriented, the residue codes provide the capability of receiving leftover bits. Residue bits are right justified in the last two data bytes received.

D4 Parity Error—If parity is enabled, this bit is set for received characters whose parity does not match the programmed sense (Even/Odd). This bit is latched. Once an error occurs, it remains set until the Error Reset command is written.
Read Register 1 (RR1): (Special Receive Condition Mode)

D5 Receive Overrun Error — this bit indicates that the receive FIFO has been overloaded by the receiver. The last character in the FIFO is overwritten and flagged with this error. Once the overwritten character is read, this error condition is latched until reset by the Error Reset command. If the MPSC is in the status affects vector mode, the overrun causes a special Receive Condition Vector.

D6 CRC/Framing Error — In async modes, a one in this bit indicates a receive framing error. In synchronous modes, a one in this bit indicates that the calculated CRC value does not match the last two bytes received. It can be reset by issuing an Error Reset command.

D7 End of Frame — this bit is valid only in SDLC mode. A one indicates that a valid ending flag has been received. This bit is reset either by an Error Reset command or upon reception of the first character of the next frame.
Read Register 2 (RR2):  

```
<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>V7</td>
<td>V6</td>
</tr>
<tr>
<td>V5</td>
<td>V4'</td>
</tr>
<tr>
<td>V2*</td>
<td>V1'</td>
</tr>
<tr>
<td>V0'</td>
<td>V0</td>
</tr>
</tbody>
</table>
```

**RR2 Channel B**

*Interrupt vector—contains the interrupt vector programmed into WR2. If the status affects vector mode is selected (WR1; D2), it contains the modified vector (See WR2). RR2 contains the modified vector for the highest priority interrupt pending. If no interrupts are pending, the variable bits in the vector are set to one.*

**SYSTEM INTERFACE**

**General**

The MPSC to Microprocessor System interface can be configured in many flexible ways. The basic interface types are polled, wait, interrupt driven, or direct memory access driven.

Polled operation is accomplished by repetitively reading the status of the MPSC, and making decisions based on that status. The MPSC can be polled at any time.

Wait operation allows slightly faster data throughput for the MPSC by manipulating the Ready input to the microprocessor. Block Read or Write Operations to the MPSC are started at will by the microprocessor and the MPSC activates its RDY signal if it is not yet ready to transmit the new byte, or if reception of new byte is not completed.

Interrupt driven operation is accomplished via an internal or external interrupt controller. When the MPSC requires service, it sends an interrupt request signal to the microprocessor, which responds with an interrupt acknowledge signal. When the internal or external interrupt controller receives the acknowledge, it vectors the microprocessor to a service routine, in which the transaction occurs.

DMA operation is accomplished via an external DMA controller. When the MPSC needs a data transfer, it request a DMA cycle from the DMA controller. The DMA controller then takes control of the bus and simultaneously does a read from the MPSC and a write to memory or vice-versa.

The following section describes the many configurations of these basic types of system interface techniques for both serial channels.

**POLLED OPERATION:**

In the polled mode, the CPU must monitor the desired conditions within the MPSC by reading the appropriate bits in the read registers. All data available, status, and error conditions are represented by the appropriate bits in read registers 0 and 1 for channels A and B.

There are two ways in which the software task of monitoring the status of the MPSC has been reduced. One is the "ORing" of all conditions into the Interrupt Pending bit. (RR0; D1 channel A only). This bit is set when the MPSC requires service, allowing the CPU to monitor one bit instead of four status registers. The other is available when the "status-affects-vector" mode is selected. By reading RR2 Channel B, the CPU can read a vector who's value will indicate that one or more of group of conditions has occurred, narrowing the field of possible conditions. See WR2 and RR2 in the Detailed Command Description section.

**Software Flow, Polled Operation**

![Software Flow Diagram](image-url)

RR0, D0 is reset automatically when the data is read
RR0, D2 is reset automatically when the data is written
**Hardware Configuration, Polled Operation**

**ADDRESS BUS**

**DATA BUS**

**RD**

**WR**

**8205**

**A0**

**A1**

**CE**

**RD**

**WR**

**MPSC**

**INTA**

**VCC**

---

**WAIT OPERATION:**
Wait Operation is intended to facilitate data transmission or reception using block move operations. If a block of data is to be transmitted, for example, the CPU can execute a String I/O instruction to the MPSC. After writing the first byte, the CPU will attempt to write a second byte immediately as is the case of block move. The MPSC forces the RDY signal low which inserts wait states in the CPU’s write cycle until the transmit buffer is ready to accept a new byte. At that time, the RDY signal is high allowing the CPU to finish the write cycle. The CPU then attempts the third write and the process is repeated.

Similar operation can be programmed for the receiver. During initialization, wait on transmit (WR1; D5 = 0) or wait on receive (WR1; D5 = 1) can be selected. The wait operation can be enabled/disabled by setting/resetting the Wait Enable Bit (WR1; D7).

**CAUTION:** ANY CONDITION THAT CAN CAUSE THE TRANSMITTER TO STOP (EG, CTS GOES INACTIVE) OR THE RECEIVER TO STOP (EG, RX DATA STOPS) WILL CAUSE THE MPSC TO HANG THE CPU UP IN WAIT STATES UNTIL RESET. EXTREME CARE SHOULD BE TAKEN WHEN USING THIS FEATURE.

**INTERRUPT DRIVEN OPERATION:**
The MPSC can be programmed into several interrupt modes: Non-Vectored, 8085 vectored, and 8088/86 vectored. In both vectored modes, multiple MPSC’s can be daisy-chained.

In the vectored mode, the MPSC responds to an interrupt acknowledge sequence by placing a call instruction (8085 mode) and interrupt vector (8085 and 8088/86 mode) on the data bus.

The MPSC can be programmed to cause an interrupt due to up to 14 conditions in each channel. The status of these interrupt conditions is contained in Read Registers 0 and 1. These 14 conditions are all directed to cause 3 different types of internal interrupt request for each channel: receive/interrupts, transmit interrupts and external/status interrupts (if enabled).

This results in up to 6 internal interrupt request signals. The priority of those signals can be programmed to one of two fixed modes:

- **Highest Priority**
  - RxA RxB TxA TxB ExTA ExTB
- **Lowest Priority**
  - RxA TxA RxB TxB ExTA ExTB

The interrupt priority resolution works differently for vectored and non-vectored modes.

**PRIORITY RESOLUTION: VECTORED MODE**
Any interrupt condition can be accepted internally to the MPSC at any time, unless the MPSC’s internal INTA signal is active, unless a higher priority interrupt is currently accepted, or if MP is inactive (high). The MPSC’s internal INTA is set on the leading (falling) edge of the first External INTA pulse and reset on the trailing (rising) edge of the second External INTA pulse. After an interrupt is accepted internally, an External INT request is generated and the IP goes inactive. IFO and IP are used for daisy-chaining MPSC’s together.
The MPSC’s internal INTA is set on the leading (falling) edge of the first external INTA pulse, and reset on the trailing (rising) edge of the second external INTA pulse. After an interrupt is accepted internally, an external INT request is generated and IP0 goes inactive (high). IP0 and IP1 are used for daisy-chaining MPSC’s together.
Each of the six interrupt sources has an associated In-Service latch. After priority has been resolved, the highest priority In-Service latch is set. After the In-Service latch is set, the INT pin goes inactive (high).
Lower priority interrupts are not accepted internally while the In-Service latch is set. However, higher priority interrupts are accepted internally and a new external INT request is generated. If the CPU responds with a new INTA sequence, the MPSC will respond as before, suspending the lower priority interrupt.

After the interrupt is serviced, the End-of-Interrupt (EOI) command should be written to the MPSC. This command will cause an internal pulse that is used to reset the In-Service Latch which allows service for lower priority interrupts in the daisy-chain to resume, provided a new INTA sequence does not start for a higher priority interrupt (higher than the highest under service). If there is no interrupt pending internally, the IPO follows IPI.
Non-Vectored Interrupt Timing

In non-vectored mode, the MPSC does not respond to interrupt acknowledge sequences. The INTA input (pin 27) must be pulled high for proper operation. The MPSC should be programmed to the Status-Affects-Vector mode, and the CPU should read RR2 (Ch. B) in its service routine to determine which interrupt requires service.

In this case, the internal pointer being set to RR2 provides the same function as the internal INTA signal in the vectored mode. It inhibits acceptance of any additional internal interrupts and its leading edge starts the interrupt priority resolution circuit. The interrupt priority resolution is ended by the leading edge of the read signal used by the CPU to retrieve the modified vector. The leading edge of read sets the In-Service latch and forces the external INT output inactive (high). The internal pointer is reset to zero after the trailing edge of the read pulse.
Note that if RR2 is specified but not read, no internal interrupts, regardless of priority, are accepted.

**DAISY CHAINING MPSC:**

In the vectored interrupt mode, multiple MPSC's can be daisy-chained on the same INT, INTA signals. These signals, in conjunction with the IPI and IPO allow a daisy-chain-like interrupt resolution scheme. This scheme can be configured for either 8085 or 8086/88 based system.

In either mode, the same hardware configuration is called for. The INT request lines are wire-OR'ed together at the input of a TTL inverter which drives the INT pin of the CPU. The INTA signal from the CPU drives all of the daisy-chained MPSC's.

The MPSC drives IPO (Interrupt Priority Output) inactive (high) if IPI (Interrupt Priority Input) is inactive (high), or if the MPSC has an interrupt pending.

The IPO of the highest priority MPSC is connected to the IPI of the next highest priority MPSC, and so on.

If IPI is active (low), the MPSC knows that all higher priority MPSC's have no interrupts pending. The IPI pin of the highest priority MPSC is strapped active (low) to ensure that it always has priority over the rest.

MPSC's Daisy-chained on an 8088/86 CPU should be programmed to the 8088/86 Interrupt mode (WR2; D4, D3 (Ch. A). MPSC's Daisy-chained on an 8085 CPU should be programmed to 8085 interrupt mode 1 if it is the highest priority MPSC. In this mode, the highest priority MPSC issues the CALL instruction during the first INTA cycle, and the interrupting MPSC provides the interrupt vector during the following INTA cycles. Lower priority MPSC's should be programmed to 8085 interrupt mode 2.

MPSC's used alone in 8085 systems should be programmed to 8085 mode 1 interrupt operation.
DMA Acknowledge Circuit

DMA Timing

DMA OPERATION
Each MPSC can be programmed to utilize up to four DMA channels: Transmit Channel A, Receive Channel A, Transmit Channel B, Receive Channel B. Each DMA Channel has an associated DMA Request line. Acknowledgement of a DMA cycle is done via normal data read or write cycles. This is accomplished by encoding the DACK signals to generate A₀, A₁, and CS signals, and multiplexing them with the normal A₀, A₁, and CS signals.

PERMUTATIONS
Channels A and B can be used with different system interface modes. In all cases it is impossible to poll the MPSC. The following table shows the possible permutations of interrupt, wait, and DMA modes for channels A and B. Bits D₁, D₀ of WR2 Ch. A determine these permutations.

<table>
<thead>
<tr>
<th>Permutation WR2 Ch. A</th>
<th>Channel A</th>
<th>Channel B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D₁, D₀</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>Wait</td>
<td>Wait</td>
</tr>
<tr>
<td></td>
<td>Interrupt Polled</td>
<td>Interrupt Polled</td>
</tr>
<tr>
<td>0 1</td>
<td>DMA Polled</td>
<td>Interrupt Polled</td>
</tr>
<tr>
<td>1 0</td>
<td>DMA Polled</td>
<td>DMA Polled</td>
</tr>
</tbody>
</table>

D₁, D₀ = 1, 1 is illegal.
PROGRAMMING HINTS

This section will describe some useful programming hints which may be useful in program development.

Asynchronous Operation

At the end of transmission, the CPU must issue "Reset Transmit Interrupt/DMA Pending" command in WR0 to reset the last transmit empty request which was not satisfied. Failing to do so will result in the MPSC locking up in a transmit empty state forever.

Non-Vectored Mode

In non-vectored mode, the Interrupt Acknowledge pin (INTA) on the MPSC must be tied high through a pull-up resistor. Failing to do so will result in unpredictable response from the 8274.

HDLC/SDLC Mode

When receiving data in SDLC mode, the CRC bytes must be read by the CPU (or DMA controller) just like any other data field. Failing to do so will result in receiver buffer overflow. Also, the End of Frame Interrupt indicates that the entire frame has been received. At this point, the CRC result (RR1:D6) and residue code (RR1:D3, D2, D1) may be checked.

Status Register RR2

RR2 contains the vector which gets modified to indicate the source of interrupt (See the section titled MPSC Modes of Operation). However, the state of the vector does not change if no new interrupts are generated. The contents of RR2 are only changed when a new interrupt is generated. In order to get the correct information, RR2 must be read only after an interrupt is generated, otherwise it will indicate the previous state.

Initialization Sequence

The MPSC initialization routine must issue a channel Reset Command at the beginning. WR4 should be defined before other registers. At the end of the initialization sequence, Reset External/Status and Error Reset commands should be issued to clear any spurious interrupts which may have been caused at power up.

Transmit Under-run/EOM Latch

In SDLC/HDLC, bisync and monosync mode, the transmit under-run/EOM must be reset to enable the CRC check bytes to be appended to the transmit frame or transmit message. The transmit under-run/EOM latch can be reset only after the first character is loaded into the transmit buffer. When the transmitter under-runs at the end of the frame, CRC check bytes are appended to the frame/message. The transmit under-run/EOM latch can be reset at any time during the transmission after the first character. However, it should be reset before the transmitter under-runs otherwise, both bytes of the CRC may not be appended to the frame/message. In the receive mode in bisync operation, the CPU must read the CRC bytes and two more SYNC characters before checking for valid CRC result in RR1.

Sync Character Load Inhibit

In bisync/monosync mode only, it is possible to prevent loading sync characters into the receive buffers by setting the sync character load inhibit bit (WR3:D1=1). Caution must be exercised in using this option. It may be possible to get a CRC character in the received message which may match the sync character and not get transferred to the receive buffer. However, sync character load inhibit should be enabled during all pre-frame sync characters so the software routine does not have to read them from the MPSC.

In SDLC/HDLC mode, sync character load inhibit bit must be reset to zero for proper operation.

EOI Command

EOI command can only be issued through channel A irrespective of which channel had generated the interrupt.

Priority in DMA Mode

There is no priority in DMA mode between the following four signals: TxDRQ(CHA), RxDRQ(CHA), TxDRQ(CHB), RxDRQ(CHB). The priority between these four signals must be resolved by the DMA controller. At any given time, all four DMA channels from the 8274 are capable of going active.
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature
Under Bias ........................................... 0°C to +70°C
Storage Temperature
(Ceramic Package) ..................... -65°C to +150°C
(Plastic Package) ..................... -40°C to +125°C
Voltage On Any Pin With
Respect to Ground ................... -0.5V to +7.0V

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS  \( (T_a = 0°C \text{ to } 70°C; V_{cc} = +5V \pm 10\%) \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{IL} )</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>+0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>( V_{IH} )</td>
<td>Input High Voltage</td>
<td></td>
<td>+2.0</td>
<td>( V_{cc} ) +0.5</td>
<td>V</td>
</tr>
<tr>
<td>( V_{OL} )</td>
<td>Output Low Voltage</td>
<td>+0.45</td>
<td></td>
<td>V</td>
<td>( I_{OL} ) = 2.0mA</td>
</tr>
<tr>
<td>( V_{OH} )</td>
<td>Output High Voltage</td>
<td>+2.4</td>
<td></td>
<td>V</td>
<td>( I_{OH} ) = -200( \mu )A</td>
</tr>
<tr>
<td>( I_{IL} )</td>
<td>Input Leakage Current</td>
<td>±10</td>
<td></td>
<td>( \mu )A</td>
<td>( V_{IN} = V_{cc} ) to 0V</td>
</tr>
<tr>
<td>( I_{OL} )</td>
<td>Output Leakage Current</td>
<td>±10</td>
<td></td>
<td>( \mu )A</td>
<td>( V_{OUT} = V_{cc} ) to 0V</td>
</tr>
<tr>
<td>( I_{CC} )</td>
<td>( V_{cc} ) Supply Current</td>
<td>200</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
</tbody>
</table>

CAPACITANCE  \( (T_a = 25°C; V_{cc} = GND = 0V) \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{IN} )</td>
<td>Input Capacitance</td>
<td>10</td>
<td></td>
<td>pF</td>
<td>( f_c = 1 ) MHz;</td>
</tr>
<tr>
<td>( C_{OUT} )</td>
<td>Output Capacitance</td>
<td>15</td>
<td></td>
<td>pF</td>
<td>Unmeasured</td>
</tr>
<tr>
<td>( C_{iO} )</td>
<td>Input/Output Capacitance</td>
<td>20</td>
<td></td>
<td>pF</td>
<td>pins returned to GND</td>
</tr>
</tbody>
</table>
### A.C. CHARACTERISTICS \( (T_a = 0^\circ\text{C} \text{ to } 70^\circ\text{C}; \ V_{cc} = +5\text{V} \pm 10\%) \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCY</td>
<td>CLK Period</td>
<td>250</td>
<td>4000</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tCL</td>
<td>CLK Low Time</td>
<td>105</td>
<td>2000</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tCH</td>
<td>CLK High Time</td>
<td>105</td>
<td>2000</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>t(R)</td>
<td>CLK Rise Time</td>
<td>0</td>
<td>30</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>t(F)</td>
<td>CLK Fall Time</td>
<td>0</td>
<td>30</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tAR</td>
<td>A0, A1 Setup to RD(\downarrow)</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tAD</td>
<td>A0, A1 to Data Output Delay</td>
<td>200</td>
<td></td>
<td>ns</td>
<td>(C_L=150\ \text{pf})</td>
</tr>
<tr>
<td>tRA</td>
<td>A0, A1 Hold After RD(\downarrow)</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tRD</td>
<td>RD(\downarrow) to Data Output Delay</td>
<td>200</td>
<td></td>
<td>ns</td>
<td>(C_L=150\ \text{pf})</td>
</tr>
<tr>
<td>tRR</td>
<td>RD Pulse Width</td>
<td>250</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tDF</td>
<td>Output Float Delay</td>
<td>120</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tAW</td>
<td>CS, A0, A1 Setup to WR(\downarrow)</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tWA</td>
<td>CS, A0, A1 Hold after WR(\uparrow)</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tWW</td>
<td>WR Pulse Width</td>
<td>250</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tDW</td>
<td>Data Setup to WR(\uparrow)</td>
<td>150</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tWD</td>
<td>Data Hold After WR(\uparrow)</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tPI</td>
<td>I(\Pi) Setup to INTA(\downarrow)</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>t(\Pi)</td>
<td>I(\Pi) Hold after INTA(\uparrow)</td>
<td>10</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tI</td>
<td>INTA Pulse Width</td>
<td>250</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>t(\Pi)(\Pi)O</td>
<td>I(\Pi)(\Pi) to I(\Pi)O Delay</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tID</td>
<td>INTA(\downarrow) to Data Output Delay</td>
<td>200</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tCO</td>
<td>RD or WR to DRO(\downarrow)</td>
<td>150</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tRV</td>
<td>Recovery Time Between Controls</td>
<td>300</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tCW</td>
<td>CS, A0, A1 to RDY(\text{A}) or RDY(\text{B}) Delay</td>
<td>140</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tDCY</td>
<td>Data Clock Cycle</td>
<td>4.5</td>
<td></td>
<td>tcy</td>
<td></td>
</tr>
<tr>
<td>tDCL</td>
<td>Data Clock Low Time</td>
<td>180</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tDCH</td>
<td>Data Clock High Time</td>
<td>180</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tTD</td>
<td>I(\pi)C to TxD Delay</td>
<td>300</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tDS</td>
<td>RxD Setup to RxC(\uparrow)</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tDH</td>
<td>RxD Hold after RxC(\uparrow)</td>
<td>140</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>t(\Pi)D</td>
<td>I(\Pi)C to INT Delay</td>
<td>4</td>
<td>6</td>
<td>tcy</td>
<td></td>
</tr>
<tr>
<td>t(\Pi)D</td>
<td>RxC to INT Delay</td>
<td>7</td>
<td>10</td>
<td>tcy</td>
<td></td>
</tr>
<tr>
<td>tPL</td>
<td>CTS, CD, SYND(\text{ET}) Low Time</td>
<td>200</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tPH</td>
<td>CTS, CD, SYND(\text{ET}) High Time</td>
<td>200</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tPD</td>
<td>External INT from CTS, CD, SYND(\text{ET})</td>
<td>500</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>
A.C. TESTING INPUT, OUTPUT WAVEFORM

INPUT/OUTPUT

2.0

0.8

TEST POINTS

A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC 1 AND 0.45V FOR A LOGIC 0. TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC 1 AND 0.8V FOR A LOGIC 0.

A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

C_L = 150 pF

C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

CLOCK CYCLE

READ CYCLE

HIGH IMPEDANCE

170102-001
WAVEFORMS (Continued)

WRITE CYCLE

INTA CYCLE

DMA CYCLE

NOTES:
1. INTA signal acts as RD signal.
2. IPI signal acts as CS signal.
WAVEFORMS (Continued)

READ/WRITE CYCLE (SOFTWARE POLLED MODE)

TRANSMIT DATA CYCLE

RECEIVE DATA CYCLE

OTHER TIMING

CTS, CD, SYNDET

6-84
82530/82530-6
SERIAL COMMUNICATIONS CONTROLLER (SCC)

- Two independent full duplex serial channels
- On chip crystal oscillator, Baud-Rate Generator and Digital Phase Locked Loop for each channel
- Programmable for NRZ, NRZI or FM data encoding/decoding
- Diagnostic local loopback and automatic echo for fault detection and isolation
- System Clock Rates:
  - 4 Mhz for 82530
  - 6 Mhz for 82530-6
- Max Bit Rate (4 Mhz)
  - Externally clocked: 1Mbps
  - Self clocked:
    250 Kbps FM coding
    125 Kbps NRZI coding
- Interfaces easily with any INTEL CPU, DMA or I/O processor
- Asynchronous Modes
  - 5-8 bit character; odd, even or no parity; 1, 1.5 or 2 stop bits
  - Independent transmit and receive clocks. 1X, 16X, 32X or 64X programmable sampling rate
  - Error Detection: Framing, Overrun and Parity
  - Break detection and generation
- Bit synchronous Modes
  - SDLC Loop/Non-Loop Operation
  - CRC-16 or CCITT Generation Detection
  - Abort generation and detection
  - I-field residue handling
  - CCITT X.25 compatible
- Byte synchronous Modes
  - Internal or external character synchronization (1 or 2 characters)
  - Automatic CRC generation and checking (CRC 16 or CCITT)
  - IBM Bisync compatible

The INTEL 82530 Serial Communications Controller (SCC) is a dual-channel, multi-protocol data communications peripheral. It is designed to interface high speed communications lines using Asynchronous, Byte synchronous and Bit synchronous protocols to INTEL's microprocessors based systems. It can be interfaced with Intel's MCS51, iAPX86/88/186 and 188 in polled, interrupt driven or DMA driven modes of operation.

The SCC is a 40 pin device manufactured using INTEL's high-performance HMOS II technology.
Figure 1. 82530 Internal Block Diagram

Figure 2. Pin configuration
The following section describes the pin functions of the SCC. Figure 2 details the pin assignments.

### Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB0</td>
<td>40</td>
<td>I/O</td>
<td>Data Bus: The Data Bus lines are bi-directional three-state lines which interface with the system’s Data Bus. These lines carry data and commands to and from the SCC.</td>
</tr>
<tr>
<td>DB1</td>
<td>1</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>DB2</td>
<td>39</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>DB3</td>
<td>2</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>DB4</td>
<td>38</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>DB5</td>
<td>3</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>DB6</td>
<td>37</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>DB7</td>
<td>4</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>INT</td>
<td>5</td>
<td>0</td>
<td>Interrupt Request: The interrupt signal is activated when the SCC requests an interrupt. It is an open drain output.</td>
</tr>
<tr>
<td>IEO</td>
<td>6</td>
<td>0</td>
<td>Interrupt Enable Out: IEO is High only if IEI is High and the CPU is not servicing an SCC interrupt or the SCC is not requesting an interrupt (Interrupt Acknowledge cycle only). IEO is connected to the next lower priority device’s IEI input and thus inhibits interrupts from lower priority devices.</td>
</tr>
<tr>
<td>IEI</td>
<td>7</td>
<td>1</td>
<td>Interrupt Enable In: IEI is used with IEO to form an interrupt daisy chain when there is more than one interrupt-driven device. A High IEI indicates that no other higher priority device has an interrupt under service or is requesting an interrupt.</td>
</tr>
<tr>
<td>INTA</td>
<td>8</td>
<td>1</td>
<td>Interrupt Acknowledge: This signal indicates an active Interrupt Acknowledge cycle. During this cycle, the SCC interrupt daisy chain settles. When RD becomes active, the SCC places an interrupt vector on the data bus (if IEI is High). INTA is latched by the rising edge of CLK.</td>
</tr>
<tr>
<td>VCC</td>
<td>9</td>
<td></td>
<td>Power: +5V Power supply</td>
</tr>
<tr>
<td>RDY A/REQ A</td>
<td>10</td>
<td>0</td>
<td>Ready/Request (output, open-drain when programmed for a Ready function, driven High or Low when programmed for a Request function). These dual-purpose outputs may be programmed as Request lines for a DMA controller or as Ready lines to synchronize the CPU to the SCC data rate. The reset state is Ready.</td>
</tr>
<tr>
<td>RDY B/REQ B</td>
<td>30</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SYNC A</td>
<td>11</td>
<td>I/O</td>
<td>Synchronization: These pins can act either as inputs, outputs or part of the crystal oscillator circuit. In the Asynchronous Receive mode (crystal oscillator option not selected), these pins are inputs similar to CTS and CD. In this mode, transitions on these lines affect the state of the Synchronous/Hunt status bits in Read Register 0 (Figure 9) but have no other function.</td>
</tr>
<tr>
<td>SYNC B</td>
<td>29</td>
<td>I/O</td>
<td></td>
</tr>
</tbody>
</table>

In External Synchronization mode with the crystal oscillator not selected, these lines also act as inputs. In this mode, SYNC must be driven LOW two receive clock cycles after the last bit in the synchronous character is received. Character assembly begins on the rising edge of the receive clock immediately preceding the activation of SYNC.

In the Internal Synchronization mode (Monosync and Bisync) with the crystal oscillator not selected, these pins act as outputs and are active only during the part of the receive clock cycle in which synchronous characters are recognized. The synchronous condition is not latched, so these outputs are active each time a synchronization pattern is recognized (regardless of characters boundaries). In SDLC mode, these pins act as outputs and are valid on receipt of a flag.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTxCA</td>
<td>12</td>
<td>I</td>
<td><strong>Receive/Transmit clocks:</strong> These pins can be programmed in several different modes of operation. In each channel, RTxC may supply the receive clock, the transmit clock, the clock for the baud rate generator, or the clock for the Digital Phase Locked Loop. These pins can be programmed for use with the respective SYNC pins as a crystal oscillator. The receive clock may be 1, 16, 32, or 64 times the data rate in Asynchronous modes.</td>
</tr>
<tr>
<td>RTxCB</td>
<td>28</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>RXDA</td>
<td>13</td>
<td>I</td>
<td><strong>Receive Data:</strong> These lines receive serial data at standard TTL levels.</td>
</tr>
<tr>
<td>RXDB</td>
<td>27</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>TRxA</td>
<td>14</td>
<td>I/O</td>
<td><strong>Transmit/Receive clocks:</strong> These pins can be programmed in several different modes of operation. TRxC may supply the receive clock or the transmit clock in the input mode or supply the output of the Digital Phase Locked Loop, the crystal oscillator, the baud rate generator, or the transmit clock in the output mode.</td>
</tr>
<tr>
<td>TRxCB</td>
<td>26</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>TxDA</td>
<td>15</td>
<td>O</td>
<td><strong>Transmit Data:</strong> These output signals transmit serial data at standard TTL levels.</td>
</tr>
<tr>
<td>Txda</td>
<td>25</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>DTRAREQA</td>
<td>16</td>
<td>O</td>
<td><strong>Data Terminal Ready/Request:</strong> These outputs follow the state programmed into the DTR bit. They can also be used as general purpose outputs or as Request lines for a DMA controller.</td>
</tr>
<tr>
<td>DTRBREQB</td>
<td>24</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td>17</td>
<td>O</td>
<td><strong>Request To Send:</strong> When the Request to Send (RTS) bit in Write Register 5 is set (figure 10), the RTS signal goes Low. When the RTS bit is reset in the Asynchronous mode and Auto Enable is on, the signal goes High after the transmitter is empty. In Synchronous mode or in Asynchronous mode with Auto Enable off, the RTS pin strictly follows the state of the RTS bit. Both pins can be used as general-purpose outputs.</td>
</tr>
<tr>
<td>RTSB</td>
<td>23</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>CTS</td>
<td>18</td>
<td>I</td>
<td><strong>Clear To Send:</strong> If these pins are programmed as Auto Enables, a Low on the inputs enables the respective transmitters. If not programmed as Auto Enables, they may be used as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow rise-time inputs. The SCC detects pulses on these inputs and can interrupt the CPU on both logic level transitions.</td>
</tr>
<tr>
<td>CTSB</td>
<td>22</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>19</td>
<td>I</td>
<td><strong>Carrier Detect:</strong> These pins function as receiver enables if they are programmed for Auto Enables, otherwise they may be used as general-purpose input pins. Both pins are Schmitt-trigger buffered to accommodate slow rise-time signals. The SCC detects pulses on these pins and can interrupt the CPU on both logic level transitions.</td>
</tr>
<tr>
<td>CD</td>
<td>21</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>CLK</td>
<td>20</td>
<td>I</td>
<td><strong>Clock:</strong> This is the system SCC clock used to synchronize internal signals. CLK is a TTL level signal.</td>
</tr>
<tr>
<td>GND</td>
<td>31</td>
<td></td>
<td><strong>Ground</strong></td>
</tr>
<tr>
<td>D/C</td>
<td>32</td>
<td>I</td>
<td><strong>Data/Command Select:</strong> This signal defines the type of information transferred to or from the SCC. A high means data is transferred, a low indicates a command.</td>
</tr>
<tr>
<td>CS</td>
<td>33</td>
<td>I</td>
<td><strong>Chip Select:</strong> This signal selects the SCC for a read or write operation.</td>
</tr>
<tr>
<td>A/8</td>
<td>34</td>
<td>I</td>
<td><strong>Channel A/Channel B Select:</strong> This signal selects the channel in which the read or write operation occurs.</td>
</tr>
<tr>
<td>WR</td>
<td>35</td>
<td>I</td>
<td><strong>Write:</strong> When the SCC is selected this signal indicates a write operation. The coincidence of RD and WR is interpreted as a reset.</td>
</tr>
<tr>
<td>RD</td>
<td>36</td>
<td>I</td>
<td><strong>Read:</strong> This signal indicates a read operation and when the SCC is selected, enables the SCC’s bus drivers. During the Interrupt Acknowledge cycle, this signal gates the interrupt vector onto the bus if the SCC is the highest priority device requesting an interrupt.</td>
</tr>
</tbody>
</table>
GENERAL DESCRIPTION

The INTEL 82530 Serial Communications Controller (SCC) is a dual-channel, multi-protocol data communications peripheral. The SCC functions as a serial-to-parallel, parallel-to-serial converter/controller. The SCC can be software-configured to satisfy a wide range of serial communications applications. The device contains new, sophisticated internal functions including on-chip baud rate generators, digital phase locked loops, various data encoding and decoding schemes, and crystal oscillators that dramatically reduce the need for external logic.

In addition, diagnostic capabilities - automatic echo and local loopback - allow the user to detect and isolate a failure in the network. They greatly improve the reliability and maintainability of the system.

The SCC handles Asynchronous formats, Synchronous byte-oriented protocols such as IBM Bisync, and Synchronous bit-oriented protocols such as HDLC and IBM SDLC. This versatile device supports virtually any serial data transfer application (Terminal, Personal Computer, Peripherals, Industrial Controller, Telecommunication system, etc.).

The 82530 can generate and check CRC codes in any Synchronous mode and can be programmed to check data integrity in various modes. The SCC also has facilities for modem controls in both channels. In applications where these controls are not needed, the modem controls can be used for general-purpose I/O.

The INTEL 82530 is designed to support INTEL's MCS51, iAPX86/88 and iAPX186/188 families.

ARCHITECTURE

The 82530 internal structure includes two full-duplex channels, two baud rate generators, internal control and interrupt logic, and a bus interface to a non-multiplexed CPU bus. Associated with each channel are a number of read and write registers for mode control and status information, as well as logic necessary to interface to modems or other external devices.

The logic for both channels provides formats, synchronization, and validation for data transferred to and from the channel interface. The modem control inputs are monitored by the control logic under program control. All of the modem control signals are general-purpose in nature and can optionally be used for functions other than modem control.

The register set for each channel includes ten control (write) registers, two synchronous character (write) registers, and four status (read) registers. In addition, each baud rate generator has two (read/write) registers for holding the time constant that determines the baud rate. Finally, associated with the interrupt logic is a write register for the interrupt vector accessible through either channel, a write-only Master Interrupt Control register and three read registers: one containing the vector with status information (Channel B only), one containing the vector without status (A only), and one containing the Interrupt Pending bits (A only).

The registers for each channel are designated as follows:

WR0-WR15 - Write Registers 0 through 15.
RR0-RR3, RR10, RR12, RR13, RR15 - Read Registers 0 through 3, 10, 12, 13, 15

Table 2 lists the functions assigned to each read or write register. The SCC contains only one WR2 and WR9, but they can be accessed by either channel. All other registers are paired (one for each channel).

DATA PATH

The transmit and receive data path illustrated in Figure 3 is identical for both channels. The receiver has three 8-bit buffer registers in a FIFO arrangement, in addition to the 8-bit receive shift register. This scheme creates additional time for the CPU to service an interrupt at the beginning of a block of high-speed data. Incoming data is routed through one of several paths (data or CRC) depending on the selected mode (the character length in asynchronous modes also determines the data path).

The transmitter has an 8-bit transmit data buffer register loaded from the internal data bus and a 20-bit transmit shift register that can be loaded either from the sync-character registers or from the transmit data register. Depending on the operational mode, outgoing data is routed through one of four main paths before it is transmitted from the Transmit Data output (TxD).
Table 2. Read and Write Register Functions

<table>
<thead>
<tr>
<th>READ REGISTER FUNCTIONS</th>
<th>WRITE REGISTER FUNCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR0 Transmit/Receive buffer status and External status</td>
<td>WR0 CRC initialize, initialization commands for the various modes, shift right/shift left command</td>
</tr>
<tr>
<td>RR1 Special Receive Condition status</td>
<td>WR1 Transmit/Receive interrupt and data transfer mode definition</td>
</tr>
<tr>
<td>RR2 Modified interrupt vector (Channel B only) Unmodified interrupt (Channel A only)</td>
<td>WR2 Interrupt vector (accessed through either channel)</td>
</tr>
<tr>
<td>RR3 Interrupt Pending bits (Channel A only)</td>
<td>WR3 Receive parameters and control</td>
</tr>
<tr>
<td>RR8 Receive buffer</td>
<td>WR4 Transmit/Receive miscellaneous parameters and modes</td>
</tr>
<tr>
<td>RR10 Miscellaneous status</td>
<td>WR5 Transmit parameters and controls</td>
</tr>
<tr>
<td>RR12 Lower byte of baud rate generator time constant</td>
<td>WR6 Sync characters or SDLC address field</td>
</tr>
<tr>
<td>RR13 Upper byte of baud rate generator time constant</td>
<td>WR7 Sync character or SDLC flag</td>
</tr>
<tr>
<td>RR15 External/Status interrupt information</td>
<td>WR8 Transmit buffer</td>
</tr>
<tr>
<td></td>
<td>WR9 Master interrupt control and reset (accessed through either channel)</td>
</tr>
<tr>
<td></td>
<td>WR10 Miscellaneous transmitter/receiver control bits</td>
</tr>
<tr>
<td></td>
<td>WR11 Clock mode control</td>
</tr>
<tr>
<td></td>
<td>WR12 Lower Byte of baud rate generator time constant</td>
</tr>
<tr>
<td></td>
<td>WR13 Upper byte of baud rate generator time constant</td>
</tr>
<tr>
<td></td>
<td>WR14 Miscellaneous control bits</td>
</tr>
<tr>
<td></td>
<td>WR15 External/Status interrupt control</td>
</tr>
</tbody>
</table>
Figure 3: Data Path
FUNCTIONAL DESCRIPTION
The functional capabilities of the SCC can be described from two different points of view: as a data communications device, it transmits and receives data in a wide variety of data communications protocols; as a microprocessor peripheral, it interacts with the CPU and provides vectored interrupts and handshaking signals.

DATA COMMUNICATIONS CAPABILITIES
The SCC provides two independent full-duplex channels programmable for use in any common asynchronous or synchronous data-communications protocol. Figure 4 and the following description briefly detail these protocols.

Asynchronous Modes
Transmission and reception can be accomplished independently on each channel with five to eight bits per character, plus optional even or odd parity. The transmitter can supply one, one-and-a-half or two stop bits per character and can provide a break output at any time. The receiver break-detection logic interrupts the CPU both at the start and at the end of a received break. Reception is protected from spikes by a transient spike-rejection mechanism that checks the signal one-half a bit time after a Low level is detected on the receive data input (RxD_A or RxD_B). If the Low does not persist (as in the case of a transient), the character assembly process does not start.

Framing errors and overrun errors are detected and buffered together with the partial character on which they occur. Vectored interrupts allow fast servicing or error conditions using dedicated routines. Furthermore, a built-in checking process avoids the interpretation of framing error as a new start bit: a framing error results in the addition of one-half a bit time to the point at which the search for the next start bit begins.

The SCC does not require symmetric transmit and receive clock signals — a feature allowing use of the wide variety of clock sources. The transmitter and receiver can handle data at a rate of 1, 1/16, 1/32, or 1/64 of the clock rate supplied to the receive and transmit clock inputs. In asynchronous modes, the SYNC pin may be programmed as an input used for functions such as monitoring a ring indicator.

![Figure 4. SCC Protocols](image-url)
Synchronous Modes

The SCC supports both byte-oriented and bit-oriented synchronous communication. Synchronous-byte-oriented protocols can be handled in several modes allowing character synchronization with a 6-bit or 8-bit synchronous character (Monosync), any 12-bit synchronous pattern (Bisync), or with an external synchronous signal. Leading synchronous characters can be removed without interrupting the CPU.

Five- or 7-bit synchronous characters are detected with 8- or 16-bit patterns in the SCC by overlapping the larger pattern across multiple incoming synchronous characters as shown in Figure 5.

CRC checking for Synchronous byte-oriented mode is delayed by one character time so that the CPU may disable CRC checking on specific characters. This permits the implementation of protocols such as IBM Bisync.

Both CRC-16 \((X^{16} + X^{15} + X^2 + 1)\) and CCITT \((X^{16} + X^{12} + X^5 + 1)\) error checking polynomials are supported. Either polynomial may be selected in all synchronous modes. Users may preset the CRC generator and checker to all 1s or all 0s. The SCC also provides a feature that automatically transmits CRC data when no other data is available for transmission.

This allows for high-speed transmissions under DMA control, with no need for CPU intervention at the end of a message. When there is no data or CRC to send in synchronous modes, the transmitter inserts 6-, 8-, or 16-bit synchronous characters, regardless of the programmed character length.

The SCC supports synchronous bit-oriented protocols, such as SDLC and HDLC, by performing automatic flag sending, zero insertion, and CRC generation. A special command can be used to abort a frame in transmission. At the end of a message, the SCC automatically transmits the CRC and trailing flag when the transmitter underruns. The transmitter may also be programmed to send an idle line consisting of continuous flag characters or a steady marking condition.

If a transmit underrun occurs in the middle of a message, an external status interrupt warns the CPU of this status change so that an abort may be issued. The SCC may also be programmed to send an abort itself in case of an underrun, relieving the CPU of this task. One to eight bits per character can be sent allowing reception of a message with no prior information about the character structure in the information field of a frame.

The receiver automatically acquires synchronization on the leading flag of a frame in SDLC or HDLC and provides a synchronization signal on the SYNC pin (an interrupt can also be programmed). The receiver can be programmed to search for frames addressed by a single byte (or four bits within a byte) of a user-selected address or to a global broadcast address. In this mode, frames not matching either the user-selected or broadcast address are ignored. The number of address bytes can be extended under software control. For receiving data, an interrupt on the first received character, or an interrupt on every character, or on special condition only (end-of-frame) can be selected. The receiver automatically deletes all Os inserted by the transmitter during character assembly. CRC is also calculated and is automatically checked to validate frame transmission. At the end of transmission, the status of a received frame is available in the status registers. In SDLC mode, the SCC must be programmed to use the SDLC CRC polynomial, but the generator and checker may be be preset to all 1s or all 0s. The CRC is inverted before transmission and the receiver checks against the bit pattern 0001110100001111.

NRZ, NRZI or FM coding may be used in any 1X mode. The parity options available in asynchronous modes are available in synchronous modes.

The SCC can be conveniently used under DMA control to provide high-speed reception or transmission. In reception, for example, the SCC can interrupt the CPU when the first character of a message is received. The CPU then enables the DMA to transfer the message to memory. The SCC then issues an end-of-frame interrupt and the CPU can

---

### Figure 5. Detecting 5- or 7- Bit Synchronous Characters
check the status of the received message. Thus, the CPU is freed for other service while the message is being received. The CPU may also enable the DMA first and have the SCC interrupt only on end-of-frame. This procedure allows all data to be transferred via DMA.

**SDLC LOOP MODE**

The SCC supports SDLC Loop mode in addition to normal SDLC. In an SDLC Loop, there is a primary controller station that manages the message traffic flow and any number of secondary stations. In SDLC Loop mode, the SCC performs the functions of a secondary station while an SCC operating in regular SDLC mode can act as a controller (Figure 6).

![Figure 6. An SDLC Loop](image)

A secondary station in an SDLC Loop is always listening to the messages being sent around the loop, and in fact must pass these messages to the rest of the loop by retransmitting them with a one-bit-time delay. The secondary station can place its own message on the loop only at specific times. The controller signals that secondary stations may transmit messages by sending a special character, called an EOP (End of Poll), around the loop. The EOP character is the bit pattern 11111110. Because of zero insertion during messages, this bit pattern is unique and easily recognized.

When a secondary station has a message to transmit and recognizes an EOP on the line, it changes the last binary one of the EOP to a zero before transmission. This has the effect of turning the EOP into a flag sequence. The secondary station now places its message on the loop and terminates the message with an EOP. Any secondary stations further down the loop with messages to transmit can then append their messages to the message of the first secondary station by the same process. Any secondary stations without messages to send merely echo the incoming messages and are prohibited from placing messages on the loop (except upon recognizing an EOP).

SDLC Loop mode is a programmable option in the SCC. NRZ, NRZI, and FM coding may all be used in SDLC Loop mode.

**BAUD RATE GENERATOR**

Each channel in the SCC contains a programmable Baud rate generator. Each generator consists of two 8-bit time constant registers that form a 16-bit time constant, a 16-bit down counter, and a flip-flop on the output producing a square wave. On startup, the flip-flop on the output is set in a High state, the value in the time constant register is loaded into the counter, and the counter starts counting down. The output of the baud rate generator toggles upon reaching zero, the value in the time constant register is loaded into the counter, and the process is repeated. The time constant may be changed at any time, but the new value does not take effect until the next load of the counter.

The output of the baud rate generator may be used as either the transmit clock, the receive clock, or both. It can also drive the digital phase-locked loop (see next section).

If the receive clock or transmit clock is not programmed to come from the TRxC pin, the output of the baud rate generator may be echoed out via the TRxC pin.

The following formula relates the time constant to the baud rate. (The baud rate is in bits/second and the BR clock period is in seconds.)

\[
\text{baud rate} = \frac{1}{2 \times (\text{time constant} + 2) \times \text{(BR clock period)}}
\]
DIGITAL PHASE LOCKED LOOP

The SCC contains a digital phase locked-loop (DPLL) to recover clock information from a data-stream with NRZI or FM encoding. The DPLL is driven by a clock that is nominally 32 (NRZI) or 16 (FM) times the data rate. The DPLL uses this clock, along with the data-stream, to construct a clock for the data. This clock may then be used as the SCC receive clock, the transmit clock, or both.

For NRZI coding, the DPLL counts the 32X clock to create nominal bit times. As the 32X clock is counted, the DPLL is searching the incoming data-stream for edges (either 1/0 or 0/1). Whenever an edge is detected, the DPLL makes a count adjustment (during the next counting cycle), producing a terminal count closer to the center of the bit cell.

For FM encoding, the DPLL still counts from 1 to 31, but with a cycle corresponding to two bit times. When the DPLL is locked, the clock edges in the data-stream should occur between counts 15 and 16 and between counts 31 and 0. The DPLL looks for edges only during a time centered on the 15/16 counting transition.

The 32X clock for the DPLL can be programmed to come from either the RTxC input or the output of the baud rate generator. The DPLL output may be programmed to be echoed out of the SCC via the TRxR pin (if this pin is not being used as an input).

DATA ENCODING

The SCC may be programmed to encode and decode the serial data in four different ways (Figure 7). In NRZ encoding, a 1 is represented by a High level and a 0 is represented by a Low level. In NRZI encoding, as 1 is represented by no change in level and a 0 is represented by a change in level. In FM₁ (more properly, bi-phase mark) a transition occurs at the beginning of every bit cell. A 1 is represented by an additional transition at the center of the bit cell and a 0 is represented by no additional transition at the center of the bit cell. In FM₂ (bi-phase space), a transition occurs at the beginning of every bit cell. A 0 is represented by an additional transition at the center of the bit cell, and a 1 is represented by no additional transition at the center of the bit cell.

In addition to these four methods, the SCC can be used to decode Manchester (bi-phase level) data by using the DPLL in the FM mode and programming the receiver for NRZ data. Manchester encoding always produces a transition at the center of the bit cell. If the transition is 0/1 the bit is a 1. If the transition is 1/0 the bit is a 0.

**AUTO ECHO AND LOCAL LOOPBACK**

The SCC is capable of automatically echoing everything it receives. This feature is useful mainly in asynchronous modes, but works in synchronous and SDLC modes as well. In Auto Echo mode TxD is RxD. Auto Echo mode can be used with NRZI or FM encoding with no additional delay, because the data-stream is not decoded before retransmission. In Auto Echo mode, the CTS input is ignored as a transmitter enable (although transitions on this input can still cause interrupts if programmed to do so). In this mode, the transmitter is actually bypassed and the programmer is responsible for disabling transmitter interrupts and READY/REQUEST on transmit.

The SCC is also capable of local loopback. In this mode, TxD is RxD just as in Auto Echo mode. However, in Local Loopback mode, the internal transmit data is tied to the internal receive data and RxD is ignored (except to be echoed out via TxD). CTS and CD inputs are also ignored as transmit and receive enables. However, transitions on these inputs can still cause interrupts. Local Loopback works in asynchronous, synchronous and SDLC modes with NRZ, NRZI or FM coding of the data stream.

**SERIAL BIT RATE**

To run the 82530 (4MHz) at 1 Mbps the receive and transmit clocks must be externally generated and synchronized to the system clock. If the serial clocks (RTxR and TRxR) and the system clock (CLK) are asynchronous, the maximum bit rate is 880 Kbps. For self-clocked operation, i.e using the on chip DPLL, the maximum bit rate is 125 Kbps if NRZI coding is used and 250 Kbps if FM coding is used.

---

**Time Constant Values for Standard Baud Rates at BR Clock ≈ 3.9936MHz**

<table>
<thead>
<tr>
<th>Rate (Baud)</th>
<th>Time Constant (decimal notation)</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>19200</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>9600</td>
<td>206</td>
<td></td>
</tr>
<tr>
<td>7200</td>
<td>275</td>
<td>0.12%</td>
</tr>
<tr>
<td>4800</td>
<td>414</td>
<td></td>
</tr>
<tr>
<td>3600</td>
<td>553</td>
<td>0.06%</td>
</tr>
<tr>
<td>2400</td>
<td>830</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>996</td>
<td>0.04%</td>
</tr>
<tr>
<td>1800</td>
<td>1107</td>
<td>0.03%</td>
</tr>
<tr>
<td>1200</td>
<td>1662</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>3326</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>6654</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>13310</td>
<td></td>
</tr>
<tr>
<td>134.5</td>
<td>14844</td>
<td>0.0007%</td>
</tr>
<tr>
<td>110</td>
<td>18151</td>
<td>0.0015%</td>
</tr>
<tr>
<td>75</td>
<td>26622</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>39934</td>
<td></td>
</tr>
</tbody>
</table>

---

**Notes:**

- The table lists time constant values for various baud rates, along with the associated error values.
- The error values are percentages indicating the deviation from the ideal time constant value.
- The table is used to determine the settings for the baud rate generator.
Figure 7. Data Encoding Methods

<table>
<thead>
<tr>
<th>Mode</th>
<th>System clock</th>
<th>System clock/Serial clock</th>
<th>Serial bit rate</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial clocks generated externally</td>
<td>4 Mhz</td>
<td>4</td>
<td>1 Mbps</td>
<td>Serial clocks synchronized with system clock. Refer to parameter #3 and #10 in general timings.</td>
</tr>
<tr>
<td></td>
<td>6 Mhz</td>
<td>4</td>
<td>1.5 Mbps</td>
<td>Serial clocks synchronized with system clock. Refer to parameter #3 and #10 in general timings.</td>
</tr>
<tr>
<td></td>
<td>4 Mhz</td>
<td>4.5</td>
<td>880 Kbps</td>
<td>Serial clocks and system clock asynchronous.</td>
</tr>
<tr>
<td></td>
<td>6 Mhz</td>
<td>4.5</td>
<td>1.3 Mbps</td>
<td>Serial clocks and system clock asynchronous.</td>
</tr>
<tr>
<td>Self-clocked operation</td>
<td>4 Mhz</td>
<td>32</td>
<td>125 Kbps</td>
<td>Seriar clocks and system clock asynchronous.</td>
</tr>
<tr>
<td></td>
<td>6 Mhz</td>
<td>32</td>
<td>187 Kbps</td>
<td>Seriar clocks and system clock asynchronous.</td>
</tr>
<tr>
<td>NRZI</td>
<td>4 Mhz</td>
<td>16</td>
<td>250 Kbps</td>
<td>Seriar clocks and system clock asynchronous.</td>
</tr>
<tr>
<td></td>
<td>6 Mhz</td>
<td>16</td>
<td>375 Kbps</td>
<td>Seriar clocks and system clock asynchronous.</td>
</tr>
</tbody>
</table>
I/O INTERFACE CAPABILITIES
The SCC offers the choice of Polling, Interrupt (vectored or nonvectored) and Block Transfer modes to transfer data, status, and control information to and from the CPU. The Block Transfer mode can be implemented under CPU or DMA control.

POLLING
All interrupts are disabled. Three status registers in the SCC are automatically updated whenever any function is performed. For example, end-of-frame in SDLC mode sets a bit in one of these status registers. The idea behind polling is for the CPU to periodically read a status register until the register contents indicate the need for data to be transferred. Only one register needs to be read; depending on its contents, the CPU either writes data, reads data, or continues. Two bits in the register indicate the need for data transfer. An alternative is a poll of the Interrupt Pending register to determine the source of an interrupt. The status for both channels resides in one register.

INTERRUPTS
When a SCC responds to an Interrupt Acknowledge signal (INTA) from the CPU, an interrupt vector may be placed on the data bus. This vector is written in WR2 and may be read in RR2A or RR2B (Figures 9 and 10).

To speed interrupt response time, the SCC can modify three bits in this vector to indicate status. If the vector is read in Channel A, status is never included; if it is read in Channel B, status is always included.

Each of the six sources of interrupts in the SCC (Transmit, Receive and External/Status interrupts in both channels) has three bits associated with the interrupt source: Interrupt Pending (IP), Interrupt Under Service (IUS), and Interrupt Enable (IE). Operation of the IE bit is straightforward. If the IE bit is set for a given interrupt source, then that source can request interrupts. The exception is when the MIE (Master Interrupt Enable) bit in WR9 is reset and no interrupts may be requested. The IE bits are write-only.

The other two bits are related to the interrupt priority chain (Figure 8). As a peripheral, the SCC may request an interrupt only when no higher-priority device is requesting one, e.g., when IEI is High. If the device in question requests an interrupt, it pulls down INT. The CPU then responds with INTA, and the interrupting device places the vector on the data bus.

In the SCC, the IP bit signals a need for interrupt servicing. When an IP bit is 1 and the IEI input is High, the INT output is pulled Low, requesting an interrupt. In the SCC, if the IE bit is not set by enabling interrupts, then the IP for that source can never be set. The IP bits are readable in RR3A.

The IUS bits signal that an interrupt request is being serviced. If an IUS is set, all interrupt sources of lower priority in the SCC and external to the SCC are prevented from requesting interrupts. The internal interrupt sources are inhibited by the state of the internal daisy chain, while lower priority devices are inhibited by the IEO output of the SCC being pulled

![Figure 8. Daisy Chaining SCC's](image-url)
Low and propagated to subsequent peripherals. An IUS bit is set during an interrupt Acknowledge cycle if there are no higher priority devices requesting interrupts.

There are three types of interrupts: Transmit, Receive and External/Status interrupts. Each interrupt type is enabled under program control with Channel A having higher priority than Channel B, and with Receiver, Transmit and External/Status interrupts prioritized in that order within each channel. When the Transmit interrupt is enabled, the CPU is interrupted when the transmit buffer becomes empty. (This implies that the transmitter must have had a data character written into it so that it can become empty.) When enabled, the receiver can interrupt the CPU in one of three ways:

- Interrupt on First Receive Character or Special Receive condition.
- Interrupt on all Receive Characters or Special Receive condition.
- Interrupt on Special Receive condition only.

Interrupt on First Character or Special Condition and Interrupt on Special Condition Only are typically used with the Block Transfer mode. A Special Receive Condition is one of the following: receiver overrun, framing error in Asynchronous mode, End-of-Frame in SDLC mode and, optionally, a parity error. The Special Receive Condition interrupt is different from an ordinary receive character available interrupt only in the status placed in the vector during the Interrupt-Acknowledge cycle. In Interrupt on First Receive Character, an interrupt can occur from Special Receive conditions any time after the first receive character interrupt.

The main function of the External/Status interrupt is to monitor the signal transitions of the CTS, CD, and SYNC pins; however, an External/Status interrupt is also caused by a Transmit Underrun condition, or a zero count in the baud rate generator, or by the detection of a Break (asynchronous mode), Abort (SDLC mode) or EOP (SDLC Loop mode) sequence in the data stream. The interrupt caused by the Abort or EOP has a special feature allowing the SCC to interrupt when the Abort or EOP sequence is detected or terminated. This feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the Abort condition in external logic in SDLC mode. In SDLC Loop mode this feature allows secondary stations to recognize the wishes of the primary station to regain control of the loop during a poll sequence.

**CPU/DMA BLOCK TRANSFER**

The SCC provides a Block Transfer mode to accommodate CPU block transfer functions and DMA controllers. The Block Transfer mode uses the READY/REQUEST output in conjunction with the READY/REQUEST bits in WR1. The READY/REQUEST output can be defined under software control as a READY line in the CPU Block Transfer mode (WR1; D6=0) or as a request line in the DMA Block Transfer mode (WR1; D6=1). To a DMA controller, the SCC REQUEST output indicates that the SCC is ready to transfer data to or from memory. To the CPU, the READY line indicates that the SCC is not ready to transfer data, thereby requesting that the CPU extend the I/O cycle. The DTR/REQUEST line allows full-duplex operation under DMA control.

**PROGRAMMING**

Each channel has fifteen Write registers that are individually programmed from the system bus to configure the functional personality of each channel. Each channel also has eight Read registers from which the CPU can read Status, Baud rate, or Interrupt information.

Only the four data registers (Read, Write for channels A and B) are directly selected by a High on the D/C input and the appropriate levels on the RD, WR and A/B pins. All other registers are addressed indirectly by the content of Write Register 0 in conjunction with a Low on the D/C input and the appropriate levels on the RD, WR and A/B pins. If bit 4 in WW0 is 1 and bits 5 and 6 are 0 then bits 0, 1, 2 address the higher registers 8 through 15. If bits 4, 5, 6 contain a different code, bits 0, 1, 2 address the lower registers 0 through 7 as shown on Table 3.

Writing to or reading from any register except RR0, WR0 and the Data Registers thus involves two operations:

First write the appropriate code into WR0, then follow this by a write or read operation on the register thus specified. Bits 0 through 4 in WW0 are automatically cleared after this operation, so that WW0 then points to WR0 or RR0 again.

Channel A/Channel B selection is made by the A/B input (High = A, Low = B)

The system program first issues a series of commands to initialize the basic mode of operation. This is followed by other commands to qualify condi-
TABLE 3. REGISTER ADDRESSING

<table>
<thead>
<tr>
<th>D/C “Point High” Code in WR0</th>
<th>D2 in WR0</th>
<th>D1 in WR0</th>
<th>D0 in WR0</th>
<th>Write Register</th>
<th>Read Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Either Way</td>
<td>X</td>
<td>X</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Low</td>
<td>Not True</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Low</td>
<td>Not True</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Low</td>
<td>Not True</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Low</td>
<td>Not True</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Low</td>
<td>Not True</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>(0)</td>
</tr>
<tr>
<td>Low</td>
<td>Not True</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>(1)</td>
</tr>
<tr>
<td>Low</td>
<td>True</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>(2)</td>
</tr>
<tr>
<td>Low</td>
<td>True</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>(3)</td>
</tr>
<tr>
<td>Low</td>
<td>True</td>
<td>1</td>
<td>0</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Low</td>
<td>True</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>Low</td>
<td>True</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Low</td>
<td>True</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>(15)</td>
</tr>
<tr>
<td>Low</td>
<td>True</td>
<td>1</td>
<td>0</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Low</td>
<td>True</td>
<td>1</td>
<td>1</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Low</td>
<td>True</td>
<td>1</td>
<td>0</td>
<td>14</td>
<td>(10)</td>
</tr>
<tr>
<td>Low</td>
<td>True</td>
<td>1</td>
<td>1</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring: e.g. when the interrupt vector indicates a Special Receive Condition interrupt, all the appropriate error bits can be read from a single register (RR1).

READ REGISTERS

The SCC contains eight read registers (actually nine, counting the receive buffer (RR8) in each channel). Four of these may be read to obtain status information (RR0, RR1, RR10, and RR15). Two registers (RR12 and RR13) may be read to earn the baud rate generator time constant. RR2 contains either the unmodified interrupt vector (Channel A) or the vector modified by status information (Channel B). RR3 contains the Interrupt Pending (IP) bits (Channel A). Figure 9 shows the formats for each read register.

WRITE REGISTERS

The SCC contains 15 write registers (16 counting WR8, the transmit buffer) in each channel. These write registers are programmed separately to configure the functional “personality” of the channels. In addition, there are two registers (WR2 and WR9) shared by the two channels that may be accessed through either of them. WR2 contains the interrupt vector for both channels, while WR9 contains the interrupt control bits. Figure 10 shows the format of each write register.
Figure 9. Read Register Bit Functions
Figure 10. Write Register Bit Functions
Figure 10. Write Register Bit Functions (Cont.)
82530 TIMING
The SCC generates internal control signals from WR and RD that are related to CLK. Since CLK has no phase relationship with WR and RD, the circuitry generating these internal control signals must provide time for metastable conditions to disappear. This gives rise to a recovery time related to CLK. The recovery time applies only between bus transactions involving the SCC. The recovery time required for proper operation is specified from the rising edge of WR or RD in the first transaction involving the SCC to the falling edge of WR or RD in the second transaction involving the SCC. This time must be at least 6 CLK cycles plus 200ns.

Read Cycle Timing
Figure 11 illustrates Read cycle timing. Addresses on A/B and D/C and the status on INTA must remain stable throughout the cycle. If CS falls after WR falls or if it rises before RD rises, the effective RD is shortened.

Write Cycle Timing
Figure 12 illustrates Write cycle timing. Addresses on A/B and D/C and the status on INTA must remain stable throughout the cycle. If CS falls after WR falls or if it rises before WR rises, the effective WR is shortened.

Interrupt Acknowledge Cycle Timing
Figure 13 illustrates Interrupt Acknowledge cycle timing. Between the time INTA goes Low and the falling edge of RD, the internal and external IEI/IEO daisy chains settle. If there is an interrupt pending in the SCC and IEI is High when RD falls, the Acknowledge cycle is intended for the SCC. In this case, the SCC may be programmed to respond to RD Low by placing its interrupt vector on D6-D7 and it then sets the appropriate Interrupt-Under-Service internally.
Figure 12. Write Cycle Timing

Figure 13. Interrupt Acknowledge Cycle Timing
ABSOLUTE MAXIMUM RATINGS*

Case Temperature
Under Bias .................. 0°C to +70°C
Storage Temperature
(Ceramic Package) ......... -65°C to +150°C
(Plastic Package) .......... -40°C to +125°C
Voltage On Any Pin With
Respect to Ground ........... -0.5V to +7.0V

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS  (Tc=0°C to 70°C; Vcc=+5V±5%)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.3</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>+2.0</td>
<td>VCC+0.3</td>
<td>V</td>
<td>IOL = 20mA</td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>-0.40</td>
<td>V</td>
<td>V</td>
<td>IOH = -250μA</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>+2.4</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>IIL</td>
<td>Input Leakage Current</td>
<td>±10</td>
<td>μA</td>
<td>0.4 to 2.4V</td>
<td></td>
</tr>
<tr>
<td>IOL</td>
<td>Output Leakage Current</td>
<td>±10</td>
<td>μA</td>
<td>0.4 to 2.4V</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>VCC Supply Current</td>
<td>250</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CAPACITANCE  (Tc=25°C; Vcc=GND=0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIN</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td>fC = 1 MHz;</td>
<td></td>
</tr>
<tr>
<td>COU</td>
<td>Output Capacitance</td>
<td>15</td>
<td>pF</td>
<td>Unmeasured</td>
<td></td>
</tr>
<tr>
<td>CIO</td>
<td>Input/Output Capacitance</td>
<td>20</td>
<td>pF</td>
<td>pins returned to GND</td>
<td></td>
</tr>
</tbody>
</table>
**A.C CHARACTERISTICS** (T<sub>c</sub>=0° C to 70° C; V<sub>cc</sub>=+5V±5%)

### READ AND WRITE TIMING

<table>
<thead>
<tr>
<th>Number</th>
<th>Symbol</th>
<th>Parameter</th>
<th>82530 (4MHz)</th>
<th>82530-6 (6 MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>tCL</td>
<td>CLK Low Time</td>
<td>105</td>
<td>2000</td>
</tr>
<tr>
<td>2</td>
<td>tCH</td>
<td>CLK High Time</td>
<td>105</td>
<td>2000</td>
</tr>
<tr>
<td>3</td>
<td>tf</td>
<td>CLK Fall Time</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>tr</td>
<td>CLK Rise Time</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>tCY</td>
<td>CLK Cycle Time</td>
<td>250</td>
<td>4000</td>
</tr>
<tr>
<td>6</td>
<td>tAW</td>
<td>Address to WR&lt;sub&gt;i&lt;/sub&gt; Setup Time</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>7</td>
<td>tWA</td>
<td>Address to WR&lt;sub&gt;i&lt;/sub&gt; Hold Time</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>tAR</td>
<td>Address to RD&lt;sub&gt;i&lt;/sub&gt; Setup Time</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>tRA</td>
<td>Address to RD&lt;sub&gt;i&lt;/sub&gt; Hold Time</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>tIC</td>
<td>INTA to CLK&lt;sub&gt;i&lt;/sub&gt; Setup Time</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>tIW</td>
<td>INTA to WR&lt;sub&gt;i&lt;/sub&gt; Setup Time (Note 1)</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>12</td>
<td>tWI</td>
<td>INTA to WR&lt;sub&gt;i&lt;/sub&gt; Hold Time</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>tIR</td>
<td>INTA to RD&lt;sub&gt;i&lt;/sub&gt; Setup Time (Note 1)</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>14</td>
<td>tRI</td>
<td>INTA to RD&lt;sub&gt;i&lt;/sub&gt; Hold Time</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>tCI</td>
<td>INTA to CLK&lt;sub&gt;i&lt;/sub&gt; Hold Time</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>16</td>
<td>tCLW</td>
<td>CS Low to WR&lt;sub&gt;i&lt;/sub&gt; Setup Time</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>tWCS</td>
<td>CS to WR&lt;sub&gt;i&lt;/sub&gt; Hold Time</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>tCHW</td>
<td>CS High to WR&lt;sub&gt;i&lt;/sub&gt; Setup Time</td>
<td>100</td>
<td>70</td>
</tr>
<tr>
<td>19</td>
<td>tCLR</td>
<td>CS Low to RD&lt;sub&gt;i&lt;/sub&gt; Setup Time (Note 1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>tRCS</td>
<td>CS to RD&lt;sub&gt;i&lt;/sub&gt; Hold Time (Note 1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>tCHR</td>
<td>CS High to RD&lt;sub&gt;i&lt;/sub&gt; Setup Time (Note 1)</td>
<td>100</td>
<td>70</td>
</tr>
<tr>
<td>22</td>
<td>tRR</td>
<td>RD Low Time (Note 1)</td>
<td>390</td>
<td>250</td>
</tr>
<tr>
<td>23</td>
<td>tRDI</td>
<td>RD&lt;sub&gt;i&lt;/sub&gt; to Data Not Valid Delay</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>tRDV</td>
<td>RD&lt;sub&gt;i&lt;/sub&gt; to Data Valid Delay</td>
<td>250</td>
<td>180</td>
</tr>
<tr>
<td>25</td>
<td>tDF</td>
<td>RD&lt;sub&gt;i&lt;/sub&gt; to Output Float Delay (Note 2)</td>
<td>70</td>
<td>45</td>
</tr>
</tbody>
</table>

**NOTES:**
1. Parameter does not apply to Interrupt Acknowledge transactions.
2. Float delay is defined as the time required for a ±0.5V change in the output with a maximum D.C load and minimum A.C load.

*Timings are preliminary and subject to change.*
A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC 1 AND 0.45V FOR A LOGIC 0. TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC 1 AND 0.8V FOR A LOGIC 0.

A.C. TESTING LOAD CIRCUIT

CL INCLUDES JIG CAPACITANCE

OPEN DRAIN TEST LOAD

FROM OUTPUT UNDER TEST
Figure 14. Read and Write Timing
## INTERRUPT ACKNOWLEDGE TIMING, RESET TIMING, CYCLE TIMING

<table>
<thead>
<tr>
<th>Number</th>
<th>Symbol</th>
<th>Parameter</th>
<th>82530 (4 MHz)</th>
<th>82530-6 (6 MHz)</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>tAD</td>
<td>Address Required Valid to Read Data Valid Delay</td>
<td>590</td>
<td>420</td>
<td>ns</td>
</tr>
<tr>
<td>28</td>
<td>TWW</td>
<td>WR! Low Time</td>
<td>390</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>29</td>
<td>tDW</td>
<td>Data to WR! Setup Time</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>30</td>
<td>tWD</td>
<td>Data to WR! Hold Time</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>31</td>
<td>tWRV</td>
<td>WR! to Ready Valid Delay (Note 4)</td>
<td>240</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>32</td>
<td>tRRV</td>
<td>RD! to Ready Valid Delay (Note 4)</td>
<td>240</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>33</td>
<td>tWRI</td>
<td>WR! to READY/REQ Not Valid Delay</td>
<td>240</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>34</td>
<td>tRR!</td>
<td>RD! to READY/REQ Not Valid Delay</td>
<td>240</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>35</td>
<td>tDWR</td>
<td>WR! to DTR/REQ Not Valid Delay</td>
<td>5 tCY</td>
<td>5 tCY</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+ 300</td>
<td>+ 250</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>tIL</td>
<td>RD! (Acknowledge) Low Time</td>
<td>285</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>40</td>
<td>tIDV</td>
<td>RD! (Acknowledge) to Read Data Valid Delay</td>
<td>190</td>
<td>180</td>
<td>ns</td>
</tr>
<tr>
<td>41</td>
<td>tEI</td>
<td>IE! to RD! (Acknowledgment) Setup Time</td>
<td>120</td>
<td>100</td>
<td>ns</td>
</tr>
<tr>
<td>42</td>
<td>tIE</td>
<td>IE! to RD! (Acknowledgment) Hold Time</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>43</td>
<td>tIEEO</td>
<td>IE! to IE! Delay Time</td>
<td>120</td>
<td>100</td>
<td>ns</td>
</tr>
<tr>
<td>44</td>
<td>tCEQ</td>
<td>CLK! to IE! Delay</td>
<td>250</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>45</td>
<td>tRI!</td>
<td>RD! to INT Inactive Delay (Note 4)</td>
<td>500</td>
<td>500</td>
<td>ns</td>
</tr>
<tr>
<td>46</td>
<td>tRW</td>
<td>RD! to WR! Delay for No Reset</td>
<td>30</td>
<td>15</td>
<td>ns</td>
</tr>
<tr>
<td>47</td>
<td>tWR</td>
<td>WR! to RD! Delay for No Reset</td>
<td>30</td>
<td>30</td>
<td>ns</td>
</tr>
<tr>
<td>48</td>
<td>tRES</td>
<td>WR! and RD Coincident Low for Reset</td>
<td>250</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>49</td>
<td>tREC</td>
<td>Valid Access Recovery Time</td>
<td>6 tCY</td>
<td>6 tCY</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+ 200</td>
<td>+ 130</td>
<td></td>
</tr>
</tbody>
</table>

### NOTES:

3. Parameter applies only between transactions involving the SCC.
4. Open-drain output, measured with open-drain test load.
5. Parameter is system dependent. For any SCC in the daisy chain, tIL must be greater than the sum of tCEQ for the highest priority device in the daisy chain, tEI for the SCC and tIEEO for each device separating them in the daisy chain.

*Timings are preliminary and subject to change.*
Figure 15. Interrupt Acknowledge Timing

Figure 16. Reset Timing

Figure 17. Cycle Timing
### GENERAL TIMING

<table>
<thead>
<tr>
<th>Number</th>
<th>Symbol</th>
<th>Parameter</th>
<th>82530 (4MHz)</th>
<th>82530-6 (6 MHz)</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
</tr>
<tr>
<td>3</td>
<td>tRCC</td>
<td>RxC! to CLK! Setup Time (Notes 1,4)</td>
<td>50</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>tRRC</td>
<td>RxD to RxC! Setup Time (X1 Mode) (Note 1)</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>tRCR</td>
<td>RxD to RxC! Hold Time (X1 Mode) (Note 1)</td>
<td>150</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>tDRC</td>
<td>RxD to RxC! Setup Time (X1 Mode) (Notes 1,5)</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>tRCD</td>
<td>RxD to RxC! Hold Time (X1 Mode) (Notes 1,5)</td>
<td>150</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>tSRC</td>
<td>SYNC to RxC! Setup Time (Note 1)</td>
<td>-200</td>
<td></td>
<td>-200</td>
</tr>
<tr>
<td>9</td>
<td>tRCS</td>
<td>Sync to RxC! Hold Time (Note 1)</td>
<td>3 tCY</td>
<td>3 tCY</td>
<td>+ 200</td>
</tr>
<tr>
<td>10</td>
<td>tTCC</td>
<td>TxCI to CLK! Setup Time (Notes 2,4)</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>tTCT</td>
<td>TxCI to TxD Delay (X1 Mode) (Note 2)</td>
<td></td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>12</td>
<td>tTCD</td>
<td>TxCI to TxD Delay (X1 Mode) (Notes 2,5)</td>
<td></td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>13</td>
<td>tTDT</td>
<td>TxD to TRxC Delay (Send Clock Echo)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>tDCH</td>
<td>RTxC High Time</td>
<td>180</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>tDCL</td>
<td>RTxC Low Time</td>
<td>180</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>tOCY</td>
<td>RTxC Cycle Time</td>
<td>4TCY</td>
<td>4TCY</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>tCLCL</td>
<td>Crystal Oscillator Period (Note 3)</td>
<td>250</td>
<td>1000</td>
<td>250</td>
</tr>
<tr>
<td>18</td>
<td>tRCH</td>
<td>TRxC High Time</td>
<td>180</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>tRCL</td>
<td>TRxC Low Time</td>
<td>180</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>tRCY</td>
<td>TRxC Cycle Time</td>
<td>4TCY</td>
<td>4TCY</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>tCC</td>
<td>CD or CTS Pulse Width</td>
<td>200</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>tSS</td>
<td>SYNC Pulse Width</td>
<td>200</td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**
1. RxC is RTxC or TRxC, whichever is supplying the receive clock.
2. TXC is TRxC or RTxC, whichever is supplying the transmit clock.
3. Both RTxC and SYNC have 30pF capacitors to ground connected to them.
4. Parameter applies only if the data rate is one-fourth the system clock (CLK) rate. In all other cases, no phase relationship between RxC and CLK or TXC and CLK is required.
5. Parameter applies only to FM encoding/decoding.

*Timings are preliminary and subject to change.
Figure 18. General Timing
Using The 8251 Universal Synchronous/Asynchronous Receiver/Transmitter
Using the 8251
Universal Synchronous/Asynchronous Receiver/Transmitter

Contents
INTRODUCTION
COMMUNICATION FORMATS
BLOCK DIAGRAM
   Receiver
   Transmitter
   Modem Control
   I/O Control
INTERFACE SIGNALS
   CPU-Related Signals
   Device-Related Signals
MODE SELECTION
PROCESSOR DATA LINK
CONCLUSION
APPENDIX A
   8251 Design Hints
INTRODUCTION

The Intel 8251 is a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) which is capable of operating with a wide variety of serial communication formats. Since many peripheral devices are available with serial interfaces, the 8251 can be used to interface a microcomputer to a broad spectrum of peripherals, as well as to a serial communications channel. The 8251 is part of the MCS-80™ Microprocessor Family, and as such it is capable of interfacing to the 8080 system with a minimum of external hardware.

This application note describes the 8251 as a component and then explains its use in sample applications via several examples. A specific use of the 8251 to facilitate communication between two MCS-80 systems is discussed in detail from both the hardware and software viewpoints. The first two sections of this application note describe the 8251 first from a functional standpoint and then on a detailed level. The function of each input and output pin is fully defined. The next section describes the various operating modes and how they can be selected, and finally, a sample design is discussed using the 8251 as a data link between the MCS-80 systems.

COMMUNICATION FORMATS

Serial communications, either on a data link or with a local peripheral, occurs in one of two basic formats; asynchronous or synchronous. These formats are similar in that they both require framing information to be added to the data to enable proper detection of the character at the receiving end. The major difference between the two formats is that the asynchronous format requires framing information to be added to each character, while the synchronous format adds framing information to blocks of data, or messages. Since the synchronous format is more efficient than the asynchronous format but requires more complex decoding, it is typically found on high-speed data links, while the asynchronous format is used on lower speed lines.

The asynchronous format starts with the basic data bits to be transmitted and adds a "START" bit to the front of them and one or more "STOP" bits behind them as they are transmitted. The START bit is a logical zero, or SPACE, and is defined as the positive voltage level by RS-232-C. The STOP bit is a logical one, or MARK, and is defined as the negative voltage level by RS-232-C. In current loop applications current flow normally indicates a MARK and lack of current a SPACE. The START bit tells the receiver to start assembling a character and allows the receiver to synchronize itself with the transmitter. Since this synchronization only has to last for the duration of the character (the next character will contain a new START bit), this method works quite well assuming a properly designed receiver. One or more STOP bits are added to the end of the character to ensure that the START bit of the next character will cause a transition on the communication line and to give the receiver time to "catch up" with the transmitter if its basic clock happens to be running slightly slower than that of the transmitter. If, on the other hand, the receiver clock happens to be running slightly faster than the transmitter clock, the receiver will perceive gaps between characters but will still correctly decode the data. Because of this tolerance to minor frequency deviations, it is not necessary that the transmitter and receiver clocks be locked to the identical frequency for successful asynchronous communication.

The synchronous format, instead of adding bits to each character, groups characters into records and adds framing characters to the record. The framing characters are generally known as SYN characters and are used by the receiver to determine where the character boundaries are in a string of bits. Since synchronization must be held over a fairly long stream of data, bit synchronization is normally either extracted from the communication channel by the modem or supplied from an external source.

An example of the synchronous and asynchronous formats is shown in Figure 1. The synchronous format shown is fairly typical in that it requires two SYN characters at the start of the message. The asynchronous format, also typical, requires a START bit preceding each character and a single STOP bit following it. In both cases, two 8-bit characters are to be transmitted. In the asynchronous mode 10*n bits are used to transmit n characters and in the synchronous mode 8N + 16 bits are used. For the example shown the asynchronous mode is actually more efficient, using 20 bits versus 32. To transmit a thousand characters in the asynchronous mode, however, takes 10,000 bits versus 8,016 for the synchronous format mode. For long messages the synchronous format becomes much more efficient than the asynchronous format; the crossover point for the examples shown in Figure 1 is eight characters, for which both formats require 80 bits.

In addition to the differences in format between synchronous and asynchronous communication, there are differences with regards to the type of modems that can be used. Asynchronous modems typically employ FSK (Frequency Shift Keying) techniques which simply generate one audio tone for a MARK and another for a SPACE. The receiving modem detects these tones on the telephone...
line, converts them to logical signals, and presents them to the receiving terminal. Since the modem itself is not concerned with the transmission speed, it can handle baud rates from zero to its maximum speed. Synchronous modems, in contrast to asynchronous modems, supply timing information to the terminal and require data to be presented to them in synchronism with this timing information. Synchronous modems, because of this extra clocking, are only capable of operating at certain preset baud rates. The receiving modem, which has an oscillator running at the same frequency as the transmitting modem, phase locks its clock to that of the transmitter and interprets changes of phase as data.

In some cases it is desirable to operate in a hybrid mode which involves transmitting data with the asynchronous format using a synchronous modem. This occurs when an increase in operating speed is required without a change in the basic protocol of the system. This hybrid technique is known as isosynchronous and involves the generation of the start and stop bits associated with the asynchronous format, while still using the modem clock for bit synchronization.

The 8251 USART has been designed to meet a broad spectrum of requirements in the synchronous, asynchronous, and isosynchronous modes. In the synchronous mode the 8251 operates with 5, 6, 7, or 8-bit characters. Even or odd parity can be optionally appended and checked. Synchronization can be achieved either externally via added hardware or internally via SYN character detection. SYN detection can be based on one or two characters which may or may not be the same. The single or double SYN characters are inserted into the data stream automatically if the software fails to supply data in time. The automatic generation of SYN characters is required to prevent the loss of synchronization. In the asynchronous mode the 8251 operates with the same data and parity structures as it does in the synchronous mode. In addition to appending a START bit to this data, the 8251 appends 1, 1½, or 2 STOP bits. Proper framing is checked by the receiver and a status flag set if an error occurs. In the asynchronous mode the USART can be programmed to accept clock rates of 16 or 64 times the required baud rate. Isosynchronous operation is a special case of asynchronous with the multiplier rate programmed as one instead of 16 or 64. Note that X1 operation is only valid if the clocks of the receiver and transmitter are synchronized.

The 8251 USART can transmit the three formats in half or full duplex mode and is double-buffered internally (i.e., the software has a complete character time to respond to a service request). Although the 8251 supports basic data set control signals (e.g., DTR and RTS), it does not fully support the signaling described in EIA-RS-232-C. Examples of unsupported signals are Carrier Detect (CF), Ring Indicator (CE), and the secondary channel signals. In some cases an additional port will be required to implement these signals. The 8251 also does not interface to the voltage levels required by EIA-RS-232-C; drivers and receivers must be added to accomplish this interface.

**BLOCK DIAGRAM**

A block diagram of the 8251 is shown in Figure 2. As can be seen in the figure, the 8251 consists of five major sections which communicate with each other on an internal data bus. The five sections are the receiver, transmitter, modem control, read/write control, and I/O Buffer. In order to facilitate discussion, the I/O Buffer has been shown broken down into its three major subsections: the status buffer, the transmit data/command buffer, and the receive data buffer.

**Receiver**

The receiver accepts serial data on the RxD pin and converts it to parallel data according to the appropriate format. When the 8251 is in the asynchronous mode and it is ready to accept a character.
(i.e., it is not in the process of receiving a character), it looks for a low level on the RxD line. When it sees the low level, it assumes that it is a START bit and enables an internal counter. At a count equivalent to one-half of a bit time, the RxD line is sampled again. If the line is still low, a valid START bit has probably been received and the 8251 proceeds to assemble the character. If the RxD line is high when it is sampled, then either a noise pulse has occurred on the line or the receiver has become enabled in the middle of the transmission of a character. In either case the receiver aborts its operation and prepares itself to accept a new character. After the successful reception of a START bit the 8251 clocks in the data, parity, and STOP bits, and then transfers the data on the internal data bus to the receive data register. When operating with less than 8 bits, the characters are right-justified. The RxRDY signal is asserted to indicate that a character is available.

In the synchronous mode the receiver simply clocks in the specified number of data bits and transfers them to the receiver buffer register, setting RxRDY. Since the receiver blindly groups data bits into characters, there must be a means of synchronizing the receiver to the transmitter so that the proper character boundaries are maintained in the serial data stream. This synchronization is achieved in the HUNT mode.

In the HUNT mode the 8251 shifts in data on the RxD line one bit at a time. After each bit is received, the receiver register is compared to a register holding the SYN character (program loaded). If the two registers are not equal, the 8251 shifts in another bit and repeats the comparison. When the registers compare as equal, the 8251 ends the HUNT mode and raises the SYNDET line to indicate that it has achieved synchronization. If the USART has been programmed to operate with two SYN characters the process is as described above, except that two contiguous characters from the line must compare to the two stored SYN characters before synchronization is declared. Parity is not checked. If the USART has been programmed to accept external synchronization, the SYNDET pin is used as an input to synchronize the receiver. The timing necessary to do this is discussed in the SIGNALS section of this note. The USART enters the HUNT mode when it is initialized into the synchronous mode or when it is commanded to do so by the command instruction. Before the receiver is operated, it must be enabled by the RxE bit (D2) of the command instructions. If this bit is not set the receiver will not assert the RxRDY bit.

Transmitter
The transmitter accepts parallel data from the processor, adds the appropriate framing information, serializes it, and transmits it on the TxD pin. In the asynchronous mode the transmitter always
adds a START bit; depending on how the unit is programmed, it also adds an optional even or odd parity bit, and either 1, 1½, or 2 STOP bits. In the synchronous mode no extra bits (other than parity, if enable) are generated by the transmitter unless the computer fails to send a character to the USART. If the USART is ready to transmit a character and a new character has not been supplied by the computer, the USART will transmit a SYN character. This is necessary since synchronous communications, unlike asynchronous communications, does not allow gaps between characters. If the USART is operating in the dual SYN mode, both SYN characters will be transmitted before the message can be resumed. The USART will not generate SYN characters until the software has supplied at least one character; i.e., the USART will fill 'holes' in the transmission but will not initiate transmission itself. The SYN characters which are to be transmitted by the USART are specified by the software during the initialization procedure. In either the synchronous or asynchronous modes, transmission is inhibited until TXEnable and the CTS input are asserted.

An additional feature of the transmitter is the ability to transmit a BREAK. A BREAK is a period of continuous SPACE on the communication line and is used in full duplex communication to interrupt the transmitting terminal. The 8251 USART will transmit a BREAK condition as long as bit 3 (SBRK) of the command register is set.

Modem Control

The modem control section provides for the generation of RTS and the reception of CTS. In addition, a general purpose output and a general purpose input are provided. The output is labeled DTR and the input is labeled DSR. DTR can be asserted by setting bit 2 of the command instruction; DSR can be sensed as bit 7 of the status register. Although the USART itself attaches no special significance to these signals, DTR (Data Terminal Ready) is normally assigned to the modem, indicating that the terminal is ready to communicate and DSR (Data Set Ready) is a signal from the modem indicating that it is ready for communications.

I/O Control

The Read/Write Control Logic decodes control signals on the 8080 control bus into signals which gate data on and off the USART's internal bus and controls the external I/O bus (DB6–DB7). The truth table for these operations is as follows:

If neither READ or WRITE is a zero, then the USART will not perform an I/O function. READ and WRITE being a zero at the same time is an illegal state with undefined results. The Read/Write Control Logic contains synchronization circuits so that the READ and WRITE pulses can occur at any time with respect to the clock inputs to the USART.

The I/O buffer contains the STATUS buffer, the RECEIVE DATA buffer and the XMIT DATA/CMD buffer as shown in Figure 2. Note that although there are two registers which store data for transfer to the CPU (STATUS and RECEIVE DATA), there is only one register which stores data being transferred to the USART. The sharing of the input register for both transmit data and commands makes it important to ensure that the USART does not have data stored in this register before sending a command to the device. The TxRDY signal can be monitored to accomplish this. Neither data nor commands should be transferred to the USART if TxRDY is low. Failure to perform this check can result in erroneous data being transmitted.

INTERFACE SIGNALS

The interface signals of the 8251 USART can be broken down into two groups — a CPU-related group and a device-related group. The CPU-related signals have been designed to optimize the attachment of the 8251 to a MCS-80™ system. The device-related signals are intended to interface a modem or like device. Since many peripherals (TTY, CRT, etc.) can be obtained with a modem-like interface, the USART has a broad range of applications which do not include a modem. Note that although the USART provides a logical interface to an EIA-RS-232 device, it does not provide EIA compatible drive, and this must be added via circuitry external to the 8251. As an example of a peripheral interface application and to aid in understanding the signal descriptions which follow, Figure 3 shows a system configured to interface with a TTY or CRT.
APPLICATIONS

Figure 3. Terminal Interface
APPLICATIONS

**CPU-Related Signals**

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vcc (26)</td>
<td>+5 Volt Supply</td>
</tr>
<tr>
<td>GND (4)</td>
<td>+5 Volt Common</td>
</tr>
<tr>
<td>CLK (20)</td>
<td>The CLK input generates internal device timing. No external inputs or outputs are referenced to CLK, but the frequency of CLK must be greater than 30 times the Receiver or Transmitter clock inputs for synchronous mode or 4.5 times the clock inputs for an asynchronous mode. An additional constraint is imposed by the electrical specifications (ref. Appendix B) which require the period of CLK be between 0.42 μsec and 1.35 μsec. The CLK input can generally be connected to the Phase 2 (TTL) output of the 8224 clock generator. A high on this input performs a master reset on the 8251. The device returns to the idle mode and will remain there until reinitialized with the appropriate control words.</td>
</tr>
<tr>
<td>WR (10)</td>
<td>A low on this input causes the USART to accept data on the data bus as either a command or as a data character.</td>
</tr>
<tr>
<td>TxRDY (15)</td>
<td>Transmitter Ready. This output signals the CPU that the USART is ready to accept a data character or command. It can be used as an interrupt to the system or, for polled operation, the CPU can check TxRDY using the status read operation. Note, however, that while the TxRDY status bit will be asserted whenever the XMIT DATA/CMD buffer is empty, the TxRDY output will be asserted only if the buffer is empty and the USART is enabled to transmit (i.e., CTS is low and TxEN is high). TxRDY will be reset when the USART receives a character from the program.</td>
</tr>
<tr>
<td>TXE (18)</td>
<td>Transmitter Empty. A high output on this line indicates that the parallel to serial converter in the transmitter is empty. In the synchronous mode, if the CPU has failed to load a new character in time, TXE will go high momentarily as SYN characters are loaded into the transmitter to fill the gap in transmission.</td>
</tr>
<tr>
<td>RXRDY (14)</td>
<td>Receiver Ready. This output goes high to indicate that the 8251 has received a character on its serial input and is ready to transfer it to the CPU. Although the receiver runs continuously, RXRDY will only be asserted if the RXE (Receive Enable) bit in the command register has been set. RXRDY can be connected to the interrupt structure or, for polled operation, the CPU can check the condition of RXRDY using a status read operation. RXRDY will be reset when the character is read by the CPU.</td>
</tr>
</tbody>
</table>

**I/O Signals**

| DB7–DB0 | The DB signals form a three-state bus which can be connected to the CPU data bus. Control, status, and data are transferred on this bus. Note that the CPU always remains in control of the bus and all transfers are initiated by it. |
| CS (11) | Chip Select. A low on this input enables communication between the USART and the CPU. Chip Select should go low when the USART is being addressed by the CPU. |
| C/D (12) | Control/Data. During a read operation this pin selects either status or data to be input to the CPU (high=status, low=data). During a write operation this pin causes the USART to interpret the data on the bus as a command if it is high or as data if it is low. |
| RD (13) | A low on this input causes the USART to gate either status or data onto the data bus. |
SYNDET (16) I/O  Synch Detect. This line is used in the synchronous mode only. It can be either an input or output, depending on whether the initialization program sets the USART for external or internal synchronization. SYNDET is reset to a zero by RESET. When in the internal synchronization mode, the USART uses SYNDET as an output to indicate that the device has detected the required SYN character(s). A high output indicates synchronization has been achieved. If the USART is programmed to operate with double SYN characters, SYNDET will go high in the middle of the last bit of the second SYN character. SYNDET will be reset by a status read operation. When in the external synchronization mode a positive-going input on the SYNDET line will cause the 8251 to start assembling characters on the next falling edge of SYNDET. The high input should be maintained at least for one TxC cycle following this edge.

Device-Related Signals

DTR (24)  O  Data Terminal Ready. This is a general purpose output signal which can be set low by programming a ‘1’ in command instruction bit 1. This signal allows additional device control.

DSR (22)  I  Data Set Ready. This is a general purpose input signal. The status of this signal can be tested by the CPU through a status read. This pin can be used to test device status and is read as bit 7 of the status register.

RTS (23)  O  Request to Send. This is a general purpose output signal equivalent to DTR. RTS is normally used to request that the modem prepare itself to transmit (i.e., establish carrier). RTS can be asserted (brought low) by setting bit 5 in the command instruction. CTS (17)  I  Clear to Send. A low on this input enables the USART to transmit data. CTS is normally generated by the modem in response to a RTS.

RxCl (25)  I  Receiver Clock. This clock controls the data rate of characters to be received by the USART. In the synchronous mode RxCl is equivalent to the baud rate, and is supplied by the modem. In asynchronous mode RxCl is 1, 16, or 64 times the baud rate. The clock division is preselected by the mode control instruction. Data is sampled by the USART on the rising edge of RxCl.

RxD (3)  I  Receiver Data. Characters are received serially on this pin and assembled into parallel characters. RxD is high true (i.e., High = MARK or ONE).

TxC (9)  I  Transmitter Clock. This clock controls the rate at which characters are transmitted by the USART. The relationship between clock rate and baud rate is the same as for RxCl. Data is shifted out of the USART on the falling edge of TxC.

TxO (19)  O  Transmit Data. Parallel characters sent by the CPU are transmitted serially by the USART on this line. TxO is high true (i.e., High = MARK or ONE).

MODE SELECTION

The 8251 USART is capable of operating in a number of modes (e.g., synchronous or asynchronous). In order to keep the hardware as flexible as possible (both at the chip and end product level), these operating modes are selected via a series of control outputs to the USART. These mode control outputs must occur between the time the USART is reset and the time it is utilized for data transfer. Since the USART needs this information to structure its internal logic it is essential to complete the initialization before any attempts are made at data transfer (including reading status).

A flowchart of the initialization process appears in Figure 4. The first operation which must occur following a reset is the loading of the mode control
The mode control register is loaded by the first control output (C/D=1, RD=1, WR=0, CS=0) following a reset. The format of the mode control instruction is shown in Figure 5. The instruction can be considered as four 2-bit fields. The first 2-bit field (D1 D0) determines whether the USART is to operate in the synchronous (00) or asynchronous mode. In the asynchronous mode this field also controls the clock scaling factor. As an example, if D1 and D0 are both ones, the RxC and TxC will be divided by 64 to establish the baud rate. The second field, D3–D2, determines the number of data bits in the character and the third, D5–D4, controls parity generation. Note that the parity bit (if enabled) is added to the data bits and is not considered as part of them when setting up the character length. As an example, standard ASCII transmission, which is seven data bits plus even parity, would be specified as:

```
X X 1 1 1 0 X X
```

The last field, D7–D6, has two meanings, depending on whether operation is to be in the synchronous or asynchronous mode. For the asynchronous mode (i.e., D1 D0 ≠ 00), it controls the number of STOP bits to be transmitted with the character. Since the receiver will always operate with only one STOP bit, D7 and D6 only control the transmitter. In the synchronous mode (D1 D0 = 00), this field controls the synchronizing process. Note that the choice of single or double SYN characters is independent of the choice of internal or external synchronization. This is because even though the receiver may operate with external synchronization logic, the transmitter must still know whether to send one or two SYN characters should the CPU fail to supply a character in time.

Following the loading of the mode instruction the appropriate SYN character (or characters) must be loaded if synchronous mode has been specified. The SYN character(s) are loaded by the same control output instruction used to load the mode instruction. The USART determines from the mode instruction whether no, one, or two SYN characters are required and uses the control output to load SYN characters until the required number are loaded.

At completion of the load of SYN characters (or after the mode instruction in the asynchronous mode), a command character is issued to the USART. The command instruction controls the operation of the USART within the basic framework established by the mode instruction. The format of the command instruction is shown in

---

**Figure 4. Initialization Flowchart**

**Figure 5. Mode Instruction Format**
Figure 6. Note that if, as an example, the USART is waiting for a SYN character load and instead is issued an internal reset command, it will accept the command as a SYN character instead of resetting. This situation, which should only occur if two independent programs control the USART, can be avoided by outputting three all zero characters as commands before issuing the internal reset command. The USART indicates its state in a status register which can be read under program control. The format of the status register read is shown in Figure 7.

When operating the receiver it is important to realize that RxE (bit 2 of the command instruction) only inhibits the assertion of RxRDY; it does not inhibit the actual reception of characters. Because the receiver is constantly running, it is possible for it to contain extraneous data when it is enabled. To avoid problems this data should be read from the USART and discarded. The read should be done immediately following the setting of Receive Enable in the asynchronous mode, and following the setting of Enter Hunt in the synchronous mode. It is not necessary to wait for RxRDY before executing the dummy read.

Figure 6. Command Instruction Format

Figure 7. Status Register Format
PROCESSOR DATA LINK

The ability to change the operating mode of the USART by software makes the 8251 an ideal device to use to implement a serial communication link. A terminal initially configured with a simple asynchronous protocol can be upgraded to a synchronous protocol such as IBM Binary Synchronous Communication by a software only upgrade. In order to demonstrate the use of the 8251 USART, the remainder of this document will describe the implementation of an interrupt-driven, full duplex communication link on the Intel MDS™ system. With minor modifications, the program developed could be used on the Intel SBC-80/10™ OEM card, thus implementing a data link between the two systems. Such a facility can be used to down-load programs, run diagnostics, and maintain common data bases in multiprocessor systems.

The factors which must be considered in the design of such a link include the desired transmission rate and format, the error checking requirements, the desirability of full duplex operation, and the physical implementation of the link. The basic requirement of the system described here is that it allow a number of sources at reasonable cost. These modems are also available in acoustically coupled versions which do not require permanent installation on the telephone network. An additional constraint is that the modem used on the switched network be readily available and inexpensive. These requirements led to the choice of a modem such as the Bell 103A to implement the link. These modems, which support full duplex communication at up to 300 baud, are readily available from a number of sources at reasonable cost. These modems are also available in acoustically coupled versions which do not require permanent installation on the telephone network. Interface to the 103A modem is accomplished with nine wires: Protective Ground, Signal Ground, Transmitted Data, Received Data, Clear to Send, Data Set Ready, Data Terminal Ready, Carrier Detector, and Ringing Indicator.

The utilization of the interface signals to the modem is as follows:

Protective Ground
Protective Ground is used to bond the chassis ground of the modem to that of the terminal.

Signal Ground
Signal Ground provides a common ground reference between the modem and the terminal.

Transmitted Data
Transmitted Data is used to transfer serial data from the terminal to the modem.

Received Data
Received Data is used to transfer serial data from the modem to the terminal.

Clear to Send
Clear to Send indicates that the modem has established a connection with a remote modem and is ready to transmit data.

Data Set Ready
Data Set Ready indicates that the modem is connected to the telephone line and is in the data mode.

Data Terminal Ready
Data Terminal Ready is a signal from the terminal which permits the modem to enter the data mode.

Carrier Detector
Carrier Detector is identical to Clear to Send in the 103 modem and will not be used in this interface.

Ringing Indicator
Ringing Indicator indicates that the modem is receiving a ringing signal from the telephone system. This signal will not be used in the interface, since it is possible for the terminal to assert Data Terminal Ready whenever it is ready for the modem to “answer the telephone”. The modem uses Data Set Ready to indicate that it has answered the call.

A block diagram showing the connections between the MDS and the SBC-80/10 through the modems is shown in Figure 8. Figure 9 shows the portion of the MDS monitor board devoted to the USARTs and Figure 10 shows the equivalent section of the SBC-80/10 board. Note that several signals on the MDS monitor board do not have the proper EIA defined voltage levels, and for this reason the adapter shown in Figure 11 was added to the MDS. The 390 pF capacitor was added to the 1488 driver to bring the rise time within EIA imposed limits of 30 volts/μsec. In Figure 7 the signal labels within the MDS and SBC-80/10 blocks correspond to the labels on the schematics, the signal labels within the modem blocks correspond to EIA conventions, and the signal labels on the wires between the blocks are abbreviations for the English language names of the signals.

As an example of how the USART clocks can be generated, circuits A27, A16, and A15 of Figure 9 form a divider of the OSC signal. The OSC signal has a frequency of 18.432 MHz and is generated by the 8224 which generates system timing for the 8080A. The 18.432 MHz signal results in a state time of 488 ns versus the normal 500 ns for the 8080A. (This does not violate 8080A specifications.) The 18.432 MHz signal can be divided by
Before the software design of the system could be undertaken, it was necessary to decide whether service requests from the USART would be handled on a polled or interrupt driven mode. Polled operation normally results in more compact code but it requires that whatever programs are running concurrently with a transmission or reception must periodically either check the status of the USART or call a routine that does. Since it was not possible to determine what program might be running during a receive or transmit operation, it was decided to operate in an interrupt driven mode.

The program which operates the 8251 must be instructed as to what data it should transmit or receive from some other program resident in the 8080 system. To facilitate the discussion of the operation of the software, the following definitions will be made:

**USRUN** is the program which controls the operation of the 8251.** USER** is a program which utilizes USRUN in order to effect a data transmission. **USER** passes commands and parameters to USRUN by means of the control block shown in Figure 12. The first byte of the block contains the command which **USER** wants **USRUN** to execute. Valid contents of this byte are “C” which causes **USRUN** to initialize itself and the 8251, “R” which causes the execution of the data input (or READ) operation, and “W” which causes a data output (WRITE) operation. The second byte of the control block is used by **USRUN** to inform **USER** of the status of the requested operation. The third and fourth bytes specify the starting address of a buffer set up by **USER** which contains the data for a transmit operation or which will be used by **USRUN** to store received data. The fifth and sixth bytes are concatenated to form a positive binary

---

**APPLICATIONS**

---

![System Block Diagram](image)

**Figure 8. System Block Diagram**

![EIA Adapter](image)

**Figure 9. EIA Adapter**

---

30 and then 64 to give a 9600 baud communication standard. The 9600 baud signal can be further divided to give 4800, 2400, 1200, 600, and 300 baud signals. The 1200 baud signal can be divided by 11 to give a 109.1 baud signal which is within 1% of the 110 baud standard signal rate. Note that because of constraints on the CLK input 9600 baud operation is not possible in the X64 mode. The divide by 64 can be accomplished by dividing by 4 with a counter and then 16 within the USART.

In order to keep the system as general purpose as possible, it was decided to transmit 8-bit data characters with an appended odd parity bit. Having a full 8-bit byte available for data enables the transmission of codes such as ASCII (which is 7-level with an additional parity bit) to be transmitted and received transparently in the system. Also, of course, it allows 8-bit bytes from the 8080A memory to be transferred in one transmission character. If error checking beyond the parity check is required, it could be added to the data record to be transmitted in the form of redundant check characters.
Figure 10. SBC 80/10 Serial I/O
Figure 11. MDS Monitor Module
number which specifies how many bytes of data USER wants transferred. The seventh and eighth bytes are concatenated and used by USRUN to count the number of bytes that have been transferred. When the required number of characters have been transferred, or if USRUN terminates a READ or WRITE due to an abnormal condition, then USRUN calls a subroutine at an address defined by the ninth and tenth bytes of the command block. This subroutine, which is provided by USER, must determine the state of the process and then take appropriate action.

Since USRUN must be capable of operation in a full duplex mode (i.e., be able to receive and transmit simultaneously), it keeps the address of two control blocks; one for a READ operation and one for a WRITE. The address of the controlling command block is kept in RAM locations labeled RCBA for the READ operation and TCBA for the WRITE operation. If RCBA (Receive Control Block Address) or TCBA (Transmit Control Block Address) is zero, it indicates that the corresponding operation is in an idle status.

Flowcharts of USRUN appear in Figure 13 and the listings appear in Figure 14. The first section of the flowcharts (Figures 13.1 and 13.2) consists of two subroutines which are used as convenient tools for operating on the control blocks. These routines are labeled LOADA and CLEAN. LOADA is entered with the address of a control block in registers H and L. Upon return registers D and E have been set equal to the address in the buffer which is the target of the next data transfer (i.e., D,E = BAD+CCT); and CCT (transferred byte count) has then been incremented. In addition, the B register is set to zero if the number of bytes that have been transferred is equal to the number requested (i.e., CCT = RCT). CLEAN, the second routine, is also entered with the address of a command block in the H and L registers. In addition, the Accumulator holds the status which will be placed in the STATUS byte of the command block. On exit the STATUS byte has been updated and the address of the completion routine has been placed in H and L.

Upon interrupt, control of the MCS-80 system is transferred to VECTOR (Figure 13.3). Vector is a program which saves the state of the system, gets the status of the USART and jumps to the RISR (Receive Interrupt Service Routine) or the TISR (Transmit Interrupt Service Routine), depending on which of the two ready flags is active. If neither ready flag is active, VECTOR restores the status of the running program, enables interrupts, and returns. (Interrupts are automatically disabled by the hardware upon an interrupt.) This exit from VECTOR, which is labeled VOUT, is used from other
portions of USRUN if return from the interrupt mode is required.

In addition to handling normal data transfers, TISR (Figure 13.4) checks a location in memory named TCMD in order to determine if the receive program wishes to send a command to the USART. Since the transmit data and command must share a buffer within the USART, any command output must occur when TxRDY is asserted. If TCMD is zero, TISR proceeds with the data transfer. If TCMD is non-zero, TISR calls TUTE (Transmit Utility, Figure 13.5) which, depending on the value...
in TCMD, turns off the receiver, turns on the receiver, or clears error conditions. Note that the error flags (parity, framing, and overrun) are always cleared by the software when the receiver is first enabled.

The flowchart of the RISR is shown in Figure 13.6. Note that in addition to terminating whenever the required number of characters have been received, the RISR also terminates if one of the error flags becomes set or if the received character matches a character found in a table pointed to by the label ETAB. This table, which starts at ETAB and continues until an all “ones” entry is found, can be used by USER to define special characters, such as EOT (End Of Transmission), which will terminate a READ operation. The remainder of Figure 13 (13.7) shows the decoding of the commands to USRUN. The listings also include a test USER which exercises USRUN. This program sets up a 256-byte transmit buffer and transfers it to a similar input buffer by means of a local loop. When both the READ and WRITE operations are complete, the test USER checks to insure that the two buffers are identical. If the buffers differ, the MDS monitor is called; if the data is correct, the test is repeated.

CONCLUSION

The 8251 USART has been described both as a device and as a component in a system. Since not only modems but also many peripheral devices have a serial interface, the 8251 is an extremely useful component in a microcomputer system. A particular advantage of the device is that it is capable of operating in various modes without requiring hardware modifications to the system of which it is a part. As with any complex subsystem, however, the 8251 USART must be carefully applied so that it can be utilized to full advantage in the overall system. It is hoped that this application note will aid in the designer in the application of the 8251 USART. As a further aid to the application of the 8251, the appendix of this document includes a list of design hints based on past experience with the 8251.

---

**Figure 13.6. Receive Interrupt Service Routine**
Figure 13.7. URUN Command Decode
APPLICATIONS

Figure 14. Program Listing

;******
; SYSTEM ORIGIN STATEMENT
;******
4000 ORG 4000H

;******
; DATA STORAGE FOR TEST USER
;******
4000 BUFIN: DS 100H ;INPUT BUFFER
4100 BUFOUT: DS 100H ;OUTPUT BUFFER
4200 5200 RBLOCK: DB 'R',00H ;RECEIVE CONTROL BLOCK
4202 0040 RBAD: DW BUFIN
4204 FF00 RRCT: DW OFFH
4206 0000 RCCT: DW 00H
4208 1742 RCRA: DW RCR
420A 5700 TBLOCK: DB 'W',00H ;TRANSMIT CONTROL BLOCK
420C 0041 TBAD: DW BUFOUT
420E FF00 TRCT: DW OFFH
4210 0000 TCCT: DW 00H
4212 2742 TCRA: DW TCR
4214 4300 GBLOCK: DB 'C',00H
4216 00 FLAG: DB 00H

;******
; COMPLETION ROUTINES
;******
4217 AF RCR: XRA A ;CLEAR A
4218 323B42 STA RCBA ;TURN OFF RECEIVE
4219 323C42 STA RCBA+1
421E 3A1642 LDA FLAG ;GET FLAG
4221 E60F ANI OFH ;CLEAR UPPER FOUR BITS
4223 321642 STA FLAG ;RESTORE FLAG
4226 C9 RET
4227 AF TCR: XRA A ;CLEAR A
4228 323942 STA TCBA ;TURN OFF TRANSMIT
4229 323A42 STA TCBA+1
422E 3A1642 LDA FLAG ;GET FLAG
4231 E60F ANI OFH ;CLEAR LOWER FOUR BITS
4233 321642 STA FLAG ;RESTORE FLAG
4236 C9 RET ;THEN RETURN
APPLICATIONS

;*****
; SYSTEM EQUATES
;*****

00F5  USTAT  EQU  0F5H  ;USART STATUS ADDRESS
00F5  USCMD  EQU  0F5H  ;USART CMD ADDRESS
00F4  USDAI  EQU  0F4H  ;USART DATA INPUT ADDRESS
00F4  USDAO  EQU  0F4H  ;USART DATA OUTPUT ADDRESS
0000  GSTAT  EQU  00H  ;GOOD STATUS
00FF  BSTAT  EQU  OFFH  ;BAD STATUS
0001  CEND   EQU  01H  

;*****
; SYSTEM DATA TABLE
;*****

4237  00  LCMD:  DB  00H  ;CURRENT OPERATING COMMAND
4238  00  TCMD:  DB  00H  ;IF NON ZERO A COMMAND TO BE SENT
4239  0000  TCBA:  DW  00H  ;ADDRESS OF XMIT CBLOCK
423B  0000  RCBA:  DW  00H  ;ADDRESS OF RECEIVE CBLOCK
423D  FF  MTAB:  DB  OFFH  ;END CHARACTER TABLE
LOAD ADDRESS ROUTINE
LOADA IS ENTERED WITH THE ADDRESS OF A CONTROL BLOCK IN H,L. ON EXIT D,E CONTAINS THE ADDRESS WHICH IS THE TARGET OF THE NEXT DATA TRANSFER (BAD+CCNT) AND B HAS BEEN SET TO ZERO IF THE REQUESTED NUMBER OF TRANSFERS HAS BEEN ACCOMPLISHED. CCNT IS INCREMENTED AFTER THE TARGET ADDRESS HAS BEEN CALCULATED.

LOADA: INX H ;D,E GETS BUFFER ADDRESS
INX H
MOV E,M
INX H
MOV D,M ;DONE
INX H
INX H
INX H
MOV C,M
INX H
MOV B,M ;DONE
XCHG B
DAD B
XCHG B ;DONE
INX B ;CCNT GETS INCREMENTED
INX B
MOV M,B
DCX H
MOV M,C ;DONE
DCX B ;DOES OLD CCNT=RCNT?
DCX H
MOV A,M
DCX H
MOV A,M
SUB B
MOV B,A
SUB B
RNZ ;NO-RETURN WITH B NOT ZERO
MOV A,M
DCX H
MOV A,M
SUB C
MOV B,A
RET ;RETURN WITH B=0 IF RCNT=CCNT
APPLICATIONS

;*****
;
; CLEAN-UP ROUTINE
; CLEAN IS ENTERED WITH THE ADDRESS OF A CONTROL
; BLOCK IN H,L AND A NEW STATUS TO BE
; ENTERED INTO IT IN A. ON EXIT THE ADDRESS OF THE
; CONTROL BLOCK IS IN D,E; THE STATUS OF THE BLOCK
; HAS BEEN UPDATED; AND THE ADDRESS OF THE COMPLETION
; ROUTINE IS IN H,L.
;*****

425B 5D    CLEAN:  MOV E,L ;SAVE THE ADDRESS OF THE COMMAND BLOCK
425C 54      MOV D,H
425D 23      INX H ;POINT AT STATUS
425E 77      MOV M,A ;SET STATUS EQUAL TO A
425F 010700   LXI B,7 ;SET INDEX TO SEVEN
4262 09      DAD B ;POINT AT COMPLETION ADDRESS
4263 7E      MOV A,M ;GET LOWER ADDRESS
4264 23      INX H ;POINT AT UPPER ADDRESS
4265 66      MOV H,M ;H GETS HIGH ADDRESS BYTE
4266 6F      MOV L,A ;L GETS LOW ADDRESS BYTE
4267 C9      RET

;*****
;
; INTERRUPT VECTOR ROUTINE
; VECTOR SAVES THE STATUS OF THE RUNNING PROGRAM
; THEN READS THE STATUS OF THE USART TO DETERMINE
; IF A RECEIVE OR TRANSMIT INTERRUPT OCCURRED.
; VECTOR THEN CALLS THE APPROPRIATE SERVICE ROUTINE.
; IF NEITHER INTERRUPTS OCCURRED THEN VECTOR RESTORES
; THE STATUS OF THE RUNNING PROGRAM. THE SERVICE
; ROUTINES USE THE EXIT CODE, LABELED VOUT, TO EFFECT
; THEIR EXIT FROM INTERRUPT MODE.
;*****

4268 F5    VECTOR:  PUSH PSW ;PUSH STATUS INTO THE STACK
4269 C5      PUSH B
426A D5      PUSH D
426B E5      PUSH H
426C DBF5    IN USTAT ;GET USART ADDRESS
426E DBFA    IN OFAH ;MDS-GET MONITOR CARD INT. STATUS
4270 0F      RRC ;ROTATE TWO PLACES
4271 0F      RRC ;SO THAT CARRY=RXRDY
4272 DA8842   JC RISR ;IF RXRDY GO TO SERVICE ROUTINE
4275 07      RLC ;IF NOT ROTATE BACK
4276 07      RLC ;LEAVING TXRDY IN CARRY
4277 DAD442   JC TISR ;IF TXRDY THEN GO TO SERVICE ROUTINE
427A 3EFC    MVI A,OFCH ;MDS-CLEAR OTHER LEVEL THREE INTERRUPTS
427C D3F3    OUT OF3H ;MDS
427E E1      VOUT: POP H ;ELSE EXIT FROM INTERRUPT MODE
427F D1      POP D
4280 C1      POP B
4281 3E20    MVI A,20H ;MDS-RESTORE CURRENT LEVEL
4283 D3FD    OUT OFDH ;MDS
4286 FB      EI ;ENABLE INTERRUPTS
4287 C9      RET
APPLICATIONS

; RECVINT: RECEIVE INTERRUPT SERVICE ROUTINE;
; RISR PROCESSES A RECEIVE INTERRUPT
; AT THE END OF RECEIVE THE USER SUPPLIED
; COMPLETION ROUTINE IS CALLED AND THEN AN
; EXIT IS TAKEN THROUGH VOUT OF THE
; VECTOR

4288 2A3B42  RISR:  LHLD  RCBA
428B 3B02    MVI  A,82H  ;MDS-CLEAR RECEIVE INTERRUPT
428D D3F3    OUT  OF3H  ;MDS
428F 2C    INR  L
4290 2D    DCR  L
4291 C29942  JNZ  RISRB
4294 24    INR  H
4295 25    DCR  H
4296 CA7E42  JZ  VOUT
4299 CD3E42  RISRB: CALL  LOADA ;READY-SET UP ADDRESS
429C DBF4    IN  USDAI ;GET INPUT DATA
429E 12    STAX  D ;AND PUT IN THE BUFFER
429F 4F    MOV  C,A ;SAVE INPUT DATA IN C
42A0 DBF5    IN  USTAT ;GET STATUS AGAIN
42A2 E638    ANI  38H ;MASK FOR ERROR FIELD
42A4 C2B942  JNZ  RISRE ;NOT ZERO-TAKE ERROR EXIT
42A7 04    INR  B ;B WAS 00 IF DONE
42A8 05    DCR  B
42A9 C2BE42  JNZ  EXCHAR ;NOT DONE-EXIT
42AC 3E00    MVI  A,GSTAT ;A GETS GOOD STATUS
42AE 217E42  RISRA: LXI  H,VOUT ;GET RETURN ADDRESS
42B1 B5    PUSH  H ;AND PUSH IT INTO THE STACK
42B2 2A3B42  LHLD  RCBA ;POINT H,L AT THE CMD BLOCK
42B5 CD5B42  CALL  CLEAN ;CALL CLEANUP ROUTINE
42B8 E9    PCHL ;EFFECTIVELY CALLS COMPLETION ROUTINE
42B9 3EFF    RISRE: MVI  A,BSTAT ;A GETS BAD STATUS
42BB C3AE42  JMP  RISRA ;OTHERWISE EXIT IS NORMAL
42BE 213D42  EXCHAR: LXI  H,MTAB ;TEST CHARACTER AGAINST EXIT TABLE
42C1 7E    EXA: MOV  A,M
42C2 F7FF    CPI OFFH ;END OF TABLE
42C4 C7E42  JZ  VOUT
42C7 B9    CMP  C
42C8 CACF42  JZ  PEND ;MATCH-TERMINATE READ
42CB 23    INX  H
42CC C3C142  JMP  EXA
42CF 3E01  PEND:  MVI  A,CEND
42D1 C3AE42  JMP  RISRA
APPLICATIONS

;*****
; TRANSMIT INTERRUPT SERVICE ROUTINE
; TISR PROCESSES TRANSMITTER INTERRUPTS
; WHEN THE END OF A TRANSMISSION IS
; DETECTED THE USER SUPPLIED COMPLETION
; ROUTINE IS CALLED AND THEN AN EXIT IS
; TAKEN THROUGH VOUT OF VECTOR
;*****

42D4 3A3842 TISR: LDA TCMD ;GET POTENTIAL COMMAND
42D7 B7 ORA A ;DESIGNATE ON IT
42D8 C40443 CNZ TUTE ;DO UTILITY COMMAND
42DB 3E81 MVI A081H ;MDS-CLEAR XMIT INTERRUPTS
42DD D3F3 OUT OF3H ;MDS
42DF 2A3942 LHLD TCBA
42E2 2C INR L ;MAKE SURE HAVE VALID CONTROL BLOCK
42E3 2D DCR L
42E4 C2EC42 JNZ TISRA ;GOOD
42E7 24 INR H
42E8 25 DCR H
42E9 CA7E42 JZ VOUT ;NON VALID BLOCK (H,L=0)
42EC CD3E42 TISRA: CALL LOADA ;SET UP ADDRESS
42EF 1A LDAX D ;GET DATA FROM BUFFER
42F0 D3F4 OUT USDAO ;AND OUTPUT IT
42F2 04 INR B ;B WAS 00 IF DONE
42F3 05 DCR B
42F4 C27E42 JNZ TISRA ;NOT DONE-EXIT FROM SERVICE ROUTINE
42F7 217E42 LXI H,VOUT ;SET UP RETURN ADDRESS
42FA E5 PUSH H ;AND PUSH IT INTO THE STACK
42FB 3E00 MVI A,GSTAT ;A GETS GOOD STATUS
42FD 2A3942 LHLD TCBA ;POINT H,L AT COMMAND BLOCK
4300 CD5B42 CALL CLEAN ;CALL CLEANUP ROUTINE
4303 E9 PCHL ;CALL COMPLETION ROUTINE
4304 FE01 OUT USCMD ;RETURN WILL BE TO VOUT
4306 CA2443 TUTE: CPI 01 ;RECEIVER OFF
4309 FE02 CPI 02 ;RECEIVER ON
430B CA1443 JZ TUTE2 ;CLEAR ERRORS.
430E FE03 CPI 03
4310 CA1C43 JZ TUTE3
4313 C9 RET
4314 3A3742 TUTE2: LDA LCMD
4317 F604 ORI 04
4319 323742 STA LCMD
431C 3A3742 TUTE3: LDA LCMD
431F F610 ORI 10H
4321 D3F5 TUTE4: OUT USCMD
4323 C9 RET
4324 3A3742 TUTE1: LDA LCMD
4327 E6FB ANI 0FBH
4329 323742 STA LCMD
432C C32143 JMP TUTE4
APPLICATIONS

; "" ;
; ;
; ;
; ;
; ;
; ;

432F 1A USRUN: LDAX D ; GET THE CMD FROM THE BLOCK
4330 FE43 CPI 'C' ; IS IT A CLEAR COMMAND?
4332 CAH043 JZ UCLEAR ; YES GO TO CLEAR ROUTINE
4335 FE52 CPI 'R' ; IS IT A READ COMMAND?
4337 CA5D43 JZ UREAD ; YES-GO TO READ ROUTINE
433A FE57 CPI 'W' ; IS IT A WRITE COMMAND?
433C CA9D43 JZ UWRITE ; GO TO WRITE ROUTINE
433F C9 RET ; NOT A GOOD COMMAND-RETURN

4340 F3 UCLEAR: DI ; DISABLE INTERRUPTS
4341 AF XRA A ; CLEAR A
4342 D3F5 OUT USCMD ; OUTPUT THREE TIMES TO ENSURE
4344 D3F5 OUT USCMD ; THAT THE USART IS IN A KNOWN STATE
4346 D3F5 OUT USCMD
4348 3E40 MVI A,40H ; CODE TO RESET USART
434A D3F5 OUT USCMD ; OUTPUT ON CMD CHANNEL
434C 3E5E MVI A,05EH ; CE IMPLIES ASYN MODE (X16)
434E D3F5 OUT USCMD ; OUTPUT ON CMD CHANNEL
4350 AF XRA A ; CLEAR A, SET ZERO
4351 213942 LXI H,Tcba ; CLEAR TCBA AND RCBA
4354 77 MOV M,A
4355 23 INX H
4356 77 MOV M,A
4357 23 INX H
4358 77 MOV M,A
4359 23 INX H
435A 77 MOV M,A
435B FB EI ; ENABLE INTERRUPTS
435C C9 RET ; AND RETURN TO USER

435D 213B42 UREAD: LXI H,RcBA ; CHECK READ IDLE
4360 7E MOV A,M
4361 B7 ORA A
4362 C26B43 JNZ UROUT
4365 23 INX H
4366 7E MOV A,M
4367 B7 ORA A
4368 CA7743 JZ URDA ; READ IS IDLE-PROCEED
436B 3EFE UROUT: MVI A,0FEH ; ALREADY RUNNING-ERROR STATUS
436D 217643 LXI H,URdb ; SET UP RETURN ADDRESS
4370 E5 PUSH H ; PUSH IT INTO STACK
4371 EB XCHG ; H GETS COMMAND BLOCK ADDRESS
4372 C65B42 CALL CLEAN ; CALL CLEANUP ROUTINE
4375 E9 PCHL ; EFFECTIVELY CALLS END ROUTINE
4376 C9 URDB: RET ; RETURN TO USER

4377 EB URDA: XCHG ; H GETS COMMAND BLOCK ADDRESS
4378 223B42 SHLD RCBA ; RCBA GETS COMMAND BLOCK ADDRESS
437B 3A3742 LDA LCmD ; GET LAST COMMAND
437E F616 ORI 16H ; SET RXE AND DTR AND RESET ERRORS
4380 323742 STA LCMD ; AND RETURN TO MEMORY
4383 0F RRC ; SET CARRY EQUAL TO TXE
APPLICATIONS

4384 D28C43 JNC URDC
4387 3E02 MVI A,2
4389 323842 STA TCMD
438C 07 URDC: RLC
438D D3F5 OUT USCMD ;OUTPUT CMD
438F DBF4 IN USDAI ;CLEAR USART OF LEFT OVER CHARACTERS
4391 DBF4 IN USDAI
4393 3E82 MVI A,82H ;MDS-CLEAR RECEIVE INTERRUPT
4395 D3F3 OUT OF3H ;MDS
4397 3EF6 MVI A,OF6H ;MDS-ENABLE LEVEL THREE
4399 D3FC OUT OFCH ;MDS
439B FB EI ;ENABLE INTERRUPTS
439C C9 RET ;RETURN TO USER

439D 213942 UWRITE: LXI H,TCBA ;CHECK WRITE IDLE
43A0 7E MOV A,M
43A1 B7 ORA A
43A2 C26B43 JNZ UROUT ;BUSY-EXIT
43A5 23 INX H
43A6 7E MOV A,M
43A7 C26B43 JNZ UROUT ;BUSY-EXIT
43AA EB XCHG ;OK-H GETS COMMAND BLOCK ADDRESS
43AB 223942 SHLD TCBA ;TCBA GETS COMMAND BLOCK ADDRESS
43AE 3A3742 LDA LCMD ;GET LAST COMMAND
43B1 F623 ORI O23H ;SET RTS,DIR, AND TXEN
43B3 323742 STA LCMD
43B6 D3F5 OUT USCMD
43B8 3EF6 MVI A,OF6H ;MDS-ENABLE LEVEL THREE INTERRUPTS
43B9 D3FC OUT OFCH ;MDS
43BC FB EI ;ENABLE SYSTEM INTERRUPTS
43BD C9 RET ;AND RETURN
APPLICATIONS

;*****
; USER IS A TEST PROGRAM WHICH EXERCISES USRUN
; *****

43BE 3EC3 USER: MVI A,OC3H ;MDS-SET INTERRUPT VECTOR
43C0 321800 STA 018H
43C3 216842 LXI H,VECTOR
43C6 221900 SHLD 019H
43C9 3B43 MVI A, C' ';SET GENERAL BLOCK TO A 'C'
43CB 111442 LXI D,GBLOCK
43CE 12 STAX D
43CF CD2F43 CALL USRUN
43D2 210040 LXI H,BUFIN ;CLEAR INPUT BUFFER
43D5 AF XRA A
43D6 77 MOV M,A
43D7 2C INR L
43D8 C2D643 JNZ $-2
43DB 210041 LXI H,BUFOUT ;INITIALIZE OUTPUT BUFFER
43DE 75 MOV M,L
43DF 2C INR L
43E0 C2DE43 JNZ $-2
43E3 65 MOV H,L ;REINITIALIZE CONTROL BLOCKS
43E4 2E52 MVI L,'R'
43E6 220042 SHLD RBLOCK
43E9 2E57 MVI L,'W'
43EB 220A42 SHLD TBLOCK
43EC 6C MOV L,H
43EF 220642 SHLD RCCT
43F2 221042 SHLD TCCT
43F5 110042 LXI D, RBLOCK ;START READ
43F8 CD2F43 CALL USRUN
43FA 110A42 LXI D, TBLOCK ;START WRITE
43FC CD2F43 CALL USRUN
4401 3EFF MVI A, OFFH ;LOOP WAITING COMPLETION
4403 321642 STA FLAG ;FLAG WILL BE SET BY COMPLETION ROUTINES
4406 3A1642 LDA FLAG
4409 B7 ORA A
440A C20644 JNZ $-4
440D 210040 LXI H,BUFIN ;TEST INPUT BUFFER=OUTPUT BUFFER
4410 7E COMLP: MOV A,M
4411 24 INR H
4412 BE CMP M
4413 C21E44 JNZ COMER
4416 25 DCR H
4417 2C INR L
4418 C21044 JNZ COMLP
441B C3BE43 JMP USER ;GOOD COMPARE-REPEAT TEST
\441E CT COMER: RST 0 ;ERROR-RETURN TO MONITOR

0000 END
<table>
<thead>
<tr>
<th>APPLICATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSTAT 00FF</td>
</tr>
<tr>
<td>CLEAN 425B</td>
</tr>
<tr>
<td>EXCHA 42BE</td>
</tr>
<tr>
<td>LCDM 4237</td>
</tr>
<tr>
<td>RBAD 4202</td>
</tr>
<tr>
<td>RCR 4217</td>
</tr>
<tr>
<td>RISR 4299</td>
</tr>
<tr>
<td>TBLOC 420A</td>
</tr>
<tr>
<td>TCR 4227</td>
</tr>
<tr>
<td>TRCT 420E</td>
</tr>
<tr>
<td>TUTE3 431C</td>
</tr>
<tr>
<td>URDB 4376</td>
</tr>
<tr>
<td>USCMD 00F5</td>
</tr>
<tr>
<td>USRUN 432F</td>
</tr>
<tr>
<td>VOUT 427E</td>
</tr>
</tbody>
</table>
1. Output of a command to the USART destroys the integrity of a transmission in progress if timed incorrectly.

Sending a command into the USART will overwrite any character which is stored in the buffer waiting for transfer to the parallel-to-serial converter in the device. This can be avoided by waiting for TxRDY to be asserted before sending a command if transmission is taking place. Due to the internal structure of the USART, it is also possible to disturb the transmission if a command is sent while a SYN character is being generated by the device. (The USART generates a SYN if the software fails to respond to TxRDY.) If this occurrence is possible in a system, commands should be transferred only when a positive-going edge is detected on the TxRDY line.

2. RxE only acts as a mask to RxRDY; it does not control the operation of the receiver.

When the receiver is enabled, it is possible for it to already contain one or two characters. These characters should be read and discarded when the RxE bit is first set. Because of these extraneous characters the proper sequence for gaining synchronization is as follows:

1. Disable interrupts
2. Issue a command to enter hunt mode, clear errors, and enable the receiver (EH,ER,RxE=1)
3. Read USART data (it is not necessary to check status)
4. Enable interrupts

The first RxRDY that occurs after the above sequence will indicate that the SYN character or characters have been detected and the next character has been assembled and is ready to be read.

3. Loss of CTS or dropping TxEnable will immediately clamp the serial output line.

TxEnable and RTS should remain asserted until the transmission is complete. Note that this implies that not only has the USART completed the transfer of all bits of the last character, but also that they have cleared the modem. A delay of 1 msec following a proper occurrence of TxEmpty is usually sufficient (see item 4). An additional problem can occur in the synchronous mode because the loss of TxEnable clamps the data in at a SPACE instead of the normal MARK. This problem, which does not occur in the asynchronous mode, can be corrected by an external gate combining RTS and the serial output data.

4. Extraneous transitions can occur on TxEmpty while data (including USART generated SYNs) is transferred to the parallel-to-serial converter.

This situation can be avoided by ensuring that TxEmpty occurs during several consecutive status reads before assuming that the transmitter is truly in the empty state.

5. A BREAK (i.e., long space) detected by the receiver results in a string of characters which have framing errors.

If reception is to be continued after a BREAK, care must be taken to ensure that valid data is being received; special care must be taken with the last character perceived during a BREAK, since its value, including any framing error associated with it, is indeterminate.
8251 PROGRAMMABLE COMMUNICATION INTERFACE
Using the 8273 SDLC/HDLC Protocol Controller

John Beaton
Microcomputer Applications
Using the 8273 SDLC/HDLC Protocol Controller

Contents

INTRODUCTION

SDLC/HDLC OVERVIEW

BASIC 8273 OPERATION

HARDWARE ASPECTS OF THE 8273

   CPU Interface
   Modem Interface

SOFTWARE ASPECTS OF THE 8273

   Command Phase Software
   Execution Phase Software
   Result Phase Software

8273 COMMAND DESCRIPTION

   Initialization/Configuration Commands
   Operating Mode Register
   Serial I/O Mode Register
   Data Transfer Mode Register
   One Bit Delay Register

   Receive Commands
   General Receive
   Selective Receive
   Selective Loop Receive
   Receive Disable

   Transmit Commands
   Transmit Frame
   Loop Transmit
   Transmit Transparent

   Abort Commands
   Reset Commands
   Modem Control Commands

HDLC CONSIDERATIONS

LOOP CONFIGURATIONS

APPLICATION EXAMPLE

CONCLUSION

APPENDIX A
INTRODUCTION
The Intel 8273 is a Data Communications Protocol Controller designed for use in systems utilizing either SDLC or HDLC (Synchronous or High-Level Data Link Control) protocols. In addition to the usual features such as full duplex operation, automatic Frame Check Sequence generation and checking, automatic zero bit insertion and deletion, and TTL compatibility found on other single component SDLC controllers; the 8273 features a frame level command structure, a digital phase locked loop, SDLC loop operation, and diagnostics.

The frame level command structure is made possible by the 8273's unique internal dual processor architecture. A high-speed bit processor handles the serial data manipulations and character recognition. A byte processor implements the frame level commands. These dual processors allow the 8273 to control the necessary byte-by-byte operation of the data channel with a minimum of CPU (Central Processing Unit) intervention. For the user this means the CPU has time to take on additional tasks. The digital phase locked loop (DPLL) provides a means of clock recovery from the received data stream on-chip. This feature, along with the frame level commands, makes SDLC loop operation extremely simple and flexible. Diagnostics in the form of both data and clock loopback are available to simplify board debug and link testing. The 8273 is a dedicated function peripheral in the MCS-8085 Microcomputer family and as such, it interfaces to the 8080/8085 system with a minimum of external hardware.

This application note explains the 8273 as a component and shows its use in a generalized loop configuration and a typical 8085 system. The 8085 system was used to verify the SDLC operation of the 8273 on an actual IBM SDLC data communications link.

The first section of this application note presents an overview of the SDLC/HDLC protocols. It is fairly tutorial in nature and may be skipped by the more knowledgeable reader. The second section describes the 8273 from a functional standpoint with explanation of the block diagram. The software aspects of the 8273, including command examples, are discussed in the third section. The fourth and fifth sections discuss a loop SDLC configuration and the 8085 system respectively.

SDLC/HDLC OVERVIEW
SDLC is a protocol for managing the flow of information on a data communications link. In other words, SDLC can be thought of as an envelope — addressed, stamped, and containing an s.a.s.e. — in which information is transferred from location to location on a data communications link. (Please note that while SDLC is discussed specifically, all comments also apply to HDLC except where noted.) The link may be either point-to-point or multi-point, with the point-to-point configuration being either switched or nonswitched. The information flow may use either full or half duplex exchanges. With this many configurations supported, it is difficult to find a synchronous data communications application where SDLC would not be appropriate.

Aside from supporting a large number of configurations, SDLC offers the potential of a 2 x increase in throughput over the presently most prevalent protocol: Bi-Sync. This performance increase is primarily due to two characteristics of SDLC: full duplex operation and the implied acknowledgement of transferred information. The performance increase due to full duplex operation is fairly obvious since, in SDLC, both stations can communicate simultaneously. Bi-Sync supports only half-duplex (two-way alternate) communication. The increase from implied acknowledgement arises from the fact that a station using SDLC may acknowledge previously received information while transmitting different information. Up to 7 messages may be outstanding before an acknowledgement is required. These messages may be acknowledged as a block rather than singly. In Bi-Sync, acknowledgements are unique messages that may not be included with messages containing information and each information message requires a separate acknowledgement. Thus the line efficiency of SDLC is superior to Bi-Sync. On a higher level, the potential of a 2 x increase in performance means lower cost per unit of information transferred. Notice that the increase is not due to higher data link speeds (SDLC is actually speed independent), but simply through better line utilization.

Getting down to the more salient characteristics of SDLC; the basic unit of information on an SDLC link is that of the frame. The frame format is shown in Figure 1. Five fields comprise each frame: flag, address, control, information, and frame check sequence. The flag fields (F) form the boundary of the frame and all other fields are positionally related to one of the two flags. All frames start with an opening flag and end with a closing flag. Flags are used for frame synchronization. They also may serve as time-fill characters between frames. There are no intraframe time-fill characters in SDLC as there are in Bi-Sync.) The opening flag serves as a reference point for the address (A) and control (C) fields. The frame check sequence (FCS) is referenced from the closing flag. All flags have the binary configuration 01111110 (7EH).

SDLC is a bit-oriented protocol, that is, the receiving station must be able to recognize a flag (or any other special character) at any time, not just on an 8-bit boundary. This, of course, implies that a frame may be N-bits in length. (The vast majority of applications tend to use frames which are multiples of 8 bits long, however.)

<table>
<thead>
<tr>
<th>OPENING FLAG</th>
<th>ADDRESS FIELD (A)</th>
<th>CONTROL FIELD (C)</th>
<th>INFORMATION FIELD (I)</th>
<th>FRAME CHECK SEQUENCE (FCS)</th>
<th>CLOSING FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111110</td>
<td>6 BITS</td>
<td>6 BITS</td>
<td>ANY LENGTH 0 TO N BITS</td>
<td>10 BITS</td>
<td>01111110</td>
</tr>
</tbody>
</table>

Figure 1. SDLC Frame Format
The fact that the flag has a unique binary pattern would seem to limit the contents of the frame since a flag pattern might inadvertently occur within the frame. This would cause the receiver to think the closing flag was received, invalidating the frame. SDLC handles this situation through a technique called zero bit insertion. This technique specifies that within a frame a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1s. Thus, no pattern of 01111110 is ever transmitted by chance. On the receiving end, after the opening flag is detected, the receiver removes any 0 following 5 consecutive 1s. The inserted and deleted 0s are not counted for error determination.

Before discussing the address field, an explanation of the roles of an SDLC station is in order. SDLC specifies two types of stations: primary and secondary. The primary is the control station for the data link and thus has responsibility for the overall network. There is only one predetermined primary station, all other stations on the link assume the secondary station role. In general, a secondary station speaks only when spoken to. In other words, the primary polls the secondaries for responses. In order to specify a secondary, each secondary is assigned a unique 8-bit address. It is this address that is used in the frame’s address field.

When the primary transmits a frame to a specific secondary, the address field contains the secondary’s address. When responding, the secondary uses its own address in the address field. The primary is never identified. This ensures that the primary knows which of many secondaries is responding since the primary may have many messages outstanding at various secondary stations. In addition to the specific secondary address, an address common to all secondaries may be used for various purposes. (An all 1s address field is usually used for this “All Parties” address.) Even though the primary may use this common address, the secondaries are expected to respond with their unique address. The address field is always the first 8 bits following the opening flag.

The 8 bits following the address field form the control field. The control field embodies the link-level control of SDLC. A detailed explanation of the commands and responses contained in this field is beyond the scope of this application note. Suffice it to say that it is in the control field that the implied acknowledgement is carried out through the use of frame sequence numbers. None of the currently available SDLC single chip controllers utilize the control field. They simply pass it to the processor for analysis. Readers wishing a more detailed explanation of the control field, or of SDLC in general, should consult the IBM documents referenced on the front page overleaf.

In some types of frames, an information field follows the control field. Frames used strictly for link management may or may not contain one. When an information field is used, it is unrestricted in both content and length. This code transparency is made possible because of the zero bit insertion mentioned earlier and the bit-oriented nature of SDLC. Even main memory core dumps may be transmitted because of this capability. This feature is unique to bit-oriented protocols. Like the control field, the information field is not interpreted by the SDLC device; it is merely transferred to and from memory to be operated on and interpreted by the processor.

The final field is the frame check sequence (FCS). The FCS is the 16 bits immediately preceding the closing flag. This 16-bit field is used for error detection through a Cyclic Redundancy Checkword (CRC). The 16-bit transmitted CRC is the complement of the remainder obtained when the A, C, and I fields are “divided” by a generating polynomial. The receiver accumulates the A, C, and I fields and also the FCS into its internal CRC register. At the closing flag, this register contains one particular number for an error-free reception. If this number is not obtained, the frame was received in error and should be discarded. Discarding the frame causes the station to not update its frame sequence numbering. This results in a retransmission after the station sends an acknowledgement from previous frames. Unlike all other fields, the FCS is transmitted MSB (Most Significant Bit) first. The A, C, and I fields are transmitted LSB (Least Significant Bit) first. The details of how the FCS is generated and checked is beyond the scope of this application note and since all single component SDLC controllers handle this function automatically, it is usually sufficient to know only that an error has or has not occurred. The IBM documents contain more detailed information for those readers desiring it.

The closing flag terminates the frame. When the closing flag is received, the receiver knows that the preceding 16 bits constitute the FCS and that any bits between the control field and the FCS constitute the information field.

SDLC does not support an interframe time-fill character such as the SYN character in Bi-Sync. If an unusual condition occurs while transmitting, such as data is not available in time from memory or CTS (Clear-to-Send) is lost from the modem, the transmitter aborts the frame by sending an Abort character to notify the receiver to invalidate the frame. The Abort character consists of eight contiguous 1s sent without zero bit insertion. Interframe time-fill consists of either flags, Abort characters, or any combination of the two.

While the Abort character protects the receiver from transmitted errors, errors introduced by the transmission medium are discovered at the receiver through the FCS check and a check for invalid frames. Invalid frames are those which are not bounded by flags or are too short, that is, less than 32 bits between flags. All invalid frames are ignored by the receiver.

Although SDLC is a synchronous protocol, it provides an optional feature that allows its use on basically asynchronous data links — NRZI (Non-Return-to-Zero-Inverted) coding. NRZI coding specifies that the signal condition does not change for transmitting a binary 1, while a binary 0 causes a change of state. Figure 2 illustrates NRZI coding compared to the normal NRZ. NRZI coding guarantees that an active line will have a transition at least every 5-bit times; long strings of zeroes cause a transition every bit time, while long strings of 1s are broken up by zero bit insertion. Since asynchronous
operation requires that the receiver sampling clock be derived from the received data, NRZI encoding plus zero bit insertion make the design of clock recovery circuitry easier.

All of the previous discussion has applied to SDLC on either point-to-point or multi-point data networks, SDLC (but not HDLC) also includes specification for a loop configuration. Figure 3 compares these three configurations. IBM uses this loop configuration in its 3650 Retail Store System. It consists of a single loop controller station with one or more down-loop secondary stations. Communications on a loop rely on the secondary stations repeating a received message down loop with a delay of one bit time. The reason for the one bit delay will be evident shortly.

Loop operation defines a new special character: the EOP (End-of-Parl) character which consists of a 0 followed by 7 contiguous, non-zero bit inserted, ones. After the loop controller transmits a message, it idles the line (sends all 1s). The final zero of the closing flag plus the first 7 1s of the idle form an EOP character. While repeating, the secondaries monitor their incoming line for an EOP character. When an EOP is detected, the secondary checks to see if it has a message to transmit. If it does, it changes the seventh 1 to a 0 (the one bit delay allows time for this) and repeats the modified EOP (now alias flag). After this flag is transmitted, the secondary terminates its repeater function and inserts its message (with multiple preceding flags if necessary). After the closing flag, the secondary resumes its one bit delay repeater function. Notice that the final zero of the secondary's closing flag plus the repeated 1s from the controller form an EOP for the next down-loop secondary, allowing it to insert a message if it desires.

One might wonder if the secondary missed any messages from the controller while it was inserting its own message. It does not. Loop operation is basically half-duplex. The controller waits until it receives an EOP before it transmits its next message. The controller's reception of the EOP signifies that the original message has propagated around the loop followed by any messages inserted by the secondary. Notice that secondaries cannot communicate with one another directly, all secondary-to-secondary communication takes place by way of the controller.
APPLICATIONS

Loop protocol does not utilize the normal Abort character. Instead, an abort is accomplished by simply transmitting a flag character. Down loop, the receiver sees the abort as a frame which is either too short (if the abort occurred early in the frame) or one with an FCS error. Either results in a discarded frame. For more details on loop operation, please refer to the IBM documents referenced earlier.

Another protocol very similar to SDLC which the 8273 supports is HDLC (High-Level Data Link Control). There are only three basic differences between the two: HDLC offers extended address and control fields, and the HDLC Abort character is 7 contiguous 1s as opposed to SDLC's 8 contiguous 1s.

Extended addressing, beyond the 256 unique addresses possible with SDLC, is provided by using the address field's least significant bit as the extended address modifier. The receiver examines this bit to determine if the octet should be interpreted as the final address octet. As long as the bit is 0, the octet that contains it is considered an extended address. The first time the bit is a 1, the receiver interprets that octet as the final address octet. Thus the address field may be extended to any number of octets. Extended addressing is illustrated in Figure 4a.

A similar technique is used to extend the control field although the extension is limited to only one extra control octet. Figure 4b illustrates control field extension.

Those readers not yet asleep may have noticed the similarity between the SDLC loop EOP character (a 0 followed by 7 1s) and the HDLC Abort (7 1s). This possible incompatibility is neatly handled by the HDLC protocol not specifying a loop configuration.

This completes our brief discussion of the SDLC/HDLC protocols. Now let us turn to the 8273 in particular and discuss its hardware aspects through an explanation of the block diagram and generalized system schematics.

BASIC 8273 OPERATION

It will be helpful for the following discussions to have some idea of the basic operation of the 8273. Each operation, whether it is a frame transmission, reception or port read, etc., is comprised of three phases: the Command, Execution, and Result phases. Figure 5 shows the sequence of these phases. As an illustration of this sequence, let us look at the transmit operation.

Figure 5. 8273 Operational Phases

When the CPU decides it is time to transmit a frame, the Command phase is entered by the CPU issuing a Transmit Frame command to the 8273. It is not sufficient to just instruct the 8273 to transmit. The frame level command structure sometimes requires more information such as frame length and address and control field content. Once this additional information is supplied, the Command phase is complete and the Execution phase is entered. It is during the Execution phase that the actual operation, in this case a frame transmission, takes place. The 8273 transmits the opening flag, A and C fields, the specified number of I field bytes, inserts the FCS, and closes with the closing flag. Once the closing flag is transmitted, the 8273 leaves the Execution phase and begins the Result phase. During the Result phase the 8273 notifies the CPU of the outcome of the command by supplying interrupt results. In this case, the results would be either that the frame is complete or that some error condition causes the transmission to be aborted. Once the CPU reads all of the results (there is only one for the Transmit Frame command), the Result phase and consequently the operation, is complete. Now that we have a general feeling for the operation of the 8273, let us discuss the 8273 in detail.

HARDWARE ASPECTS OF THE 8273

The 8273 block diagram is shown in Figure 6. It consists of two major interfaces: the CPU module interface and the modem interface. Let's discuss each interface separately.
CPU Interface

The CPU interface consists of four major blocks: Control/Read/Write logic (C/R/W), internal registers, data transfer logic, and data bus buffers.

The CPU module utilizes the C/R/W logic to issue commands to the 8273. Once the 8273 receives a command and executes it, it returns the results (good/bad completion) of the command by way of the C/R/W logic. The C/R/W logic is supported by seven registers which are addressed via the A0, A1, RD, and WR signals, in addition to CS. The A0 and A1 signals are generally derived from the two low order bits of the CPU module address bus while RD and WR are the normal I/O Read and Write signals found on the system control bus. Figure 7 shows the address of each register using the C/R/W logic. The function of each register is defined as follows:

<table>
<thead>
<tr>
<th>ADDRESS INPUTS</th>
<th>CONTROL INPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>A0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Command — 8273 operations are initiated by writing the appropriate command byte into this register.

Parameter — Many commands require more information than found in the command itself. This additional information is provided by way of the parameter register.

Immediate Result (Result) — The completion information (results) for commands which execute immediately are provided in this register.

Transmit Interrupt Result (Txl/R) — Results of transmit operations are passed to the CPU in this register.

Receiver Interrupt Result (Rxl/R) — Receive operation results are passed to the CPU via this register.

Status — The general status of the 8273 is provided in this register. The Status register supplies the handshaking necessary during various phases of the 8273 operation.

Test Mode — This register provides a software reset function for the 8273.

The commands, parameters, and bit definition of these registers are discussed in the following software section. Notice that there are not specific transmit or receive data registers. This feature is explained in the data transfer logic discussion.
The final elements of the C/R/W logic are the interrupt lines (RxINT and TxINT). These lines notify the CPU module that either the transmitter or the receiver requires service; i.e., results should be read from the appropriate interrupt result register or a data transfer is required. The interrupt request remains active until all the associated interrupt results have been read or the data transfer is performed. Though using the interrupt lines relieves the CPU module of the task of polling the 8273 to check if service is needed, the state of each interrupt line is reflected by a bit in the Status register and non-interrupt driven operation is possible by examining the contents of these bits periodically.

The 8273 supports two independent data interfaces through the data transfer logic; receive data and transmit data. These interfaces are programmable for either DMA or non-DMA data transfers. While the choice of the configuration is up to the system designer, it is based on the intended maximum data rate of the communications channel. Figure 8 illustrates the transfer rate of data bytes that are acquired by the 8273 based on link data rate. Full-duplex data rates above 9600 baud usually require DMA. Slower speeds may or may not require DMA depending on the task load and interrupt response time of the processor.

Figure 9 shows the 8273 in a typical DMA environment. Notice that a separate DMA controller, in this case the Intel 8257, is required. The DMA controller supplies the timing and addresses for the data transfers while the 8273 manages the requesting of transfers and the actual counting of the data block lengths. In this case, elements of the data transfer interface are:

- **TxDRQ**: Transmit DMA Request — Asserted by the 8273, this line requests a DMA transfer from memory to the 8273 for transmit.
- **TxACK**: Transmit DMA Acknowledge — Returned by the 8273 in response to TxDRQ, this line notifies the 8273 that a request has been granted, and provides access to the transmitter data register.
- **RxDRQ**: Receiver DMA Request — Asserted by the 8273, it requests a DMA transfer from the 8273 to memory for a receive operation.
- **RxACK**: Receiver DMA Acknowledge — Returned by the 8273, it notifies the 8273 that a receive DMA cycle has been granted, and provides access to the receiver data register.
- **RD**: Read — Supplied by the 8257 to indicate data is to be read from the 8273 and placed in memory.
- **WR**: Write — Supplied by the 8257 to indicate data is to be written to the 8273 from memory.

To request a DMA transfer the 8273 raises the appropriate DMA request line; let us assume it is a transmitter request (TxDRQ). Once the 8257 obtains control of the system bus by way of its HOLD and HLDA (hold acknowledge) lines, it notifies the 8273 that TxDRQ has been granted by returning TxACK and WR. The TxACK and WR signals transfer data to the 8273 for a transmit, independent of the 8273 chip select pin (CS). A similar sequence of events occurs for receiver requests. This "hard select" of data into the transmitter or out of the receiver alleviates the need for the normal transmit and receive data registers addressed by a combination of address lines, CS, and WR or RD. Competitive devices that do not have this "hard select" feature require the use of an external multiplexer to supply the correct inputs for register selection during DMA. (Do not forget that the SDLC controller sees both the addresses and control signals supplied by the DMA controller during DMA cycles.) Let us look at typical frame transmit and frame receive sequences to better see how the 8273 truly manages the DMA data transfer.

Before a frame can be transmitted, the DMA controller is supplied, by the CPU, the starting address for the desired information field. The 8273 is then commanded to transmit a frame. (Just how this is done is covered later during our software discussion.) After the command, but before transmission begins, the 8273 needs a little more information (parameters). Four parameters are required for the transmit frame command: the address field byte, the control field byte, and two bytes which are the least significant and most significant bytes of the information field byte length. Once all four parameters are loaded, the 8273 makes RTS (Request-to-Send) active and waits for CTS (Clear-to-Send) to go active. Once CTS is active, the 8273 starts the frame transmission. While the 8273 is transmitting the opening flag, address field, and control field; it starts making transmitter DMA requests. These requests continue at character (byte) boundaries until the pre-loaded number of bytes of information field have been transmitted. At this point the requests stop, the FCS and closing flag are transmitted, and the TxINT line is raised, signaling the CPU that the frame transmission is complete. Notice that after the initial command and parameter loading, absolutely no CPU intervention was required (since DMA is used for data transfers) until the entire frame was transmitted. Now let's look at a frame reception.
The receiver operation is very similar. Like the initial transmit sequence, the DMA controller is loaded with a
starting address for a receiver data buffer and the 8273 is commanded to receive. Unlike the transmitter, there
are two different receive commands: General Receive, where all received frames are transferred to memory,
and Selective Receive, where only frames having an address field matching one of two preprogrammed 8273
address fields are transferred to memory. Let’s assume for now that we want to general receive. After
the receive command, two parameters are required before the receiver becomes active: the least significant
and most significant bytes of the receiver buffer length. Once these bytes are loaded, the receiver is active and
the CPU may return to other tasks. The next frame appearing at the receiver input is transferred to memory
using receiver DMA requests. When the closing flag is received, the 8273 checks the FCS and raises its RxINT
line. The CPU can then read the results which indicate if the frame was error-free or not. (If the received frame
had been longer than the pre-loaded buffer length, the CPU would have been notified of that occurrence earlier
with a receiver error interrupt. The command description section contains a complete list of error conditions.)
Like the transmit example, after the initial command, the CPU is free for other tasks until a frame is com-
pletely received. These examples have illustrated the 8273’s management of both the receiver and transmitter
DMA channels.

It is possible to use the DMA data transfer interface in a non-DMA interrupt-driven environment. In this case, 4 in-
terrupt levels are used: one each for TxINT and RxINT, and one each for TxDRO and RxDRO. This configuration
is shown in Figure 10. This configuration offers the advantages that no DMA controller is required and data
requests are still separated from result (completion) re-
quests. The disadvantages of the configuration are that
4 interrupt levels are required and that the CPU must ac-
tually supply the data transfers. This, of course, reduces
the maximum data rate compared to the configuration
based strictly on DMA. This system could use an Intel
8259 8-level Priority Interrupt Controller to supply a vec-
tored CALL (subroutine) address based on requests on
its inputs. The 8273 transmitter and receiver make data requests by raising the respective DRO line. The CPU is
interrupted by the 8259 and vectored to a data transfer
routine. This routine either writes (for transmit) or reads (for receive) the 8273 using the respective TxDACK or
RxDACK line. As in the case above, the DACK lines serve as “hard” chip selects into and out of the 8273.
(TxDACK + WR writes data into the 8273 for transmit. RxDACK + RD reads data from the 8273 for receive.)
The CPU is notified of operation completion and results by way of TxINT and RxINT lines. Using the 8273, and
the 8259, in this way, provides a very effective, yet sim-
ple, interrupt-driven interface.

Figure 11 illustrates a system very similar to that
described above. This system utilizes the 8273 in a non-
DMA data transfer mode as opposed to the two DMA
approaches shown in Figures 9 and 10. In the non-DMA
case, data transfer requests are made on the TxINT and
RxINT lines. The DRQ lines are not used. Data transfer
requests are separated from result requests by a bit in
the Status register. Thus, in response to an interrupt,
the CPU reads the Status register and branches to either
a result or a data transfer routine based on the status of
one bit. As before, data transfers are made via using the
DACK lines as chip selects to the transmitter and receiver data registers.

Figure 12 illustrates the simplest system of all. This
system utilizes polling for all data transfers and results.
Since the interrupt pins are reflected in bits in the
Status register, the software can read the Status register periodically looking for one of these to be set. If
it finds an INT bit set, the appropriate Result Available
bit is examined to determine if the “interrupt” is a data transfer or completion result. If a data transfer is called
for, the DACK line is used to enter or read the data from
the 8273. If the interrupt is a completion result, the ap-
propriate result register is read to determine the good/
bad completion of the operation.

The actual selection of either DMA or non-DMA modes
is controlled by a command issued during initialization.
This command is covered in detail during the software
discussion.
The final block of the CPU module interface is the Data Bus Buffer. This block supplies the tri-state, bidirectional data bus interface to allow communication to and from the 8273.

Modem Interface
As the name implies, the modem interface is the modem side of the 8273. It consists of two major blocks: the modem control block and the serial data timing block.

The modem control block provides both dedicated and user-defined modem control functions. All signals supported by this interface are active low so that EIA inverting drivers (MC1488) and inverting receivers (MC1489) may be used to interface to standard modems.

Port A is a modem control input port. Its representation on the data bus is shown in Figure 13. Bits D0 and D1 have dedicated functions. D0 reflects the logical state of the CTS (Clear-to-Send) pin. [If CTS is active (low), D0 is a 1.] This signal is used to condition the start of a transmission. The 8273 waits until CTS is active before it starts transmitting a frame. While transmitting, if CTS goes inactive, the frame is aborted and the CPU is interrupted. When the CPU reads the interrupt result, a CTS failure is indicated.

D1 reflects the logical state of the CD (Carrier Detect) pin. CD is used to condition the start of a frame reception. CD must be active in time for a frame's address field. If CD is lost (goes inactive) while receiving a frame, an interrupt is generated with a CD failure result. CD may go inactive between frames.

Bits D2 thru D5 reflect the logical state of the PA2 thru PA5 pins respectively. These inputs are user defined. The 8273 does not interrogate or manipulate these bits. D6, D7, and D8 are not used and each is read as a 1 for a Read Port A command.

Port B is a modem control output port. Its data bus representation is shown in Figure 14. As in Port A, the bit values represent the logical condition of the pins. D0 and D1 are dedicated function outputs. D0 represents the RTS (Request-to-Send) pin. RTS is normally used to notify the modem that the 8273 wishes to transmit. This function is handled automatically by the 8273. If RTS is inactive (pin is high) when the 8273 is commanded to transmit, the 8273 makes it active and then waits for CTS before transmitting the frame. One byte time after the end of the frame, the 8273 returns RTS to its inactive state. However, if RTS was active when a transmit command is issued, the 8273 leaves it active when the frame is complete.

Bit D5 reflects the state of the Flag Detect pin. This pin is activated whenever an active receiver sees a flag character. This function is useful to activate a timer for line activity timeout purposes.

Bits D6 thru D9 provide four user-defined outputs. Pins PB6 thru PB9 reflect the logical state of these bits. The 8273 does not interrogate or manipulate these bits. D6 and D7 are not used. In addition to being able to output to Port B, Port B may be read using a Read Port B command. All Modem control output pins are forced high on reset. (All commands mentioned in this section are covered in detail later.)

The final block to be covered is the serial data timing block. This block contains two sections: the serial data logic and the digital phase locked loop (DPLL).

Elements of the serial data logic section are the data pins, TXD (transmit data output) and RXD (receive data input), and the respective data clocks, TXC and RXC. The transmit and receive data is synchronized by the TXC and RXC clocks. Figure 15 shows the timing for these signals. The leading edge (negative transition) of TXC generates new transmit data and the trailing edge (positive transition) of RXC is used to capture the receive data.

It is possible to reconfigure this section under program control to perform diagnostic functions; both data and clock loopback are available. In data loopback mode, the TXD pin is internally routed to the RXD pin. This allows simple board checkout since the CPU can send an SDLC message to itself. (Note that transmitted data will still appear on the TXD pin.)

![Figure 12. Polled System](image)

![Figure 13. Port A (Input) Bit Definition](image)

![Figure 14. Port B (Output) Bit Definition](image)
When data loopback is utilized, the receiver may be presented incorrect sample timing (RxC) by the external circuitry. Clock loopback overcomes this problem by allowing the internal routing of TxC and RxC. Thus the same clock used to transmit the data is used to receive it. Examination of Figure 15 shows that this method ensures bit synchronization. The final element of the serial data logic is the Digital Phase Locked Loop.

The DPLL provides a means of clock recovery from the received data stream. This feature allows the 8273 to interface without external synchronizing logic to low cost asynchronous modems (modems which do not supply clocks). It also makes the problem of clock timing in loop configurations trivial.

To use the DPLL, a clock at 32 times the required baud rate must be supplied to the 32 x CLK pin. This clock provides the interval that the DPLL samples the received data. The DPLL uses the 32 x clock and the received data to generate a pulse at the DPLL output pin. This DPLL pulse is positioned at the nominal center of the received data bit cell. Thus the DPLL output may be wired to RxC and/or TxC to supply the data timing. The exact position of the pulse is varied depending on the line noise and bit distortion of the received data. The adjustment of the DPLL position is determined according to the rules outlined in Figure 16.

Adjustments to the sample phase of DPLL with respect to the received data is made in discrete increments. Referring to Figure 16, following the occurrence of DPLL pulse A, the DPLL counts 32 x CLK pulses and examines the received data for a data edge. Should no edge be detected in 32 pulses, the DPLL positions the next DPLL pulse (B) at 32 clock pulses from pulse A. Since no new phase information is contained in the data stream, the sample phase is assumed to be at nominal 1 x baud rate. Now assume a data edge occurs after DPLL pulse B. The distance from B to the next pulse C is influenced according to which quadrant (A1, B1, B2, or A2) the data edge falls in. (Each quadrant represents 8 32 x CLK times.) For example, if the edge is detected in quadrant A1, it is apparent that pulse B was too close to the data edge and the time to the next pulse must be shortened. The adjustment for quadrant A1 is specified as -2. Thus, the next DPLL pulse, pulse C, is positioned 32 - 2 or 30 32 x CLK pulses following DPLL pulse B. This adjustment moves pulse C closer to the nominal bit center of the next received data cell. A data edge occurring in quadrant B2 would have caused the adjustment to be small, namely 32 + 1 or 33 32 x CLK pulses. Using this technique, the DPLL pulse converges to the nominal bit center within 12 data transitions, worse case — 4-bit times adjusting through quadrant A1 or A2 and 8-bit times adjusting through B1 or B2.
APPLICATIONS

When the receive data stream goes idle after 15 ones, DPLL pulses are generated at 32 pulse intervals of the 32x CLK. This feature allows the DPLL pulses to be used as both transmitter and receiver clocks.

In order to guarantee sufficient transitions of the received data to enable the DPLL to lock, NRZI encoding of the data is recommended. This ensures that, within a frame, data transitions occur at least every five bit times — the longest sequence of 1s which may be transmitted with zero bit insertion. It is also recommended that frames following a line idle be transmitted with pre-frame sync characters which provide a minimum of 12 transitions. This ensures that the DPLL is generating DPLL pulses at the nominal bit centers in time for the opening flag. (Two 00H characters meet this requirement by supplying 16 transitions with NRZI encoding. The 8273 contains a mode which supplies such a pre-frame sync.)

Figure 17 illustrates 8273 clock configurations using either synchronous or asynchronous modems. Notice how the DPLL output is used for both TxC and RxC in the asynchronous case. This feature eliminates the need for external clock generation logic where low cost asynchronous modems are used and also allows direct connection of 8273s for the ultimate in low cost data links. The configuration for loop applications is discussed in a following section.

This completes our discussion of the hardware aspects of the 8273. Its software aspects are now discussed.

SOFTWARE ASPECTS OF THE 8273

The software aspects of the 8273 involve the communication of both commands from the CPU to the 8273 and the return of results of those commands from the 8273 to the CPU. Due to the internal processor architecture of the 8273, this CPU-8273 communication is basically a form of interprocessor communication. Such communication usually requires a form of protocol of its own. This protocol is implemented through use of handshaking supplied in the 8273 Status register. The bit definition of this register is shown in Figure 18.

CBSY: Command Busy — CBSY indicates when the 8273 is in the command phase. CBSY is set when the CPU writes a command into the Command register, starting the Command phase. It is reset when the last parameter is deposited in the Parameter register and accepted by the 8273, completing the Command phase.

CBF: Command Buffer Full — When set, this bit indicates that a byte is present in the Command register. This bit is normally not used.

CPBF: Command Parameter Buffer Full — This bit indicates that the Parameter register contains a parameter. It is set when the CPU deposits a parameter in the Parameter register. It is reset when the 8273 accepts the parameter.

CRBF: Command Result Buffer Full — This bit is set when the 8273 places a result from an immediate type command in the Result register. It is reset when the CPU reads the result from the Result register.

RxINT: Receiver Interrupt — The state of the RxlNT pin is reflected by this bit. RxINT is set by the 8273 whenever the receiver needs servicing. RxINT is reset when the CPU reads the results or performs the data transfer.

TxINT: Transmitter Interrupt — This bit is identical to RxINT except action is initiated based on transmitter interrupt sources.

Figure 17. Serial Data Timing Configuration
**APPLICATIONS**

RxIRA: Receiver Interrupt Result Available — RxIRA is set when the 8273 places an interrupt result byte into the RxI/R register. RxIRA is reset when the CPU reads the RxI/R register.

TxIRA: Transmitter Interrupt Result Available — TxIRA is the corresponding Result Available bit for the transmitter. It is set when the 8273 places an interrupt result byte in the TxI/R register and reset when the CPU reads the register.

The significance of each of these bits will be evident shortly. Since the software requirements of each 8273 phase are essentially independent, each phase is covered separately.

---

![Command Phase Flowchart](image)

**Command Phase Software**

Recalling the Command phase description in an earlier section, the CPU starts the Command phase by writing a command byte into the 8273 Command register. If further information about the command is required by the 8273, the CPU writes this information into the Parameter register. Figure 19 is a flowchart of the Command phase. Notice that the CBSY and CPBF bits of the Status register are used to handshake the command and parameter bytes. Also note that the chart shows that a command may not be issued if the Status register indicates the 8273 is busy (CBSY = 1). If a command is issued while CBSY = 1, the original command is overwritten and lost. (Remember that CBSY signifies the command phase is in progress and not the actual execution of the command.) The flowchart also includes a Parameter buffer full check. The CPU must wait until CPBF = 0 before writing a parameter to the Parameter register. If a parameter is issued while CPBF = 1, the previous parameter is overwritten and lost. An example of command output assembly language software is provided in Figure 20a. This software assumes that a command buffer exists in memory. The buffer is pointed at by the HL register. Figure 20b shows the command buffer structure.

The 8273 is a full duplex device, i.e., both the transmitter and receiver may be executing commands or passing interrupt results at any given time. (Separate Rx and Tx interrupt pins and result registers are provided for this reason.) However, there is only one Command register. Thus, the Command register must be used for only one command sequence at a time and the transmitter and receiver may never be simultaneously in a command phase. A detailed description of the commands and their parameters is presented in a following section.

---

![Figure 20A. Command Phase Software](image)
Execution Phase Software

During the Execution phase, the operation specified by the Command phase is performed. If the system utilizes DMA for data transfers, there is no CPU involvement during this phase, so no software is required. If non-DMA data transfers are used, either interrupts or polling is used to signal a data transfer request.

For interrupt-driven transfers the 8273 raises the appropriate INT pin. When responding to the interrupt, the CPU must determine whether it is a data transfer request or an interrupt signaling that an operation is complete and results are available. The CPU determines the cause by reading the Status register and interrogating the associated IRA (Interrupt Result Available) bit (TxIRA for TxINT and RxIRA for RxINT). If the IRA = 0, the interrupt is a data transfer request. If the IRA = 1, an operation is complete and the associated Interrupt Result register must be read to determine the completion status (good/bad/etc.). A software interrupt handler implementing the above sequence is presented as part of the Result phase software.

When polling is used to determine when data transfers are required, the polling routine reads the Status register looking for one of the INT bits to be set. When a set INT bit is found, the corresponding IRA bit is examined. Like in the interrupt-driven case, if the IRA = 0, a data transfer is required. If IRA = 1, an operation is complete and the Interrupt Result register needs to be read. Again, example polling software is presented in the next section.

Result Phase Software

During the Result phase the 8273 notifies the CPU of the outcome of a command. The Result phase is initiated by either a successful completion of an operation or an error detected during execution. Some commands such as reading or writing the I/O ports provide immediate results, that is, there is essentially no delay from the issuing of the command and when the result is available. Other commands such as frame transmit, take time to complete so their result is not available immediately. Separate result registers are provided to distinguish these two types of commands and to avoid interrupt handling for simple results.

Immediate results are provided in the Result register. Validity of information in this register is indicated to the CPU by way of the CRBF bit in the Status register. When the CPU completes the Command phase of an immediate command, it polls the Status register waiting until CRBF = 1. When this occurs, the CPU may read the Result register to obtain the immediate result. The Result register provides only the results from immediate commands.

Example software for handling immediate results is shown in Figure 21. The routine returns with the result in the accumulator. The CPU then uses the result as is appropriate.

All non-immediate commands deal with either the transmitter or receiver. Results from these commands are provided in the TxIR/ (Transmit Interrupt Result) and RxIR/ (Receive Interrupt Result) registers respectively. Results in these registers are conveyed to the CPU by the TxIRA and RxIRA bits of the Status register. Results of non-immediate commands consist of one byte result interrupt code indicating the condition for the interrupt and, if required, one or more bytes supplying additional information. The interrupt codes and the meaning of the additional results are covered following the detailed command description.

Non-immediate results are passed to the CPU in response to either interrupts or polling of the Status register. Figure 22 illustrates an interrupt-driven result handler. (Please note that all of the software presented in this application note is not optimized for either speed or code efficiency. They are provided as a guide and to illustrate concepts.) This handler provides for interrupt-driven data transfers as was promised in the last section. Users employing DMA-based transfers do not need the lines where the IRA bit is tested for zero. (These lines are denoted by an asterisk in the comments column.) Note that the INT bit is used to determine when all results have been read. All results must be read. Otherwise, the INT bit (and pin) will remain high and further interrupts may be missed. These routines place the results in a result buffer pointed at by RCRBUF and TxRBUF.

A typical result handler for systems utilizing polling is shown in Figure 23. Data transfers are also handled by this routine. This routine utilizes the routines of Figure 22 to handle the results.

At this point, the reader should have a good conceptual feel about how the 8273 operates. It is now time for the particulars of each command to be discussed.
APPLICATIONS

;FUNCTION: RXI - INTERRUPT DRIVEN RESULT/DATA HANDLER
;INPUTS: RXBUF, RXPNT
;CALLS: NONE
;OUTPUTS: RXBUF, RXPNT
;DESTROY: NOTHING
;DESCRIPTION: RXI IS ENTERED AT A RECEIVER INTERRUPT.
;THE INTERRUPT IS TESTED FOR DATA TRANSFER (IRA=0).
;A RESULT (IRA=1). FOR DATA TRANSFER, THE DATA IS
;PLACED IN A BUFFER AT RXBUF. RESULTS ARE PLACED IN
;A BUFFER AT RXBUF.
;A FLAG/RXPNT IS SET IF THE INTERRUPT WAS A RESULT.
;DATA TRANSFER INSTRUCTIONS ARE DENOTED BY (*) AND
;MAYBE ELIMINATED BY USERS USING DMA.

# RXI:

;RXI 1,

;RXI 2,

;RXI 3,

;RXI 4,

;RXI 5,

;RXI 6,

;FUNCTION: TXI - INTERRUPT DRIVEN RESULT/DATA HANDLER
;INPUTS: TXBUF, TXPNT, TXFLAG
;OUTPUTS: TXBUF, TXPNT, TXFLAG
;CALLS: NONE
;DESTROY: NOTHING
;DESCRIPTION: TXI IS ENTERED AT A TRANSMITTER INTERRUPT.
;THE INTERRUPT IS TESTED BY WAY OF THE IRA BIT TO SEE
;IF A DATA TRANSFER OR RESULT COMPLETION HAS OCCURRED.
;DATA TRANSFERS (IRA=0). THE DATA IS OBTAINED FROM
;A BUFFER LOCATION POINTED AT BY TXPNT. FOR COMPLETION,
;IRA=1). THE RESULTS ARE READ AND PLACED AT A RESULT
;BUFFER POINTED AT BY TXBUF, AND THE TXPNT IS SET
;TO INDICATE TO THE MAIN PROGRAM THAT A OPERATION IS
;COMPLETE. TX OPERATIONS HAVE ONLY ONE RESULT.
;DATA TRANSFER INSTRUCTIONS ARE DENOTED BY (*). THESE
;MAY BE REMOVED BY USERS USING DMA.

;TXI:

;TXI 2,

;TXI 3,

;TXI 4,

;TXI 5,

;TXI 6,

;FUNCTION: POLOP
;INPUTS: NONE
;OUTPUTS: C# (NO STATUS), =1 (TX COMPLETION),
;=2 (TX COMPLETION) ( BOTH)
;CALLS: TXI, RXI
;DESTROYS: B, C
;DESCRIPTION: POLOP IS CALLED TO POLL THE 6273 FOR
;DATA TRANSFERS AND COMPLETION RESULTS. THE
;ROUTINES TXI AND RXI ARE USED FOR THE ACTUAL
;TRANSFERS AND BUFFER WORK. POLOP RETURMS
;THE STATUS OF THEIR ACTION.

;POLOP:

;POLOP 1,

;POLOP 2,

;POLOP 3,

;POLOP 4,

;POLOP 5,

;POLOP 6,

;8273 COMMAND DESCRIPTION

In this section, each command is discussed in detail. In
order to shorten the notation, please refer to the com-
mand key in Table 1. The 8273 utilizes five different
command types: Initialization/Configuration, Receive,
Transmit, Reset, and Modem Control.

Initialization/Configuration Commands

The Initialization/Configuration commands manipulate
registers internal to the 8273 that define the various
operating modes. These commands either set or reset
specified bits in the registers depending on the type of
command. One parameter is required. Set commands
perform a logical OR operation of the parameter (mask)
and the internal register. This mask contains 1s where
register bits are to be set. A 0 in the mask causes no
change in the corresponding register bit. Reset com-
mands perform a logical AND operation of the param-
eter (mask) and the internal register, i.e., the mask is 0
to reset a register bit and a 1 to cause no change. Before
presenting the commands, the register bit definitions are
discussed.

TABLE 1. COMMAND SUMMARY KEY

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Mask</th>
<th>Register Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>LSB AND MSB OF RECEIVE BUFFER LENGTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R0</td>
<td>LSB AND MSB OF RECEIVED FRAME LENGTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L0</td>
<td>LSB AND MSB OF TRANSMIT FRAME LENGTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>MATCH ADDRESSES FOR SELECTIVE RECEIVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RIC</td>
<td>RECEIVER INTERRUPT RESULT CODE-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIC</td>
<td>TRANSMITTER INTERRUPT RESULT CODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>ADDRESS FIELD OF RECEIVED FRAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>CONTROL FIELD OF RECEIVED FRAME</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 22. Interrupt-Driven Result Handlers

with Non-DMA Data Transfers

Figure 23. Polling Result Handler
Operating Mode Register (Figure 24)

D7–D0: **Not Used** — These bits must not be manipulated by any command; i.e., D7–D0 must be 0 for the Set command and 1 for the Reset command.

D5: **HDLC Abort** — When this bit is set, the 8273 will interrupt when 7 1s (HDLC Abort) are received by an active receiver. When reset, an SDLC Abort (8 1s) will cause an interrupt.

D4: **EOP Interrupt** — Reception of an EOP character (0 followed by 7 1s) will cause the 8273 to interrupt the CPU when this bit is set. Loop controller stations use this mode as a signal that a polling frame has completed the loop. No EOP interrupt is generated when this bit is reset.

**Early Tx Interrupt** — This bit specifies when the transmitter should generate an end of frame interrupt. If this bit is set, an interrupt is generated when the last data character has been passed to the 8273. If the user software issues another transmit command within two byte times, the final flag interrupt does not occur and the new frame is transmitted with only one flag of separation. If this restriction is not met, more than one flag will separate the frames and a frame complete interrupt is generated after the closing flag. If the bit is reset, only the frame complete interrupt occurs. This bit, when set, allows a single flag to separate consecutive frames.

D2: **Buffered Address and Control** — When set, the address and control fields of received frames are buffered in the 8273 and passed to the CPU as results after a received frame interrupt (they are not transferred to memory with the information field). On transmit, the A and C fields are passed to the 8273 as parameters. This mode simplifies buffer management. When this bit is reset, the A and C fields are passed to and from memory as the first two data transfers.

D1: **Preframe Sync** — When set, the 8273 prefaces each transmitted frame with two characters before the opening flag. These two characters provide 16 transitions to allow synchronization of the opposing receiver. To guarantee 16 transitions, the two characters are 55H-55H for non-NRZI mode (see Serial I/O Register description) or 00H-00H for NRZI mode. When reset, no preframe characters are transmitted.

D0: **Flag Stream** — When set, the transmitter will start sending flag characters as soon as it is idle; i.e., immediately if idle when the command is issued or after a transmission if the transmitter is active when this bit is set. When reset, the transmitter starts sending Idle characters on the next character boundary if idle already, or at the end of a transmission if active.

**APPLICATIONS**

Serial I/O Mode Register (Figure 25)

D7–D0: **Not Used** — These bits must be 0 for the Set command and 1 for the Reset command.

D2: **Data Loopback** — When set, transmitted data (TXD) is internally routed to the receive data circuitry. When reset, TXD and RXD are independent.

D1: **Clock Loopback** — When set, TXC is internally routed to RXC. When reset, the clocks are independent.

D0: **NRZI (Non-Return to Zero Inverted)** — When set, the 8273 assumes the received data is NRZI encoded, and NRZI encodes the transmitted data. When reset, the received and transmitted data are treated as a normal positive logic bit stream.

Data Transfer Mode Register (Figure 26)

D7–D1: **Not Used** — These bits must be 0 for the Set command and 1 for the Reset command.

D0: **Interrupt Data Transfer** — When set, the 8273 will interrupt the CPU when data transfers are required (the corresponding IRA Status register bit will be 0 to signify a data transfer interrupt rather than a Result phase interrupt). When reset, 8273 data transfers are performed through DMA requests on the DRQ pins without interrupting the CPU.
APPLICATIONS

One Bit Delay Register (Figure 27)

D7: One Bit Delay — When set, the 8273 retransmits the received data stream one bit delayed. This mode is entered and exited at a received character boundary. When reset, the transmitted and received data are independent. This mode is utilized for loop operation and is discussed in a later section.

D6-D0: Not Used — These bits must be 0 for the Set command and 1 for the Reset command.

Figure 27. One Bit Delay Mode Register

Figure 28 shows the Set and Reset commands associated with the above registers. The mask which sets or resets the desired bits is treated as a single parameter. These commands do not interrupt nor provide results during the Result phase. After reset, the 8273 defaults to all of these bits reset.

<table>
<thead>
<tr>
<th>REGISTER</th>
<th>COMMAND</th>
<th>HEX CODE</th>
<th>PARAMETER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONE BIT DELAY MODE</td>
<td>SET</td>
<td>A4</td>
<td>SET MASK</td>
</tr>
<tr>
<td></td>
<td>RESET</td>
<td>64</td>
<td>RESET MASK</td>
</tr>
<tr>
<td>DATA TRANSFER MODE</td>
<td>SET</td>
<td>97</td>
<td>SET MASK</td>
</tr>
<tr>
<td></td>
<td>RESET</td>
<td>57</td>
<td>RESET MASK</td>
</tr>
<tr>
<td>OPERATING MODE</td>
<td>SET</td>
<td>91</td>
<td>SET MASK</td>
</tr>
<tr>
<td></td>
<td>RESET</td>
<td>51</td>
<td>RESET MASK</td>
</tr>
<tr>
<td>SERIAL I/O MODE</td>
<td>SET</td>
<td>A0</td>
<td>SET MASK</td>
</tr>
<tr>
<td></td>
<td>RESET</td>
<td>50</td>
<td>RESET MASK</td>
</tr>
</tbody>
</table>

Figure 28. Initialization/Configuration Command Summary

Receive Commands

The 8273 supports three receive commands plus a receiver disable function.

General Receive

When commanded to General Receive, the 8273 passes all frames either to memory (DMA mode) or to the CPU (non-DMA mode) regardless of the contents of the frame’s address field. This command is used for primary and loop controller stations. Two parameters are required: B0 and B1. These parameters are the LSB and MSB of the receiver buffer size. Giving the 8273 this extra information alleviates the CPU of the burden of checking for buffer overflow. The 8273 will interrupt the CPU if the received frame attempts to overfill the allotted buffer space.

Selective Receive

In Selective Receive, two additional parameters besides B0 and B1 are required: A1 and A2. These parameters are two address match bytes. When commanded to Selective Receive, the 8273 passes to memory or the CPU only those frames having an address field matching either A1 or A2. This command is usually used for secondary stations with A1 being the secondary address and A2 is the "All Parties" address. If only one match byte is needed, A1 and A2 should be equal. As in General Receive, the 8273 counts the incoming data bytes and interrupts the CPU if B0, B1 is exceeded.

Selective Loop Receive

This command is very similar in operation to Selective Receive except that One Bit Delay mode must be set and that the loop is captured by placing transmitter in Flag Stream mode automatically after an EOP character is detected following a selectively received frame. The details of using the 8273 in loop configurations is discussed in a later section so please hold questions until then.

The handling of interrupt results is common among the three commands. When a frame is received without error, i.e., the FCS is correct and CD (Carrier Detect) was active throughout the frame or no attempt was made to overfill the buffer; the 8273 interrupts the CPU following the closing flag to pass the completion results. These results, in order, are the receiver interrupt result code (RIC), and the byte length of the information field of the received frame (R0, R1). If Buffered mode is selected, the address and control fields are passed as two additional results. If Buffered mode is not selected, the address and control fields are passed as the first two data transfers and R0, R1 reflect the information field length plus two.

Receive Disable

The receiver may also be disabled using the Receive Disable command. This command terminates any receive operation immediately. No parameters are required and no results are returned.

The details for the Receive command are shown in Figure 29. The interrupt result code key is shown in Figure 30. Some explanation of these result codes is appropriate.

The interrupt result code is the first byte passed to the CPU in the Rx/I register during the Result phase. Bits D7-D0 define the cause of the receiver interrupt. Since each result code has specific implications, they are discussed separately below.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>HEX CODE</th>
<th>PARAMETERS</th>
<th>RESULTS*</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERAL RECEIVE</td>
<td>C0</td>
<td>B0, B1</td>
<td>RIC, R0, R1, A, C</td>
</tr>
<tr>
<td>SELECTIVE RECEIVE</td>
<td>C1</td>
<td>B0, B1, A1, A2</td>
<td>RIC, R0, R1, A, C</td>
</tr>
<tr>
<td>SELECTIVE LOOP RECEIVE</td>
<td>C2</td>
<td>B0, B1, A1, A2</td>
<td>RIC, R0, R1, A, C</td>
</tr>
<tr>
<td>DISABLE RECEIVER</td>
<td>C5</td>
<td>NONE</td>
<td>NONE</td>
</tr>
</tbody>
</table>

*A AND C ARE PASSED AS RESULTS ONLY IN BUFFERED MODE

Figure 29. Receiver Command Summary
The first two result code results from the error-free reception of a frame. If the frame is received correctly after a General Receive command, the first result is returned. If either Selective Receive command was used (normal or loop), a match with $A_1$ generates the first result code and a match with $A_2$ generates the second. In either case, the receiver remains active after the interrupt; however, the internal buffer size counters are not reset. That is, if the receive command indicated 100 bytes were allocated to the receive buffer ($B_0, B_1$) and an 8-byte frame was received correctly, the maximum next frame size that could be received without recomputing the receiver (resetting $B_0$ and $B_1$) is 20 bytes. Thus, it is common practice to recommend the receiver after each frame reception. DMA and/or memory pointers are usually updated at this time. (Note that users who do not wish to take advantage of the 8273's buffer management features may simply use $B_0, B_1 = 00$ for each receive command. Then frames of 65K bytes may be received without buffer overflow errors.)

The third result code is a CRC error. This indicates that a frame was received in the correct format (flags, etc.); however, the received FCS did not check with the internally generated FCS. The frame should be discarded. The receiver remains active. (Do not forget that even though an error condition has been detected, all frame information up until that error has either been transferred to memory or passed to the CPU. This information should be invalidated. This applies to all receiver error conditions.) Note that the FCS, either transmitted or received, is never available to the CPU.

The Abort Detect result occurs whenever the receiver sees either an SDLC (8 1s) or an HDLC (7 1s), depending on the Operating Mode register. However, the intervening Abort character between a closing flag and an Idle does not generate an interrupt. If an Abort character (seen by an active receiver within a frame) is not preceded by a flag and is followed by an Idle, an interrupt will be generated for the Abort, followed by an Idle inter-rupt one character time later. The Idle Detect result occurs whenever 15 consecutive 1s are received. After the Abort Detect interrupt, the receiver remains active. After the Idle Detect interrupt, the receiver is disabled and must be recomputed before further frames may be received.

If the EOP Interrupt bit is set in the Operating Mode register, the EOP Detect result is returned whenever an EOP character is received. The receiver is disabled, so the Idle following the EOP does not generate an Idle Detect interrupt.

The minimum number of bits in a valid frame between the flags is 32. Fewer than 32 bits indicates an error. If Buffered mode is selected, such frames are ignored, i.e., no data transfers or interrupts are generated. In non-Buffered mode, a $<32$-bit frame generates an interrupt with the $<32$-bit Frame result since data transfers may already have disturbed the 8257 or interrupt handler. The receiver remains active.

The DMA Overrun result results from the DMA controller being too slow in extracting data from the 8273, i.e., the RXDACK signal is not returned before the next received byte is ready for transfer. The receiver is disabled if this error condition occurs.

The Memory Buffer Overflow result occurs when the number of received bytes exceeds the receiver buffer length supplied by the $B_0$ and $B_1$ parameters in the receive command. The receiver is disabled.

The Carrier Detect Failure result occurs whenever the $CD$ pin goes high (inactive) during reception of a frame. The $CD$ pin is used to qualify reception and must be active by the time the address field starts to be received. If $CD$ is lost during the frame, a $CD$ Failure interrupt is generated and the receiver is disabled. No interrupt is generated if $CD$ goes inactive between frames.

If a condition occurs requiring an interrupt be generated before the CPU has finished reading the previous interrupt results, the second interrupt is generated after the current Result phase is complete (the RxINT pin and status bit go low then high). However, the interrupt result for this second interrupt will be a Receive Interrupt Overrun. The actual cause of the second interrupt is lost. One case where this may occur is at the end of a received frame where the line goes idle. The 8273 generates a received frame interrupt after the closing flag and then 15-bit times later, generates an Idle Detect interrupt. If the interrupt service routine is slow in reading the first interrupt's results, the internal RxIR register still contains result information when the Idle Detect interrupt occurs. Rather than wiping out the previous results, the 8273 adds a Receive Interrupt Overrun result as an extra result. If the system's interrupt structure is such that the second interrupt is not acknowledged (interrupts are still disabled from the first interrupt), the Receive Interrupt Overrun result is read as an extra result, after those from the first interrupt. If the second interrupt is serviced, the Receive Interrupt Overrun is returned as a single result. (Note that the INT pins supply the necessary transitions to support a Program-
mable Interrupt Controller such as the Intel 8259. Each interrupt generates a positive-going edge on the appropriate INT pin and the high level is held until the interrupt is completely serviced.) In general, it is possible to have interrupts occurring at one character time intervals. Thus the interrupt handling software must have at least that much response and service time.

The occurrence of Receive Interrupt Overruns is an indication of marginal software design; the system’s interrupt response and servicing time is not sufficient for the data rates being attempted. It is advisable to configure the interrupt handling software to simply read the interrupt results, place them into a buffer, and clear the interrupt as quickly as possible. The software can then examine the buffer for new results at its leisure, and take appropriate action. This can easily be accomplished by using a result buffer flag that indicates when new results are available. The interrupt handler sets the flag and the main program resets it once the results are retrieved.

Both SDLC and HDLC allow frames which are of arbitrary length (>32 bits). The 8273 handles this N-bit reception through the high order bits (D7-D3) of the result code. These bits code the number of valid received bits in the last received information field byte. This coding is shown in Figure 30. The high order bits of the received partial byte are indeterminate. [The address, control, and information fields are transmitted least significant bit (Ao) first. The FCS is complemented and transmitted most significant bit first.]

Transmit Commands

The 8273 transmitter is supported by three Transmit commands and three corresponding Abort commands.

Transmit Frame

The Transmit Frame command simply transmits a frame. Four parameters are required when Buffered mode is selected and two when it is not. In either case, the first two parameters are the least and the most significant bytes of the desired frame length (Lo, L1). In Buffered mode, Lo and L1 equal the length in bytes of the desired information field, while in the non-Buffered mode, Lo and L1 must be specified as the information field length plus two. (Lo and L1 specify the number of data transfers to be performed.) In Buffered mode, the address and control fields are presented to the transmitter as the third and fourth parameters respectively. In non-Buffered mode, the A and C fields must be passed as the first two data transfers.

When the Transmit Frame command is issued, the 8273 makes RTS (Request-to-Send) active (pin low) if it was not already. It then waits until CTS (Clear-to-Send) goes active (pin low) before starting the frame. If the Preframe Sync bit in the Operating Mode register is set, the transmitter prefixes two characters (16 transitions) before the opening flag. If the Flag Stream bit is set in the Operating Mode register, the frame (including Preframe Sync if selected) is started on a flag boundary. Otherwise the frame starts on a character boundary.

At the end of the frame, the transmitter interrupts the CPU (the interrupt results are discussed shortly) and returns to either Idle or Flag Stream, depending on the Flag Stream bit of the Operating Mode register. If RTS was active before the transmit command, the 8273 does not change it. If it was inactive, the 8273 will deactivate it within one character time.

Loop Transmit

Loop Transmit is similar to Frame Transmit (the parameter definition is the same). But since it deals with loop configurations, One Bit Delay mode must be selected.

If the transmitter is not in Flag Stream mode when this command is issued, the transmitter waits until after a received EOP character has been converted to a flag (this is done automatically) before transmitting. (The one bit delay is, of course, suspended during transmit.) If the transmitter is already in Flag Stream mode as a result of a selectively received frame during a Selective Loop Receive command, transmission will begin at the next flag boundary for Buffered mode or at the third flag boundary for non-Buffered mode. This discrepancy is to allow time for enough data transfers to occur to fill up the internal transmit buffer. At the end of a Loop Transmit, the One Bit Delay mode is re-entered and the flag stream mode is reset. More detailed loop operation is covered later.

Transmit Transparent

The Transmit Transparent command enables the 8273 to transmit a block of raw data. This data is without SDLC protocol, i.e., no zero bit insertion, flags, or FCS. Thus it is possible to construct and transmit a Bi-Sync message for front-end processor switching or to construct and transmit an SDLC message with incorrect FCS for diagnostic purposes. Only the Lo and L1 parameters are used since there are not fields in this mode. (the 8273 does not support a Receive Transparent command.)

Abort Commands

Each of the above transmit commands has an associated Abort command. The Abort Frame Transmit command causes the transmitter to send eight contiguous ones (no zero bit insertion) immediately and then revert to either idle or flag streaming based on the Flag Stream bit. (The 8 1s as an Abort character is compatible with both SDLC and HDLC.)

For Loop Transmit, the Abort Loop Transmit command causes the transmitter to send one flag and then revert to one bit delay. Loop protocol depends upon FCS errors to detect aborted frames. The Abort Transmit Transparent simply causes the transmitter to revert to either idles or flags as a function of the Flag Stream mode specified. The Abort commands require no parameters, however, they do generate an interrupt and return a result when complete.

A summary of the Transmit commands is shown in Figure 31. Figure 32 shows the various transmit interrupt result codes. As in the receiver operation, the transmitter generates interrupts based on either good
completion of an operation or an error condition to start the Result phase.

The Early Transmit Interrupt result occurs after the last data transfer to the 8273 if the Early Transmit Interrupt bit is set in the Operating Mode register. If the 8273 is commanded to transmit again within two character times, a single flag will separate the frames. (Buffered mode must be used for a single flag to separate the frames. If non-Buffered mode is selected, three flags will separate the frames.) If this time constraint is not met, another interrupt is generated and multiple flags or idles will separate the frames. The second interrupt is the normal Frame Transmit Complete Interrupt. The Frame Transmit Complete result occurs at the closing flag to signify a good completion.

The DMA Underrun result is analogous to the DMA Overrun result in the receiver. Since SDLC does not support intraframe time fill, if the DMA controller or CPU does not supply the data in time, the frame must be aborted. The action taken by the transmitter on this error is automatic. It aborts the frame just as if an Abort command had been issued.

Clear-to-Send Error result is generated if CTS goes inactive during a frame transmission. The frame is aborted as above.

The Abort Complete result is self-explanatory. Please note however that no Abort Complete interrupt is generated when an automatic abort occurs. The next command type consists of only one command.

The action taken by the transmitter on this error is automatic. It aborts the frame just as if an Abort command had been issued.

Clear-to-Send Error result is generated if CTS goes inactive during a frame transmission. The frame is aborted as above.

The Abort Complete result is self-explanatory. Please note however that no Abort Complete interrupt is generated when an automatic abort occurs. The next command type consists of only one command.

- The modem control outputs are forced high inactive.
- The 8273 Status register is cleared.
- Any commands in progress cease.
- The 8273 enters an idle state until the next command is issued.

Modem Control Commands

The modem control ports were discussed earlier in the Hardware section. The commands used to manipulate these ports are shown in Figure 33. The Read Port A and Read Port B commands are immediate. The bit definition for the returned byte is shown in Figures 13 and 14. Do not forget that the returned value represents the logical condition of the pin, i.e., pin active (low) = bit set.

<table>
<thead>
<tr>
<th>PORT</th>
<th>COMMAND</th>
<th>HEX CODE</th>
<th>PARAMETER</th>
<th>REQ RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A INPUT</td>
<td>READ</td>
<td>22</td>
<td>NONE</td>
<td>PORT VALUE</td>
</tr>
<tr>
<td>READ</td>
<td>22</td>
<td>NONE</td>
<td>PORT VALUE</td>
<td></td>
</tr>
<tr>
<td>B OUTPUT</td>
<td>SET</td>
<td>A3</td>
<td>SET MASK</td>
<td>NONE</td>
</tr>
<tr>
<td>RESET</td>
<td>63</td>
<td>RESET MASK</td>
<td>NONE</td>
<td></td>
</tr>
</tbody>
</table>

Figure 33. Modem Control Command Summary

The Set and Reset Port B commands are similar to the Initialization commands in that they use a mask parameter which defines the bits to be changed. Set Port B utilizes a logical OR mask and Reset Port B uses a logical AND mask. Setting a bit makes the pin active (low). Resetting the bit deactivates the pin (high).

To help clarify the numerous timing relationships that occur and their consequences, Figures 34 and 35 are provided as an illustration of several typical sequences. It is suggested that the reader go over these diagrams and re-read the appropriate part of the previous sections if necessary.

HLDC CONSIDERATIONS

The 8273 supports HDLC as well as SDLC. Let’s discuss how the 8273 handles the three basic HDLC/SDLC differences: extended addressing, extended control, and the 7 1s Abort character.

Recalling Figure 4A, HDLC supports an address field of indefinite length. The actual amount of extension used is determined by the least significant bit of the characters immediately following the opening flag. If the LSB is 0, more address field bytes follow. If the LSB is 1, this byte is the final address field byte. Software must be used to determine this extension.

If non-Buffered mode is used, the A, C, and I fields are in memory. The software must examine the initial characters to find the extent of the address field. If Buffered mode is used, the characters corresponding to the SDLC A and C fields are transferred to the CPU as interrupt results. Buffered mode assumes the two characters following the opening flag are to be transferred as interrupt results regardless of content or meaning. (The 8273...
does not know whether it is being used in an SDLC or an HDLC environment.) In SDLC, these characters are necessarily the A and C field bytes, however in HDLC, their meaning may change depending on the amount of extension used. The software must recognize this and examine the transferred results as possible address field extensions.

Frames may still be selectively received as is needed for secondary stations. The Selective Receive command is still used. This command qualifies a frame reception on the first byte following the opening flag matching either of the A1 or A2 match byte parameters. While this does not allow qualification over the complete range of HDLC addresses, it does perform a qualification on the first address byte. The remaining address field bytes, if any, are then examined via software to completely qualify the frame.

Once the extent of the address field is found, the following bytes form the control field. The same LSB test used for the address field is applied to these bytes to determine the control field extension, up to two bytes maximum. The remaining frame bytes in memory represent the information field.

The Abort character difference is handled in the Operating Mode register. If the HDLC Abort Enable bit is set, the reception of seven contiguous ones by an active receiver will generate an Abort Detect interrupt rather than eight ones. (Note that both the HDLC Abort Enable bit and the EOP Interrupt bit must not be set simultaneously.)

Now let’s move on to the SDLC loop configuration discussion.

**LOOP CONFIGURATION**

Aside from use in the normal data link applications, the 8273 is extremely attractive in loop configuration due to the special frame-level loop commands and the Digital Phase Locked Loop. Toward this end, this section details the hardware and software considerations when using the 8273 in a loop application.

The loop configuration offers a simple, low-cost solution for systems with multiple stations within a small physical location, i.e., retail stores and banks. There are two primary reasons to consider a loop configuration. The interconnect cost is lower for a loop over a multi-point configuration since only one twisted pair or fiber optic cable is used. (The loop configuration does not support the passing of distinct clock signals from station to station.) In addition, loop stations do not need the intelligence of a multi-point station since the loop protocol is simpler. The most difficult aspects of loop station design are clock recovery and implementation of one bit delay (both are handled neatly by the 8273).

Figure 36 illustrates a typical loop configuration with one controller and two down-loop secondaries. Each station must derive its own data timing from the received data stream. Recalling our earlier discussion of the DPLL notice that TxC and RxC clocks are provided by the DPLL output. The only clock required in the secondaries is a simple, non-synchronized clock at 32 times the desired baud rate. The controller requires both 32 x and 1 x clocks. (The 1 x is usually implemented by dividing the 32 x clock with a 5-bit divider. However, there is no synchronism requirement between these clocks so any convenient implementation may be used.)
APPLICATIONS

Figure 35. Sample Transmitter Timing Diagrams
A quick review of loop protocol is appropriate. All communication on the loop is controlled by the loop controller. When the controller wishes to allow the secondaries to transmit, it sends a polling frame (the control field contains a poll code) followed by an EOP (End-of-Poll) character. The secondaries use the EOP character to capture the loop and insert a response frame as will be discussed shortly.

The secondaries normally operate in the repeater mode, retransmitting received data with one bit time of delay. All received frames are repeated. The secondary uses the one bit time of delay to capture the loop.

When the loop is idle (no frames), the controller transmits continuous flag characters. This keeps transitions on the loop for the sake of down-loop phase locked loops. When the controller has a non-polling frame to transmit, it simply transmits the frame and continues to send flags. The non-polling frame is then repeated around the loop and the controller receives it to signify a complete traversal of the loop. At the particular secondary addressed by the frame, the data is transferred to memory while being repeated. Other secondaries simply repeat it.

If the controller wants to poll the secondaries, it transmits a polling frame followed by all 1s (no zero bit insertion). The final zero of the closing frame plus the first seven 1s form an EOP. While repeating, the secondaries monitor their incoming line for an EOP. When an EOP is received, the secondary checks if it has any response for the controller. If not, it simply continues repeating. If the secondary has a response, it changes the seventh EOP one into a zero (the one bit time of delay allows time for this) and repeats it, forming a flag for the down-loop stations. After this flag is transmitted, the secondary terminates its repeater function and inserts its response frame (with multiple preceding flags if necessary). After the closing flag of the response, the secondary re-enters its repeater function, repeating the up-loop controller 1s. Notice that the final zero of the response’s closing flag plus the repeated 1s from the controller form a new EOP for the next down-loop secondary. This new EOP allows the next secondary to insert a response if it desires. This gives each secondary a chance to respond.

Back at the controller, after the polling frame has been transmitted and the continuous 1s started, the controller waits until it receives an EOP. Receiving an EOP signifies to the controller that the original frame has propagated around the loop followed by any responses inserted by the secondaries. At this point, the controller may either send flags to idle the loop or transmit the next frame. Let’s assume that the loop is implemented completely with the 8273s and describe the command flows for a typical controller and secondary.

The loop controller is initialized with commands which specify that the NRZI, Preframe Sync, Flag Stream, and EOP interrupt modes are set. Thus, the controller encodes and decodes all data using NRZI format. Preframe Sync mode specifies that all transmitted frames be prefaces with 16 line transitions. This ensures that the minimum of 12 transitions needed by the DPLLs to lock after an all 1s line have occurred by the time the secondary sees a frame’s opening flag. Setting the Flag Stream mode starts the transmitter sending flags which idles the loop. And the EOP interrupt mode specifies that the controller processor will be interrupted whenever the active receiver sees an EOP, indicating the completion of a poll cycle.

When the controller wishes to transmit a non-polling frame, it simply executes a Frame Transmit command. Since the Flag Stream mode is set, no EOP is formed after the closing flag. When a polling frame is to be transmitted, a General Receive command is executed first. This enables the receiver and allows reception of all incoming frames; namely, the original polling frame plus any response frames inserted by the secondaries. After the General Receive command, the frame is transmitted with a Frame Transmit command. When the frame is complete, a transmitter interrupt is generated. The loop controller processor uses this interrupt to reset Flag Stream mode. This causes the transmitter to start sending all 1s. An EOP is formed by the last flag and the first 7 1s. This completes the loop controller transmit sequence.

At any time following the start of the polling frame transmission the loop controller receiver will start receiving frames. (The exact time difference depends, of course, on the number of down-loop secondaries due to each inserting one bit time of delay.) The first received frame is simply the original polling frame. However, any additional frames are those inserted by the secondaries. The loop controller processor knows all frames have been received when it sees an EOP interrupt. This interrupt is generated by the 8273s since the EOP interrupt mode was set during initialization. At this point, the transmitter may be commanded either to enter Flag
Stream mode, idling the loop, or to transmit the next frame. A flowchart of the above sequence is shown in Figure 37.

The secondaries are initialized with the NRZI and One Bit Delay modes set. This puts the 8273 into the repeater mode with the transmitter repeating the received data with one bit time of delay. Since a loop station cannot transmit until it sees and EOP character, any transmit command is queued until an EOP is received. Thus whenever the secondary wishes to transmit a response, a Loop Transmit command is issued. The 8273 then waits until it receives an EOP. At this point, the receiver changes the EOP into a flag, repeats it, resets One Bit Delay mode stopping the repeater function, and sets the transmitter into Flag Stream mode. This captures the loop. The transmitter now inserts its message. At the closing flag, Flag Stream mode is reset, and One Bit Delay mode is set, returning the 8273 to repeater function and forming an EOP for the next down-loop station. These actions happen automatically after a Loop Transmit command is issued.

When the secondary wants its receiver enabled, a Selective Loop Receive command is issued. The receiver then looks for a frame having a match in the Address field. Once such a frame is received, repeated, and transferred to memory, the secondary's processor is interrupted with the appropriate Match interrupt result and the 8273 continues with the repeater function until an EOP is received, at which point the loop is captured as above. The processor should use the interrupt to determine if it has a message for the controller. If it does, it simply issues a Loop Transmit command and things progress as above. If the processor has no message, the software must reset the Flag Stream mode bit in the Operating Mode register. This will inhibit the 8273 from capturing the loop at the EOP. (The match frame and the EOP may be separated in time by several frames depending on how many up-loop stations inserted messages of their own.) If the timing is such that the receiver has already captured the loop when the Flag Stream mode bit is reset, the mode is exited on a flag boundary and the frame just appears to have extra closing flags before the EOP. Notice that the 8273 handles the queuing of the transmit commands and the setting and resetting of the mode bits automatically. Figure 38 illustrates the major points of the secondary command sequence.

---

**Figure 37. Loop Controller Flowchart**

**Figure 38. Loop Secondary Flowchart**

---

*6-167*
When an off-line secondary wishes to come on-line, it must do so in a manner which does not disturb data on the loop. Figure 39 shows a typical hardware interface. The line labeled Port could be one of the 8273 Port B outputs and is assumed to be high (1) initially. Thus up-loop data is simply passed down-loop with no delay; however, the receiver may still monitor data on the loop. To come on-line, the secondary is initialized with only the EOP Interrupt mode set. The up-loop data is then monitored until an EOP occurs. At this point, the secondary’s CPU is interrupted with an EOP interrupt. This signals the CPU to set One Bit Delay mode in the 8273 and then to set Port low (active). These actions switch the secondary’s one bit delay into the loop. Since after the EOP only 1s are traversing the loop, no loop disturbance occurs. The secondary now waits for the next EOP, captures the loop, and inserts a “new on-line” message. This signals the controller that a new secondary exists and must be acknowledged. After the secondary receives its acknowledgement, the normal command flow is used.

It is hopefully evident from the above discussion that the 8273 offers a very simple and easy to implement solution for designing loop stations whether they are controllers or down-loop secondaries.

APPLICATIONS

APPLICATION EXAMPLE

This section describes the hardware and software of the 8273/8085 system used to verify the 8273 implementation of SDLC on an actual IBM SDLC Link. This IBM link was gratefully volunteered by Raytheon Data Systems in Norwood, Mass. and I wish to thank them for their generous cooperation. The IBM system consisted of a 370 Mainframe, a 3705 Communications Processor, and a 3271 Terminal Controller. A Comlink II Modem supplied the modern interface and all communications took place at 4800 baud. In addition to observing correct responses, a Spectron D601 Datascope was used to verify the data exchanges. A block diagram of the system is shown in Figure 40. The actual verification was accomplished by the 8273 system receiving and responding to polls from the 3705. This method was used on both point-to-point and multi-point configurations. No attempt was made to implement any higher protocol software over that of the poll and poll responses since such software would not affect the verification of the 8273 implementation. As testimony to the ease of use of the 8273, the system worked on the first try.

An SDK-85 (System Design Kit) was used as the core 8085 system. This system provides up to 4K bytes of ROM/EPIOM, 512 bytes of RAM, 76 I/O pins, plus two timers as provided in two 8755 Combination EPROM/I/O devices and two 8155 Combination RAM/I/O/Timer devices. In addition, 5 interrupt inputs are supplied on the 8085. The address, data, and control buses are buffered by the 8212 and 8216 latches and bidirectional bus drivers. Although it was not used in this application, an 8229 Display Driver/Keyboard Encoder is included to interface the on-board display and keyboard. A block diagram of the SDK-85 is shown in Figure 41. The 8273 and associated circuitry was constructed on the ample wire-wrap area provided for the user.

The example 8273/8085 system is interrupt-driven and uses DMA for all data transfers supervised by an 8257 DMA Controller. A 2400 baud asynchronous line, implemented with an 8251A USART, provides communication between the software and the user. 8253 Programmable Interval Timer is used to supply the baud rate clocks for the 8251A and 8273. (The 8273 baud rate clocks were used only during initial system debug. In actual operation, the modem supplied these clocks via the RS-232 interface.) Two 2142 1K x 4 RAMs provided 512 bytes of transmitter and 512 bytes of receiver buffer memory. (Command and result buffers, plus miscellaneous variables are stored in the 8155s.) The RS-232 interface utilized MC1488 and MC1489 RS-232 drivers and receivers. The schematic of the system is shown in Figure 42.

One detail to note is the DMA and interrupt structure of the transmit and receive channels. In both cases, the receiver is always given the higher priority (8257 DMA channel 0 has priority over the remaining channels and the 8085 RST 7.5 interrupt input has priority over the RST 6.5 input.) Although the choice is arbitrary, this technique minimizes the chance that received data could be lost due to other processor or DMA commitments.

Also note that only one 8205 Decoder is used for both the peripherals’ and the memories’ Chip Selects. This was done to eliminate separate memory and I/O decoders since it was known beforehand that neither address space would be completely filled.

The 4 MHz crystal and 8224 Clock Generator were used only to verify that the 8273 operates correctly at that maximum spec speed. In a normal system, the 3.072 MHz clock from the 8085 would be sufficient. (This fact was verified during initial checkout.)
Figure 41. SDK-85 Functional Block Diagram

Figure 42. 8273/SDK-85 System
The software consists of the normal monitor program supplied with the SDK-85 and a program to input commands to the 8273 and to display results. The SDK-85 monitor allows the user to read and write on-board RAM, start execution at any memory location, to single-step through a program, and to examine any of the 8085's internal registers. The monitor drives either the on-board keyboard/LED display or a serial TTY interface. This monitor was modified slightly in order to use the 8251A with a 2400 baud CRT as opposed to the 110 baud normally used. The 8273 program implements monitor-like user interface. 8273 commands are entered by a two-character code followed by any parameters required by that command. When 8273 interrupts occur, the source of the interrupt is displayed along with any results associated with it. To gain a flavor of how the user/program interface operates, a sample output is shown in Figure 43. The 8273 program prompt character is a "-" and user inputs are underlined.

The "SO 05" implements the Set Operating Mode command with a parameter of 05H. This sets the Buffer and Flag Stream modes. "SS 01" sets the 8273 in NRZl mode using the Set Serial I/O Mode command. The next command specifies General Receiver with a receiver buffer size of 0100H bytes (B0 = 00, B1 = 01). The "TF" command causes the 8273 to transmit a frame containing an address field of C2H and control field of 11H. The information field is 001122. The "TF" command has a special format. The L0 and L1 parameters are computed from the number of information field bytes entered.

After the TF command is entered, the 8273 transmits the frame (assuming that the modem protocol is observed). After the closing flag, the 8273 interrupts the 8085. The 8085 reads the interrupt results and places them in a buffer. The software examines this buffer for new results and if new results exist, the source of the interrupt is displayed along with the results.

In this example, the 0DH result indicates a Frame Complete interrupt. There is only one result for a transmitter interrupt, the interrupt's trailing zero results were included to simplify programming.

The next event is a frame reception. The interrupt results are displayed in the order read from the 8273. The E0H indicates a General Receive interrupt with the last byte of the information field received on an 8-bit boundary. The 03 00 (R0, Ri) results show that there are 3H bytes of information field received. The remaining two results indicate that the received frame had a C2H address field and a 34H control field. The 3 bytes of information field are displayed on the next line.

Figures 44 through 51 show the flowcharts used for the 8273 program development. The actual program listing is included as Appendix A. Figure 44 is the main status poll loop. After all devices are initialized and a prompt character displayed, a loop is entered at LOOPIT. This loop checks for a change of status in the result buffer or if a keyboard character has been received by the 8251 or if a poll frame has been received. If any of these conditions are met, the program branches to the appropriate routine. Otherwise, the loop is traversed again.

The result buffer is implemented as a 255-byte circular buffer with two pointers: CNADR and LDADR. CNADR is the console pointer. It points to the next result to be displayed LDADR is the load pointer. It points to the next empty position in the buffer into which the interrupt handler places the next result. The same buffer is used for both transmitter and receiver results. LOOPIT examines these pointers to detect when CNADR is not equal to LDADR indicating that the buffer contains results which have not been displayed. When this occurs, the program branches to the DISPLAY routine.

DISPLAY determines the source of the undisplayed results by testing the first result. This first result is necessarily the interrupt result code. If this result is 0CH or greater, the result is from a transmitter interrupt. Otherwise it is from a receiver source. The source of the result code is then displayed on the console along with the next four results from the buffer. If the source was a transmitter interrupt, the routine merely repoints the pointer CNADR and returns to LOOPIT. For a receiver source, the receiver data buffer is displayed in addition to the receiver interrupt results before returning to LOOPIT.

![Flowchart](https://example.com/flowchart.png)

Figure 44. Main Status Poll Loop

---

**8273 MONITOR V1.2**

- SO 05
- SS 01
- G 00 00 01
- TF C2 11 00 11 32

T x INT — 00 00 00 00 00

R x INT — E0 03 00 C2 34

FF EE DD

Figure 43. Sample 8273 Monitor I/O
If the result buffer pointers indicate an empty buffer, the 8251A is polled for a keyboard character. If the 8251 has a character, GETCMD is called. There the character is read and checked if legal. Illegal characters simply cause a reprompt. Legal characters indicate the start of a command input. Most commands are organized as two characters signifying the command action; i.e., GR — General Receive. The software recognizes the two character command code and takes the appropriate action. For non-Transmit type commands, the hex equivalent of the command is placed in the C register and the number of parameters associated with that command is placed in the B register. The program then branches to the COMM routine.

The COMM routine builds the command buffer by reading the required number of parameters from the keyboard and placing them at the buffer pointed at by CMDBUF. The routine at COMM2 then issues this command buffer to the 8273.

If a Transmit type command is specified, the command buffer is set up similarly to the the COMM routine; however, since the information field data is entered from the keyboard, an intermediate routine, TF, is called. TF loads the transmit data buffer pointed at by TxBUF. It counts the number of data bytes entered and loads this number into the command buffer as \(L_0\), \(L_1\). The command is then issued to the 8273 by jumping to CMDOUT.

One command does not directly result in a command being issued to the 8273. This command, Z, operates a software flip-flop which selects whether the software will respond automatically to received polling frames. If the Poll-Response mode is selected, the prompt character is changed to a ‘+’. If a frame is received which contains a prearranged poll control field, the memory location POLIN is made nonzero by the receiver interrupt handler. LOOPIT examines this location and if it is nonzero, causes a branch to the TxPOL routine. The TxPOL routine clears POLIN, sets a pointer to a special command buffer at CMDBUF1, and issues the command by way of the COMM2 entry in the COMM routine. The special command buffer contains the appropriate response frame for the poll frame received. These actions only occur when the Z command has changed the prompt to a ‘+’. If the prompt is normal ‘-’, polling frames are displayed as normal frames and no response is transmitted. The Poll-Response mode was used during the IBM tests.
APPLICATIONS

The final two software routines are the transmitter and receiver interrupt handlers. The transmit interrupt handler, Txl, simply saves the registers on the stack and checks if loading the result buffer will fill it. If the result buffer will overfill, the program is exited and control is passed to the SDK-85 monitor. If not, the results are read from the Txl/R register and placed in the result buffer at LDADR. The DMA pointers are then reset, the registers restored, and interrupts enabled. Execution then returns to the pre-interrupt location.

The receiver interrupt handler, Rxl, is only slightly more complex. As in Txl, the registers are saved and the possibility of overfilling the result buffer is examined. If the result buffer is not full, the results are read from Rxl/R and placed in the buffer. At this point the prompt character is examined to see if the Poll-Response mode is selected. If so, the control field is compared with two possible polling control fields. If there is a match, the special command buffer is loaded and the poll indicator, POLIN, is made nonzero. If no match occurred, no action is taken. Finally, the receiver DMA buffer pointers are reset, the processor status restored, and interrupts are enabled. The RET instruction returns execution to the pre-interrupt location.

This completes the discussion of the 8273/8085 system design.

CONCLUSION

This application note has covered the 8273 in some detail. The simple and low cost loop configuration was explored. And an 8273/8085 system was presented as a sample design illustrating the DMA/interrupt-driven interface. It is hoped that the major features of the 8273, namely the frame-level command structure and the Digital Phase Locked Loop, have been shown to be a valuable asset in an SDLC system design.
## APPLICATIONS

### APPENDIX A

ISIS-II 8038/8085 MACRO ASSEMBLER, XL08

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>SEQ</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>2</td>
<td>TRUE EQU 00H</td>
<td>.00 FOR RAYTHEON</td>
</tr>
<tr>
<td>0000</td>
<td>3</td>
<td>:FF FOR SELF-TEST</td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>4</td>
<td>TRUE1 EQU 00H</td>
<td>:00 FOR NORMAL RESPONSE</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>:FF FOR LOOP RESPONSE</td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>6</td>
<td>DEM EQU 00H</td>
<td>:00 FOR NO DEMO</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>:FF FOR DEMO</td>
<td></td>
</tr>
</tbody>
</table>

10  GENERAL 8273 MONITOR WITH RAYTHEON POLL MODE ADDED

19  COMMANDS SUPPORTED ARE:

- RS - RESET SERIAL I/O MODE
- SS - SET SERIAL I/O MODE
- PO - RESET OPERATING MODE
- SO - SET OPERATING MODE
- PD - RECEIVER DISABLE
- G - GENERAL RECEIVE
- SP - SELECTIVE RECEIVE
- TF - TRANSMIT FRAME
- AF - ABORT FRAME
- SP - SET PORT B
- RP - RESET PORT B
- RB - RESET ONE BIT DELAY (PAR = 7F)
- SB - SET ONE BIT DELAY (PAR = 80)
- SL - SELECTIVE LOOP RECEIVE
- TL - TRANSMIT LOOP
- Z - CHANGE MODES FLIP/FLOP

38  40 ; NOTE. 'SEI' COMMANDS IMPLEMENT LOGICAL 'OR' FUNCTIONS
41  'RESET' COMMANDS IMPLEMENT LOGICAL 'AND' FUNCTIONS
43  44 ; BUFFERED MODE MUST BE SELECTED WHEN SELECTIVE RECEIVE IS USED.
45  46 ; COMMAND FORMAT IS: 'COMMAND (2 LTRS)' 'PAR #1' 'PAR #2' ETC.
48  50 ; THE TRANSMIT FRAME COMMAND FORMAT IS: 'TF' 'A' 'C' 'BUFFER CONTENTS'.
51  NO LENGTH COUNT IS NEEDED. BUFFER CONTENTS IS ENDED WITH A CR.
APPLICATIONS

56 ; A SNRM-P OR RR(0)-P IS RECEIVED. A RESPONSE FRAME OF NSA-F
57 ; OR RR(0)-F IS TRANSMITTED. OTHER COMMANDS OPERATE NORMALLY.
62 ;
63 ;***********************************************************************
64 ,
65 :8273 EQUATES
66 :
0000 67 STAT73 EQU 90H ; STATUS REGISTER
0000 69 COMMAND EQU 90H ; COMMAND REGISTER
0001 69 PARM73 EQU 91H ; PARAMETER REGISTER
0001 70 REJ73 EQU 91H ; RESULT REGISTER
0002 71 TXIP73 EQU 92H ; TX INTERRUPT RESULT REGISTER
0003 72 RXIR73 EQU 93H ; RX INTERRUPT RESULT REGISTER
0002 73 TEST73 EQU 92H ; TEST MODE REGISTER
0000 74 CPBF EQU 20H ; PARAMETER BUFFER FULL BIT
0004 75 TXINT EQU 04H ; TX INTERRUPT BIT IN STATUS REGISTER
0008 76 RXINT EQU 08H ; RX INTERRUPT BIT IN STATUS REGISTER
0001 77 TXINT EQU 01H ; TX INT RESULT AVAILABLE BIT
0002 78 RXINT EQU 02H ; RX INT RESULT AVAILABLE BIT
73 ,
80 :8255 EQUATES
81 ;
0008 82 MODE52 EQU 98H ; 8255 MODE WORD REGISTER
000C 83 CNT0P52 EQU 9CH ; COUNTER 0 REGISTER
000D 84 CNT1P52 EQU 9DH ; COUNTER 1 REGISTER
000E 85 CNT2P52 EQU 9EH ; COUNTER 2 REGISTER
000C 86 CNT0 EQU 000CH ; CONSOLE BAUD RATE (2400)
0036 87 MOCNT0 EQU 36H ; MODE FOR COUNTER 0
0066 88 MOCNT2 EQU 06H ; MODE FOR COUNTER 2
2017 89 LKBR1 EQU 2017H ; 8253 BAUD RATE LSB ADR
2018 90 LKBR2 EQU 2018H ; 8253 BAUD RATE MSB ADR
91 ,
92 : BAUD RATE TABLE BAUD RATE LKBR1 LKBR2
93 ; ************* ***** ****
94 ; 9600 2E 00
95 ; 4800 5C 00
96 ; 2400 E9 00
97 ; 1200 72 01
98 ; 600 E5 02
99 ; 300 C9 05
100 ,
101 ;
102 :8257 EQUATES
103 ;
0006 104 MODE57 EQU 0A8H ; 8257 MODE PORT
0000 105 CHAROR EQU 0A8H ; CH0 CHAR ADR REGISTER
0001 106 CHORC EQU 0A1H ; CH0 TERMINAL COUNT REGISTER
0002 107 CHAROR EQU 0A2H ; CH1 CHAR ADR REGISTER
0003 108 CHORC EQU 0A3H ; CH1 TERMINAL COUNT REGISTER
0008 109 STAT57 EQU 0ABH ; STATUS REGISTER
8200 110 R/BUF EQU 8200H ; RX BUFFER START ADDRESS
8200 111 T/BUF EQU 8000H ; TX BUFFER START ADDRESS
0062 112 DEMA EQU 62H ; DISABLE RX DMA CHANNEL, TX STILL ON
41FF 113 RXCT EQU 41FFH ; TERMINAL COUNT AND MODE FOR RX CHANNEL
0063 114 ENDMA EQU 63H ; ENABLE BOTH TX AND RX CHANNELS-EXT. MTR, TX STOP
0061 115 DEMA EQU 61H ; DISABLE TX DMA CHANNEL, RX STILL ON
81FF 116 TXCT EQU 81FFH ; TERMINAL COUNT AND MODE FOR TX CHANNEL
117 ;

6-176 231311-001
APPLICATIONS

118. 8251A EQUATES

0009 120 CNTL51 EQU 89H ; CONTROL WORD REGISTER
0009 121 STATS51 EQU 89H ; STATUS REGISTER
0008 122 TX51 EQU 88H ; TX DATA REGISTER
0008 122 RX51 EQU 88H ; RX DATA REGISTER
000E 124 MDS51 EQU B7EH ; MODE 16X2 STOP, NO PARITY
0027 125 CMD51 EQU 27H ; COMMAND, ENABLE TX/RX
0002 126 RDY EQU 92H ; READY BIT

128. MONITOR SUBROUTINE EQUATES

061F 130 GETCH EQU 061FH ; GET CHAR FROM KEYBOARD, ASCII IN CH
05F8 131 ECHO EQU 05F8H ; ECHO CHAR TO DISPLAY
075E 132 VALDG EQU 075EH ; CHECK IF VALID DIGIT, CARRY SET IF VALID
0588 133 ONBN EQU 0588H ; CONVERT ASCII TO HEX
05EB 134 CPLF EQU 05EBH ; DISPLAY CR, HENCE LF TOO
06C7 135 NOUT EQU 06C7H ; CONVERT BYTE TO 2 ASCII CHR AND DISPLAY

136 ;
137. MISC EQUATES

138 ;
139 STKSRT EQU 2000H ; STACK START
0003 140 CNTLC EQU 83H ; CNTL-C EQUIVALENT
0008 141 MONITOR EQU 0008H ; MONITOR
2000 142 CMDUF EQU 2000H ; START OF COMMAND BUFFER
2020 143 CMDF1 EQU 2020H ; POLL MODE SPECIAL TX COMMAND BUFFER
0000 144 CR EQU 00H ; ASCII CR
000A 145 LF EQU 0AH ; ASCII LF
2004 146 RST75 EQU 2004H ; RST7 5 JUMP ADDRESS
20CE 147 RST65 EQU 20CEH ; RST6 5 JUMP ADDRESS
2010 148 LOADR EQU 2010H ; RESULT BUFFER LOAD POINTER STORAGE
2013 149 CNADR EQU 2013H ; RESULT BUFFER CONSOLE POINTER STORAGE
2800 150 RESBUF EQU 2800H ; RESULT BUFFER START - 255 BYTES
0002 151 SNRF EQU 93H ; SNRM-P CONTROL CODE
0011 152 RRAP EQU 11H ; RRAP(8)-P CONTROL CODE
007C 153 NSAF EQU 73H ; NSA-F CONTROL CODE
0011 154 RRBF EQU 11H ; RRBF(8)-F CONTROL CODE
2015 155 PRMTFT EQU 2015H ; PRMT FT STORAGE
2016 156 POLIN EQU 2016H ; POLL MODE SELECTION INDICATOR
2027 157 DEMODE EQU 2027H ; DEMO MODE INDICATOR

161 ;
162 ;**************************************************************************
163 ;
164 ; RAM STORAGE DEFINITIONS:
165 ; LOC   DEF
166 ;
167 ; 2000-200F COMMAND BUFFER
168 ; 2010-2011 RESULT BUFFER LOAD POINTER
169 ; 2013-2014 RESULT BUFFER CONSOLE POINTER
170 ; 2015 PROMPT CHARACTER STORAGE
171 ; 2016 POLL MODE INDICATOR
172 ; 2017 BAUD RATE LSB FOR SELF-TEST
173 ; 2018 BAUD RATE MSB FOR SELF-TEST
177 ; 2019 SPARE
179 ; 2020-2026 RESPONSE COMMAND BUFFER FOR POLL MODE
180 ; 2800-28FF RESULT BUFFER

181 ;
182 ;**************************************************************************
APPLICATIONS

137: ; PROGRAM START
138: ; INITIALIZE 8253, 8257, 8251A and RESET 8273
139: ; ALSO SET NORMAL MODE AND PRINT SIGNON MESSAGE
140: ;
141: 0000 189 ORG 000H
142: 0000 31C020 191 START: LXI SP. STKSRT ; INITIALIZE SP
143: 0003 3E36 192 MVI A. MDNT0 ; 8253 MODE SET
144: 0005 D39B 193 OUT MODE53 ; 8253 MODE PORT
145: 0007 3A1729 194 LDA LXBR1 ; GET 8273 BAUD RATE LSB
146: 0009 D39C 195 OUT CNT053 ; USING COUNTER 0 AS BAUD RATE GEN
147: 000B 3E18 196 LDA LXBR2 ; GET 8273 BAUD RATE MSB
148: 000D D39C 197 OUT CNT053 ; COUNTER 0
149: 0011 CD350B 198 CALL RXDMA ; INITIALIZE 8257 RX DMA CHANNEL
1410 CD350B 199 CALL TXDMA ; INITIALIZE 8257 TX DMA CHANNEL
1411 3E01 200 MVI A. 01H ; OUTPUT 1 FOLLOWED BY A 0
1412 D39C 201 OUT TEST73 ; TO TEST MODE REGISTER
1414 3E00 202 MVI A. 00H ; TO RESET THE 8273
1416 D39C 203 OUT TEST73 ;
1417 3E20 204 MVI A. /- ; NORMAL MODE PROMPT CHR
1419 321520 205 STA PRNTPT ; PUT IN STORAGE
141A 3E00 206 MVI A. 00H ; TX POLL RESPONSE INDICATOR
141B 321620 207 STA POLIN ; 0 MEANS NO SPECIAL TX
141C 322720 208 STA DEMODE ; CLEAR DEMO MODE
141D 2134 MVI A, 00H ; SIGNON MESSAGE ADR
141E 201 OUT TEST73 ; TO TEST LOAD REGISTER
141F 2134 MVI A. 00H ; POLL RESPONSE INDICATOR
1420 3A18 212 LXI H. SIGNON ; SIGNON MESSAGE ADDR
1421 205 OUT TEST73 ;
1422 3E20 213 CALL TXMSG ; DISPLAY SIGNON
1423
1424 ; MONITOR USES JUMPS IN RAM TO DIRECT INTERRUPTS
1425 ;
1426 21D420 217 LXI H. RST75 ; RST 7. 5 JUMP LOCATION USED BY MONITOR
1427 01000C 218 LXI B. RXI ; ADDRESS OF RX INT ROUTINE
1428 36C3 219 MVI M. 0C3H ; LOAD ' JMP' OPCODE
142A 220 INX H ; INC POINTER
142B 71 221 MOV M. C ; LOAD RXI LSB
142C 22 222 INX H ; INC POINTER
142D 70 223 MOV M. B ; LOAD RXI MSB
142E 21CE20 224 LXI H. RST65 ; RST 6. 5 JUMP LOCATION USED BY MONITOR
142F 0100B 225 LXI B. TXI ; ADDRESS OF TX INT ROUTINE
1430 36C3 226 MVI M. 0C3H ; LOAD ' JMP' OPCODE
1432 227 INX H ; INC POINTER
1433 71 228 MOV M. C ; LOAD TXI LSB
1434 22 229 INX H ; INC POINTER
1435 70 230 MOV M. B ; LOAD TXI MSB
1436 3E18 231 MVI A. 18H ; GET SET TO RESET INTERRUPTS
1438 30 232 SIM ; RESET INTERRUPTS
1439 40 FB 233 EI ; ENABLE INTERRUPTS
143A
143B ; INITIALIZE BUFFER POINTER
143C
143D 210028 238 LXI H. RESBUF ; SET RESULT BUFFER POINTERS
143E 211220 239 SHLD CNADR ; RESULT CONSOLE POINTER
1440 221220 240 SHLD LDADR ; RESULT LOAD POINTER
1441
1442 ; MAIN PROGRAM LOOP - CHECKS FOR CHANGE IN RESULT POINTERS, USART STATUS,
1443 ; OR POLL STATUS

6-178
APPLIC ATIONS

0857 CDEB05 245 CMDREC CALL CRLF ; DISPLAY CR
0859 3A1520 246 LDR FRMPT ; GET CURRENT PROMPT CHR
085D 4F 247 MOV C.A ; MOVE TO C
085E CDF805 248 CALL ECHO ; DISPLAY IT
0861 2A1320 249 LOOPIT: LHLD CNDR ; GET CONSOLE POINTER
0864 7D 250 MOV A.L ; SAVE POINTER LSB
0865 2A1020 251 LHLD LOADR ; GET LOAD POINTER
0868 8D 252 CMP L ; SAME LSB?
0869 C239A0 253 JNZ DISPY ; NO, RESULTS NEED DISPLAYING
086C D689 259 IN STAT51 ; YES, CHECK KEYBOARD
086E E682 260 CALL ECHO ; DISPLY IT
0870 C2708 261 JNZ GETCMD ; MUST BE CHR SO GO GET IT
0873 3A1629 262 LDR POLIN ; GET POLL MODE STATUS
0875 A7 263 ANA A ; IS IT O?
0877 C24C09 264 JNZ TXPOL ; NO, THEN POLL OCCURRED
087A C36108 265 JMP LOOPIT ; YES, TRY AGAIN

266 ;
267 ;
268 ; COMMAND RECOGNIZER ROUTINE
269 ;
270 ;
087D CDIF06 271 GETCMD CALL GETCH ; GET CHR
0880 CDF805 272 CALL ECHO ; ECHO IT
0883 79 273 MOV A.C ; SETUP FOR COMPARE
0884 FE52 274 CPI 'R' ; R?
0886 CAFB08 275 JZ ROWN ; GET MORE
0889 FE53 276 CPI 'S' ; S?
088B CAD708 277 JZ SDWN ; GET MORE
088E FE47 278 CPI 'G' ; G?
0890 CAF808 279 JZ GDWN ; GET MORE
0893 FE54 280 CPI 'T' ; T?
0895 C9E009 281 JZ TDWN ; GET MORE
0898 FE41 282 CPI 'A' ; A?
0899 C2209 283 JZ ADWN ; GET MORE
089D FE5A 284 CPI 'Z' ; Z?
089F C3109 285 JZ CNODE ; YES, GO CHANGE MODE
08A2 FE83 289 CPI CNTLC ; CNTLC-C?
08A4 CA6800 291 JZ MONITOR ; EXIT TO MONITOR
08A7 BCF3 292 ILLEG. ANI C,'?' ; PRINT ?
08A9 CDIF05 293 CALL ECHO ; DISPLAY IT
08AC C35708 294 JMP CMDREC ; LOOP FOR COMMAND
295 ;
08AF CDIF06 296 ROWN: CALL GETCH ; GET NEXT CHR
08B2 CDF805 297 CALL ECHO ; ECHO IT
08B5 79 298 MOV A.C ; SETUP FOR COMPARE
08B6 FE4F 299 CPI 'O' ; O?
08B8 C5D609 300 JZ ROCMD ; RO COMMAND
08B8 FE53 301 CPI 'S' ; S?
08BD CA7609 302 JZ RSCMD ; RS COMMAND
08D0 FE44 303 CPI 'D' ; D?
08D2 CA7109 304 JZ RDCMD ; RD COMMAND
08D5 FE50 305 CPI 'P' ; P?
08D7 CA8809 306 JZ RPCMD ; RP COMMAND
08DA C5E800 308 CPI 'A' ; A?
08D9 FE42 309 CPI 'B' ; B?
08E1 CA7B09 310 JZ RBCMD ; RB COMMAND

6-179 231311-001
APPLICATIONS

00D4 C3708 311  JMP  ILLEG  ; ILLEGAL, TRY AGAIN
00D7 CDLF06 313  SDWN: CALL GETCH  ; GET NEXT CHR
00DA CDF085 314  CALL ECHO ; ECHO IT
00D0 78 315  MOV A, B  ; SETUP FOR COMPARE
00DE FE4F 316  CPI '0'  ; 0?
00E8 CDLF89 317  JZ SOCMD  ; SO COMMAND
00E3 FE53 318  CPI 'S'  ; 0?
00E5 CDLF99 319  JZ SSCMD  ; SS COMMAND
00E8 FE52 320  CPI 'R'  ; R?
00EA CDLF99 321  JZ SRCMD  ; SR COMMAND
00ED FE58 322  CPI 'P'  ; P?
00EF CDFE89 323  JZ SPCMD  ; SP COMMAND
00F2 FE42 324  CPI 'B'  ; B?
00F4 CDLF09 325  JZ SBCMD  ; SB COMMAND
00F7 FE4C 326  CPI 'L'  ; L?
00F9 CDLF09 327  JZ SLCMD  ; SL COMMAND
00FC C3708 328  JMP  ILLEG  ; ILLEGAL, TRY AGAIN
00FF CDLF06 330  SDWN: CALL GETCH  ; GET NEXT CHR
0102 CDF085 331  CALL ECHO ; ECHO IT
0105 78 332  MOV A, B  ; SETUP FOR COMPARE
0106 FE52 333  CPI 'R'  ; R?
0108 CDLF89 334  JZ GRCMD  ; GR COMMAND
010B C3708 335  JMP  ILLEG  ; ILLEGAL, TRY AGAIN
010B CDF085 337  SDWN: CALL GETCH  ; GET NEXT CHR
0111 CDF085 338  CALL ECHO ; ECHO IT
0114 78 339  MOV A, B  ; SETUP FOR COMPARE
0115 FE46 340  CPI 'F'  ; F?
0117 CDFE09 341  JZ TFCMD  ; TF COMMAND
011A FE4C 342  CPI 'L'  ; L?
011C CDLF99 343  JZ TCLMD  ; TL COMMAND
011F C3708 344  JMP  ILLEG  ; ILLEGAL, TRY AGAIN
0123 CDLF06 346  SDWN: CALL GETCH  ; GET NEXT CHR
0125 CDF085 347  CALL ECHO ; ECHO IT
0128 78 348  MOV A, B  ; SETUP FOR COMPARE
0129 FE46 349  CPI 'F'  ; F?
012B CDFE89 350  JZ AFCMD  ; AF COMMAND
012E C3708 351  JMP  ILLEG  ; ILLEGAL, TRY AGAIN
0131 337  SDWN: CALL GETCH  ; GET NEXT CHR
0135 CDF085 338  CALL ECHO ; ECHO IT
0138 78 339  MOV A, B  ; SETUP FOR COMPARE
0139 FE46 340  CPI 'F'  ; F?
013B CDFE89 341  JZ AFCMD  ; AF COMMAND
013E C3708 351  JMP  ILLEG  ; ILLEGAL, TRY AGAIN
0141 337  SDWN: CALL GETCH  ; GET NEXT CHR
0145 CDF085 338  CALL ECHO ; ECHO IT
0148 78 339  MOV A, B  ; SETUP FOR COMPARE
0149 FE46 340  CPI 'F'  ; F?
014B CDFE89 341  JZ AFCMD  ; AF COMMAND
014E C3708 351  JMP  ILLEG  ; ILLEGAL, TRY AGAIN
014F 337  SDWN: CALL GETCH  ; GET NEXT CHR
0153 CDF085 338  CALL ECHO ; ECHO IT
0156 78 339  MOV A, B  ; SETUP FOR COMPARE
0157 FE46 340  CPI 'F'  ; F?
015A CDFE89 341  JZ AFCMD  ; AF COMMAND
015E C3708 351  JMP  ILLEG  ; ILLEGAL, TRY AGAIN

0301 F3 355  CMODE  DI  ; DISABLE INTERRUPTS
0302 3A1520 356  LDA PRMPT  ; GET CURRENT PROMPT
0305 FE20 357  CPI  ; NORMAL MODE?
0307 C24309 358  JNZ SW  ; NO. CHANGE IT
0309 3E2B 359  MVI A, 4  ; NEW PROMPT
030C 321520 360  STA PRMPT  ; STORE NEW PROMPT
030F FB 365  EI  ; ENABLE INTERRUPTS
0310 C3788 366  JMP CHDREC  ; RETURN TO LOOP
0313 3E2D 367  SW: MVI A, '4'  ; NEW PROMPT CHR
0315 321520 368  STA PRMPT  ; STORE IT
0318 FB 369  EI  ; ENABLE INTERRUPTS
0319 C3788 370  JMP CHDREC  ; RETURN TO LOOP

6-180
APPLICATIONS

373 ; TRANSMIT ANSWER TO POLL SETUP
374 ;
094C 3E00 382 TXPOL: MVI B,00H ; CLEAR POLL INDICATOR
094E 321620 384 STA POLIN ; INDICATOR ADDR
0951 216108 385 LXI H,LOOPIT ; SETUP STACK FOR COMMAND OUTPUT
0954 E5 386 PUSH H ; PUT RETURN TO CMDREC ON STACK
0955 06B4 387 MVI B,04H ; GET # OF PARAMETERS READY
0957 212020 388 LXI H,CMD8FL ; POINT TO SPECIAL BUFFER
095A C3FOA 389 JMP COMM2 ; JUMP TO COMMAND OUTPUTER
390 ;
391 ;
392 ; COMMAND IMPLEMENTING ROUTINES
394 ;
095D 0601 398 ROCMD: MVI B,01H ; # OF PARAMETERS
095F 0E51 399 MVI C,05H ; COMMAND
0961 C0E50A 400 CALL COMM ; GET PARAMETERS AND ISSUE COMMAND
0964 C35708 401 JMP CMDREC ; GET NEXT COMMAND
402 ;
403 ; RS - RESET SERIAL I/O MODE COMMAND
404 ;
0967 0601 405 RSCMD: MVI B,01H ; # OF PARAMETERS
0969 0E50 406 MVI C,00H ; COMMAND
096B C0E50A 407 CALL COMM ; GET PARAMETERS AND ISSUE COMMAND
096E C35708 408 JMP CMDREC ; GET NEXT COMMAND
409 ;
410 ; RD - RECEIVER DISABLE COMMAND
411 ;
0971 0600 412 ROCMD: MVI B,00H ; # OF PARAMETERS
0973 0EC5 413 MVI C,0C5H ; COMMAND
0975 C0E50A 414 CALL COMM ; ISSUE COMMAND
0978 C35708 415 JMP CMDREC ; GET NEXT COMMAND
416 ;
417 ; RB - RESET ONE BIT DELAY COMMAND
418 ;
097B 0601 419 ROCMD: MVI B,01H ; # OF PARAMETERS
097D 0E44 420 MVI C,04H ; COMMAND
097F C0E50A 421 CALL COMM ; GET PARAMETER AND ISSUE COMMAND
0982 C35708 422 JMP CMDREC ; GET NEXT COMMAND
423 ;
424 ; SB - SET ONE BIT DELAY COMMAND
425 ;
0985 0601 426 SBCMD: MVI B,01H ; # OF PARAMETERS
0987 0E44 427 MVI C,04H ; COMMAND
0989 C0E50A 428 CALL COMM ; GET PARAMETER AND ISSUE COMMAND
098C C35708 429 JMP CMDREC ; GET NEXT COMMAND
430 ;
431 ; SL - SELECTIVE LOOP RECEIVE COMMAND
432 ;
098F 0604 433 SLCMD: MVI B,04H ; # OF PARAMETERS
0991 0CE2 434 MVI C,0C2H ; COMMAND
0993 C0E50A 435 CALL COMM ; GET PARAMETERS AND ISSUE COMMAND
0996 C35708 436 JMP CMDREC ; GET NEXT COMMAND
437 ;
438 ; TL - TRANSMIT LOOP COMMAND

6-181
APPLICATIONS

0999 210020 448 TCMD: LXI H.CHDBUF,SET COMMAND BUFFER POINTER
099C 0682 441 MVI B.02H,LOAD PARAMETER COUNTER
099E 36CA 442 MVI M.0CAH,LOAD COMMAND INTO BUFFER
099B 218220 443 LXI H.CHDBUF+2,POINT AT ADDR AND CTRL POSITIONS
0993 C3F609 444 JMP TFCMD1,FINISH OFF COMMAND IN TF ROUTINE

445;

446; 50 - SET OPERATING MODE COMMAND
447;

09A6 0601 448 SCMD: MVI B.01H,# OF PARAMETERS
09A8 0E31 449 MVI C.91H,COMMAND
09AA CDE50A 450 CALL COMM,GET PARAMETER AND ISSUE COMMAND
09AD C35708 451 JMP CMDREC,GET NEXT COMMAND

452;

453; SS - SET SERIAL I/O COMMAND
454;

09B0 0601 455 SCMD: MVI B.01H,# OF PARAMETERS
09B2 0E00 456 MVI C.0A0H,COMMAND
09B4 CDE50A 457 CALL COMM,GET PARAMETER AND ISSUE COMMAND
09B7 C35708 458 JMP CMDREC,GET NEXT COMMAND

459;

460; SR - SELECTIVE RECEIVE COMMAND
461;

09B9 0604 462 SCMD: MVI B.04H,# OF PARAMETERS
09BC 0EC1 463 MVI C.0C1H,COMMAND
09BE CDE50A 464 CALL COMM,GET PARAMETERS AND ISSUE COMMAND
09C1 C35708 465 JMP CMDREC,GET NEXT COMMAND

466;

467; GR - GENERAL RECEIVE COMMAND
468;

09C4 0682 469 GRCMD: MVI B.02H,NO PARAMETERS
09C6 0EC0 470 MVI C.0C0H,COMMAND
09C8 CDE50A 471 CALL COMM,ISSUE COMMAND
09CB C35708 472 JMP CMDREC,GET NEXT COMMAND

473;

474; AF - ABORT FRAME COMMAND
475;

09CE 0600 476 AFCMD: MVI B.00H,NO PARAMETERS
09D0 0ECC 477 MVI C.0CCH,COMMAND
09D2 CDE50A 478 CALL COMM,ISSUE COMMAND
09D5 C35708 479 JMP CMDREC,GET NEXT COMMAND

480;

481; RP - RESET PORT COMMAND
482;

09D8 0601 483 RPMC: MVI B.01H,# OF PARAMETERS
09DA 0E53 484 MVI C.63H,COMMAND
09DC CDE50A 485 CALL COMM,GET PARAMETER AND ISSUE COMMAND
09DF C35708 486 JMP CMDREC,GET NEXT COMMAND

487;

488; SP - SET PORT COMMAND
489;

09E2 0601 490 SCMD: MVI B.01H,# OF PARAMETERS
09E4 0E33 491 MVI C.033H,COMMAND
09E6 CDE50A 492 CALL COMM,GET PARAMETER AND ISSUE COMMAND
09E9 C35708 493 JMP CMDREC,GET NEXT COMMAND

494;

495; TF - TRANSMIT FRAME COMMAND
496;

6-182
APPLICATIONS

09EC 210020 497 FCMD: LXI H. CMDBUF ;SET COMMAND BUFFER POINTER
09EF 69B2 498 MVI B. 02H ;LOAD PARAMETER COUNTER
09F1 36C8 499 MVI M. 0CH ;LOAD COMMAND INTO BUFFER
09F3 210220 500 LXI H. CMDBUF+2 ;POINT AT ADR AND CNTL POSITIONS
09F6 78 501 FCMD1: MOV A, B ;TEST PARAMETER COUNT
09F7 A7 502 ANA A ;IS IT 0?
09FB 503 TFCMD1: MOV A, B ;TEST PARAMETER COUNT
09F7 8F 504 CALL PARIN ;GET BUFFER POINTER
09F6 505 JZ TBUFL ;YES, LOAD TX DATA BUFFER
09FE 506 INX M ;INC COMMAND BUFFER POINTER
09F8 507 DCR B ;DEC PARAMETER COUNTER
09F3 77 508 MOV M. A ;LOAD PARAMETER INTO COMMAND BUFFER
09F4 C3F609 509 JMP TFCMD1 ;GET NEXT PARAMETER
09F7 210080 510 TBUL: LXI H. TXBUF ;LOAD TX DATA BUFFER POINTER
09F8 010080 511 LXI B. 0000H ;CLEAR BC - BYTE COUNTER
09F0 C5 512 TBUL1: PUSH B ;SAVE BYTE COUNTER
09F8 C0DA0A 513 CALL PARIN ;GET DATA ALIAS PARAMETER
09A1 DBA0A 514 JC ENDC; MAYBE END IF ILLEGAL
09A4 77 515 MOV M. A ;LOAD DATA BYTE INTO BUFFER
09A5 23 516 INX H ;INC BUFFER POINTER
09A6 C1 517 POP B ;RESTORE BYTE COUNTER
09A7 83 518 INX B ;INC BYTE COUNTER
09A8 C3000A 519 JMP TBUFFL ;GET NEXT DATA
09A9 FEB0 520 ENDC: CPI CR ;RETURNED ILLEGAL CR CR?
09A9 C240A 521 JZ TBUFFL ;YES, THEN TX BUFFER FULL
09A9 C3001 522 POP B ;RESTORE B TO SAVE STACK
09A9 C3708 523 JMP ILLEG ;ILLEGAL CHR
09A9 C3BF0A 524 JMP TBUFFL ;RESTORE BYTE COUNTER
09A9 210120 525 LXI H. CMDBUF+1 ;POINT INTO COMMAND BUFFER
09AB 71 526 MOV M. C ;STORE BYTE COUNT LSB
09A9 23 527 INX H ;INC POINTER
09A9 70 528 MOV M. B ;STORE BYTE COUNT MSB
09A9 69B4 529 MVI B. 04H ;LOAD PARAMETER COUNT INTO B
09A9 21360A 530 LXI H. TFRET ;GET RETURN ADDR FOR THIS ROUTINE
09A9 C5 531 PUSH B ;PUSH ONCE
09A9 E3 532 XTHL ;PUT RETURN ON STACK
09A9 C5 533 POP B ;PUSH IT SO CMDOUT CAN USE IT
09A9 C3F609 534 JMP CMDOUT ;ISSUE COMMAND
09A9 C3570B 535 JMP TFRET ;GET NEXT COMMAND
09A9 1695 536 DISP Y H. 05H ;D IS RESULT COUNTER
09A9 211320 537 LHL DNRADR ;GET CONSOLE POINTER
09A9 E5 538 PUSH H ;SAVE IT
09A9 7E 539 MOV A. M ;GET RESULT IC
09A4 E51F 540 ANI 1FH ;LIMIT TO RESULT CODE
09A2 FC8C 541 CPI OCH ;TEST IF RX OR TX SOURCE
09A4 D920A 542 JC RXSRC ;CARRY, THEN RX SOURCE
09A4 21C30C 543 TXSRC: LXI H. TXMSG ;TX INT MESSAGE
09A4 C0920C 544 CALL TMS Note ;DISPLAY IT
09A4 7E 545 DISPY2; POP H ;RESTORE CONSOLE POINTER
09A4 7E 546 DISPY3; MOV A. M ;GET RESULT
09A4 CDC705 547 CALL NMOUT ;CONVERT AND DISPLAY

6-183
APPLICATIONS

0A52 0E20 555 MVI C,' ;SP CHR
0A54 COF805 556 CALL ECHO ;DISPLAY IT
0A57 2C 557 INR L ;INC BUFFER POINTER
0A58 13 558 DCR D ;DEC RESULT COUNTER
0A59 C24E0A 559 JNZ DISPLAY ;NOT DONE
0A5C 221320 560 SHLD CNADR ;UPDATE CONSOLE POINTER
0A5F C35708 561 JMP CNRECC ;RETURN TO LOOP
562 ;
563 ;
564 ;RECEIVER SOURCE - DISPLAY RESULTS AND RECEIVE BUFFER CONTENTS
565 ;
566 ;
0A62 21080C 567 RXSORC: LXI H,RXMSG ;RX INT MESSAGE ADDR
0A65 CO920C 568 CALL TYMSG ;DISPLAY MESSAGE
0A68 E1 569 POP H ;RESTORE CONSOLE POINTER
0A69 7E 570 RXS1: MOV A,M ;RETRIEVE RESULT FROM BUFFER
0A6A C0C706 571 CALL NMOUT ;CONVERT AND DISPLAY IT
0A6D 0E20 572 MVI C,' ;ASCII SP
0A6F CDFB05 573 CALL ECHO ;DISPLAY IT
0A72 2C 574 INR L ;INC CONSOLE POINTER
0A73 15 575 DCR D ;DEC RESULT COUNTER
0A74 7A 576 MOV A,D ;GET SET TO TEST COUNTER
0A75 FE84 577 CPI 04H ;IS THE RESULT R0?
0A77 C9A20A 578 JZ ROPT ;YES, GO SAVE IT
0A79 FE83 579 CPI 03H ;IS THE RESULT R1?
0A7C C9A70A 580 JZ ROPT ;YES, GO SAVE IT
0A7F 87 581 RXS2: ANA A ;TEST RESULT COUNTER
0A80 2690A 582 JNZ RXS1 ;NOT DONE YET, GET NEXT RESULT
0A83 221320 583 SHLD CNADR ;DONE, SO UPDATE CONSOLE POINTER
0A86 COE805 584 CALL CRFL ;DISPLAY CR
0A89 210802 585 LXI H,RXBUF ;POINT AT RX BUFFER
0A8C C1 586 POP B ;RETRIEVE RECEIVED COUNT
0A8D 78 587 RXS3: MOV A,B ;IS COUNT 07?
0A8E 81 588 ORA C ;
0A8F C9S708 589 JZ CNRECC ;YES, GO BACK TO LOOP
0A92 7E 590 MOV A,M ;NO, GET CHR
0A93 C5 591 PUSH B ;SAVE BC
0A94 C0C706 592 CALL NMOUT ;CONVERT AND DISPLAY CHR
0A97 0E20 593 MVI C,' ;ASCII SP
0A99 COF805 594 CALL ECHO ;DISPLAY IT TO SEPARATE DATA
0A9C C1 595 POP B ;RESTORE BC
0A9B 8B 596 DEX B ;DEC COUNT
0A9E 23 597 INX H ;INC POINTER
0A9F C38D0A 598 JMP RXS3 ;GET NEXT CHR
599 ;
0A92 4E 580 ROPT: MOV C,M ;GET R0 FOR RESULT BUFFER
0A93 C5 581 PUSH B ;SAVE IT
0A94 C37F0A 582 JMP RXS2 ;RETURN
583 ;
0A97 C1 584 ROPT: POP B ;GET R0
0A98 46 585 MOV B,M ;GET R1 FOR RESULT BUFFER
0A99 C5 586 PUSH B ;SAVE IT
0A9A C37F0A 587 JMP RXS2
588 ;
589 ;
590 ;
591 ;PARAMETER INPUT - PARAMETER RETURNED IN E REGISTER
592 ;
APPLICATIONS

613

614 PARIN: PUSH B ;SAVE BC
615 MVI D, 01H ;SET CHR COUNTER
616 CALL GETCH ;GET CHR
617 CALL ECHO ;ECHO IT
618 MOV A, C ;PUT CHR IN A
619 CPI ;SP?
620 JNZ PARINL ;NO, ILLEGAL, TRY AGAIN
621 CALL GETCH ;GET CHR OF PARAMETER, 'I'
622 CALL VALDG ;IS IT A VALID CHR?
623 JNC PARINL ;NO, TRY AGAIN
624 CALL CNNBN ;CONVERT IT TO HEX
625 MOV C, A ;SAVE IT IN C
626 JMP PARIN2 ;YES, DONE WITH THIS PARAMETER
627 MOV A, D ;GET CHR COUNTER
628 ANA A ;IS IT 0?
629 JZ PARIN2 ;YES, DONE WITH THIS PARAMETER
630 DCR D ;DEC CHR COUNTER
631 XRA A ;CLEAR CARRY
632 MOV A, C ;RECOVER 1ST CHR
633 RAL ;ROTATE LEFT 4 PLACES
634 RAL
635 RAL
636 RAL
637 MOV E, A ;SAVE IT IN E
638 JMP PARIN3 ;GET NEXT CHR
639 PARIN2: MOV A, C ;2ND CHR IN A
640 ORA E ;COMBINE BOTH CHR'S
641 POP B ;RESTORE BC
642 RET ;RETURN TO CALLING PROGRAM
643 MOV A, C ;PUT ILLEGAL CHR IN A
644 STC ;SET CARRY AS ILLEGAL STATUS
645 POP B ;RESTORE BC
646 RET ;RETURN TO CALLING PROGRAM
647 ;
648 ;
649 JUMP HERE IF BUFFER FULL
650 ;
651 BUUFFUL: DB 0CFH ;EXIT TO MONITOR
652 ;
653 ;
654 COMMAND DISPATCHER
655 ;
656 ;
657 COMM: LXI H, CMDBUF ;SET POINTER
658 PUSH B ;SAVE BC
659 MOV M, C ;LOAD COMMAND INTO BUFFER
660 COMM: MOV A, B ;CHECK PARAMETER COUNTER
661 ANA A ;IS IT 0?
662 JZ CMDOUT ;YES, GO ISSUE COMMAND
663 CALL PARIN ;GET PARAMETER
664 JC ILLEG ;ILLEGAL CHR RETURNED
665 INV H ;INC BUFFER POINTER
666 DCR B ;DEC PARAMETER COUNTER
667 MOV M, A ;PARAMETER TO BUFFER
668 JMP COMML ;GET NEXT PARAMETER
669 CMDOUT: LXI H, CMDBUF ;REPOINT POINTER
670 POP B ;RESTORE PARAMETER COUNT
6-185  231311-001
APPLICATIONS

0AFF DB90  671  COM2: IN  STATUS : READ 8273 STATUS
0B01 07  672  RLC  : ROTATE CBSY INTO CARRY
0B02 DAF0A  673  JC  : WAIT FOR OK
0B05 7E  674  MOV  A,M  ; OK, MOVE COMMAND INTO A
0B06 D390  675  OUT  COMM73  ; OUTPUT COMMAND
0B08 78  676  PAR1: MOV  A,B  ; GET PARAMETER COUNT
0B09 87  677  ANA  A  ; IS IT 0?
0B0A C8  678  RZ  ; YES, DONE, RETURN
0B0B 23  679  INK  H  ; INC COMMAND BUFFER POINTER
0B0C 85  680  DCR  B  ; DEC PARAMETER COUNT
0B0D DB90  681  PAR2: IN  STATUS : READ STATUS
0B0F E2B0  682  ANI  CPAF  ; IS CPAF BIT SET?
0B11 C00B  683  JNZ  PAR2  ; WAIT TIL ITS 0
0B14 7E  684  MOV  A,M  ; OK, GET PARAMETER FROM BUFFER
0B15 D391  685  OUT  PARM73  ; OUTPUT PARAMETER
0B17 C00B  686  JMP  PAR1  ; GET NEXT PARAMETER

687 ;
688 ;
689 ; INITIALIZE AND ENABLE RX DMA CHANNEL
690 ;
691 ;
0B1A 3E62  692  RXDMA: MVI  A,DRMA  ; DISABLE RX DMA CHANNEL
0B1C D398  693  OUT  MODE57  ; 8257 MODE PORT
0B1E 010080  694  LXI  B,ROBUF  ; RX BUFFER START ADDRESS
0B21 79  695  MOV  A,C  ; RX BUFFER LSB
0B22 D390  696  OUT  CHAADR  ; RX CH ADDR PORT
0B24 78  697  MOV  A,B  ; RX BUFFER MSB
0B25 D390  698  OUT  CHAADR  ; RX CH ADDR PORT
0B27 01FF41  699  LXI  B,RTC  ; RX CH TERMINAL COUNT
0B28 79  700  MOV  A,C  ; RX TERMINAL COUNT LSB
0B28 D391  701  OUT  CHOTC  ; RX CH PORT
0B29 78  702  MOV  A,B  ; RX TERMINAL COUNT MSB
0B2E D391  703  OUT  CHOTC  ; RX CH PORT
0B30 3E63  704  MVI  A,ENDMA  ; ENABLE DMA WORD
0B32 D398  705  OUT  MODE57  ; 8257 MODE PORT
0B34 C9  706  RET  ; RETURN
707 ;
708 ;
709 ; INITIALIZE AND ENABLE TX DMA CHANNEL
710 ;
711 ;
0B35 3E61  712  TXDMA: MVI  A,DTMA  ; DISABLE TX DMA CHANNEL
0B37 D398  713  OUT  MODE57  ; 8257 MODE PORT
0B39 010080  714  LXI  B,TXBUF  ; TX BUFFER START ADDRESS
0B3C 79  715  MOV  A,C  ; TX BUFFER LSB
0B3D D392  716  OUT  CHLADR  ; CHL ADDR PORT
0B3F 78  717  MOV  A,B  ; TX BUFFER MSB
0B40 D392  718  OUT  CHLADR  ; CHL ADDR PORT
0B42 01FF81  719  TXDMA1: LXI  B,TXTC  ; TX CH TERMINAL COUNT
0B45 79  720  MOV  A,C  ; TX TERMINAL COUNT LSB
0B46 D3A3  721  OUT  CHLC  ; CHL PORT
0B48 78  722  MOV  A,B  ; TX TERMINAL COUNT MSB
0B49 D3A3  723  OUT  CHLC  ; CHL PORT
0B4B 3E63  724  MVI  A,ENDMA  ; ENABLE DMA WORD
0B4C D398  725  OUT  MODE57  ; 8257 MODE PORT
0B4F C9  726  RET  ; RETURN
727 ;
728 ;
APPLICATIONS

729; INERRUPT PROCESSING SECTION
730;

0C00 ORG 0C00H
731
732;
733
734; RECEIVER INTERRUPT - RST 7.5 (LOC 3CH)
735:

0C00 05 736 RXI1 PUSH H ; SAVE HL
0C01 F5 737 PUSH PSW ; SAVE PSW
0C02 C5 738 PUSH B ; SAVE BC
0C03 05 739 PUSH D ; SAVE DE
0C04 3E62 740 MVI A.DROMA ; DISABLE RX DMA
0C05 0308 741 OUT MODE5; ; 8257 MODE PORT
0C06 3E18 742 MVI A.18H ; RESET RST 7.5 F/F
0C08 30 743 SIM
0C0B 1604 744 MVI D.04H ; D IS RESULT COUNTER
0C10 2A120 745 LHLD LAADR ; GET LOAD POINTER
0C10 E5 746 PUSH H ; SAVE IT
0C11 E5 747 PUSH H ; SAVE IT AGAIN
0C12 45 748 MOV B.L ; SAVE LSB
0C13 4A1230 749 LHLD CNADR ; GET CONSOLE POINTER
0C16 04 750 RXI1 INR B ; BUMP LOAD POINTER LSB
0C17 7B 751 MOV A.B ; GET SET TO TEST
0C18 BD 752 CMP L ; LOAD=CONSOLE?
0C19 CAE40A 753 JZ BUFFUL ; YES, BUFFER FULL
0C1C 15 754 DCR D ; DEC COUNTER
0C1D C2160C 755 JNZ RXI1 ; NOT DONE, TRY AGAIN
0C20 1605 756 MVI D.05H ; RESET COUNTER
0C22 E1 757 POP H ; RESTORE LOAD POINTER
0C23 0B90 758 RXI2 IN STAT73 ; READ STATUS
0C25 E608 759 ANI RXINT ; TEST RX INT BIT
0C27 C9390C 760 JZ RXI3 ; DONE; GO FINISH UP
0C29 0B90 761 IN STAT73 ; READ STATUS AGAIN
0C2C E802 762 ANI RXIRA ; IS RESULT READY?
0C2E C9230C 763 JZ RXI2 ; NO; TEST AGAIN
0C31 0B93 764 IN RXIR73 ; YES, READ RESULT
0C33 77 765 MOV M.A ; STORE IN BUFFER
0C34 2C 766 INR L ; INC BUFFER POINTER
0C35 15 767 DCR D ; DEC COUNTER
0C36 C3230C 768 JMP RXI2 ; GET MORE RESULTS
0C39 7A 769 RXI3 MOV A.D ; GET SET TO TEST
0C3A 87 770 ANA A ; ALL RESULTS?
0C3B CAH50C 771 JZ RXI4 ; YES, SO FINISH UP
0C3E 3600 772 MVI M.00H ; NO, LOAD @ TIL DONE
0C40 2C 773 INR L ; BUMP POINTER
0C41 15 774 DCR D ; DEC COUNTER
0C42 C3390C 775 JMP RXI3 ; GO AGAIN
0C45 221020 776 RXI4 SHLD LAADR ; UPDATE LOAD POINTER
0C48 3A1529 777 LDA PROMPT ; GET MODE INDICATOR
0C4B FE24 778 CPI ' ' ; NORMAL MODE?
0C4D C9E50C 779 JZ RX16 ; YES, CLEAN UP BEFORE RETURN

780; POLL MODE SO CHECK CONTROL BYTE
782; IF CONTROL IS A POLL, SET UP SPECIAL TX COMMAND BUFFER
783; AND RETURN WITH POLL INDICATOR NOT 0
784;

0C50 E1 785 POP H ; GET PREVIOUS LOAD ADR POINTER
0C51 7E 786 MOV A.M ; GET IC BYTE FROM BUFFER

6-187 231311-001
APPLICATIONS

OC52 E61E 787 ANI 1EH ; LOOK AT GOOD FRAME BITS
OC54 C290C 788 JNZ RX15 ; IF NOT 0, INTERRUPT WASN'T FROM A GOOD FRAME
OC57 2C 789 INR L ; BYPASS R0 AND R1 IN BUFFER
OC58 2C 790 INR L
OC59 2C 791 INR L
OC5A 56 792 MOV D, M ; GET ADDR BYTE AND SAVE IT IN D
OC5B 2C 793 INR L
OC5C 7E 794 MOV A, M ; GET CTRL BYTE FROM BUFFER
OC5D FE33 795 CPI SNMP ; WAS IT SNRM-P?
OC5E CA6C0C 796 JZ TXRET ; YES, GO SET RESPONSE
OC5F C60C 797 CPI RR0P ; WAS IT RR(0)-P?
OC64 C290C 798 JNZ RX15 ; YES, GO SET RESPONSE, OTHERWISE RETURN
OC67 1E11 799 MVI E, RR0F ; RR(0)-F SO SET RESPONSE TO RR(0)-F
OC69 C3E0C 800 JMP TXRET ; GO FINISH LOADING SPECIAL BUFFER
OC6A 1E73 801 TI1: MVI E, NSA-F ; SNRM-P SO SET RESPONSE TO NSA-F
OC6B 21820 802 TXRET LXI H, CMD0F1 ; SPECIAL BUFFER ADDR
OC6C 36C8 806 MVI M, 008H ; LOAD TX FRAME COMMAND
OC6D 3680 808 INX H ; INC POINTER
OC6E 3680 809 MVI M, 00H ; LR=0
OC6F 3680 810 INX H ; INC POINTER
OC70 3680 811 MVI M, 00H ; LR=0
OC71 72 812 INX H ; INC POINTER
OC72 72 813 MOV M, D ; LOAD RCVD ADDR BYTE
OC73 72 814 INX H ; INC POINTER
OC74 73 815 MOV M, E ; LOAD RESPONSE CTRL BYTE
OC75 32160 816 MVI A, 00H ; SET POLL INDICATOR NOT 0
OC76 32620 817 STA POLIN ; LOAD POLL INDICATOR
OC77 C390C 818 JMP RX15 ; RETURN
OC78 3680 819
OC79 E1 820 RXI6: POP H ; CLEAN UP STACK IF NORMAL MODE
OC7A C390C 821 JMP RX15 ; RETURN
OC7B 21820 822
OC7C CD10B 823 RX15: CALL RXDMA ; RESET DMA CHANNEL
OC7D D1 824 POP D ; RESTORE REGISTERS
OC7E C1 825 POP B
OC7F E1 826 POP PSW
OC80 FB 827 POP H
OC81 C9 828 EI ; ENABLE INTERRUPTS
OC82 98 829 RET ; RETURN
OC83 3F 830
OC84 831
OC85 E1 832 MESSAGE TYPE - ASSUMES MESSAGE STARTS AT HL
OC86 C390C 833 RET
OC87 21820 834
OC88 C5 835 TXMSG: PUSH B ; SAVE BC
OC89 7E 836 TXMSG2: MOV A, M ; GET ASCII CHR
OC8A 23 837 INX H ; INC POINTER
OC8B FEFF 838 CPI 0FFH ; STOP?
OC8C C40C 839 JZ TXMSG1 ; YES, SET FOR EXIT
OC8D 4F 840 MOV C, A ; SET UP FOR DISPLAY
OC8E CDF805 841 CALL ECHO ; DISPLAY CHR
OC8F C390C 842 JMP TXMSG2 ; GET NEXT CHR
OC90 C1 843 TXMSG1: POP B ; RESTORE BC
OC91 C9 844 RET ; RETURN
OC92 C5 845
OC93 7E 846
OC94 23 847 ; SIGNON MESSAGE
OC95 98 848

6-188
APPLICATIONS

0CA3 00  849 SIGNON: DB CR. '8273 MONITOR VOL 1'; CR. 0FFH
0CA4 38323733
0CA5 284D4FE4
0CA6 49544F52
0CA7 20205631
0CA8 2E71
0CA9 0D
0CAF 00
0CB6 0D
0CB7 FF

850 ;
851 ;
852 ;
853 ; RECEIVER INTERRUPT MESSAGES
854 ;
855 ;

0CB8 0D  856 RXIMSG: DB CR. 'RX INT - %0FFH
0CB9 52582049
0CBA 4E542020
0CB1 20
0CB2 FF

857 ;
858 ; TRANSMITTER INTERRUPT MESSAGES
859 ;

0CB3 0D  860 TXIMSG: DB CR. 'TX INT - %0FFH
0CB4 54582049
0CB5 4E542020
0CB6 20
0CB7 FF

861 ;
862 ;
863 ; TRANSMITTER INTERRUPT ROUTINE
864 ;

0CEC 0E  865 TX1: PUSH H ; SAVE HL
0CFD F5  866 PUSH PSW ; SAVE PSW
0CD0 C5  867 PUSH B ; SAVE BC
0CD1 D5  868 PUSH D ; SAVE DE
0DD2 3E61  869 MVI A, DTDMA ; DISABLE TX DMA
0DD3 39B  870 OUT MODES7 ; 8257 MODE PORT
0DD4 1604  871 MVI D, 04H ; SET COUNTER
0DD5 8202  872 LHLD LDADR ; GET LOAD POINTER
0DD6 E5  873 PUSH H ; SAVE IT
0DD7 45  874 MOV B, L ; SAVE LSB IN B
0DD8 2A320  875 LHLD CNADR ; GET CONSOLE POINTER
0DE0 04  876 TX11: INR B ; INC POINTER
0DE1 78  877 MOV A, B ; GET SET TO TEST
0DE2 BD  878 CMP L ; LOAD=CONSOLE?
0DE3 C4E0A  879 JZ BUFFUL ; YES, BUFFER FULL
0DE4 15  880 DCR D ; NO, TEST NEXT LOCATION
0DE5 72B0C  881 JNZ TX11 ; TRY AGAIN
0DE6 E1  882 POP H ; RESTORE LOAD POINTER
0DE7 BB92  883 IN TXIR73 ; READ RESULT
0DE8 77  884 MOV M, A ; STORE IN BUFFER
0DE9 2C  885 INR L ; INR POINTER
0DEA 3600  886 MVI M, 00H ; EXTRA RESULT SPOTS 0
0DEB 2C  887 INR L
0DEC 3600  888 MVI M, 00H
0DEF 2C  889 INR L
0DF1 3600  890 MVI M, 00H
0DF2 2C  891 INR L

6-189
APPLICATIONS

0CF8 3680 892 MVI M_OFFSET
0CF8 221A20 894 SHLD LOADR ; UPDATE LOAD POINTER
0CF8 CD5500 899 CALL TXDMR ; RESET DMA CHANNEL
0001 D1 900 POP D ; RESTORE DE
0002 C1 901 POP B ; RESTORE BC
0003 F1 902 POP PSW ; RESTORE PSW
0004 E1 903 POP H ; RESTORE HL
0005 FB 904 EI ; ENABLE INTERRUPTS
0006 C9 905 RET ; RETURN
096 ;
097 ;
092 ;
093 ;
094 END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

ADIN A 0922 AFDMD A 09CE BUFDL A 0A4D CADRDR A 089A CHCTC A 08A1 CHLADR A 08A2 CHLTC A 08A3
CMD51 A 0987 CMD8F A 2020 CMD8BU A 2000 CMOUT A 0A4F CMDREC A 0857 CMODE A 0831 CMIADR A 2193
CNT83 A 099C CNT153 A 099D _CNT253 A 099E CNT51 A 0899 CNTLC A 0803 CNVBN A 0858 COBR A 080C
COMM A 08E5 COMM1 A 08EA COMM2 A 08AF COMM73 A 0899 CPBF A 0820 CR A 08D0 CRF A 08EB
DEN A 0900 DENODE A 2277 DISPY A 0893 DISPY1 A 0894 DISPY2 A 084D DMAP A 0862 DTMIA A 0861
ECHO A 08F8 ENDCHE A 0818 ENDMA A 0863 GDW A 086F GETCH A 081F GETCMD A 0870 GRCHD A 08C4
ILLEG A 0887 LDADR A 2010 LF A 0823 _LKBR1 A 0867 _LKBR2 A 086B LOOP0 A 0811 MDNTO A 083D
MDNTO A 0866 MODE51 A 08CE MODE53 A 085B MODE57 A 0898 MONTOR A 0808 MNOUT A 0867 NSAF A 0873
PAR1 A 0988 PAR2 A 080D PARIN A 08AD PARIN1 A 08E0 PARIN2 A 08DC PARIN3 A 08B8 PARIN3 A 0891
POLIN A 2196 PRMT A 2015 RPT A 08A2 RFT A 0897 RCHD A 0897 RCHD A 0871 RDWN A 08AF
ROY A 0802 RESBUF A 2200 RESLJ3 A 0891 ROCMD A 085D RPCMD A 0808 RRF A 0811 RAP A 0811
RSCMD A 0867 RST65 A 08CE RST75 A 2084 RBUF A 0820 RX51 A 0808 RXMT A 081A RXI A 080B
RX11 A 0C16 RX12 A 0C23 RX13 A 0C39 RX14 A 0C45 RX15 A 0C3A RX16 A 0C55 RXIMSG A 0C8B
RXINT A 0900 RXINT2 A 0893 RXINTA A 0802 RXTI A 0869 RXT2 A 08F7 RXT3 A 08D0 RXSRC A 0862
RXTC A 41FF SBCMD A 0895 SDWN A 0807 SIGNED A 08A3 SCLMD A 08A6 SNMAP A 0893 SOCMD A 0896
SPCMD A 09E2 SCMD A 08A9 SCCMD A 0890 STAT A 0800 STAT51 A 08A9 STAT73 A 0808 STAT75 A 08A9
STK5T A 2080 SW A 0843 TI A 08CA TBUFFL A 0824 TBUFL A 0807 TBUFL A 0810 TDIAN A 080E
TEST73 A 0992 TFCMD A 089C TFCMD1 A 0896 TFRET A 0836 TCLMD A 0899 TRUE A 0800 TRUE1 A 0800
TXBUF A 0800 TXD51 A 0808 TXMA A 0835 TXDMA A 0842 TXI A 08CE TX11 A 08B8 TXIMSG A 08CC
TXINT A 0904 TXINT2 A 0895 TXIRA A 0801 TXPOL A 084C TXRET A 08CE TXSRC A 0847 TXTC A 081F
TWMS A A 0C92 TWMSG A A 0C91 TWMSG2 A A 0C93 VALDG A 075E

ASSEMBLY COMPLETE, NO ERRORS
Asynchronous Communication with the 8274 Multiple-Protocol Serial Controller
INTRODUCTION

The 8274 Multiprotocol serial controller (MPSC) is a sophisticated dual-channel communications controller that interfaces microprocessor systems to high-speed serial data links (at speeds to 800K bits per second) using synchronous or asynchronous protocols. The 8274 interfaces easily to most common microprocessors (e.g., 8048, 8051, 8085, 8086, and 8088), to DMA controllers such as the 8237 and 8257, and to the 8089 I/O processor. Both MPSC communication channels are completely independent and can operate in a full-duplex communication mode (simultaneous data transmission and reception).

Communication Functions

The 8274 performs many communications-oriented functions, including:

- Converting data bytes from a microprocessor system into a serial bit stream for transmission over the data link to a receiving system.
- Receiving serial bit streams and reconverting the data into parallel data bytes that can easily be processed by the microprocessor system.
- Performing error checking during data transfers. Error checking functions include computing/transmitting error codes (such as parity bits or CRC bytes) and using these codes to check the validity of received data.
- Operating independently of the system processor in a manner designed to reduce the system overhead involved in data transfers.

System Interface

The MPSC system interface is extremely flexible, supporting the following data transfer modes:

1. Polled Mode. The system processor periodically reads (polls) an 8274 status register to determine when a character has been received, when a character is needed for transmission, and when transmission errors are detected.

2. Interrupt Mode. The MPSC interrupts the system processor when a character has been received, when a character is needed for transmission, and when transmission errors are detected.

3. DMA Mode. The MPSC automatically requests data transfers from system memory for both transmit and receive functions by means of two DMA request signals per serial channel. These DMA request signals may be directly interfaced to an 8237 or 8257 DMA controller or to an 8089 I/O processor.

4. WAIT Mode. The MPSC ready signal is used to synchronize processor data transfers by forcing the processor to enter wait states until the 8274 is ready for another data byte. This feature enables the 8274 to interface directly to an 8086 or 8088 processor by means of string I/O instructions for very high-speed data links.

Scope

This application note describes the use of the 8274 in asynchronous communication modes. Asynchronous communication is typically used to transfer data to/from video display terminals, modems, printers, and other low-to-medium-speed peripheral devices. Use of the 8274 in both interrupt-driven and polled system environments is described. Use of the DMA and WAIT modes are not described since these modes are employed mainly in synchronous communication systems where extremely high data rates are common. Programming examples are written in PL/M-80 (Appendix B and Appendix C). PL/M-86 is executed by the iAPX-86 and iAPX-88 processor families. In addition, PL/M-86 is very similar to PL/M-80 (executed by the MCS-80 and MCS-85 processor families). In addition, Appendix D describes a simple application example using an SDK-86 in an iAPX-86/88 environment.

SERIAL-ASYNCHRONOUS DATA LINKS

A serial asynchronous interface is a method of data transmission in which the receiving and transmitting systems need not be synchronized. Instead of transmitting clocking information with the data, locally generated clocks (16, 32 or 64 times as fast as the data transmission rate) are used by the transmitting and receiving systems. When a character of information is sent by the transmitting system, the character data is framed (preceded and followed) by special START and STOP bits. This framing information permits the receiving system to temporarily synchronize with the data transmission. (Refer to Figure 1 during the following discussion of asynchronous data transmission.)

Figure 1. Transmission of a 7-Bit ASCII Character with Even Parity
Normally the data link is in an idle or marking state, continuously transmitting a "mark" (binary 1). When a character is to be sent, the character data bits are immediately preceded by a "space" (binary 0 START bit). The mark-to-space transition informs the receiving system that a character of information will immediately follow the start bit. Figure 1 illustrates the transmission of a 7-bit ASCII character (upper case S) with even parity. Note that the character is transmitted immediately following the start bit. Data bits within the character are transmitted from least-significant to most-significant. The parity bit is transmitted immediately following the character data bits and the STOP framing bit (binary 1) signifies the end of the character.

Asynchronous interfaces are often used with human interface devices such as CRT/keyboard units where the time between data transmissions is extremely variable.

**Framing**

Character framing is accomplished by the START and STOP bits described previously. When the START bit transition (mark-to-space) is detected, the receiving system assumes that a character of data will follow. In order to test this assumption (and isolate noise pulses on the data link), the receiving system waits one-half bit time and samples the data link again. If the link has returned to the marking state, noise is assumed, and the receiver waits for another START bit transition.

When a valid START bit is detected, the receiver samples the data link for each bit of the following character. Character data bits and the parity bit (if required) are sampled at their nominal centers until all required characters are received. Immediately following the data bits, the receiver samples the data link for the STOP bit, indicating the end of the character. Most systems permit specification of 1, 1½, or 2 stop bits.

**Timing**

The transmitter and receiver in an asynchronous data link arrangement are clocked independently. Normally, each clock is generated locally and the clocks are not synchronized. In fact, each clock may be a slightly different frequency. (In practice, the frequency difference should not exceed a few percent. If the transmitter and receiver clock rates vary substantially, errors will occur because data bits may be incorrectly identified as START or STOP framing bits.) These clocks are designed to operate at 16, 32, or 64 times the communications data rate. These clock speeds allow the receiving device to correctly sample the incoming bit stream.

Serial-interface data rates are measured in bits/second. The term "baud" is used to specify the number of times per second that the transmitted signal level can change states. In general, the baud is not equal to the bit rate. Only when the transmitted signal has two states (electrical levels) is the baud rate equal to the bit rate. Most point-to-point serial data links use RS-232-C, RS-422, or RS-423 electrical interfaces. These specifications call for two electrical signal levels (the baud is equal to the bit rate). Modem interfaces, however, may often have differing bit and baud rates.

While there are generally no limitations on the data transmission rates used in an asynchronous data link, a limited set of rates has been standardized to promote equipment interconnection. These rates vary from 75 bits per second to 38,400 bits per second. Table 1 illustrates typical asynchronous data rates and the associated clock frequencies required for the transmitter and receiver circuits.
Table 1. Communication Data Rates and Associated Transmitter/Receiver Clock Rates

<table>
<thead>
<tr>
<th>Data Rate (bits/second)</th>
<th>Clock Rate (kHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X16</td>
</tr>
<tr>
<td>75</td>
<td>1.2</td>
</tr>
<tr>
<td>150</td>
<td>2.4</td>
</tr>
<tr>
<td>300</td>
<td>4.8</td>
</tr>
<tr>
<td>600</td>
<td>9.6</td>
</tr>
<tr>
<td>1200</td>
<td>19.2</td>
</tr>
<tr>
<td>2400</td>
<td>38.4</td>
</tr>
<tr>
<td>4800</td>
<td>76.8</td>
</tr>
<tr>
<td>9600</td>
<td>153.6</td>
</tr>
<tr>
<td>19200</td>
<td>307.2</td>
</tr>
<tr>
<td>38400</td>
<td>614.4</td>
</tr>
</tbody>
</table>

Parity

In order to detect transmission errors, a parity bit may be added to the character data as it is transferred over the data link. The parity bit is set or cleared to make the total number of "one" bits in the character even (even parity) or odd (odd parity). For example, the letter "A" is represented by the seven-bit ASCII code 1000001 (41H). The transmitted data code (with parity) for this character contains eight bits: 01000001 (41H) for even parity and 11000001 (OC1H) for odd parity. Note that a single bit error changes the parity of the received character and is therefore easily detected. The 8274 supports both odd and even parity checking as well as a parity disable mode to support binary data transfers.

Communication Modes

Serial data transmission between two devices can occur in one of three modes. In the simplex transmission mode, a data link can transmit data in one direction only. In the half-duplex mode, the data link can transmit data in both directions, but not simultaneously. In the full-duplex mode (the most common), the data link can transmit data in both directions simultaneously. The 8274 directly supports the full-duplex mode and will interface to simplex and half-duplex communication data links with appropriate software controls.

BREAK Condition

Asynchronous data links often include a special sequence known as a break condition. A break condition is initiated when the transmitting device forces the data link to a spacing state (binary 0) for an extended length of time (typically 150 milliseconds). Many terminals contain keys to initiate a break sequence. Under software control, the 8274 can initiate a break sequence when transmitting data and detect a break sequence when receiving data.

MPSC SYSTEM INTERFACE

Hardware Environment

The 8274 MPSC interfaces to the system processor over an 8-bit data bus. Each serial I/O channel responds to two I/O or memory addresses as shown in Table 2. In addition, the MPSC supports vectored and daisy-chained interrupts.

The 8274 may be configured for memory-mapped or I/O-mapped operation.

Table 2. 8274 Addressing

<table>
<thead>
<tr>
<th>CS</th>
<th>A1</th>
<th>A0</th>
<th>Read Operation</th>
<th>Write Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Ch. A Data Read</td>
<td>Ch. A Data Write</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Ch. A Status Read</td>
<td>Ch. A Command/Parameter</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Ch. B Data Read</td>
<td>Ch. B Data Write</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Ch. B Status Read</td>
<td>Ch. B Command/Parameter</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>High Impedance</td>
<td>High Impedance</td>
</tr>
</tbody>
</table>

The 8274-processor hardware interface can be configured in a flexible manner, depending on the operating mode selected—polled, interrupt-driven, DMA, or WAIT. Figure 3 illustrates typical MPSC configurations for use with an 8088 microprocessor in the polled and interrupt-driven modes.

All serial-to-parallel conversion, parallel-to-serial conversion, and parity checking required during asynchronous serial I/O operation is automatically performed by the MPSC.

Operational Interface

Command, parameter, and status information is stored in 22 registers within the MPSC (8 writable registers and 3 readable registers for each channel). These registers are all accessed by means of the command/status ports for each channel. An internal pointer register selects which of the command or status registers will be written or read during a command/status access of an MPSC channel. Figure 4 diagrams the command/status register architecture for each serial channel. In the following discussion, the writable registers will be referred to as WR0 through WR7 and the readable registers will be referred to as RR0 through RR2.
a) Polled Configuration

The least-significant three bits of WR0 are automatically loaded into the pointer register every time WR0 is written. After reset, WR0 is set to zero so that the first write to a command register causes the data to be loaded into WR0 (thereby setting the pointer register). After WR0 is written, the following read or write accesses the register selected by the pointer. The pointer is reset after the read or write operation is completed. In this manner, reading or writing an arbitrary MPSC channel register requires two I/O accesses. The first access is always a write command. This write command is used to set the pointer register. The second access is either a read or write command; the pointer register (previously set) will ensure that the correct internal register is read or written. After this second access, the pointer register is automatically reset. Note that writing WR0 and reading RR0 does not require presetting of the pointer register.

b) Daisy-chained Interrupt Configuration

During initialization and normal MPSC operation, various registers are read and/or written by the system processor. These actions are discussed in detail in the following paragraphs. Note that WR6 and WR7 are not used in the asynchronous communication modes.

RESET

When the 8274 RESET line is activated, both MPSC channels enter the idle state. The serial output lines are forced to the marking state (high) and the modem interface signals (RTS, DTR) are forced high. In addition, the pointer register is set to zero.
External/Status Latches

The MPSC continuously monitors the state of four external/status conditions:

1. CTS—clear-to-send input pin.
2. CD—carrier-detect input pin.
3. SYNDET—sync-detect input pin. This pin may be used as a general-purpose input in the asynchronous communication mode.
4. BREAK—a break condition (series of space bits on the receiver input pin).

A change of state in any of these monitored conditions will cause the associated status bit in RR0 (Appendix A) to be latched (and optionally cause an interrupt).

Error Reporting

Three error conditions may be encountered during data reception in the asynchronous mode:
1. Parity. If parity bits are computed and transmitted with each character and the MPSC is set to check parity (bit 0 in WR4 is set), a parity error will occur whenever the number of “1” bits within the character (including the parity bit) does not match the odd/even setting of the parity check flag (bit 1 in WR4).

2. Framing. A framing error will occur if a stop bit is not detected immediately following the parity bit (if parity checking is enabled) or immediately following the most-significant data bit (if parity checking is not enabled).

3. Overrun. If an input character has been assembled but the receiver buffers are full (because the previously received characters have not been read by the system processor), an overrun error will occur. When an overrun error occurs, the input character that has just been received will overwrite the immediately preceding character.

Transmitter/Receiver Initialization

In order to operate in the asynchronous mode, each MPSC channel must be initialized with the following information:

1. Clock Rate. This parameter is specified by bits 6 and 7 of WR4. The clock rate may be set to 16, 32, or 64 times the data-link bit rate. (See Appendix A for WR4 details.)

2. Number of Stop Bits. This parameter is specified by bits 2 and 3 of WR4. The number of stop bits may be set to 1, 1½, or 2. (See Appendix A for WR4 details.)

3. Parity Selection. Parity may be set for odd, even, or no parity by bits 0 and 1 of WR4. (See Appendix A for WR4 details.)

4. Receiver Character Length. This parameter sets the length of received characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 6 and 7 of WR3. (See Appendix A for WR3 details.)

5. Receiver Enable. The serial-channel receiver operation may be enabled or disabled by setting or clearing bit 0 of WR3. (See Appendix A for WR3 details.)

6. Transmitter Character Length. This parameter sets the length of transmitted characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 5 and 6 of WR5. (See Appendix A for WR5 details.) Characters of less than 5 bits in length may be transmitted by setting the transmitted length to five bits (set bits 5 and 6 of WR5 to 1).

   The MPSC then determines the actual number of bits to be transmitted from the character data byte. The bits to be transmitted must be right justified in the data byte, the next three bits must be set to 0 and all remaining bits must be set to 1. The following table illustrates the data formats for transmission of 1 to 5 bits of data:

<table>
<thead>
<tr>
<th>Character Length</th>
<th>Bits Transmitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7 D6 D5 D4 D3 D2 D1 D0</td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 0 0 0 c</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1 0 0 0 c c</td>
<td>2</td>
</tr>
<tr>
<td>1 1 0 0 0 c c c</td>
<td>3</td>
</tr>
<tr>
<td>1 0 0 0 c c c c</td>
<td>4</td>
</tr>
<tr>
<td>0 0 0 c c c c c</td>
<td>5</td>
</tr>
</tbody>
</table>

7. Transmitter Enable. The serial-channel transmitter operation may be enabled or disabled by setting or clearing bit 3 of WR5. (See Appendix A for WR5 details.)

For data transmissions via a modem or RS-232-C interface, the following information must also be specified:

1. Request-to-Send/Data-Terminal-Ready. Must be set to indicate status of data terminal equipment. Request-to-send is controlled by bit 1 of WRS and data terminal ready is controlled by bit 7. (See Appendix A for WRS details.)

2. Auto Enable. May be set to allow the MPSC to automatically enable the channel transmitter when the clear-to-send signal is active and to automatically enable the receiver when the carrier-detect signal is active. Auto Enable is controlled by bit 5 of WR3. (See Appendix A for WR3 details.)

During initialization, it is desirable to guarantee that the external/status latches reflect the latest interface information. Since up to two state changes are internally stored by the MPSC, at least two Reset External/Status Interrupt commands must be issued. This procedure is most easily accomplished by simply issuing this reset command whenever the pointer register is set during initialization.

An MPSC initialization procedure (MPSCRX$INIT) for asynchronous communication is listed in Appendix B. Figure 5 illustrates typical MPSC initialization parameters for use with this procedure.

**Figure 5. Sample 8274 Initialization Procedure for Polled Operation**

<table>
<thead>
<tr>
<th>Call</th>
<th>MPSCRX$INIT(41, 1, 1, 0, 1, 3, 1, 1, 3, 1, 1, 0, 1).</th>
</tr>
</thead>
<tbody>
<tr>
<td>X16 clock rate</td>
<td>Enable transmitter and receiver</td>
</tr>
<tr>
<td>1 stop bit</td>
<td>Auto enable set</td>
</tr>
<tr>
<td>Odd parity</td>
<td>DTR and RTS set</td>
</tr>
<tr>
<td>8-bit characters (Tx and Rx)</td>
<td>Break transmission disabled</td>
</tr>
</tbody>
</table>

6-197 210311-001
Polled Operation

In the polled mode, the processor must monitor the MPSC status by testing the appropriate bits in the read register. Data available, status, and error conditions are represented in RR0 and RR1 for channels A and B. An example of MPSC-pollled transmitter/receiver routines are given in Appendix B. The following routines are detailed:

1. MPSC$POLL$RCV$CHARACTER—This procedure receives a character from the serial data link. The routine waits until the character-available flag in RR0 has been set. When this flag indicates that a character is available, RR1 is checked for errors (overrun, parity, or framing). If an error is detected, the character in the MPSC receive buffer must be read and discarded and the error routine (RECEIVESERROR) is called. If no receive errors have been detected, the character is input from the 8274 data port and returned to the calling program.

MPSC$POLL$RCV$CHARACTER requires three parameters—the address of the 8274 channel data port (data$port), the address of the 8274 channel command port (cmd$port), and the address of a byte variable in which to store the received character (character$ptr).

2. MPSC$POLL$TRAN$CHARACTER—This procedure transmits a character to the serial data link. The routine waits until the transmitter-buffer-empty flag has been set in RR0 before writing the character to the 8274.

MPSC$POLL$TRAN$CHARACTER requires three parameters—the address of the 8274 channel data port (data$port), the address of the 8274 channel command port (cmd$port), and the character of data that is to be transmitted (character).

3. RECEIVESERROR—This procedure processes receiver errors. First, an Error Reset command is written to the affected channel. All additional error processing is dependent on the specific application. For example, the receiving device may immediately request retransmission of the character or wait until a message has been completed.

RECEIVESERROR requires two parameters—the address of the affected 8274 command port (cmd$port) and the error status (status) from 8274 register RR1.

Interrupt-driven Operation

In an interrupt-driven environment, all receiver operations are reported to the system processor by means of interrupts. Once a character has been received and assembled, the MPSC interrupts the system processor. The system processor must then read the character from the MPSC data buffer and clear the current interrupt. During transmission, the system processor starts serial I/O by writing the first character of a message to the MPSC. The MPSC interrupts the system processor whenever the next character is required (i.e., when the transmitter buffer is empty) and the processor responds by writing the next character of the message to the MPSC data port for the appropriate channel.

By using interrupt-driven I/O, the MPSC proceeds independently of the system processor, signalling the processor only when characters are required for transmission, when characters are received from the data link, or when errors occur. In this manner, the system processor may continue execution of other tasks while serial I/O is performed concurrently.

Interrupt Configurations

The 8274 is designed to interface to 8085- and 8086-type processors in much the same manner as the 8259A is designed. When operating in the 8085 mode, the 8274 causes a "call" to a prespecified, interrupt-service routine location. In the 8086 mode, the 8274 presents the processor with a one-byte interrupt-type number. This interrupt-type number is used to "vector" through the 8086 interrupt service table. In either case, the interrupt service address or interrupt-type number is specified during MPSC initialization.

To shorten interrupt latency, the 8274 can be programmed to modify the prespecified interrupt vector so that no software overhead is required to determine the cause of an interrupt. When this "status affects vector" mode is enabled, the following eight interrupts are differentiated automatically by the 8274 hardware:

1. Channel B Transmitter Buffer Empty.
2. Channel B External/Status Transition.
3. Channel B Character Available.
5. Channel A Transmitter Buffer Empty.
6. Channel A External/Status Transition.
7. Channel A Character Available.
8. Channel A Receive Error.

Interrupt Sources/Priorities

The 8274 has three interrupt sources for each channel:

1. Receiver (RxA, RxB). An interrupt is initiated when a character is available in the receiver buffer or when a receiver error (parity, framing, or overrun) is detected.
2. Transmitter (TxA, TxB). An interrupt is initiated when the transmitter buffer is empty and the 8274 is ready to accept another character for transmission.

3. External/Status (ExTA, ExTB). An interrupt is initiated when one of the external/status conditions (CD, CTS, SYNDET, BREAK) changes state.

The 8274 supports two interrupt priority orderings (selectable during MPSC initialization) as detailed in Appendix A, WR2, CH-A.

**Interrupt Initialization**

In addition to the initialization parameters required for polled operation, the following parameters must be supplied to the 8274 to specify interrupt operation:

1. Transmit Interrupt Enable. Transmitter-buffer-empty interrupts are separately enabled by bit 1 of WR1. (See Appendix A for WR1 details.)

2. Receive Interrupt Enable. Receiver interrupts are separately enabled in one of three modes: a) interrupt on first received character only and on receive errors (used for message-oriented transmission systems), b) interrupt on all received characters and on receive errors, but do not interrupt on parity errors, and c) interrupt on all received characters and on receive errors (including parity errors). The ability to separately disable parity interrupts can be extremely useful when transmitting messages. Since the parity error bit in RR1 is latched, it will not be reset until an error reset operation is performed. Therefore, the parity error bit will be set if any parity errors were detected in a multicharacter message. If this mode is used, the serial I/O software must poll the parity error bit at the completion of a message and issue an error reset if appropriate. The receiver interrupt mode is controlled by bits 3 and 4 of WR1. (See Appendix A for WR1 details.)

3. External/Status Interrupts. External/Status interrupts can be separately enabled by bit 0 of WR1. (See Appendix A for WR1 details.)

4. Interrupt Vector. An eight-bit interrupt-service routine location (8085) or interrupt type (8086) is specified through WR2 of channel B. (See Appendix A for WR2 details.) Table 3 lists interrupt vector addresses generated by the 8274 in the "status affects vector" mode.

5. "Status Affects Vector" Mode. The 8274 will automatically modify the interrupt vector if bit 3 of WR1 is set. (See Appendix A for WR1 details.)

6. System Configuration. Specifies the 8274 data transfer mode. Three configuration modes are available: a) interrupt-driven operation for both channels, b) DMA operation for both channels, and c) DMA operation for channel A, interrupt-driven operation for channel B. The system configuration is specified by means of bits 0 and 1 of WR2 (channel A). (See Appendix A for WR2 details.)

7. Interrupt Priorities. The 8274 permits software specification of receive/transmit priorities by means of bit 2 of WR2 (channel A). (See Appendix A of WR2 details.)

8. Interrupt Mode. Specifies whether the MPSC is to operate in a non-vectored mode (for use with an external interrupt controller), in an 8086-vectored mode, or in an 8085-vectored mode. This parameter is specified through bits 3 and 4 of WR2 (channel A). (See Appendix A for WR2 details.)

<table>
<thead>
<tr>
<th>Table 3. MPSC-generated Interrupt Vectors in &quot;Status Affects Vector&quot; Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>V7 V6 V5 V4 V3 V2 V1 V0</strong></td>
</tr>
<tr>
<td><strong>8086 Interrupt Type</strong></td>
</tr>
<tr>
<td>V7 V6 V5 V4 V3 0 0 0</td>
</tr>
<tr>
<td>V7 V6 V5 V4 V3 0 0 1</td>
</tr>
<tr>
<td>V7 V6 V5 V4 V3 0 1 0</td>
</tr>
<tr>
<td>V7 V6 V5 V4 V3 0 1 1</td>
</tr>
<tr>
<td>V7 V6 V5 V4 V3 1 0 0</td>
</tr>
<tr>
<td>V7 V6 V5 V4 V3 1 0 1</td>
</tr>
<tr>
<td>V7 V6 V5 V4 V3 1 1 0</td>
</tr>
<tr>
<td>V7 V6 V5 V4 V3 1 1 1</td>
</tr>
</tbody>
</table>

An MPSC interrupt initialization procedure (MPSC$INT$INIT) is listed in Appendix C.

**Interrupt Service Routines**

Appendix C lists four interrupt service procedures, a buffer transmission procedure, and a buffer reception procedure that illustrate the use of the 8274 in interrupt-driven environments. Use of these procedures assumes that the 8086/8088 interrupt vector is set to 20H and that channel B is used with the "status affects vector" mode enabled.

1. TRANSMIT$BUFFER$—This procedure begins serial transmission of a data buffer. Two parameters are required—a pointer to the buffer (buf$ptr) and the length of the buffer (buf$length). The procedure first sets the global buffer pointer, buffer length, and...
initial index for the transmitter-interrupt service routine and initiates transmission by writing the first character of the buffer to the 8274. The procedure then enters a wait loop until the I/O completion status is set by the transmit-interrupt service routine (MPSC$TRANSMIT$CHARACTERS$INT).

2. RECEIVE$BUFFER—This procedure inputs a line (terminated by a line feed) from a serial I/O port. Two parameters are required—a pointer to the input buffer (buf$ptr) and a pointer to the buffer length variable (buf$length$ptr). The buffer length will be set by this procedure when the complete line has been input. The procedure first sets the global buffer pointer and initial index for the receiver interrupt service routine. RECEIVE$BUFFER then enters a wait loop until the I/O completion status is set by the receive interrupt routine (MPSC$RECEIVE$­CHARACTERS$INT).

3. MPSC$RECEIVE$CHARACTERS$INT—This procedure is executed when the MPSC Tx-buffer-empty interrupt is acknowledged. If the current transmit buffer index is less than the buffer length, the next character in the buffer is written to the MPSC data port and the buffer pointer is updated. Otherwise, the transmission complete status is posted.

4. MPSC$RECEIVE$CHARACTERS$INT—This procedure is executed when a character has been assembled by the MPSC and the MPSC has issued a character-available interrupt. If no input buffer has been set up by RECEIVE$BUFFER, the character is ignored. If a buffer has been set up, but it is full, a receive overrun error is posted. Otherwise, the received character is read from the MPSC data port and the buffer index is updated. Finally, if the received character is a line feed, the reception complete status is posted.

5. RECEIVE$ERRORS$INT—This procedure is executed when a receive error is detected. First, the error conditions are read from RR1 and the character currently in the MPSC receive buffer is read and discarded. Next, an Error Reset command is written to the affected channel. All additional error processing is application dependent.

6. EXTERNAL$STATUS$CHANGE$INT—This procedure is executed when an external status condition change is detected. The status conditions are read from RR0 and a Reset External/Status Interrupt command is issued. Further error processing is application dependent.

DATA LINK INTERFACE

Serial Data Interface

Each serial I/O channel within the 8274 MPSC interfaces to two data link lines—one line for transmitting data and one for receiving data. During transmission, characters are converted from parallel data format (as supplied by the system processor or DMA device) into a serial bit stream (with START and STOP bits) and clocked out on the TxD pin. During reception, a serial bit stream is input on the RxD pin, framing bits are stripped out of the data stream, and the resulting character is converted to parallel data format and passed to the system processor or DMA device.

Data Clocking

As discussed previously, the frequency of data transmission/reception on the data link is controlled by the MPSC clock in conjunction with the programmed clock divider (in register WR4). The 8274 is designed to permit all four serial interface lines (TxD and RxD for each channel) to operate at different data rates. Four clock input pins (TxC and RxC for each channel) are available for this function. Note that the clock rate divider specified in WR4 is used for both RxC and TxC on the appropriate channel; clock rate dividers for each channel are independent.

Modem Control

The following four modem interface signals may be connected to the 8274:

1. Data Terminal Ready (DTR). This interface signal (output by the 8274) is software controlled through bit 7 of WR5. When active, DTR indicates that the data terminal/computer equipment is active and ready to interact with the data communications channel. In addition, this signal prepares the modem for connection to the communication channel and maintains connections previously established (e.g., manual call origination).

2. Request To Send (RTS). This interface signal (output by the 8274) is software controlled through bit 1 of WR5. When active, RTS indicates that the data terminal/computer equipment is ready to transmit data.

3. Clear To Send (CTS). This interface signal (input to the 8274) is supplied by the modem in response to an active RTS signal. CTS indicates that the data terminal/computer equipment is permitted to transmit...
data. The state of CTS is available to the programmer as bit 5 of RR0. In addition, if the auto enable control is set (bit 5 of WR3), the 8274 will not transmit data bytes until RTS has been activated. If CTS becomes inactive during transmission of a character, the current character transmission is completed before the transmitter is disabled.

4. Carrier Detect (CD). This interface signal (input to the 8274) is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RxD line. The state of CD is available to the programmer as bit 3 of RR0. In addition, if the auto enable control is set (bit 5 of WR3), the 8274 will not enable the serial receiver until CD has been activated. If the CD signal becomes inactive during reception of a character, the receiver is disabled, and the partially received character is lost.

In addition to the above modem interface signals, the 8274 SYNDET input pin for channel A may be used as a general-purpose input in the asynchronous communication mode. The status of this signal is available to the programmer as bit 4 of status register RR0.
APPENDIX A
COMMAND/STATUS DETAILS FOR ASYNCHRONOUS COMMUNICATION

Write Register 0 (WR0):

Command/Status Register Pointer bits determine which write-register the next byte is to be written into, or which read-register the next byte is to be read from. After reset, the first byte written into either channel goes into WR0. Following a read or write to any register (except WR0) the pointer will point to WR0.

Command bits determine which of the basic seven commands are to be performed.

Command 0 Null—has no effect.

Command 1 Not used in asynchronous modes.

Command 2 Reset External/Status Interrupts—resets the latched status bits of RR0 and reenables them, allowing interrupts to occur again.

Command 3 Channel Reset—resets the Latched Status bits of RR0, the interrupt prioritization logic and all control registers for the channel. Four extra system clock cycles should be allowed for MPSC reset time before any additional commands or controls are written into the channel.

Command 4 Enable Interrupt on Next Receive Character—if the Interrupt-on-First-Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the MPSC for the next message.

Command 5 Reset Transmitter Interrupt Pending—if the Transmit Interrupt mode is selected, the MPSC automatically interrupts data when the transmit buffer becomes empty. When there are no more characters to be sent, issuing this command prevents further transmitter interrupts until the next character has been completely sent.

Command 6 Error Reset—error latches, Parity and Overrun errors in RR1 are reset.

Command 7 End of Interrupt—resets the interrupt-in-service latch of the highest-priority internal device under service.

D0 External/Status Interrupt Enable—allows interrupt to occur as the result of transitions on the CD, CTS or SYNDIT inputs. Also allows interrupts as the result of a Break/Abort detection and termination, or at the beginning of CRC, or sync character transmission when the Transmit Underrun/EOM latch becomes set.

D1 Transmitter Interrupt/DMA Enable—allows the MPSC to interrupt or request a DMA transfer when the transmitter buffer becomes empty.

D2 Status Affects Vector—(WR1, D2 active in channel B only.) If this bit is not set,
Write Register 1 (WR1):

<table>
<thead>
<tr>
<th>MSB</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**EXT INTERRUPT ENABLE**

**TX INTERRUPT/DMA ENABLE**

<table>
<thead>
<tr>
<th>STATUS AFFECTS VECTOR/CH B ONLY</th>
<th>0 - FIXED VECTOR</th>
</tr>
</thead>
</table>

0 0 RxINT/DMA DISABLE

0 1 RxINT ON FIRST CHAR OR SPECIAL CONDITION

1 0 INT ON ALL Rx CHAR (PARITY AFFECTS VECTOR) OR SPECIAL CONDITION

1 1 INT ON ALL Rx CHAR (PARITY DOES NOT AFFECT VECTOR) OR SPECIAL CONDITION

1 = VARIABLE VECTOR

0 = FIXED VECTOR

0 0 RxA > RxB > TxA > TxB

0 1 RxA > RxB > TxA > EXT A > EXT B

1 0 8085 MODE 1

1 1 8086 MODE 2

1 1 ILLEGAL

1 = PRIORITY RxA > RxB > TxA > TxB

0 = PRIORITY RxA > RxB > TxA > EXT A > EXT B

1 = PRIORITY RxA > RxB > TxA > EXT A > EXT B

0 0 BOTH INTERRUPT

0 1 A DMA B INT

1 0 BOTH DMA

1 1 ILLEGAL

D6 Must be Zero.

D7 Wait Enable—enables the wait function.

Write Register 2 (WR2): Channel A

<table>
<thead>
<tr>
<th>MSB</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**WAIT ENABLE 1 = ENABLE, 0 = DISABLE**

**WAIT ON Rx, 0 - WAIT ON Tx**

then the fixed vector, programmed in WR2, is returned from an interrupt acknowledge sequence. If the bit is set, then the vector returned from an interrupt acknowledge is variable as shown in the Interrupt Vector Table.

D4,D3 Receive Interrupt Mode.

0 0 Receive Interrupts/DMA Disabled.

0 1 Receive Interrupt on First Character Only or Special Condition.

1 0 Interrupt on All Receive Characters of Special Condition (Parity Error is a Special Receive Condition).

1 1 Interrupt on All Receive Characters or Special Condition (Parity Error is not a Special Receive Condition).

D5 Wait on Receive/Transmit—when the following conditions are met, the RDY pin is activated, otherwise it is held in the High-Z state. (Conditions: Interrupt Enabled Mode, Wait Enabled, CS=0, A0=0/1, and A1=0). The RDY pin is pulled low when the transmitter buffer is full or the receiver buffer is empty and it is driven High when the transmitter buffer is empty or the receiver buffer is full. The RDYa and RDYB may be wired or connected since only one signal is active at any one time while the other is in the High Z state.

D1,D0 System Configuration—These specify the data transfer from MPSC channels to the CPU, either interrupt or DMA based.
0 1  Channel A uses DMA, Channel Buses interrupt.
1 0  Channel A and Channel B both use DMA.
1 1  Illegal Code.

D2  Priority—this bit specifies the relative priorities of the internal MPSC interrupt/DMA sources.
0  (Highest) RxA, TxA, RxA, RxB, TxBExTA, ExTB (Lowest).
1  (Highest) RxA, RxB, TxA, TxB, ExTA, ExTB (Lowest).

D5,D4,D3  Interrupt Code—specifies the behavior of the MPSC when it receives an interrupt acknowledge sequence from the CPU. (See Interrupt Vector Mode Table).
0 X X  Non-vectored interrupts—intended for use with an external interrupt controller such as the 8259A.
1 0 0  8085 Vector Mode 1—intended for use as the primary MPSC in a daisy-chained priority structure.
1 0 1  8085 Vector Mode 2—intended for use as any secondary MPSC in a daisy-chained priority structure.
1 1 0  8086/88 Vector Mode—intended for use as either a primary or secondary in a daisy-chained priority structure.

D6  Must be Zero.

D7  
0  Pin 10 = RTSB.
1  Pin 10 = SYNDETB.

Write Register 2 (WR2): Channel B

D7-D0  Interrupt vector—this register contains the value of the interrupt vector placed on the data bus during acknowledge sequences.

Write Register 3 (WR3):

D0  Receiver Enable—A one enables the receiver to begin. This bit should be set only after the receiver has been initialized.
D5  Auto Enables—A one written to this bit causes CD to be an automatic enable signal for the receiver and CTC to be an automatic enable signal for the transmitter. A zero written to this bit limits the effect of CD and CTS signals to setting/resetting their corresponding bits in the status register (RR0).

Write Register 2 (WR2): Channel B

D7,D6  Receiver Character length.
0 0  Receive 5 Data bits/character.
0 1  Receive 7 Data bits/character.
1 0  Receive 6 Data bits/character.
1 1  Receive 8 Data bits/character.
Write Register 4 (WR4):

Parity—a one in this bit causes a parity bit to be added to the programmed number of data bits per character for both the transmitted and received character. If the MPSC is programmed to receive 8 bits per character, the parity bit is not transferred to the microprocessor. With other receiver character lengths, the parity bit is transferred to the microprocessor.

Even/Odd Parity—if parity is enabled, a one in this bit causes the MPSC to transmit and expect even parity, and zero causes it to send and expect odd parity.

Stop Bits.

Clock mode—selects the clock/data rate multiplier for both the receiver and the transmitter. If the 1x mode is selected, bit synchronization must be done externally.

Write Register 5 (WR5):

Request to Send—a one in this bit forces the RTS pin active (low) and zero in this bit forces the RTS pin inactive (high).

Transmitter Enable—a zero in this bit forces a marking state on the transmitter output. If this bit is set to zero during data or sync character transmission, the marking state is entered after the character has been sent. If this bit is set to zero during transmission of a CRC character, sync or flag bits are substituted for the remainder of the CRC bits.

Send Break—a one in this bit forces the transmit data low. A zero in this bit allows normal transmitter operation.

Transmit Character length.
11 Transmit 8 bits/character.

Bits to be sent must be right justified, least-significant bit first, e.g.:

D7 D6 D5 D4 D3 D2 D1 D0
0 0 B5 B4 B3 B2 B1 B0

Read Register 0 (RR0):

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX CHIR AVAILABLE</td>
<td>INT PENDING (CHA ONLY)</td>
<td>CARRIER DETECT</td>
<td>SYNDET</td>
<td>CTS</td>
<td>NOT USED IN ASYNCHRONOUS MDES</td>
<td>BREAK</td>
<td></td>
</tr>
</tbody>
</table>

D0 Receive Character Available—this bit is set when the receive FIFO contains data and is reset when the FIFO is empty.

D1 Interrupt Pending—This Interrupt-Pending bit is reset when an EOI command is issued and there is no other interrupt request pending at that time. In vector mode, this bit is set at the falling edge of the second INTA in an INTA cycle for an internal interrupt request. In non-vector mode, this bit is set at the falling edge of RD input after pointer 2 is specified. This bit is always zero in Channel B.

D2 Transmit Buffer Empty—This bit is set whenever the transmit buffer is empty except when CRC characters are being sent in a synchronous mode. This bit is reset when the transmit buffer is loaded. This bit is set after an MPSC reset.

D3 Carrier Detect—This bit contains the state of the CD pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the CD pin causes the CD bit to be latched and causes an External/Status interrupt. This bit indicates current state of the CD pin immediately following a Reset External/Status Interrupt command.

D4 SYNDET—In asynchronous modes, the operation of this bit is similar to the CD status bit, except that it shows the state of the SYNDET input. Any High-to-Low transition on the SYNDET pin sets this bit, and causes an External/Status interrupt (if enabled). The Reset External/Status interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the SYNDET pin at time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the SYNDET input.

D5 Clear to Send—this bit contains the inverted state of the CTS pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the CTS pin causes the CTS bit to be latched and causes an External/Status interrupt. This bit indicates the inverse of the current state of the CTS pin immediately following a Reset External/Status Interrupt command.

D7 Break—in the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, Command 2) to the break detection logic so the Break sequence termination can be recognized.

The Break bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single, extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.
Read Register 1 (RR1)

D0  All sent—this bit is set when all characters have been sent, in asynchronous modes. It is reset when characters are in the transmitter, in asynchronous modes. In synchronous modes, this bit is always set.

D4  Parity Error—if parity is enabled, this bit is set for received characters whose parity does not match the programmed sense (Even/Odd). This bit is latched. Once an error occurs, it remains set until the Error Reset command is written.

D5  Receive Overrun Error—this bit indicates that the receive FIFO has been overloaded by the receiver. The last character in the FIFO is overwritten and flagged with this error. Once the overwritten character is read, this error condition is latched until reset by the Error Reset command. If the MPSC is in the “status affects vector” mode, the overrun causes a special Receive Error Vector.

D6  Framing Error—in async modes, a one in this bit indicates a receive framing error. It can be reset by issuing an Error Reset command.

Read Register 2 (RR2):

D7–D0  Interrupt vector—contains the interrupt vector programmed into WR2. If the “status affects vector” mode is selected, it contains the modified vector. (See WR2.) RR2 contains the modified vector for the highest priority interrupt pending. If no interrupts are pending, the variable bits in the vector are set to one.
APPENDIX B
MPSC-POLLED TRANSMIT/RECEIVE CHARACTER Routines

MPSC$RX$INIT: procedure (cmd$port, clock$rate, stop$bits, parity$type, parity$enable, rx$char$length, rx$enable, auto$enable, tx$char$length, tx$enable, dtr, brk, rts);

declare cmd$port byte,
    clock$rate byte,
    stop$bits byte,
    parity$type byte,
    parity$enable byte,
    rx$char$length byte,
    rx$enable byte,
    auto$enable byte,
    tx$char$length byte,
    tx$enable byte,
    dtr byte,
    brk byte,
    rts byte;

output(cmd$port)=30H;             /* channel reset */
output(cmd$port)=14H;             /* point to WR4 */
/* set clock rate, stop bits, and parity information */
output(cmd$port)=shl(clock$rate,6) or shl(stop$bits,2) or shl(parity$type,1)
    or parity$enable;
output(cmd$port)=13H;             /* point to WR3 */
/* set up receiver parameters */
output(cmd$port)=shl(rx$char$length,6) or rx$enable or shl(auto$enable,5);
output(cmd$port)=15H;             /* point to WR5 */
/* set up transmitter parameters */
output(cmd$port)=shl(tx$char$length,5) or shl(tx$enable,3) or shl(dtr,7)
    or shl(brk,4) or shl(rts,1);
end MPSC$RX$INIT;
MPSC$POLL$RCV$CHARACTER: procedure(data$port, cmd$port, character$ptr) byte;

declare data$port byte,
    cmd$port byte,
    character$ptr pointer,
    character based character$ptr byte,
    status byte;

declare char$avail literally '1',
    rcv$error literally '70H';

/* wait for input character ready */
while (input(cmd$port) and char$avail) <> 0 do; end;

/* check for errors in received character */
output(cmd$port)=1; /* point to RR1 */
if (status:=input(cmd$port) and rcv$error)
    then do;
        character=input(data$port); /* read character to clear MPSC */
        call RECEIVE$ERROR(cmd$port,status); /* clear receiver errors */
        return 0; /* error return - no character avail */
    end;
else do;
    character=input(data$port); /* good return - character avail */
    return OFFH;
end;
end MPSC$POLL$RCV$CHARACTER;

MPSC$POLL$TRAN$CHARACTER: procedure(data$port, cmd$port, character);

declare data$port byte,
    cmd$port byte,
    character byte;

declare tx$buffer$empty literally '4';

/* wait for transmitter buffer empty */
while not (input(cmd$port) and tx$buffer$empty) do; end;

/* output character */
output(data$port)=character;
end MPSC$POLL$TRAN$CHARACTER;

RECEIVE$ERROR: procedure(cmd$port, status);

declare cmd$port byte,
    status byte;

output(cmd$port)=30H; /* error reset */

/* *** other application dependent error processing should be placed here *** */
end RECEIVE$ERROR;
TRANSMIT$BUFFER: procedure (buf$ptr,buf$length)

declare
    buf$ptr    pointer,
    buf$length byte;

    /* set up transmit buffer pointer and buffer length in global variables for interrupt service */
    tx$buffer$ptr=buf$ptr;
    transmit$length=buf$length;

    transmit$status=not$complete;
    output(data$port)=transmit$buffer(0);
    transmit$index=1;
    /* setup status for not complete */
    /* first character transmitted */

    /* wait until transmission complete or error detected */
    while transmit$status = not$complete do; end;
    if transmit$status <> complete
        then return false;
        else return true;
    end transmit$BUFFER;

RECEIVE$BUFFER: procedure (buf$ptr,buf$length$ptr);

declare
    buf$ptr pointer,
    buf$length$ptr pointer,
    buf$length based buf$length$ptr byte;

    /* set up receive buffer pointer in global variable for interrupt service */
    rx$buffer$ptr=buf$ptr;
    receive$index=0;

    receive$status=not$complete;
    /* set status to not complete */

    /* wait until buffer received */
    while receive$status = not$complete do; end;
    buf$length=receive$length;
    if receive$status = complete
        then return true;
        else return false;
    end receive$BUFFER;
MPSC$RECEIVE$CHARACTER$INT: procedure interrupt 22H;

    /* ignore input if no open buffer */
    if receive$status <> not$complete then return;

    /* check for receive buffer overrun */
    if receive$index = 128
        then receive$status=overrun;
    else do;
        /* read character from MPSC and place in buffer - note that the
         * parity of the character must be masked off during this step if
         * the character is less than 8 bits (e.g., ASCII) */
        receive$buffer(receive$index),character=input(data$port) and 7FH;
        receive$index=receive$index+1; /* update receive buffer index */

        /* check for line feed to end line */
        if character = line$feed
            then do; receive$length=receive$index; receive$status=complete; end;
        end;
    end MPSC$RECEIVE$CHARACTER$INT;

MPSC$TRANSMIT$CHARACTER$INT: procedure interrupt 20H;

    /* check for more characters to transfer */
    if transmit$index < transmit$length
        then do;
            /* write next character from buffer to MPSC */
            output(data$port)=transmit$buffer(transmit$index);
            transmit$index=transmit$index+1; /* update transmit buffer index */
        end;
    else transmit$status=complete;
end MPSC$TRANSMIT$CHARACTER$INT;

RECEIVE$ERROR$INT: procedure interrupt 23H;

    declare
temp byte; /* temporary character storage */
output(cmd$port)=l; /* point to RR1 */
receive$status=input(cmd$port);
temp=input(data$port); /* discard character */
output(cmd$port)=error$reset; /* send error reset */

    /* *** other application dependent
     * error processing should be placed here  *** */
end RECEIVE$ERROR$INT;

EXTERNAL$STATUS$CHANGE$INT: procedure interrupt 21H;

    transmit$status=input(cmd$port) /* input status change information */
    output(cmd$port)=reset$ext$status;

    /* *** other application dependent
     * error processing should be placed here  *** */
end EXTERNAL$STATUS$CHANGE$INT;
APPENDIX C
INTERRUPT-DRIVEN TRANSMIT/RECEIVE SOFTWARE

declare
/* global variables for buffer manipulation */

rx$buffer$ptr pointer, /* pointer to receive buffer */
receive$buffer based rx$buffer$ptr(128) byte,
receive$status byte initial(0), /* indicates receive buffer status */
receive$index byte, /* current index into receive buffer */
receive$length byte, /* length of final receive buffer */

rx$buffer$ptr pointer, /* pointer to transmit buffer */
transmit$buffer based tx$buffer$ptr(128) byte,
transmit$status byte initial(0), /* indicates transmit buffer status */
transmit$index byte, /* current index into transmit buffer */
transmit$length byte, /* length of buffer to be transmitted */

cmd$port literally '43H',
data$port literally '41H',
a$cmd$port literally '42H',
b$cmd$port literally '43H',
line$feed literally '0AH',
not$complete literally '0',
complete literally 'OFFH',
overrun literally '1',

channel$reset literally '18H',
error$reset literally '30H',
reset$ext$status literally '10H';
MPSC$INT$INIT: procedure (clock$rate, stop$bits, parity$type, parity$enable, rx$char$length, rx$enable, auto$enable, tx$char$length, tx$enable, dtr, brk, rts, ext$en, tx$en, rx$en, stat$affects$vector, config, priority, vector$int$mode, int$vector);

declare
clock$rate byte, /* 2-bit code for clock rate divisor */
stop$bits byte, /* 2-bit code for number of stop bits */
parity$type byte, /* I-bit parity type */
parity$enable byte, /* I-bit parity enable */
rx$char$length byte, /* 2-bit receive character length */
rx$enable byte, /* I-bit receiver enable */
auto$enable byte, /* I-bit auto enable flag */
tx$char$length byte, /* 2-bit transmit character length */
tx$enable byte, /* I-bit transmitter enable */
dtr byte, /* I-bit status of DTR pin */
brk byte, /* I-bit data link break enable */
rts byte, /* I-bit status of RTS pin */
ext$en byte, /* I-bit external/status enable */
tx$en byte, /* I-bit Tx interrupt enable */
rx$en byte, /* 2-bit Rx interrupt enable/mode */
stat$affects$vector byte, /* I-bit status affects vector flag */
config byte, /* 2-bit system config - int/DMA */
priority byte, /* I-bit priority flag */
vector$int$mode byte, /* 3-bit interrupt mode code */
int$vector byte; /* 8-bit interrupt type code */

output(b$cmd$port)=channel$reset; /* channel reset */
output(b$cmd$port)=14H; /* point to WR4 */
/* set clock rate, stop bits, and parity information */
output(b$cmd$port)=shl(clock$rate, 6) or shl(stop$bits, 2) or shl(parity$type, 1) or parity$enable;

output(b$cmd$port)=13H; /* point to WR3 */
/* set up receiver parameters */
output(b$cmd$port)=shl(rx$char$length, 6) or rx$enable or shl(auto$enable, 5);

output(b$cmd$port)=15H; /* point to WR5 */
/* set up transmitter parameters */
output(b$cmd$port)=shl(tx$char$length, 5) or shl(tx$enable, 3) or shl(dtr, 7) or shl(brk, 4) or shl(rts, 1);

output(b$cmd$port)=12H; /* point to WR2 */
/* set up interrupt vector */
output(b$cmd$port)=int$vector;

output(a$cmd$port)=12H; /* point to WR2, channel A */
/* set up interrupt modes */
output(a$cmd$port)=shl(vector$int$mode, 3) or shl(priority, 2) or config;

output(b$cmd$port)=11H; /* point to WR1 */
/* set up interrupt enables */
output(b$cmd$port)=shl(rx$en, 3) or shl(stat$affects$vector, 2) or shl(tx$en, 1) or ext$en;

end MPSC$INT$INIT;
This application example shows the 8274 in a simple iAPX-86/88 system. The 8274 controls two separate asynchronous channels using its internal interrupt controller to request all data transfers. The 8274 driver software is described which transmits and receives data buffers provided by the CPU. Also, status registers are maintained in system memory to allow the CPU to monitor progress of the buffers and error conditions.

**THE HARDWARE INTERFACE**

Nothing could be easier than the hardware design of an interrupt-driven 8274 system. Simply connect the data bus lines, a few bus control lines, supply a timing clock for baud rate and, voila, it's done! For this example, the ubiquitous SDK-86 is used as the host CPU system. The 8274 interface is constructed on the wire-wrap area provided. While discussing the hardware interface, please refer to Diagram 1.

Placing the 8274 on the lower 8 bits of the 8086 data bus allows byte-wide data transfers at even I/O addresses. For simplicity, the 8274's CS/ input is generated by combining the M-IO/ select line with address line A7 via a 7432. This places the 8274 address range in multiple spots within the 8086 I/O address space. (While fine for this example, a more complete address decoding is recommended for actual prototype systems.) The 8086's A1 and A2 address lines are connected to the A0 and A1 8274 register select inputs respectively. Although other port assignments are possible because of the overlapping address spaces, the following I/O port assignments are used in this example:

<table>
<thead>
<tr>
<th>Port Function</th>
<th>I/O Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data channel A</td>
<td>0000H</td>
</tr>
<tr>
<td>Command/status A</td>
<td>0002H</td>
</tr>
<tr>
<td>Data channel B</td>
<td>0004H</td>
</tr>
<tr>
<td>Command/status B</td>
<td>0006H</td>
</tr>
</tbody>
</table>

To connect the 8274's interrupt controller into the system an inverter and pull-up resistor are needed to convert the 8274's active-low, interrupt-request output, IRQ, into the correct polarity for the 8086's INTR interrupt input. The 8274 recognizes interrupt-acknowledge bus cycles by connecting the INTA (INTerrupt Acknowledge) lines of the 8274 and 8086 together.

The 8274 ReaD and WRite lines directly connect to the respective 8086 lines. The RESET line requires an inverter. The system clock for the 8274 is provided by the PCLK (peripheral clock) output of the 8284A clock generator.

On the 8274's serial side, traditional 1488 and 1489 RS-232 drivers and receivers are used for the serial interface. The onboard baud rate generator supplies the channel baud rate timing. In this example, both sides of both channels operate at the same baud rate although this certainly is not a requirement. (On the SDK-86, the baud rate selection is hard-wired thru jumpers. A more flexible approach would be to incorporate an 8253 Programmable Interval Timer to allow software-configurable baud rate selection.)

That's all there is to it. This hardware interface is completely general-purpose and supports all of the 8274 features except the DMA data transfer mode which requires an external DMA controller. Now let's look at the software interface.

**SOFTWARE INTERFACE**

In this example, it is assumed that the 8086 has better things to do rather than continuously run a serial channel. Presenting the software as a group of callable procedures lets the designer include them in the main body of another program. The interrupt-driven data transfers give the effect that the serial channels are handled in the background while the main program is executing in the foreground. There are five basic procedures: a serial channel initialization routine and buffer handling routines for the transmit and receive data buffers of each channel. Appendix D-I shows the entire software listing. Listing line numbers are referenced as each major routing is discussed.

The channel initialization routine (INITIAL 8274), starting with line #203, simply sets each channel into a particular operating mode by loading the command registers of the 8274. In normal operation, once these registers are loaded, they are rarely changed. (Although this example assumes a simple asynchronous operating mode, the concept is easily extended for the byte- and bit-synchronous modes.)
Figure D-1. 8274/SDK-86 Hardware Interface
The channel operating modes are contained in two tables starting with line #163. As the 8274 has only one command register per channel, the remaining seven registers are loaded indirectly through the WR0 (Write Register 0) register. The first byte of each table entry is the register pointer value which is loaded into WR0 and the second byte is the value for that particular register.

The indicated modes set the 8274 for asynchronous operation with data characters 8 bits long, no parity, and 2 stop bits. An X16 baud rate clock is assumed. Also selected is the “interrupt on all RX character” mode with a variable interrupt vector compatible with the 8086/8088. The transmitters are enabled and all model control lines are put in their active state.

In addition to initializing the 8274, this routine also sets up the appropriate interrupt vectors. The 8086 assumes the first 1K bytes of memory contain up to 256 separate interrupt vectors. On the SDK-86 the initial 2K bytes of memory is RAM and therefore must be initialized with the appropriate vectors. (In a prototype system, this initial memory is probably ROM, thus the vector set-up is not needed.) The 8274 supplies up to eight different interrupt vectors. These vectors are developed from internal conditions such as data requests, status changes, or error conditions for each channel. The initialization routine arbitrarily assumes that the initial 8274 vector corresponds to 8086 vector location 80H (memory location 200H). This choice is arbitrary since the 8274 initial vector location is programmable.

Finally, the initialization routine sets up the status and flag in RAM. The meaning and use of these locations are discussed later.

Following the initialization routine are those for the transmit commands (starting with line #268). These commands assume that the host CPU has initialized the publically declared variables for the transmit buffer pointer, TX_POINTER_CHx, and the buffer length, TX_LENGTH_CHx. The transmit command routines simply clear the transmitter empty flag, TX_EMPTY_CHx, and load the first character of the buffer into the transmitter. It is necessary to load the first character in this manner since transmitter interrupts are generated only when the 8274’s transmit data buffer becomes empty. It is the act of becoming empty which generates the interrupt not simply the buffer being empty, thus the transmitter needs one character to start.

The host CPU can monitor the transmitter empty flag, TX_EMPTY_CHx, in order to determine when transmission of the buffer is complete. Obviously, the CPU should only call the command routine after first checking that the empty flag is set.

After returning to the main program, all transmitter data transfers are handled via the transmitter interrupt service routines starting at lines #360 and #443. These routines start by issuing an End-Of-Interrupt command to the 8274. (This command resets the internal interrupt controller logic of the 8274 for this particular vector and opens the logic for other internal interrupt requests. The routines next check the length count. If the buffer is completely transmitted, the transmitter empty flag, TX_EMPTY_CHx, is set and a command is issued to the 8274 to reset its interrupt line. Assuming that the buffer is not completely transmitted, the next character is output to the transmitter. In either case, an interrupt return is executed to return to the main CPU program.

The receiver commands start at line #314. Like the transmit commands, it is assumed that the CPU has initialized the receive-buffer-pointer public variable, RX_POINTER_CHx. This variable points to the first location in an empty receive buffer. The command routines clear the receiver ready flag, RX_READY_CHx, and then set the receiver enable bit in the 8274 WR3 register. With the receiver now enabled, any received characters are placed in the receive buffer using interrupt-driven data transfers.

The received data service routines, starting at lines #402 and #485, simply place the received character in the buffer after first issuing the EOI command. The character is then compared to an ASCII CR. An ASCII CR causes the routine to set the receiver ready flag, RX_READY_CHx, and to disable the receiver. The CPU can interrogate this flag to determine when the buffer contains a new line of data. The receive buffer pointer, RX_POINTER_CHx, points to the last received character and the receive counter, RX_COUNTER_CHx, contains the length.

That completes our discussion of the command routines and their associated interrupt service routines. Although not used by the commands, two additional service routines are included for completeness. These routines handle the error and status-change interrupt vectors.

The error service routines, starting at lines #427 and #510, are vectored to if a special receive condition is detected by the 8274. These special receive conditions include parity, receiver overrun, and framing errors. When this vector is generated, the error condition is indicated in RRI (Read Register 1). The error service routine issues an EOI command, reads RRI and places it in the ERROR_MSG_CHx variable, and then issues
a reset error command to the 8274. The CPU can monitor the error message location to detect error conditions. The designer, of course, can supply his own error service routine.

Similarly, the status-change routines (starting lines #386 and #469) are initiated by a change in the modem-control status lines CTS/, CD/, or SYNDET/. (Note that WR2 bit 0 controls whether the 8274 generates interrupts based upon changes in these lines. Our WR2 parameter is such that the 8274 is programmed to ignore changes for these inputs.) The service routines simply read RR0, place its contents in the STATUS_MSG-_CHx variable and then issue a reset external status command. Read Register 0 contains the state of the modem inputs at the point of the last change.

Well, that's it. This application example has presented useful, albeit very simple, routines showing how the 8274 might be used to transmit and receive buffers using an asynchronous serial format. Extensions for byte- or bit-synchronous formats would require no hardware changes due to the highly programmable nature of the 8274's serial formats.

---

**8274 APPLICATION BRIEF PROGRAM**

MCS-86 MACRO ASSEMBLER  ASYNCH

IS15-11 MCS-86 MACRO ASSEMBLER V2.1 ASSEMBLY OF MODULE ASYNCH

OBJECT MODULE PLACED IN F1.ASYNCH.OBJ

ASSEMBLER INVOKED BY: ASMB6 .F1.ASYNCH.SRC

<table>
<thead>
<tr>
<th>LOC OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>;************************************************************************</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>;* 8274 APPLICATION BRIEF PROGRAM *</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>;* THE 8274 IS INITIALIZED FOR SIMPLE ASYNCHRONOUS SERIAL</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>;* FORMAT AND VECTORED INTERRUPT-DRIVEN DATA TRANSFERS *</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>;* THE INITIALIZATION ROUTINE ALSO LOOPS THE 8086'S INTERRUPT</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>;* VECTOR TABLE FROM THE CODE SEGMENT INTO LOW RAM ON THE</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>;* SX-86 THE TRANSMITTER AND RECEIVER ARE LEFT ENABLED. *</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>;* FOR TRANSMIT, THE CPU PASSES IN MEMORY THE POINTER OF A</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>;* BUFFER TO TRANSMIT AND THE BYTE LENGTH OF THE BUFFER.</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>;* THE DATA TRANSFER PROCEED USING INTERRUPT-DRIVEN TRANSFERS. *</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>;* A STATUS BIT IN MEMORY IS SET WHEN IF BUFFERS IS EMPTY. *</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>;* FOR RECEIVE, THE CPU PASSES THE POINTER OF A BUFFER TO FILL</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>;* THE BUFFER IS FILLED UNTIL A 'CR,CHR' CHARACTER IS RECEIVED *</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>;* A STATUS BIT IS SET AND THE CPU MAY READ THE RX POINTER TO</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>;* DETERMINE THE LOCATION OF THE LAST CHARACTER. *</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>;* ALL ROUTINES ARE ASSUMED TO EXIST IN THE SAME CODE SEGMENT.</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>;* CALL'S TO THE SERVICE ROUTINES ARE ASSUMED TO BE &quot;SHORT&quot; OR</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>;* INTRASEGMENT (ONLY THE RETURN ADDRESS IP IS ON THE STACK). *</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>;*</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>;************************************************************************</td>
</tr>
</tbody>
</table>
MCS-86 MACRO ASSEMBLER

LOC OBJ

NAME ASYNCO , MODULE NAME

PUBLIC DECLARATIONS FOR COMMAND ROUTINES

PUBLIC INIT2274 , INITIALIZATION ROUTINE
PUBLIC TXCMDMAND,CHB , TX BUFFER COMMAND CHANNEL B
PUBLIC TXCOMMAND,CHA , TX BUFFER COMMAND CHANNEL A
PUBLIC RXCOMMAND,CHB , RX BUFFER COMMAND CHANNEL B
PUBLIC RXCOMMAND,CHA , RX BUFFER COMMAND CHANNEL A

PUBLIC DECLARATIONS FOR STATUS VARIABLES

PUBLIC RXRDY,CHB , RX READY FLAG CHB
PUBLIC RXRDY,CHA , RX READY FLAG CHA
PUBLIC TXEMPT,CHB , TX EMPTY FLAG CHB
PUBLIC TXEMPT,CHA , TX EMPTY FLAG CHA
PUBLIC RXCNT,CHB , RX BUFFER COUNTER CHB
PUBLIC RXCNT,CHA , RX BUFFER COUNTER CHA
PUBLIC ERRORMSG,CHB , ERROR FLAG CHB
PUBLIC ERRORMSG,CHA , ERROR FLAG CHA
PUBLIC STATMSG,CHB , STATUS FLAG CHB
PUBLIC STATMSG,CHA , STATUS FLAG CHA

PUBLIC DECLARATIONS FOR VARIABLES PASSED TO THE TRANSIT AND RECEIVE COMMANDS.

PUBLIC TXPOINTER,CHB , TX BUFFER POINTER FOR CHB
PUBLIC TXLENGTH,CHB , TX LENGTH OF BUFFER FOR CHB
PUBLIC TXPOINTER,CHA , TX BUFFER POINTER FOR CHA
PUBLIC TXLENGTH,CHA , TX LENGTH OF BUFFER FOR CHA
PUBLIC RXPOINTER,CHB , RX BUFFER POINTER FOR CHB
PUBLIC RXPOINTER,CHA , RX BUFFER POINTER FOR CHA

I/O PORT ASSIGNMENTS

CHANNEL A PORT ASSIGNMENTS

DATA PORT,CHA EQU 0 , DATA I/O PORT
COMMAND PORT,CHA EQU 2 , COMMAND PORT

CHANNEL B PORT ASSIGNMENTS

DATA PORT,CHB EQU 4 , DATA I/O PORT
COMMAND PORT,CHB EQU 6 , COMMAND PORT
STATUS PORT,CHB EQU 2 , COMMAND PORT

-MISC SYSTEM EQUATES

CR CHR EQU 0OH , ASCII CR CHARACTER CODE
INT_TABLE_BASE EQU 200H , INT VECTOR BASE ADDRESS
CODE_START EQU 500H , START LOCATION FOR CODE

+1 #EJECT

-PAM ASSIGNMENTS FOR DATA SEGMENT

DATA SEGMENT
LOC  OBJ  LINE  SOURCE

91  .VECTOR INTERRUPT TABLE - ASSUME INITIAL 8274 INTERRUPT
92  .VECTOR IS NUMBER 88 (<232H>) FOR EACH VECTOR THE TABLE
93  .CONTAINS START LOCATION AND CODE SEGMENT REGISTER VALUE
94  .THE TABLE IS LINKED FROM ROM
95
96  ORG INT_TABLE_BASE
97
98  TABLE_BASE DW 0  // TX INTERRUPT VECTOR FOR CHB
99
100  STATS_VECTOR_CHB DW 0  // STATUS INTERRUPT VECTOR FOR CHB
101  RX_VECTOR_CHB DW 0  // RX INTERRUPT VECTOR FOR CHB
102  RX_CS_CHB DW 0  // RX BUFFER DESCRIPTION FOR CHB
103  ERR_VECTOR_CHB DW 0  // ERROR INTERRUPT VECTOR FOR CHB
104  ERR_CS_CHB DW 0  // ERROR CMOS DESCRIPT
105  TX_POINTER_CHB DW 0  // TX BUFFER POINTER FOR CHB
106  TX_LENGTH_CHB DW 0  // TX BUFFER LENGTH OF CHB
107  RX_POINTER_CHB DW 0  // RX BUFFER POINTER FOR CHB
108  RX_COUNT_CHB DW 0  // RX LENGTH COUNTER FOR CHB
109  TX_EMPTY_CHB DB 0  // TX DONE FLAG
110  RX_READY_CHB DB 0  // READY Flag (+1 IF CR.CHB RECEIVED, ELSE 0)
111  STATUS_MSG_CHB DB 0  // STATUS CHANGE MESSAGE
112  ERROR_MSG_CHB DB 0  // ERROR MESSAGE LOCATION (+1 IF NO ERROR)
113
114  CHANNEL A POINTERS AND STATUS
115
116  TABLE_BASE DW 0  // TX INTERRUPT VECTOR FOR CHB
117  TX_LENGTH_CHB DW 0  // TX BUFFER LENGTH OF CHB
118  RX_POINTER_CHB DW 0  // RX BUFFER POINTER FOR CHB
119  RX_COUNT_CHB DW 0  // RX LENGTH COUNTER FOR CHB
120  TX_EMPTY_CHB DB 0  // TX DONE FLAG
121  RX_READY_CHB DB 0  // READY Flag (+1 IF CR.CHB RECEIVED, ELSE 0)
122  STATUS_MSG_CHB DB 0  // STATUS CHANGE MESSAGE
123  ERROR_MSG_CHB DB 0  // ERROR MESSAGE LOCATION (+1 IF NO ERROR)
124
125  CHANNEL B POINTERS AND STATUS
126
127  TABLE_BASE DW 0  // TX INTERRUPT VECTOR FOR CHB
128  TX_LENGTH_CHB DW 0  // TX BUFFER LENGTH OF CHB
129  RX_POINTER_CHB DW 0  // RX BUFFER POINTER FOR CHB
130  RX_COUNT_CHB DW 0  // RX LENGTH COUNTER FOR CHB
131  RXReady_CHB DB 0  // READY Flag (+1 IF CR.CHB RECEIVED, ELSE 0)
132  STATUS_MSG_CHB DB 0  // STATUS CHANGE MESSAGE
133  ERROR_MSG_CHB DB 0  // ERROR MESSAGE LOCATION (+1 IF NO ERROR)
134
135  DATA ENDS
136
137 *1 EJECT
MCS-86 MACRO ASSEMBLER ASYNCB

LOC OBJ LINE SOURCE

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

AP-134
; START OF COMMAND ROUTINES

; INITIALIZATION COMMAND FOR THE 8274 - THE 8274

; IS SETUP ACCORDING TO THE PARAMETERS STORED IN

; PROM ABOVE STARTING AT CMSWB FOR CHANNEL B AND

; CMSTRA FOR CHANNEL A

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*
TX_COMMAND.CHB:

TX_CHANNEL B COMMAND ROUTINE - ROUTINE IS CALLED TO
TRANSMIT A BUFFER, THE BUFFER STARTING ADDRESS,
TX POINTER, AND THE BUFFER LENGTH, TX LENGTH CHB.
MUST BE INITIALIZED BY THE CALLING PROGRAM.
BOTH ITEMS ARE WORD VARIABLES.

TX_CHANNEL A COMMAND ROUTINE - ROUTINE IS CALLED TO
TRANSMIT A BUFFER, THE BUFFER STARTING ADDRESS,
TX POINTER, AND THE BUFFER LENGTH, TX LENGTH CHA.
MUST BE INITIALIZED BY THE CALLING PROGRAM.
BOTH ITEMS ARE WORD VARIABLES.

TX_CHANNEL.CHA:

TX COMMAND FOR CHANNEL B - THE CALLING ROUTINE MUST
INITIALIZE RX_POINTER.CHB TO POINT AT THE RECEIVE
BUFFER BEFORE CALLING THIS ROUTINE.
MACIO ASSEMBLER

LOC OBJ LINE SOURCE
311 *,
312 .*************************************************************************** *
313
0506 314 RX.CMD
315 S0 315 PUSH AX
316 SAVE REGISTERS
317 S1 316 POP DX
318 MOV RX,READY.8: CLEAR RX READY FLAG
319 S2 317 MOV RX,COUNT.8: CLEAR RX COUNTER
320 S3 318 MOV DX,CMD.8,POINT AT CMD PORT
321 S4 320 MOV AL,3: SET UP FOR RX
322 S5 321 OUT DX,AL
323 S6 322 MOV AL,BCCH: YOUR 8 BITS/CHAR ENABLE RX
324 S7 323 OUT DX,AL
325 S8 324 POP DX
326 S9 325 POP RX
327 C3 326 RET: RETURN
328 *
329 RX COMMAND FOR CHANNEL A: THE CALLING ROUTINE MUST
330 INITIALIZE RX.POINTER.CHB TO POINT AT THE RECEIVE BUFFER BEFORE CALLING THIS ROUTINE
331 *
332 *
333 *
334 *
335 *
336 RX.CMD
337 S0 336 PUSH AX
338 SAVE REGISTERS
339 S1 337 POP DX
340 MOV RX,READY.8: CLEAR RX READY FLAG
341 S2 340 MOV RX,COUNT.8: CLEAR RX COUNTER
342 S3 341 MOV DX,CMD.8,POINT AT CMD PORT
343 S4 343 MOV AL,3: SET UP FOR RX
344 S5 342 OUT DX,AL
345 S6 343 MOV AL,BCCH: YOUR 8 BITS/CHAR ENABLE RX
346 S7 344 OUT DX,AL
347 S8 345 POP DX
348 S9 346 POP RX
349 C3 347 RET: RETURN
350 *
351 RX.TX
352 .*************************************************************************** *
353 START OF INTERRUPT SERVICE ROUTINES
354 *
355 *
356 *
357 /CHANNEL B TRANSMIT DATA SERVICE ROUTINE
358
0600 S2 358 XMITIN. PUSH DX
359 SAVE REGISTERS
360 S3 360 PUSH DI
361 S4 361 PUSH AX
362 S5 362 CALL EOI: SEND EOI COMMAND TO 8274
363 S6 363 INC TX.POINTER.CHB: POINT TO NEXT CHAR
364 S7 364 DEC TX.LENGTH.CHB: DEC LENGTH COUNTER
365 S8 365 JE XIB: TEST IF DONE
366 7406 S9 366 MOV DI: TX.POINTER.CHB
367 B8000000 S10 367 MOV DX,DATA.PORT.CHB: NOT DONE - GET NEXT CHARACTER
368 S11 368 MOV DI: TX.POINTER.CHB
369 S12 369 MOV AL,DI: PUT CHARACTER IN AL
369 S13 370 OUT DX,AL: OUTPUT IT TO 8274
370
LOC OBJ LINE SOURCE

0622 58 371 POP AX ;RESTORE REGISTERS
0623 5F 372 POP DI
0624 5A 373 POP DX
0625 CF 374 IRET ;RETURN TO FOREGROUND
0626 B66088 375 MOV DX, COMMAND_PORT.CHB ;ALL CHARACTERS HAVE BEEN SEND
0629 B628 376 MOV AL, 20H ;RESET TRANSMITTER INTERRUPT PENDING
062B EE 377 OUT DX, AL
062C CGA62880201 378 MOV TL,EMPTY.CHB, 1 ;DONE - SET TX EMPTY FLAG CHB
0631 58 379 POP AX ;RESTORE REGISTERS
0632 5F 380 POP DI
0633 5A 381 POP DX
0634 CF 382 IRET ;RETURN TO FOREGROUND
384 ,CHANNEL B STATUS CHANGE SERVICE ROUTINE
0635 52 386 STAINB: PUSH DX ;SAVE REGISTERS
0636 57 387 PUSH DI
0637 5B 388 PUSH AX
0638 E80500 389 CALL EOI ;SEND EOI COMMAND TO 8274
063B B66088 390 MOV DX, COMMAND_PORT.CHB
063E EC 391 IN AL, DX ;READ RR8
0640 A22A92 392 MOV STATUS.MSG.CHB, AL ;PUT RR8 IN STATUS MESSAGE
0642 B818 393 MOV AL, 1AH ;SEND RESET STATUS INT COMMAND TO 8274
0644 EE 394 OUT DX, AL
0645 58 395 POP AX ;RESTORE REGISTERS
0646 5F 396 POP DI
0647 5A 397 POP DX
0648 CF 398 IRET
399 ,CHANNEL B RECEIVED DATA SERVICE ROUTINE
400 0649 52 402 RCVIND: PUSH DX ;SAVE REGISTERS
064A 57 403 PUSH DI
064B 5B 404 PUSH AX
064C E81000 405 CALL EOI ;SEND EOI COMMAND TO 8274
064F B632482 406 MOV DI, RX_POINTER.CHB ;GET RX CHB BUFFER POINTER
0522 B66088 407 MOV DX, DATA_PORT.CHB
0556 EC 408 IN AL, DX ;READ CHARACTER
0557 B66085 409 MOV [DI], AL ;STORE IN BUFFER
0559 FF60240C 410 INC RX_POINTER.CHB ;BUMP THE BUFFER POINTER
0562 FF602602 411 INC RX_COUNT.CHB ;BUMP THE COUNTER
0566 3C00 412 CMP AL, CR.CHAR ;TEST IF LAST CHARACTER TO BE RECEIVED?
0567 730E 413 JNE RIB
0565 C8A62960201 414 MOV RX,READY.CHB, 1 ;YES, SET READY FLAG
066A B66088 415 MOV DX, COMMAND_PORT.CHB ;POINT AT COMMAND PORT
066B B800 416 MOV AL, 2 ;POINT AT HRS
066F EE 417 OUT DX, AL
0679 B608 418 MOV AL, B0H ;DISABLE RX
0672 EE 419 OUT DX, AL
0673 58 420 POP AX ;EITHER WAY: RESTORE REGISTERS
0674 5F 421 POP DI
0675 5A 422 POP DX
0676 CF 423 IRET ;RETURN TO FOREGROUND
424 ,CHANNEL B ERROR SERVICE ROUTINE
425 0677 52 427 ERRIND: PUSH DX ;SAVE REGISTERS
0678 5F 428 PUSH AX
0679 E89400 429 CALL EOI ;SEND EOI COMMAND TO 8274
067C B66088 430 MOV DX, COMMAND_PORT.CHB

6-224
067F B081 431 MOV AL, 1 ;POINT AT RRL
0681 EE 432 OUT DX, AL
0682 EC 433 IN AL, DX ;READ RRL
0683 A22082 434 MOV ERROR, MSG, CHB, AL ;SAVE IT IN ERROR FLAG
0686 B030 435 MOV AL, 3BH ;SEND RESET ERROR COMMAND TO 8274
0688 EE 436 OUT DX, AL
0689 58 437 POP AX ;RESTORE REGISTERS
068A 5A 438 POP DX
068B CF 439 IRET ;RETURN TO FOREGROUND

068C 52 440 ;CHANNEL A TRANSMIT DATA SERVICE ROUTINE
068D 57 441 XATINA: PUSH DX ;SAVE REGISTERS
068E 59 442 PUSH DI
068F 837E90 443 PUSH AX
0692 E862E802 444 CALL EOI ;SEND EOI COMMAND TO 8274
0696 FF62E802 445 INC TX.POINTER.CHAR ;POINT TO NEXT CHARACTER
0699 74E8 446 JE XIA ;TEST IF DONE
069C B0000 447 MOV DX, DATA.PORT.CHAR ;NOT DONE - GET NEXT CHARACTER
069F B00E802 451 MOV DI, TX.POINTER.CHAR
06A0 B045 452 MOV AL, DI1 ;PUT CHARACTER IN AL
06A5 EE 453 OUT DX, AL ;OUTPUT IT TO 8274
06A6 58 454 POP AX ;RESTORE REGISTERS
06A7 5F 455 POP DI
06A8 5A 456 POP DX
06A9 CF 457 IRET ;RETURN TO FOREGROUND
06AB B0200 458 XIA: MOV DX, COMMAND.PORT.CHAR ;ALL CHARACTERS HAVE BEEN SEND
06AD B028 459 MOV AL, 28H ;RESET TRANSMITTER INTERRUPT PENDING
06AF EE 460 OUT DX, AL
06B0 C8A348201 461 MOV TX.EMPTY.CHAR.1, 1 ;DONE - 5O SET TX EMPTY FLAG CHB
06B5 58 462 POP AX ;RESTORE REGISTERS
06B6 5F 463 POP DI
06B7 5A 464 POP DX
06B8 CF 465 IRET ;RETURN TO FOREGROUND

06B9 52 466 ;CHANNEL A STATUS CHANGE SERVICE ROUTINE
06BA 57 467 STRINA: PUSH DX ;SAVE REGISTERS
06BB 59 468 PUSH DI
06BC E85180 469 CALL EOI ;SEND EOI COMMAND TO 8274
06BF B0200 470 MOV DX, COMMAND.PORT.CHAR
06C2 EC 471 IN AL, DX ;SEND RRA
06C3 R23082 472 MOV STATUS, MSG, CHB, AL ;PUT RRA IN STATUS MESSAGE
06C6 B010 473 MOV AL, 1AH ;SEND RESET STATUS INT COMMAND TO 8274
06C8 EE 474 OUT DX, AL
06CA 58 475 POP AX ;RESTORE REGISTERS
06CB 5F 476 POP DI
06CC 5A 477 POP DX
06CD CF 478 IRET

06DE 52 479 ;CHANNEL A RECEIVED DATA SERVICE ROUTINE
06DF 57 480 ROVINA: PUSH DX ;SAVE REGISTERS
06E0 59 481 PUSH DI
06E1 5F 482 PUSH RX
06E4 E83D99 483 CALL EOI ;SEND EOI COMMAND TO 8274
06E7 B00000 484 MOV DI, RX.POINTER.CHAR ;GET RX CHAR BUFFER POINTER
06E7 B0000 485 MOV DX, DATA.PORT.CHAR
AP-134

MC6800 MARCH ASSEMBLY

LOC OBJ LINE SOURCE

060A EC 491 IN AL, DX ; READ CHARACTER
060B 8005 492 MOV [DI], AL ; STORE IN BUFFER
060C FF063002 493 INC RCL, POINTER CHAR ; BUMP THE BUFFER POINTER
060D FF063202 494 INC RCL, COUNT CHAR ; BUMP THE COUNTER
060E 0D00 495 CMP AL, CR CHAR ; TEST IF LAST CHARACTER TO BE RECEIVED?
060F 799E 496 JNE RIA
0610 C6003E8020B1 497 MOV RCL, READY CHAR ; YES, SET READY FLAG
0611 890200 498 MOV DX, COMMAND PORT CHAR ; POINT AT COMMAND PORT
0612 F003 499 MOV AL, 3 ; POINT AT MR3
0613 FF EE 500 OUT DX, AL
0614 F00C 501 MOV AL, RC8H ; DISABLE RX
0615 EE EE 502 OUT DX, AL
0616 58 503 RIA POP AX ; EITHER WAY, RESTORE REGISTERS
0617 5F 504 POP DI
0618 5A 505 POP DX
0619 CF 506 IRET ; RETURN TO FOREGROUND
061A 508 ; CHANNEL A ERROR SERVICE ROUTINE
061B 509

06F8 52 510 ERROR, PUSH DX ; SAVE REGISTERS
06F9 59 511 PUSH AX
06FA 810000 512 CALL EOI ; SEND EOI COMMAND TO 8274
0700 890200 513 MOV DX, COMMAND PORT CHAR
0701 0F03 514 MOV AL, 1 ; POINT AT RR1
0702 EE 515 OUT DX, AL
0703 58 516 IN AL, DX ; READ RR1
0704 232782 517 MOV ERROR, MSG CHAR, AL ; SAVE IT IN ERROR FLAG
0705 8008 518 MOV AL, 3BH ; SEND RESET ERROR COMMAND TO 8274
0706 EE 519 OUT DX, AL
0707 58 520 POP AX ; RESTORE REGISTERS
0708 5A 521 POP DX
0709 CF 522 IRET ; RETURN TO FOREGROUND
070A 524 ; END-OF-INTERUPT ROUTINE - SENDS EOI COMMAND TO 8274
070B 525 ; THIS COMMAND MUST ALWAYS TO ISSUED ON CHANNEL A
070C 526

0710 59 527 EOI: PUSH AX ; SAVE REGISTERS
0711 52 528 PUSH DX
0712 890200 529 MOV DX, COMMAND PORT CHAR ; ALWAYS FOR CHANNEL A
0713 0F08 530 MOV AL, 3BH
0714 EE 531 OUT DX, AL
0715 5A 532 POP DX
0716 58 533 POP AX
0717 C3 534 RET
0718 535
0719 536 ; END OF CODE ROUTINE
071A 537

578 ABC ENDS
539

ASSEMBLY COMPLETE. NO ERRORS FOUND

210311-001
REFERENCES


Synchronous Communication with the 8274 Multiple Protocol Serial Controller

Sikander Naqvi
Application Engineer
INTRODUCTION:

The INTEL 8274 is a Multi-Protocol Serial Controller, capable of handling both asynchronous and synchronous communication protocols. Its programmable features allow it to be configured in various operating modes, providing optimization to given data communication application.

This application note describes the features of the MPSC in Synchronous Communication applications only. It is strongly recommended that the reader read the 8274 Data Sheet and Application Note AP134 "Asynchronous Communication with the 8274 Multi-Protocol Serial Controller" before reading this Application Note. This Application note assumes that the reader is familiar with the basic structure of the MPSC, in terms of pin description, Read/Write registers and asynchronous communication with the 8274. Appendix A contains the software listings of the Application Example and Appendix B shows the MPSC Read/Write Registers for quick reference.

The first section of this application note presents an overview of the various synchronous protocols. The second section discusses the block diagram description of the MPSC. This is followed by the description of MPSC interrupt structure and mode of operation in the third and fourth sections. The fifth section describes a hardware/software example, using the INTEL single board computer iXBC88/45 as the hardware vehicle. The sixth section consists of some specialized applications of the MPSC. Finally, in section seven, some useful programming hints are summarized.

<table>
<thead>
<tr>
<th>OPENING FLAG BYTE</th>
<th>ADDRESS FIELD(A)</th>
<th>CONTROL FIELD(C)</th>
<th>DATA FIELD</th>
<th>FRAME CHECK SEQUENCE</th>
<th>CLOSING FLAG BYTE</th>
</tr>
</thead>
</table>

* Extendable to 2 or More Bytes
** Extendable to 2 Bytes

SYNCHRONOUS PROTOCOL OVERVIEW

This section presents an overview of various synchronous protocols. The contents of this section are fairly tutorial and may be skipped by the more knowledgeable reader.

Bit Oriented Protocols Overview

Bit oriented protocols have been defined to manage the flow of information on data communication links. One of the most widely known protocol is the one defined by the International Standards Organization: HDLC (High Level Data Link Control). The American Standard Associations' protocol, ADCCP is similar to HDLC. CCITT Recommendation X.25 layer 2 is also an acceptable version of HDLC. Finally, IBM's SDLC (Synchronous Data Link Control) is also a subset of the HDLC.

In this section, we will concentrate most of our discussion on HDLC. Figure 1 shows a basic HDLC frame format.

A frame consists of five basic fields: Flag, Address, Control, Data and Error Detection. A frame is bounded by flags - opening and closing flags. An address field is 8 bits wide, extendable to 2 or more bytes. The control field is also 8 bits wide, extendable to two bytes. The data field or information field may be any number of bits. The data field may or may not be on an 8 bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags.

ZERO BIT INSERTION

The flag has a unique binary bit pattern: 7E HEX. To eliminate the possibility of the data field containing a 7E HEX pattern, a bit stuffing technique called Zero Bit Insertion is used. This technique specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 01111110 is ever transmitted between flags. On the receiving side, after receiving the flag, the receiver hardware automatically deletes any 0 following five consecutive 1's. The 8274 performs zero bit insertion and deletion automatically in the SDLC/HDLC mode. The zero-bit stuffing ensures periodic transitions in the data stream. These transitions are necessary for a phase lock circuit, which may be used at the receiver end to generate a receive clock which is in phase to the received data. The inserted and deleted 0's are not included in the CRC checking. The address field is used to address a given secondary station. The control field contains the link-level control information which includes implied acknowledgement, supervisory commands and responses, etc. A more detailed discussion of higher level protocol functions is beyond the scope of this application note. Interested readers may refer to the references at the end of this application note.

The data field may be of any length and content in HDLC. Note that SDLC specifies that data field be a multiple of bytes only. In data communications, it is gen-
erally desirable to transmit data which may be of any content. This requires that data field should not contain characters which are defined to assist the transmission protocol (like opening flag 7EH in HDLC/SDLC communications). This property is referred to as "data transparency". In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion discussed earlier and the bit orientated nature of the protocol.

The last field is the FCS (Frame Check Sequence). The FCS uses the error detecting techniques called Cyclic Redundancy Check. In SDLC/HDLC, the CCITT-CRC must be used.

**NON-RETURN TO ZERO INVERTED (NRZI)**

NRZI is a method of clock and data encoding that is well suited to the HDLC protocol. It allows HDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while a 0 causes a change of state. NRZI coding ensures that an active data line will have transition at least every 5-bit times (recall Zero Bit Insertion), while contiguous 0's will cause a change of state. Thus, ZBI and NRZI encoding makes it possible for a phase lock circuit at the receiver end to derive a receive clock (from received data) which is synchronized to the received data and at the same time ensure data transparency.

**Byte Synchronous Communication**

As the name implies, Byte Synchronous Communication is a synchronous communication protocol which means that the transmitting station is synchronized to the receiving station through the recognition of a special sync character or characters. Two examples of Byte Synchronous protocol are the IBM Bisync and Monosync. Bisync has two starting sync characters per message while monosync has only one sync character. For the sake of brevity, we will only discuss Bisync here. All the discussion is valid for Monosync also. Any exceptions will be noted. Figure 2 shows a typical Bisync message format.

The Bisync protocol is defined for half duplex communication between two or more stations over point to point or multipoint communication lines. Special characters control link access, transmission of data and termination of transmission operations for the system. A detailed discussion of these special control characters (SYN, ENQ, STX, ITB, ETB, ETX, DLE, SOH, ACK0, ACK1, WACK, NAK and EOT, etc) is beyond the scope of this Application Note. Readers interested in more detailed discussion are directed to the references listed at the end of this Application Note.

As shown in Figure 2, each message is preceded by two sync characters. Since the sync characters are defined at the beginning of the message only, the transmitter must insert fill characters (sync) in order to maintain synchronization with the receiver when no data is being transmitted.

**TRANSPARENT TRANSMISSION**

Bisync protocol requires special control characters to maintain the communication link over the line. If the data is EBCDIC encoded, then transparency is ensured by the fact that the data field will not contain any of the bisync control characters. However, if data does not conform to standard character encoding techniques, transparency in bisync is achieved by inserting a special character DLE (Data Link Escape) before and after a string of characters which are to be transmitted transparently. This ensures that any data characters which match any of the special characters are not confused for special characters. An example of a transparent block is shown in Figure 3.

In a transparent mode, it is required that the CRC(BCC) is not performed on special characters. Later on, we will show how the 8274 can be used to achieve transparent transmission in Bisync mode.

<table>
<thead>
<tr>
<th>SYNC</th>
<th>SYNC</th>
<th>SOH</th>
<th>HEADER</th>
<th>STX TEXT</th>
<th>ETX OR ETB</th>
<th>CRC1</th>
<th>CRC2</th>
</tr>
</thead>
</table>

**Figure 2. Bisync Message Format**

<table>
<thead>
<tr>
<th>DLE</th>
<th>STX</th>
<th>TRANSPARENT TRANSMISSION</th>
<th>DLE</th>
<th>ETX</th>
<th>BCC</th>
</tr>
</thead>
</table>

Enter transparent mode

return to normal mode

**Figure 3. Bisync Transparent Format**
This section discusses the block diagram view of the 8274. The CPU interface and serial interface is discussed separately. This will be followed by a hardware example in the fifth section, which will show how to interface the 8274 with the Intel CPU 8088. The 8274 block diagram is shown in Figure 4.

### CPU Interface

The CPU interface to the system interface logic block utilizes the A0, A1, CS, RD and WR inputs to communicate with the internal registers of the 8274. Figure 5 shows the address of the internal registers. The DMA interface is achieved by utilizing DMA request lines for each channel: TxDQA, TxDQB, RxDQA, RxDQB. Note that

<table>
<thead>
<tr>
<th>CS</th>
<th>A1</th>
<th>A0</th>
<th>Read Operation</th>
<th>Write Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>CHA DATA READ</td>
<td>CHA DATA WRITE</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>CHA STATUS REGISTER (RR0,RR1)</td>
<td>CHA COMMAND/PARAMETER (WR0–WR7)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>CHB DATA READ</td>
<td>CHB DATA WRITE</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>CHB STATUS REGISTER (RR0,RR1,RR2)</td>
<td>CHB COMMAND/PARAMETER (WR0–WR7)</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>HIGH Z</td>
<td>HIGH Z</td>
</tr>
</tbody>
</table>

Figure 4. 8274 Block Diagram

Figure 5. Bus Interface
TxDRQ_B and RxDRQ_A becomes IPO and IPI respectively in non-DMA mode. IPI is the Interrupt Priority Input and IPO is the Interrupt Priority Output. These two pins can be used for connecting multiple MPSCs in a daisy chain. If the Wait Mode is programmed, then TxRDQ_A and RxRDQ_A pins become RDY_B and RDY_A pins. These pins can be wire-ored and are usually hooked up to the CPU RDY line to synchronize the CPU for block transfers. The INT pin is activated whenever the MPSC requires CPU attention. The INTA may be used to utilize the powerful vectored mode feature of the 8274. Detailed discussion on these subjects will be done later in this Application Note. The Reset pin may be used for hardware reset while the clock is required to click the internal logic on the MPSC.

Serial Interface

On the serial side, there are two completely independent channels: Channel A and Channel B. Each channel consists of a transmitter block, receiver block and a set of read/write registers which are used to initialize the device. In addition, a control logic block provides the modem interface pins. Channel B serial interface logic is a mirror image of Channel A serial interface logic, except for one exception: there is only one pin for RTS_B and SYNDET_B. At a given time, this pin is either RTS_B or SYNDET_B. This mode is programmable through one of the internal registers on the MPSC.

Transmit And Receive Data Path

Figure 6 shows a block diagram for transmit and receive data path. Without describing each block on the diagram, a brief discussion of the block diagram will be presented here.

TRANSMIT DATA PATH

The transmit data is transferred to the twenty-bit serial shift register. The twenty-bits are needed to store two bytes of sync characters in bisync mode. The last three bits of the shift register are used to indicate to the internal control logic that the current data byte has been shifted out of the shift register. The transmit data in the transmit shift register is shifted out through a two bit delay onto the TxData line. This two bit delay is used to synchronize the internal shift clock with the external transmit clock. The data in the shift register is also presented to zero bit insertion logic which inserts a zero after sensing five contiguous ones in the data stream. In parallel to all this activity, the CRC-generator is computing CRC on the transmitted data and appends the frame with CRC bytes at the end of the data transmission.
RECEIVE DATA PATH

The received data is passed through a one-bit delay before it is presented for flag(sync) comparison. In bisync mode, after the synchronization is achieved, the incoming data bypasses the sync register and enters directly into the three bit buffer on its way to receive shift register. In SDLC mode, the incoming data always passes through the sync register where data pattern is continuously monitored for contiguous ones for zero deletion logic. The data then enters the three bit buffer and the receive shift register. From the receive shift register, the data is transferred to the three byte deep FIFO. The data is transferred to the top of the FIFO at the chip clock rate (not the receiver clock). It takes three chip clock/periods to transfer data from the serial shift register to the top of the FIFO. The three bit deep Receive Error FIFO shifts any error condition which may have occurred during a frame reception. While all this is happening, the CRC checker is checking the CRC on the incoming data. The computed CRC is checked with the CRC bytes attached to the incoming frame and an error generated under a no-check condition. Note that the bisync data is presented to the CRC checker with an 8-bit delay. This is necessary to achieve transparency in bisync mode as will be shown later in this Application Note.

![Diagram of MPSC Interrupt Structure](image-url)

Figure 7. MPSC Interrupt Structure
MULTI-PROTOCOL SERIAL CONTROLLER (MPSC) INTERRUPT STRUCTURE

The MPSC offers a very powerful interrupt structure, which helps in responding to an interrupt condition very quickly. There are multiple sources of interrupts within the MPSC. However, the MPSC resolves the priority between various interrupting sources and interrupts the CPU for service through the interrupt line. This section presents a comprehensive discussion on all the 8274 interrupts and the priority resolution between these interrupts.

All the sources of interrupts on the 8274 can be grouped into three distinct categories. (See Figure 7)

1. Receive Interrupts
2. Transmit Interrupts
3. External/Status Interrupts.

An internal interrupt priority structure sets the priority between the interrupts. There are two programmable options available on the MPSC. The priority is set by WR2A, D2. (Figure 8)

![Figure 8. Interrupt Priority](image)

<table>
<thead>
<tr>
<th>PRIORITY</th>
<th>WR2A:D2</th>
<th>Highest</th>
<th>Lowest</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RXA</td>
<td>TxA</td>
<td>RXB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TxB</td>
<td>EXT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXTB</td>
</tr>
<tr>
<td>1</td>
<td>RXA</td>
<td>TxA</td>
<td>RXB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TxB</td>
<td>EXT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXTB</td>
</tr>
</tbody>
</table>

Receive Interrupt

All receive interrupts may be categorized into two distinct groups: Receive Interrupt on Receive Character and Special Receive Condition Interrupts.

RECEIVE INTERRUPT ON RECEIVE CHARACTER

A receive interrupt is generated when a character is received by the MPSC. However, as will be discussed later, this is a programmable feature on the MPSC. A Rx character available interrupt is generated by the MPSC after the receive character has been assembled by the MPSC. It may be noted that in DMA transfer mode too, a receive interrupt on the first receive character should be programmed. In SDLC mode, if address search mode has been programmed, this interrupt will be generated only after a valid address match has occurred. In monosync mode, this interrupt is generated on receipt of a character after at least two valid sync characters. In monosync mode, a character followed by at least a single valid sync character will generate this interrupt. An interrupt on first receive character signifies the beginning of a valid frame. An end of the frame is characterized by an “End of Frame” Interrupt (RR1: D7).* This bit (RR1:D7) is set in SDLC/HDLC mode only and signifies that a valid ending flag (7EH) has been received. This bit gets reset either by an “Error Reset” command (WR0: D5D4D3 = 110) or upon reception of the first character of the next frame. In multiframe reception, on receiving the interrupt at the “End of Frame” the CPU may issue an Error Reset command which will reset the interrupt. In DMA mode, the interrupt on first receive character is accompanied by a RxDREQ (Receiver DMA request) on the appropriate channel. At the end of the frame, an End of Frame interrupt is generated. The CPU may use this interrupt to jump into a routine which may redefine the receive buffer for the next incoming frame.

*SRR1:D7 is bit D7 in Read Register 1.

SPECIAL RECEIVE CONDITION INTERRUPTS

So far, we have assumed that the reception is error free. But this is not a ‘typical’ case in most real-life applications. Any error condition during a frame reception generates yet another interrupt — special receive condition interrupt. There are four different error conditions which can generate this interrupt.

(i) Parity error
(ii) Receive Overrun error
(iii) Framing error
(iv) End of Frame

(i) Parity error: Parity error is encountered in asynchronous (start-stop bits) and in bisync/monosync protocols. Both odd or even parity can be programmed. A parity error in a received byte will generate a special receive condition interrupt and sets bit 4 in RR1.

(ii) Receive Overrun error: If the CPU or the DMA controller (in DMA mode) fails to read a received character within three byte times after the received character interrupt (or DMA request) was generated, the receiver buffer will overflow and this will generate a special receive condition interrupt and sets bit 5 in RR1.

(iii) Framing error: In asynchronous mode, a framing error will generate a special receive interrupt and set bit D6 in RR1. This bit is not latched and is updated on the next received character.

(iv) End of frame: This interrupt is encountered in SDLC/HDLC mode only. When the MPSC receives the closing flag, it generates the special receive condition interrupt and sets bit D7 in RR1.

All the special receive condition interrupts may be reset by issuing an Error Reset Command.

CRC Error: In SDLC/HDLC and synchronous modes, a CRC error is indicated by bit D6 in RR1. When used to check CRC error, this bit is normally set until a correct CRC match is obtained which resets this bit. After receiving a frame, the CPU must read this bit (RR1:D6) to determine if a valid CRC check had occurred. It may be noted that a CRC error does not generate an interrupt.
It may be also be pointed out that in SDLC/HDLC mode, receive DMA requests are disabled by a special receive condition and can only be re-enabled by issuing an Error Reset Command.

**Transmit Interrupt**

A transmit buffer empty generates a transmit interrupt. This has been discussed earlier under “Transmit in Interrupt Mode” and it would be sufficient to note here that a transmit buffer empty interrupt is generated only when the transmit buffer gets empty — assuming it had a data character loaded into it earlier. This is why on starting a frame transmission, the first data character is loaded by the CPU without a transmit empty interrupt (or DMA request in DMA mode). After this character is loaded into the serial shift register, the buffer becomes empty, and an interrupt (or DMA request) is generated. This interrupt is reset by a “Reset Tx Interrupt/DMA Pending” command (WR0: D5 D4 D3 = 101).

**External/Status Interrupt**

Continuing our discussion on transmit interrupt, if the transmit buffer is empty and the transmit serial shift register also becomes empty (due to the data character shifted out of the MPSC), a transmit under-run interrupt will be generated. This interrupt may be reset by “Reset/External Status Interrupt” command (WR0: D5 D4 D3 = 101).

The External Status Interrupt can be caused by five different conditions:

(i) DCD Transition
(ii) CTS Transition
(iii) Sync/Hunt Transition
(iv) Tx under-run/EOM condition
(v) Break/Abort Detection.

**DCD,CTS TRANSITION**

Any transition on these inputs on the serial interface will generate an External/Status interrupt and set the corresponding bits in status register RR0. This interrupt will also be generated in DMA as well as in Wait Mode. In order to find out the state of the CTS or DCD pins before the transition had occurred, RR0 must be read before issuing a Reset External/Status Command through WR0. A read of RR0 after the Reset External/Status Command will give the condition of CTS or DCD pins after the transition had occurred. Note that bit D5 in RR0 gives the complement of the state of CTS pin while D3 in RR0 reflects the actual state of the DCD pin.

**SYNC HUNT TRANSITION**

Any transition on the SYNDET input generates an interrupt. However, sync input has different functions in different modes and we shall discuss them individually.

**SDLC Mode**

In SDLC mode, the SYNDET pin is an output. Status register RR1, D4 contains the state of the SYNDET pin. The Enter Hunt Mode initially sets this bit in R0. An opening flag in a received SDLC frame resets this bit and generates an external status interrupt. Every time the receiver is enabled or the Enter Hunt Code Command is issued, an external status interrupt will be generated on receiving a valid flag followed by a valid address/data character. This interrupt may be reset by the “Reset External Status Interrupt” command.

**External SYNC Mode**

The MPSC can be programmed into External Sync Mode by setting WR4, D5 D4 = 11. The SYNDET pin is an input in this case and must be held high until an external character synchronization is established. However, the External Sync mode is enabled by the Enter Hunt Mode control bit (WR3: D4). A high at the SYNDET pin holds the sync/Hunt bit (RR0,D4) in the reset state. When external synchronization is established, SYNDET must be driven low on second rising edge of RxC after the rising edge of RxC on which the last bit of sync character was received. This high to low transition sets the Sync/Hunt bit and generates an external status interrupt, which must be reset by the Reset External/Status command. If the SYNDET input goes high again, another External Status Interrupt is generated, which may be cleared by Reset External Status command.

**Mono-Sync/Bisync Mode**

SYNDET pin acts as an output in this case. The Enter Hunt Mode sets the Sync/Hunt bit in R0. Sync/Hunt bit is reset when the MPSC achieves character synchronization. This high to low transition will generate an external status interrupt. The SYNDET pin goes active every time a sync pattern is detected in the data stream. Once again, the external status interrupt may be reset by the Reset External Status command.

**Tx UNDER-RUN/END OF MESSAGE (EOM)**

The transmitter logic includes a transmit buffer and a transmit serial shift register. The CPU loads the character into the transmit buffer which is transferred into the transmit shift register to be shifted out of the MPSC. If the transmit buffer gets empty, a transmit buffer empty interrupt is generated (as discussed earlier). However, if the transmit buffer gets empty and the serial shift register gets empty, a transmit under-run condition will be created. This generates an External Status Interrupt and the interrupt can be cleared by the Reset External Status command. The status register RR0, D6 bit is set when the transmitter under-runs. This bit plays an important role in controlling a transmit operation, as will be discussed later in this application note.
BREAK/ABORT DETECTION

In asynchronous mode, bit D7 in RR0 is set when a break condition is detected on the receive data line. This also generates an External/Status interrupt which may be reset by issuing a Reset External/Status Interrupt command to the MPSC. Bit D7 in RR0 is reset when the break condition is terminated on the receive data line and this causes another External/Status interrupt to be generated. Again, a Reset External/Status Interrupt command will reset this interrupt and will enable the break detection logic to look for the next break sequence.

In SDLC Receive Mode, an Abort sequence (seven or more 1's) detection on the receive data line will generate an External/Status interrupt and set RR0,D7. A Reset External/Status command will clear this interrupt. However, a termination of the Abort sequence will generate another interrupt and set RR0, D7 again. Once again, it may be cleared by issuing Reset External/Status Command.

This concludes our discussion on External Status Interrupts.

Interrupt Priority Resolution

The internal interrupt priority between various interrupt sources is resolved by an internal priority logic circuit, according to the priority set in WR2A. We will now discuss the interrupt timings during the priority resolution. Figures 9 and 10 show the timing diagrams for vectored and non-vectored modes.

VECTORED MODE

We shall assume that the MPSC accepted an internal request for an interrupt by activating the internal INT signal. This leads to generating an external interrupt signal on the INT pin. The CPU responds with an interrupt acknowledge (INTA) sequence. The leading edge of the first INTA pulse sets an internal interrupt acknowledge signal (we will call it Internal INTA). Internal INTA is reset by the high going edge of the third INTA pulse. The MPSC will not accept any internal requests for an interrupt during the period when Internal INTA is active (high). The MPSC resolves the priority during various existing internal interrupt requests during the Interrupt Request Priority Resolve Time, which is defined as the time between the leading edge of the first INTA and the leading edge of the second INTA from the CPU. Once the internal priorities have been resolved, an internal Interrupt-in-Service Latch is set. The external INT is also deactivated when the Interrupt-in-Service Latch is set.

The lower priority interrupt requests are not accepted internally until an EOI (WR0: D5 D4 D3 = 111) command is issued by the CPU. The EOI command enables the lower priority interrupts. However, a higher priority interrupt

Figure 9. 8274 in 8085 Vectored Mode Priority Resolution Time

---

210403-001
request will still be accepted (except during the period when internal INTA is active) even though the Internal-in-Service Latch is set. This higher priority request will generate another external INT and will have to be handled by the CPU according to how the CPU is set up. If the CPU is set up to respond to this interrupt, a new INTA cycle will be repeated as discussed earlier. It may also be noted that a transmitter buffer empty and receive character available interrupts are cleared by loading a character into the MPSC and by reading the character received by the MPSC respectively.

NON-VECTORED MODE

Figure 10 shows the timing of interrupt sequence in non-vectored mode. The explanation for non-vectored is similar to the vector mode, except for the following exceptions.

- No internal priority requests are accepted during the time when pointer 2 for Channel B is specified.
- The interrupt request priority resolution time is the time between the leading edge of pointer 2 and leading edge of RD active. It may be pointed out that in non-vectored mode, it is assumed that the status affects vector mode is used to expedite interrupt response.

On getting an interrupt in non-vectored mode, the CPU must read status register RR2 to find out the cause of the interrupt. In order to do so, first a pointer to status register RR2 is specified and then the status read from RR2. It may be noted here that after specifying the pointer, the CPU must read status register RR2 otherwise, no new interrupt requests will be accepted internally.

Just like the vectored mode, no lower internal priority requests are accepted until an EOI command is issued by the CPU. A higher priority request can still interrupt the CPU (except during the priority request inhibit time). It is important to note here that if the CPU does not perform a read operation after specifying the pointer 2 for Channel B, the interrupt request accepted before the pointer 2 was activated will remain valid and no other request (high or low priority) will be accepted internally. In order to complete a correct priority resolution, it is advised that a read operation be done after specifying the pointer 2B.

IPI and IPO

So far, we have ignored the IPI and IPO signals shown in Figures 9 and 10. We may recall that IPI is the Interrupt-Priority-Input to the MPSC. In conjunction with the IPO (Interrupt Priority Output), it is used to daisy chain multiple MPSC's. MPSC daisy chaining will be discussed in detail later in this application note.
EOI Command

The EOI command as explained earlier, enables the lower priority interrupts by resetting the internal In-Service-Latch, which consequently resets the IPO output to a low state. See Figures 9 and 10 for details. Note that before issuing any EOI command, the internal interrupting source must be satisfied otherwise, same source will interrupt again. The Internal Interrupt is the signal which gets reset when the internal interrupting source is satisfied (see Figure 9).

This concludes our discussion on the MPSC Interrupt Structure.

MULTI-PROTOCOL SERIAL CONTROLLER (MPSC) MODES OF OPERATION

The MPSC provides two fully independent channels that may be configured in various modes of operations. Each channel can be configured into full duplex mode and may operate in a mode or protocol different from the other channel. This feature will be very efficient in an application which requires service. In the following discussion, we will discuss how to transmit and receive in interrupt driven mode.

TRANSMIT IN INTERRUPT MODE

The MPSC can be configured into interrupt mode by appropriately setting the bits in WR2 A (Write Register 2, Channel A). Figure 11 shows the modes of operation.

<table>
<thead>
<tr>
<th>WR2A</th>
<th>MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>D0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 11. MPSC Mode Selection for Channel A and Channel B.

We will limit our discussion to SDLC transmit and receive only. However, exceptions for other synchronous protocols will be pointed out. To initiate a frame transmission, the first data character must be loaded from the CPU, in all cases. (DMA Mode too, as you will notice later in this application note). Note that in SDLC mode, this first data character may be the address of the station addressed by the MPSC. The transmit buffer consists of a transmit buffer and a serial shift register. When the character is transferred from the buffer into the serial shift register, an interrupt due to transmit buffer empty is generated. The CPU has one byte time to service this interrupt and load another character into the transmitter buffer. The MPSC will generate an interrupt due to transmit buffer under-run condition if the CPU does not service the Transmit Buffer Empty Interrupt within one byte time.

This process will continue until the CPU is out of any more data characters to be sent. At this point, the CPU does not respond to the interrupt with a character but simply issues a Reset Tx INT/DMA pending command (WR0: D5 D4 D3 = 101). The MPSC will ultimately under-run, which simply means that both the transmit buffer and transmit shift registers are empty. At this point, flag character (7EH) or CRC byte is loaded into the transmit shift register. This sets the transmit under-run bit in RR0 and generates “Transmit Under-run/EOM” interrupt (RR0:D6 = 1). You will recall that an SDLC frame has two CRC bytes after the data field. 8274 generates the CRC on all the data that is loaded from the CPU. During initialization, there is a choice of selecting a CRC-16 or CCITT-CRC (WR5: D2). In SDLC/HDLC operation, CCITT-CRC must be selected. We will now see how the CRC gets inserted at the end of the data field. Here we have a choice of having the CRC attached to the data field or sending the frame without the CRC bytes. During transmission, a “Reset Tx Under-run/EOM Latch” command (WR0: D7 D6 = 11) will ensure that at the end of the frame when the transmitter underruns, CRC bytes will be automatically inserted at the end of the data field. If the “Reset Tx Under-run/EOM Latch” command was not issued during the transmission of data characters, no CRC would be inserted and the MPSC will transmit flags (7EH) instead.

However, in case of CRC transmission, the CRC transmission sets the Tx Under-run/EOM bit and generates a Transmitter Under-run/EOM Interrupt as discussed earlier. This will have to be reset in the next frame to ensure CRC insertion in the next frame. It is recommended that Tx Under-run/EOM latch be reset very early in the transmission mode, preferably after loading the first character. It may be noted here that Tx Under-run/EOM latch cannot be reset if there is no data in the transmit buffer. This means that at least one character has to be loaded into the MPSC before a “Reset Transmit Under-run/EOM Latch” command will be accepted by the MPSC.

When the transmitter is under-run, an interrupt is generated. This interrupt is generated at the beginning of the CRC transmission, thus giving the user enough time (minimum 22 transmit clock cycles) to issue an Abort.
command (WR0: D5 D4 D3 = 0 0 1) in case if the transmitted data had an error. The Abort Command will ensure that the MPSC transmits at least eight 1's but less than fourteen 1's before the line reverts to continuous flags. The receiver will scratch this frame because of bad CRC.

However, assuming the transmission was good (no Abort Command issued), after the CRC bytes have been transmitted, closing flag (7EH) is loaded into the transmit buffer. When the flag (7EH) byte is transferred to the serial shift register, a transmit buffer empty interrupt is generated. If another frame has to be transmitted, a new data character has to be loaded into the transmit buffer and the complete transmit sequence repeated. If no more frames are to be transmitted, a "Reset Transmit INT/DMA Pending" command (WR0: D5 D4 D3 = 1 0 1) will reset the transmit buffer empty interrupt.

For character oriented protocols (Bisync, Monosync), the same discussion is valid, except that during transmit under-run condition and transmit under-run/EOM bit in set state, instead of flags, filler sync characters are transmitted.

**CRC Generation:**

The transmit CRC enable bit (WR5: D0) must be set before loading any data into the MPSC. The CRC generator must be reset to all 1's at the beginning of each frame before CRC computation has begun. The CRC computation starts on the first data character loaded from the CPU and continues until the last data character. The CRC generated is inverted before it is sent on the Tx Data line.

**Transmit Termination:**

A successful transmission can be terminated by issuing a "Reset Transmit Interrupt/DMA Pending" command, as discussed earlier. However, the transmitter may be disabled any time during the transmission and the results will be as shown in Figure 12.

**RECEIVE IN INTERRUPT MODE**

The receiver has to be initialized into the appropriate receive mode (see sample program later in this application note). The receiver must be programmed into Hunt Mode (WR3: D4) before it is enabled (WR3: D0). The receiver will remain in the Hunt Mode until a flag (or sync character) is received. While in the SDLC/Bisync/Monosync mode, the receiver does not enter the Hunt Mode unless the Hunt bit (WR3, D4) is set again or the receiver is enabled again.

SDLC Address byte is stored in WR6. A global address (FFH) has been hardwired on the MPSC. In address search mode (WR3: D2 = 1), any frame with address matching with the address in WR6 will be received by the MPSC. Frames with global address (FFH) will also be received, irrespective of the condition of address search.

<table>
<thead>
<tr>
<th>Transmitter Disabled during</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Data Transmission</td>
<td>Tx Data will send idle characters* which will be zero inserted.</td>
</tr>
<tr>
<td>2. CRC Transmission</td>
<td>16 bit transmission, corresponding to 16 bits of CRC will be completed. However, flag bits will be substituted in the CRC field.</td>
</tr>
<tr>
<td>3. Immediately after issuing ABORT command.</td>
<td>Abort will still be transmitted — output will be in the mark state.</td>
</tr>
</tbody>
</table>

*Idle characters are defined as a string of 15 or more contiguous ones.

mode bit (WR3: D2). In general receive mode (WR3: D2=0), all frames will be received.

Since the MPSC only recognizes single byte address field, extended address recognition will have to be done by the CPU on the data passed on by the MPSC. If the first address byte is checked by the MPSC, and the CPU determines that the second address byte does not have the correct address field, it must set the Hunt Mode (WR3: D2 = 1) and the MPSC will start searching for a new address byte preceded by a flag.

Programmable Interrupts: The receiver may be programmed into any one of the four modes. See Figure 13 for details.

<table>
<thead>
<tr>
<th>WR1, CHA</th>
<th>Rx Interrupt Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>D4 D3</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>Rx INT/DMA disable</td>
</tr>
<tr>
<td>0 1</td>
<td>Rx INT on first character</td>
</tr>
<tr>
<td>1 0</td>
<td>INT on all Rx characters (Parity affects vector)</td>
</tr>
<tr>
<td>1 1</td>
<td>INT on all Rx characters (Parity does not affect vector)</td>
</tr>
</tbody>
</table>

**Figure 12. Transmitter Disabled During Transmission**

**Figure 13. Receiver Interrupt Modes**

All receiver interrupts can be disabled by WR1: D4 D3 = 0 0. Receiver interrupt on first character is normally used to start a DMA transfer or a block transfer sequence using WAIT to synchronize the data transfer to received or transmitted data.

**External Status Interrupts:**

Any change in DCD input or Abort detection in the received data, will generate an interrupt if External Status Interrupt was enabled (WR1: D0).
Special Receive Conditions:
The receiver buffer is quadruply buffered. If the CPU fails to respond to "receive character" available interrupt within a period of three byte times (received bytes), the receiver buffer will overflow and generate an interrupt. Finally, at the end of the received frame, an interrupt will be generated when a valid ending flag has been detected.

Receive Character Length:
The receive character length (6, 7 or 8 bits/character) may be changed during reception. However, to ensure that the change is effective on the next received character, this must be done fast enough such that the bits specified for the next character have not been assembled.

CRC Checking:
The opening flag in the frame resets the receive CRC generator and any field between the opening and closing flag is checked for the CRC. In case of a CRC error, the CRC/Framing Error bit in status register I is set (RRI: D6=1). Receiver CRC may be disabled/enabled by WR3,D3. The CRC bytes on the received frame are passed on to the CPU just like data, and may be discarded by the CPU.

Receive Terminator:
An end of frame is indicated by End of Frame interrupt. The CPU may issue an "Error Reset" command to reset this interrupt.

DMA (Direct Memory Access) Mode
The 8274 can be interfaced directly to the Intel DMA Controllers 8237A, 8257A and Intel I/O Processor 8089. The 8274 can be programmed into DMA mode by setting appropriate bits in WR2A. See Figure 11 for details.

TRANSMIT IN DMA MODE:
After initializing the 8274 into the DMA mode, the first character must be loaded from the CPU to start the DMA cycle. When the first data character (may be the address byte in SDLC) is transferred from the transmit buffer to the transmit serial shift register, the transmit buffer gets empty and a transmit DMA request (TxDRQ) is generated for the channel. Just like the interrupt mode, to ensure that the CRC bytes are included in the frame, the transmit under-run/EOM latch must be reset. This should preferably be done after loading the first character from the CPU. The DMA will progress without any CPU intervention. When the DMA controller reaches the terminal count, it will not respond to the DMA request, thus letting the MPSC under-run. This will ensure CRC transmission. However, the under-run condition will generate an interrupt due to the Tx under-run/EOM bit getting set (RR0: D6). The CPU should issue a "Reset TxInt/DRQ pending" command to reset TxDRQ and issue a "Reset External Status" command to reset Tx Under-run/EOM interrupt. Following the CRC transmission, flag (7EH) will be loaded into the transmit buffer. This will also generate the TxDRQ since the transmit buffer is empty following the transmission of the CRC bytes. The CPU may issue a "Reset TxINT/DRQ pending" command to reset the TxDRQ. "Reset TxINT/DRQ pending" command must be issued before setting up the transmit DMA channel on the DMA Controller, otherwise the MPSC will start the DMA transfer immediately after the DMA channel is set up.

RECEIVE IN DMA MODE
The receiver must be programmed in RxINT on first receive character mode (WR1: D4 D3 = 0 1). Upon receiving the first character, which may be the address byte in SDLC, the MPSC generates an interrupt and also generates a Rx DMA Request (Rx DRQ) for the appropriate channel. The CPU has three byte times to service this interrupt (enable the DMA controller, etc.) before the receiver buffer will overflow. It is advisable to initialize the DMA controller before receiving the first character. In case of high bit rates, the CPU will have to service the interrupt very fast in order to avoid receiver over-run.

Once the DMA is enabled, the received data is transferred to the memory under DMA control. Any received error conditions or external status change condition will generate an interrupt as in the interrupt driven mode. The End of Frame is indicated by the End of Frame interrupt which is generated on reception of the closing flag of the SDLC frame. This End of Frame condition also disables the Receive DMA request. The End of Frame interrupt may be reset by issuing an "Error Reset" command to the MPSC. The "Error Reset" command also re-enables the Receive DMA request. It may be noted that the End of Frame condition sets bits D7 in RR1. This bit gets reset by "Error Reset" command. However, End of Frame bit (RR1:D7) can also be reset by the flag of the next incoming frame. For proper operation, Error Reset Command should be issued "after" the End of Frame Bit (RR1:D7) is set. In a more general case, "Error Reset" command should be issued after End of Frame, Receive over-run or Receive parity bit are set in RR1.

Wait Mode
The wait mode is normally used for block transfer by synchronizing the data transfer through the Ready output from the MPSC, which may be connected to the Ready input of the CPU. The mode can be programmed by WR 1, D7 D5 and may be programmed separately and independently on CH A and CH B. The Wait Mode will be operative if the following conditions are satisfied.
(i) Interrupts are enabled.
(ii) Wait Mode is enabled (WR1: D7)
(iii) CS = 0, AI = 0

The RDY output becomes active when the transmitter buffer is full or receiver buffer is empty. This way the RDY output from the MPSC can be used to extend the CPU read and write cycle by inserting WAIT states. RDY_A or RDY_B are in high impedance state when the corresponding channel is not selected. This makes it possible to connect RDY_A and RDY_B outputs in wired OR configuration. Caution must be exercised here in using the ROY outputs of the MPSC or else the CPU may hang up for indefinite period. For example, let us assume that transmitter buffer is full and ROY_A is active, forcing the CPU into a wait state. If the CTS goes inactive during this period, the RDY_A will remain active for indefinite period and CPU will continue to insert wait states.

**Vectored/Non-Vectored Mode**

The MPSC is capable of providing an interrupt vector in response to the interrupt acknowledge sequence from the CPU. WR2, CH B contains this vector and the vector can be read in status register RR2. WR2, CH A (bit D5) can program the MPSC in vectored or non-vectored mode. See Figure 14 for details.

In both cases, WR2 may still have the vector stored in it. However, in vectored mode, the MPSC will put the vector on the data bus in response the INTA (Interrupt Acknowledge) sequence as shown in Figure 15. In non-vectored mode, the MPSC will not respond to the INTA sequence. However, the CPU can read the vector by polling Status Register RR2. WR2A, D4 and D3 can be programmed to respond to 8085 or 8086 INTA sequence. It may be noted here that IPI (Interrupt Priority In) pin on the MPSC must be active for the vector to appear on the data bus.

**Status Affect Vector**

The vector stored in WR2B can be modified by the source of the interrupt. This can be done by setting the Status Affect Vector bit (WR1: D2). This powerful feature of the MPSC provides fast interrupt response time, by eliminating the need of writing a routine to read the status of the MPSC. Three bits of the vector are modified in two different ways as shown on Figure 16. Bits V4,V3,V2 are modified in 8085 based system and bits V2, V1, V0 are modified in 8086/88 based system.

In non-vectored mode, the status affect vector mode can still be used and the vector read by the CPU. Status Register RR2B (Read Register 2 in Channel B) will contain this modified vector.

### Table: Interrupt Mode

<table>
<thead>
<tr>
<th>WR2A,D5</th>
<th>Interrupt Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-vectored Interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Vectored Interrupt</td>
</tr>
</tbody>
</table>

**Figure 14. Vectored Interrupts**

The vector stored in WR2B can be modified by the source of the interrupt. This can be done by setting the Status Affect Vector bit (WR1: D2). This powerful feature of the MPSC provides fast interrupt response time, by eliminating the need of writing a routine to read the status of the MPSC. Three bits of the vector are modified in two different ways as shown on Figure 16. Bits V4,V3,V2 are modified in 8085 based system and bits V2, V1, V0 are modified in 8086/88 based system.

### Figure 15. MPSC Vectored Interrupts

<table>
<thead>
<tr>
<th>WR2A</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>IPI</th>
<th>MODE</th>
<th>1ST INTA</th>
<th>2ND INTA</th>
<th>3RD INTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>NON-VECTORED</td>
<td>HIGH-Z</td>
<td>V7 V6 V5 V4 V3 V2 V1 V0</td>
<td>HI-Z</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8085-1</td>
<td>1100 1101</td>
<td>HI-Z</td>
<td>0000 0000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8085-1</td>
<td>1100 1101</td>
<td>HI-Z</td>
<td>0000 0000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8085-2</td>
<td>HI-Z</td>
<td>V7 V6 V5 V4 V3 V2 V1 V0</td>
<td>HI-Z</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8086</td>
<td>HI-Z</td>
<td>V7 V6 V5 V4 V3 V2 V1 V0</td>
<td>HI-Z</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8086</td>
<td>HI-Z</td>
<td>HI-Z</td>
<td>HI-Z</td>
</tr>
</tbody>
</table>

**Figure 16. Status Affect Vector Mode**

<table>
<thead>
<tr>
<th>(8085) V4</th>
<th>V3</th>
<th>V2</th>
<th>V1</th>
<th>V0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Rx Special Condition: Parity Error, Framing Error, Rx Over-run Error, EOF (SDLC)

EXT/STAT Change: Change in Modem Control Pin Status: CTS, DCD, SYNC, EOM, Break/Abort Detection

---

6-241

210403-001
APPLICATION EXAMPLE

This section describes the hardware and software of an 8274/8088 system. The hardware vehicle used is the INTEL Single Board Computer iSBC 88/45 - Advanced Communication Controller. The software which exercises the 8274 is written in PLM 86. This example will demonstrate how 8274 can be configured into the SDLC mode and transfer data through DMA control. The hardware example will help the reader configure his hardware and the software examples will help in developing an application software. Most software examples closely approximate a real data link controller software in the SDLC communication and may be used with very little modification.

iSBC® 88/45

A brief description of the iSBC 88/45 board will be presented here. For more detailed information on the board and the schematics, refer to Hardware Manual for the iSBC 88/45, Advanced Communication Controller. iSBC 88/45 is an intelligent slave/multimaster communication board based on the 8088 processor, the 8274 and the 8273 SDLC/HDLC controller. Figure 17 shows the functional block diagram of the board. The iSBC 88/45 has the following features.

- 8 MHz processor
- 16K bytes of static RAM (12K dual port)
- Multimaster/Intelligent Slave Multibus Interface
- Nine Interrupt Levels 8259A
- Two serial channels through 8274
- One Serial channel through 8273
- S/W programmable baud rate generator
- Interfaces: RS 232, RS422/449, CCITT V.24
- 8237A DMA controller
- Baud Rate to 800k Baud
INITIALIZE_8274:PROCEDURE PUBLIC;

/****************************************************************************
** INITIALIZE THE 8274 FOR SDLC MODE
****************************************************************************

1. RESET CHANNEL
2. EXTERNAL INTERRUPTS ENABLED
3. NO WAIT
4. PIN 10 = RTS
5. NON-VECTORED INTERRUPT-8086 MODE
6. CHANNEL A DMA, CH B INT
7. TX AND RX = 8 BITS/CHAR
8. ADDRESS SEARCH MODE
9. CD AND CTS AUTO ENABLE
10. TX AND RX = 8 BITS/CHAR
11. X1 CLOCK
12. NO PARITY
13. SDLC/HDLC MODE
14. RTS AND DTR
15. CCITT - CRC
16. TRANSMITTER AND RECEIVER ENABLED
17. 7EH = FLAG

DECLARE C BYTE;

DECLARE TABLE_74_A(*1) BYTE DATA
(00H.1BH, /* CHANNEL RESET */
  00H.80H, /* RESET TX CRC */
  02H.11H, /* PIN 10=RTS, A DMA, B INT */
  04H.20H, /* SDLC/H DLC MODE, NO PARITY */
  07H.07EH, /* SDLC FLAG */
  01H.08H, /* RX DMA ENABLE */
  05H.0EBH, /* DTR, RTS, B TX Bits, TX ENAB */
  06H.55H, /* DEFAULT ADDRESS */
  03H.0D9H, /* B RX Bits, AUTO ENABLES, HUNT Mode */
  OFFH), /* END OF INITIALIZATION TABLE */

DECLARE TABLE_74_B(*1) BYTE DATA
(02H.00H, /* INTERRUPT VECTOR */
  01H.1CH, /* STATUS AFFECTS VECTOR */
  OFFH), /* END */

/* INITIALIZE THE 8274 */

C=0;
DO WHILE TABLE_74_B(C) <> OFFH;
  OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
  C=C+1;
  OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
  C=C+1;
END;

C=0;
DO WHILE TABLE_74_A(C) <> OFFH;
  OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
  C=C+1;
  OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
  C=C+1;
END;
RETURN;
END INITIALIZE_8274;

Figure 18. Typical MPSC SDLC Initialization Sequence
Interrupt\) as the name

The application example is interrupt driven and uses DMA for all data transfers under 8237A control. The 8254 provides the transmit and receive clocks for the 8274. The 8274 was run at 400K baud with a local loopback (jumper wire) on Channel A data. The board was also run at 800K baud by modifying the software as will be discussed later in the Special Applications section. One detail to note is that the Rx Channel DMA request line from the 8274 has higher priority than the Tx Channel DMA request line. The 8274 master clock was 4.0 MHz. The on-board RAM is used to define transmit and receive data buffers. In this application, the data is read from memory location 800H through 810H and transferred to memory location 900H to 910H through the 8274 Serial Link. The operation is full duplex. 8274 modem control pins, CTS and CD have been tied low (active).

Software

The software consists of a monitor program and a program to exercise the 8274 in the SDLC mode. Appendix A contains the entire program listing. For the sake of clarity, each source module has been rewritten in a simple language and will be discussed here individually. Note that some labels in the actual listings in the Appendix will not match with the labels here. Also the listing in the Appendix sets up some flags to communicate with the monitor. Some of these flags are not explained in detail for the reason that they are not pertinent to this discussion. The monitor takes the command from a keyboard and executes this program, logging any error condition which might occur.

8274 Initialization

The MPSC is initialized in the SDLC mode for Channel A. Channel B is disabled. See Figure 18 for the initialization routine. Note that WR4 is initialized before setting up the transmitter and receive parameters. However, it may also be pointed out that other than WR4, all the other registers may be programmed in any order. Also SDLC-CRC has been programmed for correct operation. An incorrect CRC selection will result in incorrect operation. Also note that receive interrupt on first receive character has been programmed although Channel A is in the DMA mode.

Interrupt Routines

The 8274 interrupt routines will be discussed here. On an 8274 interrupt, program branches off to the “Main Interrupt Routine”. In main interrupt routine, status register RR2 is read. RR2 contains the modified vector. The cause of the interrupt is determined by reading the modified bits of the vector. Note that the 8274 has been programmed in the non-vectored mode and status affects vector bit has been set. Depending on the value of the modified bits, the appropriate interrupt routine is called. See Figure 19 for the flow diagram and Figure 20 for the source code. Note that an End of Interrupt Command is issued after servicing the interrupt. This is necessary to enable the lower priority interrupts.

Figure 21 shows all the interrupt routines called by the Main Interrupt Routine. “Ignore - Interrupt” as the name implies, ignores any interrupts and sets the FAIL flag. This is done because this program is for Channel A only and we are ignoring any Channel B interrupts. The important thing to note is the Channel A Receiver Character available routine. This routine is called after receiving the first character in the SDLC frame. Since the transfer mode is DMA, we have a maximum of three character times to service this interrupt by enabling the DMA controller.

![Figure 19. Interrupt Response Flow Diagram](image-url)
Figure 20. Typical Main Interrupt Routine

```c
/*********************/
/* MAIN INTERRUPT ROUTINE */
/*********************/
OUTPUT(COMMAND_B_74) = 2; /* SET POINTER TO 2*/
TEMP = INPUTSTATUS_B_74) AND 07H; /* READ INTERRUPT VECTOR */
/* CHECK FOR CHA INT ONLY*/
/* FOR THIS APPLICATION CH B INTERRUPTS ARE IGNORED*/
DO CASE TEMP,
    CALL IGNORE_INT; /* V2Vivo = 000*/
    CALL IGNORE_INT; /* V2Vivo = 001*/
    CALL CHB_RX_CHAR; /* V2Vivo = 010*/
    CALL IGNORE_INT; /* V2Vivo = 011*/
    CALL CHA_EXTERNAL_CHANGE; /* V2Vivo = 100*/
    CALL CHA_RX_CHAR; /* V2Vivo = 110*/
    CALL CHA_RX_SPECIAL; /* V2Vivo = 111*/
END;
OUTPUT(COMMAND_A_74) = 38H; /* END OF INTERRUPT FOR 8274 */
RETURN;
END INTERRUPT_8274;
```

Figure 21. 8274 Typical Interrupt Handling Routines

```c
/*********************/
/* CHANNEL A EXTERNAL/STATUS CHANGE INTERRUPT HANDLER */
/*********************/
CHA_EXTERNAL_CHANGE: PROCEDURE;
TEMP = INPUTSTATUS_A_74); /* STATUS REQ 1*/
IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
    TXDONE_S = DONE;
ELSE DO;
    TXDONE_S = DONE;
    RESULTS_S = FAIL;
END;
OUTPUT(COMMAND_A_74) = 10H; /* RESET EXT/STATUS INTERRUPTS */
RETURN;
END CHA_EXTERNAL_CHANGE;
/*********************/
/* CHANNEL A SPECIAL RECEIVE CONDITIONS INTERRUPT HANDLER */
/*********************/
CHA_RX_SPECIAL: PROCEDURE;
OUTPUT(COMMAND_A_74) = 1;
TEMP = INPUTSTATUS_A_74),
IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
    DO;
        IF (TEMP AND 040H) = 040H THEN
            RESULTS_S = FAIL; /* CRC ERROR */
            RXDONE_S = DONE;
            OUTPUT(COMMAND_A_74) = 30H; /* ERROR RESET*/
        END;
    ELSE DO;
        IF (TEMP AND 20H) = 20H THEN DO;
            RESULTS_S = FAIL; /* RX OVERRUN ERROR*/
            RXDONE_S = DONE;
            OUTPUT(COMMAND_A_74) = 30H; /* ERROR RESET*/
        END;
    END;
END;
RETURN;
END CHA_RX_SPECIAL;
/*********************/
/* CHANNEL A RECEIVE CHARACTER AVAILABLE */
/*********************/
CHA_RX_CHAR: PROCEDURE;
OUTPUT(SINGLE_MASK) = CHO_SEL; /* ENABLE RX DMA CHANNEL*/
RETURN;
END CHA_RX_CHAR;
```
It may be recalled that the receiver buffer is three bytes deep in addition to the receiver shift register. At very high data rates, it may not be possible to have enough time to read RR2, enable the DMA controller without overrunning the receiver. In a case like this, the DMA controller may be left enabled before receiving the Receive Character Interrupt. Remember, the Rx DMA request and interrupt for the receive character appear at the same time. If the DMA controller is enabled, it would service the DMA request by reading the received character. This will make the 8274 interrupt line go inactive. However, the 8259A has latched the interrupt and a regular interrupt acknowledge sequence still occurs after the DMA controller has completed the transfer and given up the bus. The 8259A will return Level 7 interrupt since the 8274 interrupt has gone away. The user software must take this into account, otherwise the CPU will hang up.

The procedure shown for the Special Receive Condition Interrupt checks if the interrupt is due to the End of Frame. If this is not TRUE, the FAIL flag is set and the program aborted. For a real life system, this must be followed up by error recovery procedures which obviously are beyond the scope of this Application Note.

The transmission is terminated when the End of Message (RR0, D6) interrupt is generated. This interrupt is serviced in the Channel A External/Status Change interrupt procedure. For any other change in external status conditions, the program is aborted and a FAIL flag set.

**Main Program**

Finally, we will briefly discuss the main program. Figure 22 shows the source program. It may be noted that the Transmit Under-run latch is reset after loading the first character into the 8274. This is done to ensure CRC transmission at the end of the frame. Also, the first character is loaded from the CPU to start DMA transfer of subsequent data. This concludes our discussion on hardware and software example. Appendix A also includes the software written to exercise the 8274 in the vectored mode by disabling the 8259A.

```
CH_A_SDLC_TEST: PROCEDURE BYTE PUBLIC;
    CALL ENABLE_INTERRUPTS_S;
    CALL INIT_8274_SDLC_S;
    ENABLE;
    OUTPUT(COMMAND_A_74) = 28H;  /* RESET TX INT/DMA */
    OUTPUT(COMMAND_B_74) = 28H;  /* BEFORE INITIALIZING 8237*/
    CALL INIT_8237_S;
    OUTPUT(DATA_A_74) = 55H;  /*LOAD FIRST CHARACTER FROM*/
    /* CPU */
    OUTPUT(COMMAND_A_74) = 28H;  /* TO ENSURE CRC TRANSMISSION. RESET TX UNDERRUN LATCH */
    RXDONE_S.TXDONE_S=NOT_DONE;  /* CLEAR ALL FLAGS */
    RESULTS_S=PASS;  /* FLAG SET FOR MONITOR */
    DO WHILE TXDONE_S=NOT_DONE;  /* DO UNTIL TERMINAL COUNT */
        END;
    DO WHILE INPUT(STATUS_A_74) AND 04H) <> 04H;
        /* WAIT FOR CRC TO GET TRANSMITTED */
        /* TEST FOR TX BUFFER EMPTY TO VERIFY THIS*/
        END;
    DO WHILE RXDONE_S=NOT_DONE;  /* DO UNTIL TERMINAL COUNT */
        END;
    CALL STOP_8237_S;
END CHA_SDLC_TEST;
```

*Figure 22. Typical 8274 Transmit/Receive Set-Up in SDLC Mode*
SPECIAL APPLICATIONS

In this section, some special application issues will be discussed. This will be useful to a user who may be using a mode which is possible with the 8274 but not explicitly explained in the data sheet.

MPSC Daisy Chain Operation

Multiple MPSC can be connected in a daisy-chain configuration (see Figure 23). This feature may be useful in an application where multiple communication channels may be required and because of high data rates, conventional interrupt controller is not used to avoid long interrupt response times. To configure the MPSCs for the daisy chain operation, the interrupt priority input pins (IPI) and interrupt priority output pins (IPO) of the MPSC should be connected as shown. The highest priority device has its IPI pin connected to ground. Each MPSC is programmed in a vectored mode with status affects vector bit set. In the 8085 basic systems, only one MPSC should be programmed in 8085 Mode 1. This is the MPSC which will put the call vector (CD Hex) on the data bus in response to the first INTA pulse (See Figure 15). It may be pointed out that the MPSC in 8085 Mode 1 will provide the call vector irrespective of the state of IPI pin. Once a higher priority MPSC generates an interrupt, its IPO pin goes inactive thus preventing lower priority MPSCs from interrupting the CPU. Preferably the highest priority MPSC should be programmed in 8085 Mode 1. It may be recalled that the Priority Resolve Time on a given MPSC extends from the falling edge of the first INTA pulse to the falling edge of the second INTA pulse. During this period, no new internal interrupt requests are accepted. The maximum number of the MPSCs that can be connected in a daisy chain is limited by the Priority Resolution Time. Figure 24 shows a maximum number of MPSCs that can be connected in various CPU systems. It may be pointed out that IPO to IPI delay time specification is 100ns.

Bisync Transparent Communication

Bisync applications generally require that data transparency be established during communication. This requires that the special control characters may not be included in the CRC accumulation. Refer to the Synchronous Protocol Overview section for a more detailed discussion on data transparency. The 8274 can be used for transparent communication in Bisync communications. This is made possible by using the 8274 Daisy Chain Vectored Mode.

<table>
<thead>
<tr>
<th>System Configuration</th>
<th>Priority Resolution Time</th>
<th>Number of 8274s Daisy Chained</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min (ns)</td>
<td>(Max)</td>
</tr>
<tr>
<td>8086-1</td>
<td>400</td>
<td>4</td>
</tr>
<tr>
<td>8086-2</td>
<td>500</td>
<td>5</td>
</tr>
<tr>
<td>8086</td>
<td>800</td>
<td>8</td>
</tr>
<tr>
<td>8088</td>
<td>800</td>
<td>8</td>
</tr>
<tr>
<td>8085-2</td>
<td>1200</td>
<td>12</td>
</tr>
<tr>
<td>8085A</td>
<td>1920</td>
<td>19</td>
</tr>
</tbody>
</table>

Note: Zero wait states have been assumed.
possible by the capability of the MPSC to selectively turn-on/turn-off the CRC accumulation while transmitting or receiving. In bisync transparent transmit mode, the special characters (DLE, DLE SYN etc) are excluded from CRC calculation. This can be easily accomplished by turning off the transmit CRC calculation (WR5:D5=0) before loading the special character into the transmit buffer. If the next character is to be included in the CRC accumulation, then the CRC can be enabled (WR5: D5=1). See Figure 25 for a typical flow diagram.

During reception, it is possible to exclude received character from CRC calculation by turning off the Receive CRC after reading the special character. This is made possible by the fact that the received data is presented to receive CRC checker 8 bit times after the character has been received. During this 8 bit times, the CPU must read the character and decide if it wants it to be included in the CRC calculation. Figure 26 shows the typical flow diagram to achieve this.

It should be noted that the CRC generator must be enabled during CRC reception. Also, after reading the CRC bytes, two more characters (SYNC) must be read before checking for CRC check result in RRI.

**Auto Enable Mode**

In some data communication applications, it may be required to enable the transmitter or the receiver when the CTS or the DCD lines respectively, are activated by the modems. This may be done very easily by programming the 8274 into the Auto Enable Mode. The auto enable mode is set by writing a ‘1’ to WR3:D5. The function of this mode is to enable the transmitter automatically when CTS goes active. The receiver is enabled when DCD goes active. An in-active state of CTS or DCD pin will disable the transmitter or the receiver respectively. However, the Transmit Enable bit (WR5:D3) and Receive Enable bit

Note that in auto enable mode, the character to be transmitted must be loaded into the transmit buffer after the CTS becomes active, not before. Any character loaded into the transmit buffer before the CTS became active will not be transmitted.

**High Speed DMA Operation**

In the section titled Application Example, the MPSC has been programmed to operate in DMA mode and receiver is programmed to generate an interrupt on the first receive character. You may recall that the receive FIFO is three bytes deep. On receiving the interrupt on the first receive character, the CPU must enable the DMA controller within three received byte times to avoid receiver over-run condition. In the application example, at 400K baud, the CPU had approximately 60 μs to enable the DMA controller to avoid receiver buffer overflow. However, at higher baud rates, the CPU may not have enough time to enable the DMA controller in time. For example, at 1M baud, the CPU should enable the DMA controller within approximately 24 μs to avoid receiver buffer overrun. In most applications, this is not sufficient time. To solve this problem, the DMA controller should be left enabled before getting the interrupt on the first receive character (which is accompanied by the Rx DMA request for the appropriate channel). This will allow the DMA controller to start DMA transfer as soon as the Rx DMA request becomes active without giving the CPU enough time to re-
spond to the interrupt on the first receive character. The CPU will respond to the interrupt after the DMA transfer has been completed and will find the 8259A (See Application Example) responding with interrupt level 7, the lowest priority level. Note that the 8274 interrupt request was satisfied by the DMA controller, hence the interrupt on the first receive character was cleared and the 8259A had no pending interrupt. Because of no pending interrupt, the 8259A returned interrupt level 7 in response to the INTA sequence from the CPU. The user software should take care of this interrupt.

PROGRAMMING HINTS
This section will describe some useful programming hints which may be useful in program development.

Asynchronous Operation
At the end of transmission, the CPU must issue “Reset Transmit Interrupt/DMA Pending” command in WR0 to reset the last transmit empty request which was not satisfied. Failing to do so will result in the MPSC locking up in a transmit empty state forever.

Non-Vectored Mode
In non-vectored mode, the Interrupt Acknowledge pin (INTA) on the MPSC must be tied high through a pull-up resistor. Failing to do so will result in unpredictable response from the 8274.

HDLC/SDLC Mode
When receiving data in SDLC mode, the CRC bytes must be read by the CPU (or DMA controller) just like any other data field. Failing to do so will result in receiver buffer overflow. Also, the End of Frame Interrupt indicates that the entire frame has been received. At this point, the CRC result (RR1:D6) and residue code (RR1:D3, D2, D1) may be checked.

Status Register RR2
RR2 contains the vector which gets modified to indicate the source of interrupt (See the section titled MPSC Modes of Operation). However, the state of the vector does not change if no new interrupts are generated. The contents of RR2 are only changed when a new interrupt is generated. In order to get the correct information, RR2 must be read only after an interrupt is generated, otherwise it will indicate the previous state.

Initialization Sequence
The MPSC initialization routine must issue a channel Reset Command at the beginning. WR4 should be defined before other registers. At the end of the initialization sequence, Reset External/Status and Error Reset commands should be issued to clear any spurious interrupts which may have been caused at power up.

Transmit Under-run/EOM Latch
In SDLC/HDLC, bisync and monosync mode, the transmit underrun/EOM must be reset to enable the CRC check bytes to be appended to the transmit frame or transmit message. The transmit under-run/EOM latch can be reset only after the first character is loaded into the transmit buffer. When the transmitter under-runs at the end of the frame, CRC check bytes are appended to the frame/message. The transmit under-run/EOM latch can be reset at any time during the transmission after the first character. However, it should be reset before the transmitter under-runs otherwise, both bytes of the CRC may not be appended to the frame/message. In the receive mode in bisync operation, the CPU must read the CRC bytes and two more SYNC characters before checking for valid CRC result in RR1.

Sync Character Load Inhibit
In bisync/monosync mode only, it is possible to prevent loading sync characters into the receive buffers by setting the sync character load inhibit bit (WR3:D1 = 1). Caution must be exercised in using this option. It may be possible to get a CRC character in the received message which may match the sync character and not get transferred to the receive buffer. However, sync character load inhibit should be enabled during all pre-frame sync characters so the software routine does not have to read them from the MPSC.

In SDLC/HDLC mode, sync character load inhibit bit must be reset to zero for proper operation.

EOI Command
EOI Command can only be issued through channel A irrespective of which channel had generated the interrupt.

Priority in DMA Mode
There is no priority in DMA mode between the following four signals: TxDRQ(CH), RxDRQ(CH), TxDRQ(CHB), RxDRQ(CHB). The priority between these four signals must be resolved by the DMA controller. At any given time, all four DMA channels from the 8274 are capable of going active.
APPENDIX A
APPLICATION EXAMPLE: SOFTWARE LISTINGS
AP-145

PL/M-86 COMPILER  iSBC 88/45 B274 CHANNEL A SDLC TEST

SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INIT_B274_S
OBJECT MODULE PLACED IN :Fl.SINI74.OBJ
COMPILER INVOKED BY: PLM&6 B6 :Fl:SINI74.PLM TITLE(iSBC 88/45 B274 CHANNEL
A SDLC TEST) COMPACT NOINTVECTOR ROM

1

INIT_B274_S: DO;
$INCLUDE (:Fl:PORTS.PL)

2

DECLARE LIT LITERALLY 'LITERALLY';
= "8237A-5 PORTS */

3

DECLARE CH0_ADDR LIT '080H',
= CH0_COUNT LIT '081H',
= CH1_ADDR LIT '082H',
= CH1_COUNT LIT '083H',
= CH2_ADDR LIT '084H',
= CH2_COUNT LIT '085H',
= CH3_ADDR LIT '086H',
= CH3_COUNT LIT '087H',
= STATUS_37 LIT '08BH',
= COMMAND_37 LIT '088H',
= REQUEST_REG_37 LIT '089H',
= SINGLE_MASK LIT '08AH',
= MODE_REG_37 LIT '08BH',

PL/M-86 COMPILER  iSBC 88/45 B274 CHANNEL A SDLC TEST

= CLR_BYTE_PTR_37 LIT '08CH',
= TEMP_REG_37 LIT '08DH',
= MASTER_CLEAR_37 LIT '08DH',
= ALL_MASK_37 LIT '08FH';

4

DECLARE CTR_00 LIT '090H',
= CTR_01 LIT '091H',
= CTR_02 LIT '092H',

6-251

210403-001
8255 PORTS

DECLARE PORTA_55 LIT '0A0H',
PORTB_55 LIT '0A1H',
PORTC_55 LIT '0A2H',
CONTROL_55 LIT '0A3H';

8274 PORTS

DECLARE DATA_A_74 LIT '0D0H',
DATA_B_74 LIT '0D1H',
STATUS_A_74 LIT '0D2H',
COMMAND_A_74 LIT '0D3H',
STATUS_B_74 LIT '0D3H',
COMMAND_B_74 LIT '0D3H';

8259A PORTS

DECLARE STATUS_POL_59 LIT '0EOH',
ICW1_59 LIT '0EOH',
OCW2_59 LIT '0EOH',
OCW3_59 LIT '0EOH',
OCW1_59 LIT '0E1H',
ICW2_59 LIT '0E1H',
ICW3_59 LIT '0E1H',
ICW4_59 LIT '0E1H';

8274 REGISTER BIT ASSIGNMENTS

DECLARE RX_AVAIL LIT '01H',
INT_PENDING LIT '02H',
TX_EMPTY LIT '04H',
CARRIER_DETECT LIT '08H',
SYNC_HUNT LIT '10H',
CLEAR_TO_SEND LIT '20H';

PL/M-86 COMPILER

DECLARE ALL_SENT LIT '01H',
PARITY_ERROR LIT '10H',
RX_OVERRUN LIT '20H',
CRC_ERROR LIT '40H',
END_OF_FRAME LIT '80H';

DECLARE TX_B_EMPTY LIT '00H',
EXT_B_CHANGE LIT '01H',
RX_B_AVAIL LIT '02H',
RX_B_SPECIAL LIT '03H',
TX_A_EMPTY LIT '04H',
EXT_A_CHANGE LIT '05H',
RX_A_AVAIL LIT '06H',
RX_A_SPECIAL LIT '07H';
DECLARE CH0_SEL LIT '00H',
    CH1_SEL LIT '01H',
    CH2_SEL LIT '02H',
    CH3_SEL LIT '03H',
    WRITE_XFER LIT '04H',
    READ_XFER LIT '05H',
    DEMAND_MODE LIT '06H',
    SINGLE_MODE LIT '07H',
    BLOCK_MODE LIT '08H',
    SET_MASK LIT '09H';

12 1 DELAY_S: PROCEDURE PUBLIC;
13 2 DECLARE D WORD;
14 2 D=0;
15 2 DO WHILE D<800H;
16 3 D=D+1;
17 3 END;
18 2 END DELAY_S;

19 1 INIT_B274_SDLC_S: PROCEDURE PUBLIC;
20 2 DECLARE C BYTE;
21 2 $EJECT;

PL/M-86 COMPILER  
ISBC B8/45 B274 CHANNEL A SDLC TEST

/** TABLE TO INITIALIZE THE B274 CHANNEL A AND B */
/** FORMAT IS: WRITE REGISTER, REGISTER DATA */
/** INITIALIZE CHANNEL ONLY */

21 2 DECLARE TABLE_74_A(*) BYTE DATA
   (00H,18H, /* CHANNEL RESET */
    00H,08H, /* RESET TX CRC */
    02H,11H, /* PIN 10=RTSB, A DMA, B INT */
    04H,20H, /* SDLC/HDLC MODE, NO PARITY */
    07H,07EH, /* SDLC FLAG */
    01H,0BH, /* RX DMA ENABLE */
    05H,0E8H, /* DTR, RTS, B TX BITS, TX ENABLE, TX CRC ENABLE */
    06H,35H, /* DEFAULT ADDRESS */
    03H,0DH, /* B RX BITS, AUTO ENABLES, HUNT MODE, */
                /* RX CRC ENABLE */
    OFFH), /* END OF INITIALIZATION TABLE */

22 2 DECLARE TABLE_74_B(*) BYTE DATA
   (02H,00H, /* INTERRUPT VECTOR */
    01H,1CH, /* STATUS AFFECTS VECTOR */
    OFFH), /* END */

/** INITIALIZE THE B254 */

23 2 OUTPUT(CONTROL_54)=36H;
24 2 OUTPUT(CTR_00) = LOW(20), /* BAUD RATE = 400K_BAUD*/
25 2 OUTPUT(CTR_00) = HIGH(20), /* BAUD RATE = 400K_BAUD*/

/** INITIALIZE THE B274 */

26 2 C=0;
27 2 DO WHILE TABLE_74_B(C) <> OFFH;
28 3 OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
29 3 C=C+1;
30 3 OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
31 3 C=C+1;
32 3 END;
REFERENCES

1. IBM Document No. GA27-3004-2: General Information — Binary Synchronous Communications


3. 8274 MPSC Data Sheet, Intel Corporation, Ca.


5. Computer Networks and Distributed Processing by James Martin. Prentice Hall, Inc., N.J.
33 2 C=0;
34 2 DO WHILE TABLE_74_A(C) <> OFFH;
35 3 OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
36 3 C=C+1;
37 3 OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
38 3 C=C+1;
39 3 END;
40 2 CALL DELAY_S;
41 2 RETURN;
42 2 END INIT_8274_SDLC_S;
43 1 END INIT_8274_S;

PL/M-86 COMPILER IBM 88/45 8274 CHANNEL A SDLC TEST

MODULE INFORMATION:

| CODE AREA SIZE | 0008H | 168D |
| CONSTANT AREA SIZE | 0000H | 0D |
| VARIABLE AREA SIZE | 0003H | 3D |
| MAXIMUM STACK SIZE | 0006H | 6D |
| 213 LINES READ |
| 0 PROGRAM WARNINGS |
| 0 PROGRAM ERRORS |

END OF PL/M-86 COMPILATION

PL/M-86 COMPILER IBM 88/45 8274 CHANNEL A SDLC TEST

SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INIT_8237_CHA
OBJECT MODULE PLACED IN: F1:SIN137.OBJ
COMPILER INVOKED BY: PLM86.86 :F1:SIN137.PLM
TITLE(IBM 88/45 8274 CHANNEL A SDLC TEST) COMPACT NOINTVECTOR ROM

/***********************************************************************************/
/* * 8237 INITIALIZATION ROUTINE FOR DMA TRANSFER */
/***********************************************************************************/
1 INIT_8237_CHA: DO;
NOLIST

12 1 INIT_8237_S: PROCEDURE PUBLIC;

13 2 OUTPUT(MASTER_CLEAR_37)=0;
14 2 OUTPUT(COMMAND_37) = 20H; /* EXTENDED WRITE */
15 2 OUTPUT(ALL_MASK_37) = 0FH; /* MASK ALL REQUESTS */
16 2 OUTPUT(MODE_REQ_37) = (SINGLE_MODE OR WRITE_XFER OR CH1_SEL);
17 2 OUTPUT(MODE_REQ_37) = (SINGLE_MODE OR READ_XFER OR CH1_SEL);
18 2 OUTPUT(CLR_BYTE_PTR_37) = 0;
19 2 OUTPUT(CH0_ADDR) = 00; /* RECEIVE BUFF AT 9000H */
20 2 OUTPUT(CH0_ADDR) = 09H;
21 2 OUTPUT(CH0_COUNT) = 0H;
22 2 OUTPUT(CH0_COUNT) = 01;
23 2 OUTPUT(CH1_ADDR) = 00; /* TRANSMIT BUFF AT 8000H */
24 2 OUTPUT(CH1_ADDR) = 08H;
25 2 OUTPUT(CH1_COUNT) = 010H;
26 2 OUTPUT(CH1_COUNT) = 00H;

6-255
/* ENABLE TRANSFER */
OUTPUT(SINGLE_MASK) = CH1_SEL; /* ENABLE TX DMA */
RETURN;

END INIT_8237_S;

/* TURN OFF THE 8237 CHANNELS 0 AND 1 */
STOP_8237_S: PROCEDURE PUBLIC;
OUTPUT(SINGLE_MASK) = CH1_SEL OR SET_MASK;
OUTPUT(SINGLE_MASK) = CHO_SEL OR SET_MASK;
RETURN;
END STOP_8237_S;
END INIT_8237_CHA;

MODULE INFORMATION:

CODE AREA SIZE = 004CH 76D
CONSTANT AREA SIZE = 0000H 0D
VARIABLE AREA SIZE = 0000H 0D

PL/M-B6 COMPILER ISBC B8/45 8274 CHANNEL A SDLC TEST

MAXIMUM STACK SIZE = 0002H 2D
163 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS

END OF PL/M-B6 COMPILATION

PL/M-B6 COMPILER ISBC B8/45 8274 CHANNEL A SDLC TEST

SERIES-III PL/M-B6 V2 0 COMPILATION OF MODULE INTR_8274_S
OBJECT MODULE PLACED IN F1 SINTR OBJ
COMPILER INVOKED BY PLMB6 B6 F1 SINTR PLM TITLE(ISBC B8/45 8274 CHANNEL A SDLC TEST) COMPACT NOINTVECTOR ROM

*******************************************************************************
/ * 8274 INTERRUPT ROUTINE */
*******************************************************************************

1  INTR_8274_S DD.
$NOLIST
12 1 DECLARE TEMP BYTE.
13 1 DECLARE (RESULTS_S,TXDONE_S, RXDONE_S) BYTE EXTERNAL,
14 1 DECLARE INT_VEC POINTER AT (140),
15 1 DECLARE INT_VEC_STORE POINTER,
16 1 DECLARE MASK_59 BYTE,
17 1 DECLARE DONE LIT 'OFFH',
      NOT_DONE LIT 'O0H',
      PASS LIT 'OFFH',
      FAIL LIT 'O0H',

*******************************************************************************
/ * IGNORE INTERRUPT HANDLER */
*******************************************************************************

18 1 IGNORE_INT PROCEDURE,
19 2 RESULTS_S = FAIL.
20 2 RETURN,
21 2 END IGNORE_INT,
CHN EXTERNAL CHANGE. PROCEDURE.

TEMP = INPUT (STATUS_A_74),  /* STATUS REG 1*/
IF (TEMP AND END_OF_TX_MESSAGE) = END_OF.Tx MESSAGE THEN
TXDONE_S = DONE,
ELSE DO,
TXDONE_S = DONE,
RESULTS_S = FAIL,
END;
OUTPUT (COMMAND_A_74) = 10H,  /* RESET EXT/STATUS INTERRUPTS */
RETURN;
END CHA EXTERNAL_CHANGE.

$EJECT

PL/M-6b COMPILER 1SBC 88/45 8274 CHANNEL A SDLC TEST

CHN SPECIAL RECEIVE CONDITIONS INTERRUPT HANDLER */
************************************************************
33 1 CHA_RX_SPECIAL. PROCEDURE.
OUTPUT (COMMAND_A_74) = 1;
TEMP = INPUT (STATUS_A_74),
IF (TEMP AND END_OF_FRAME) = END_OF_Frame THEN
DO,
IF (TEMP AND 040H) = 040H THEN
RESULTS_S = FAIL,  /* CRC ERROR */
RXDONE_S = DONE,
OUTPUT (COMMAND_A_74) = 30H;  /* ERROR RESET*/
END;
ELSE DO,
IF (TEMP AND 02H) = 02H THEN DO;
RESULTS_S = FAIL,  /* RX OVERRUN ERROR*/
RXDONE_S = DONE,
OUTPUT (COMMAND_A_74) = 30H;  /* ERROR RESET*/
END;
END;
RETURN.
END CHA_RX_SPECIAL.

$EJECT

PL/M-6b COMPILER 1SBC 88/45 8274 CHANNEL A SDLC TEST

CHN RECEIVE CHARACTER AVAILABLE */
************************************************************
53 1 CHA_RX_CHAR. PROCEDURE.
OUTPUT (SINGLE_MASK) = CHO_SEL,  /* ENABLE RX DMA CHANNEL*/
RETURN.
END CHA_RX_CHAR.

$EJECT

ENABLE INTERRUPTIONS S. PROCEDURE PUBLIC,
DECLARE CHA_INT_ON LIT ‘OF7H’,
DISABLE.
CALL SET_INTERRUPT (39, INT_39).

/* ENABLE 8274 INTERRUPTS - SET UP THE 8259A */
INT_VEC_STORE = INT_VEC;
INT_VEC = INTERRUPT$PTR(INT_8274_S);
MASK_59 = INPUT(OCW1_59);
OUTPUT(OCW1_59) = MASK_59 AND CHA_INT.ON.
RETURN.
END ENABLE_INTERRUPTS_S;

DISABLE INTERRUPTS_S PROCEDURE PUBLIC,
DISABLE.
INT_VEC = INT_VEC_STORE;
OUTPUT(OCW1_59) = MASK_59,
ENABLE,
RETURN.
END DISABLE_INTERRUPTS_S,

CHANNEL B RECEIVE CHARACTER AVAILABLE */

INTERRUPT ROUTINE

INT_8274_S PROCEDURE INTERRUPT 35 PUBLIC;
OUTPUT(COMMAND_B_74) = 2; /* SET POINTER TO 2*/
TEMP = INPUT(STATUS_B_74) AND 07H; /* READ INTERRUPT VECTOR */
/* FOR THIS APPLICATION CH B INTERRUPTS ARE IGNORED*/
DO CASE TEMP:
    CALL IGNORE_INT; /* V2V1VO = 000*/
    CALL IGNORE_INT; /* V2V1VO = 001*/
    CALL CHB_RX_CHAR; /* V2V1VO = 010*/
    CALL IGNORE_INT; /* V2V1VO = 011*/
    CALL IGNORE_INT; /* V2V1VO = 100*/
    CALL CHB_EXTERNAL_CHANGE; /* V2V1VO = 101*/
    CALL CHB_RX_CHAR; /* V2V1VO = 110*/
    CALL CHA_RX_SPECIAL, /* V2V1VO = 111*/
END;
OUTPUT(COMMAND_A_74) = 3BH; /* END OF INTERRUPT FOR 8274 */
OUTPUT(OCW2_59) = 63H, /* B259 EDI */
OUTPUT(OCW1_59) = INPUT(OCW1_59) AND 0F7H;
RETURN;
END INT_8274_S,

DEFAULT INTERRUPT ROUTINE - B259A INTERRUPT 7 */
/* REQUIRED ONLY WHEN DMA CONTROLLER IS ENABLED */
/* BEFORE RECEIVING FIRST CHARACTER WHICH IS */
/* AT HIGH BAUD RATES LIKE BOOK BAUD READ APP. */
/* NOTE SECTION 6 FOR DETAILS */
97  1  INT_39: PROCEDURE INTERRUPT 39;
98  2  OUTPUT(OCW2_59) = 20H;  /* NON-SPECIFIC EOI */
99  2  OUTPUT(OCW1_59) = INPUT(OCW1_59) AND OF7H,
100  2  RESULTS_S = FAIL,
101  2  END INT_39.
102  1  END INTR_8274_S.

MODULE INFORMATION:
CODE AREA SIZE    = 01BFH  447D
CONSTANT AREA SIZE = 0000H  0D
VARIABLE AREA SIZE = 0006H  6D
MAXIMUM STACK SIZE = 0022H  34D
295 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION

PL/M-86 COMPILER  iSBC 88/45 8274 CHANNEL A SDLC TEST

SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE STEST
OBJECT MODULE PLACED IN :FI:STEST.OBJ
COMPILER INVOKED BY: PLM86.86 :FI:STEST.PLML TITLE(iSBC 88/45 8274 CHANNEL A SDLC TEST)
COMPACT NOINTVECTOR ROM

************************************************************************1
1* *1
*1
************************************************************************1
STEST: DD;

  2  1  DELAY_S: PROCEDURE EXTERNAL;
  3  2  END DELAY_S;
  4  1  ENABLE_INTERRUPTS_S: PROCEDURE EXTERNAL;
  5  2  END ENABLE_INTERRUPTS_S;
  6  1  DISABLE_INTERRUPTS_S: PROCEDURE EXTERNAL;
  7  2  END DISABLE_INTERRUPTS_S;
  8  1  INIT_8274_SDLC_S: PROCEDURE EXTERNAL;
  9  2  END INIT_8274_SDLC_S;
 10  1  INIT_8237_S: PROCEDURE EXTERNAL;
 11  2  END INIT_8237_S;
 12  1  STOP_8237_S: PROCEDURE EXTERNAL;
 13  2  END STOP_8237_S;
 14  1  VERIFY_TRANSFER_S: PROCEDURE EXTERNAL;
 15  2  END VERIFY_TRANSFER_S;
 16  1  INT_8274_S: PROCEDURE INTERRUPT 35 EXTERNAL;
 17  2  END INT_8274_S;
 18  1  DECLARE DONE LIT 'OFFH';
 19  1  DECLARE NOT_DONE LIT '00H';
 20  1  DECLARE PASS LIT 'OFFH';
 21  1  DECLARE FAIL LIT '00H';

END OF PL/M-86 COMPILATION
*EJECT

PL/M-86 COMPILER  ISBC 88/45 8274 CHANNEL A SDLC TEST

30  1  CHA_SDLC_TEST: PROCEDURE BYTE PUBLIC;

31  2  CALL  ENABLE_INTERRUPTS_S;
32  2  CALL  INIT_6274_SDLC_S;
33  2  CALL  ENABLE_INTERRUPTS_S;
34  2  OUTPUT(COMMAND_A_74) = 28H;  /* RESET TX INT/DMA */
35  2  OUTPUT(COMMAND_B_74) = 28H;  /* BEFORE INITIALIZING 8237*/
36  2  CALL  INIT_6237_S;
37  2  OUTPUT(DATA_A_74) = 55H;  /* LOAD FIRST CHARACTER FROM CPU*/
/* TO ENSURE CRC TRANSMISSION RESET TX UNDERRUN LATCH*/
38  2  OUTPUT(COMMAND_A_74) = 00H;
39  2  RXDONE_S.TXDONE_S=NOT_DONE;  /* CLEAR ALL FLAGS */
40  2  RESULTS_S=PASS;  /* SET RESULT FOR MANUAL*/
41  2  DO WHILE TXDONE_S.NOT_DONE;  /* DO UNTIL TERMINAL COUNT*/
42  3  END;
43  2  DO WHILE (INPUT(STATUS_A_74) AND 04H) <> 04H;
44  3  /* WAIT FOR CRC TO GET TRANSMITTED */
45  2  DO WHILE RXDONE_S.NOT_DONE;  /* DO UNTIL TERMINAL COUNT*/
46  3  END;
47  2  CALL  STOP_6237_S;
48  2  CALL  DISABLE_INTERRUPTS_S;
49  2  CALL  VERIFY_TRANSFER_S;
50  2  RETURN RESULTS_S;
51  2  END  CHA_SDLC_TEST;
52  1  END TEST;

MODULE INFORMATION:

   CODE AREA SIZE = 0063H  99D
   CONSTANT AREA SIZE = 0000H  0D
   VARIABLE AREA SIZE = 0003H  3D
   MAXIMUM STACK SIZE = 0004H  4D
   198 LINES READ
   0 PROGRAM WARNINGS
   0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION

PL/M-86 COMPILER  ISBC 88/45 8274 CHANNEL A SDLC TEST

SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE VECTOR_MODE
OBJECT MODULE PLACED IN:Fi:VECTOR.OBJ
COMPILER INVOKED BY: PLM86.86:Fi:VECTOR.PLML TITLE(ISBC 88/45 8274 CHANNEL A SDLC TEST)

*******************************************************************************/
/* 8274 INTERRUPT HANDLING ROUTINE FOR */
/* 8274 VECTOR MODE */
/* STATUS AFFECTS VECTOR */
*******************************************************************************/

6-260
/* THIS IS AN EXAMPLE OF HOW 8274 CAN BE USED IN VECTORED MODE */
/* THE 8BC8/45 BOARD WAS REWIRED TO DISABLE THE PIT 8259A AND */
/* ENABLE THE 8274 TO PLACE ITS VECTOR ON THE DATABUS IN RESPONSE */
/* TO THE INTA SEQUENCE FROM THE 8088. OTHER MODIFICATIONS INCLUDED */
/* CHANGES TO 8274 INITIALIZATION PROGRAM (SINIT4) TO PROGRAM 8274 */
/* INTO VECTORED MODE (WRITE REGISTER 2A D9=1) */

VECTOR_MODE: DO;
*NOLIST

DECLARE TEMP BYTE;
DECLARE (RESULTS_S, TXDONE, RXDONE) BYTE EXTERNAL;
DECLARE DONE LITERALLY 'OFFH', NOT_DONE LITERALLY '0OH',
PASS LITERALLY 'OFFH', FAIL LITERALLY '00H';

TX_INTERRUPT_CHA PROCEDURE INTERRUPT B4;
OUTPUT(COMMAND_A_74) = 00101000B; /*RESET TXINT PENDING*/
OUTPUT (COMMAND_A_74) = 00111000B; /*EOI*/
END TX_INTERRUPT:CHA;

EXT_STAT_CHANGE_CHA PROCEDURE INTERRUPT B5;
TEMP = INPUT(STATUS_A_74).
IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
TXDONE = DONE;
ELSE DO;
TXDONE = DONE.
END RX_CHAR_AVAILABLE_CHA;

PL/M-86 COMPILER 8BC 88/45 8274 CHANNEL A SDLC TEST

RESULTS_S = FAIL;
END;

OUTPUT(COMMAND_A_74) = 00010000B; /*RESET EXT STAT INT*/
OUTPUT (COMMAND_A_74) = 00111000B; /*EOI*/
RETURN;
END EXT_STAT_CHANGE_CHA.

RX_CHAR_AVAILABLE_CHA PROCEDURE INTERRUPT B6;
OUTPUT(COMMAND_A_74) = 00111000B; /*EOI*/
RETURN;
END RX_CHAR_AVAILABLE_CHA;

EJECT
SPECIAL RECEIVE CONDITION INTERRUPT SERVICE ROUTINE CHECKS FOR END OF FRAME BIT ONLY. SEE SPECIAL SERVICE ROUTINE FOR NON-VECTORED MODE FOR CRC CHECK AND OVERRUN ERROR CHECK.

SPECIAL_RX_CONDITION_CHA: PROCEDURE INTERRUPT 87;

OUTPUT(COMMAND_A_74) = 1; /*POINTER 1*/
TEMP = INPUTSTATUS_A_74);
IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
RXDONE = DONE;
ELSE DO;
END;
OUTPUT(COMMAND_A_74) = 00110000B; /*ERROR RESET*/
OUTPUT(COMMAND_A_74) = 00110000B; /*EOI*/
RETURN;
END SPECIAL_RX_CONDITION_CHA;

ENABLE_INTERRUPTS: PROCEDURE PUBLIC;
DISABLE;
CALL SET$INTERRUPT(84.TX_INTERRUPT_CHA);
CALL SET$INTERRUPT(85.EXT_STAT_CHANGE_CHA);
CALL SET$INTERRUPT(86.R,X_CHAR_AVAILABLE_CHA);  
CALL SET$INTERRUPT(87.SPECIAL_RX_CONDITION_CHA);
RETURN;
END ENABLE_INTERRUPTS;

END VECTOR_MODE;

MODULE INFORMATION:

<table>
<thead>
<tr>
<th>CODE AREA SIZE</th>
<th>CONSTANT AREA SIZE</th>
<th>VARIABLE AREA SIZE</th>
<th>MAXIMUM STACK SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>012EH 302D</td>
<td>0000H 0D</td>
<td>0001H 1D</td>
<td>001EH 30D</td>
</tr>
</tbody>
</table>

6-262
APPENDIX B
MPSC READ/WRITE REGISTER DESCRIPTIONS
WRITE REGISTER 0 (WR0):

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>D5</td>
<td>D4</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D1</td>
<td>D0</td>
</tr>
</tbody>
</table>

COMMAND STATUS POINTER
REGISTER POINTER

0 0 0: NULL CODE
0 0 1: SEND ABORT (SDLC)
0 1 0: RESET EXT STATUS INTERRUPTS
0 1 1: CHANNEL RESET
1 0 0: ENABLE INTERRUPT ON NEXT RX CHARACTER
1 0 1: RESET Tx INT DMA PENDING
1 1 0: ERROR RESET
1 1 1: END OF INTERRUPT

0 0: NULL CODE
0 1: RESET Rx CRC CHECKER
1 0: RESET Tx CRC GENERATOR
1 1: RESET Tx UNDERRUN EOM LATCH

WRITE REGISTER 1 (WR1):

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>D5</td>
<td>D4</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D1</td>
<td>D0</td>
</tr>
</tbody>
</table>

EXT INTERRUPT ENABLE

Tx INTERRUPT DMA ENABLE

STATUS AFFECTS VECTOR

0: FIXED VECTOR
1: VARIABLE VECTOR

0: (CHB ONLY)
1: (NULL CODE CH A)

0 0: Rx INT/DMA DISABE
0 1: Rx INT ON FIRST CHAR OR SPECIAL CONDITION
1 0: INT ON Rx CHAR (PARITY AFFECTS VECTOR) OR SPECIAL CONDITION
1 1: INT ON ALL Rx CHAR (PARITY DOES NOT AFFECT VECTOR) OR SPECIAL CONDITION

1: WAIT ON Rx, 0 WAIT ON Tx

MUST BE ZERO

WAIT ENABLE, 1 ENABLE, 0 DISABLE

6-264
210403-001
WRITE REGISTER 2 (WR2): CHANNEL A

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D0</td>
</tr>
</tbody>
</table>

- **00**: BOTH INTERRUPT
- **01**: DMA B INT
- **10**: BOTH DMA
- **11**: ILLEGAL

- **00**: 8085 MODE 1
- **01**: 8085 MODE 2
- **10**: 8086/88 MODE
- **11**: ILLEGAL

**MUST BE ZERO**

- **1**: PIN 10 SYNDetect
- **0**: PIN 10 RTS

*EXTERNAL STATUS INTERRUPT ONLY IF EXT INTERRUPT ENABLE (WR1:DO) IS SET*

WRITE REGISTER 2 (WR2): CHANNEL B

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>V7</td>
<td>V0</td>
</tr>
</tbody>
</table>

- **INTERRUPT VECTOR**

WRITE REGISTER 3 (WR3):

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D0</td>
</tr>
</tbody>
</table>

- **Rx ENABLE**
- **SYNC CHAR LOAD INHIBIT**
- **ADDR SRCH MODE (SDLC)**
- **Rx CRC ENABLE**
- **ENTER HUNT MODE**
- **AUTO ENABLES**

- **00**: Rx5 BITS/CHAR
- **01**: Rx7 BITS/CHAR
- **10**: Rx8 BITS/CHAR
- **11**: Rx8 BITS/CHAR
WRITE REGISTER 4 (WR4):

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>D5</td>
<td>D4</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D1</td>
<td>D0</td>
</tr>
</tbody>
</table>

1 ENABLE PARITY
0 DISABLE PARITY
1 EVEN PARITY
0 ODD PARITY
0 0 ENABLE SYNC MODES
0 1 1 STOP BIT
1 0 1.5 STOP BITS
1 1 2 STOP BITS
0 0 8 BIT SYNC CHAR
0 1 16 BIT SYNC CHAR
1 0 SDLC/HDL/C1111110|FLA
1 1 1 EXTERNAL SYNC MODE
0 0 X1 CLOCK
0 1 X16 CLOCK
1 0 X32 CLOCK
1 1 X64 CLOCK

WRITE REGISTER 5 (WR5):

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>D5</td>
<td>D4</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D1</td>
<td>D0</td>
</tr>
</tbody>
</table>

Tx CRC ENABLE
RTS
SDLC/CRC -16 (CRC MODE)
Tx ENABLE
SEND BREAK
DTR

WRITE REGISTER 6 (WR6):

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>D5</td>
<td>D4</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D1</td>
<td>D0</td>
</tr>
</tbody>
</table>

LEAST SIGNIFICANT
SYNC BYTE (ADDRESS IN SDLC/HDL/C MODE)

WRITE REGISTER 7 (WR7):

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>D5</td>
<td>D4</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D1</td>
<td>B0</td>
</tr>
</tbody>
</table>

MOST SIGNIFICANT
SYNC BYTE (MUST BE 0111111 IN SDLC/HDL/C MODE)
READ REGISTER 0 (RRO):

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>D5</td>
<td>D4</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D1</td>
<td>D0</td>
</tr>
</tbody>
</table>

- Rx CHAR AVAILABLE
- INT PENDING (CHA ONLY)
- Tx BUFFER EMPTY
- CARRIER DETECT
- SYNC/HUNT
- CTS
- Tx UNDERRUN/EOM
- BREAK/ABORT

READ REGISTER 1 (RR1):

(SPECIAL RECEIVE CONDITION MODE)

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D6</td>
</tr>
<tr>
<td>D5</td>
<td>D4</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D1</td>
<td>D0</td>
</tr>
</tbody>
</table>

- ALL SENT
- I FIELD BYTE
- PREVIOUS BYTE
- 2ND PREVIOUS BYTE
- RESIDUE DATA
- BITS CHAR MODE
- PARITY ERROR
- Rx OVERRUN ERROR
- CRC/FRAMING ERROR
- END OF FRAME (SDLC HDLC MODE)

READ REGISTER 2 (RR2):

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>V7</td>
<td>V6</td>
</tr>
<tr>
<td>V5</td>
<td>V4*</td>
</tr>
<tr>
<td>V3*</td>
<td>V2*</td>
</tr>
<tr>
<td>V1*</td>
<td>V0*</td>
</tr>
</tbody>
</table>

*VARIABLES IN STATUS AFFECTS VECTOR MODE
Asynchronous and SDLC Communications with 82530

SHARAD GANDHI
CONTENTS

INTRODUCTION
1. SCC Port Definition
2. Accessing the SCC Registers
3. Initialization for ASYNC Operation
4. ASYNC Communication in Polling Mode
5. ASYNC Communication in Interrupt Mode
6. Initialization for SDLC Communication
7. SDLC Frame Reception
8. SDLC Frame Transmission
9. SDLC Interrupt Routines

CONCLUSIONS

REFERENCES

APPENDIX A—82530-BAUD RATE GENERATORS

APPENDIX B—MODEM CONTROL PINS ON THE 82350

APPENDIX C—INTERFACING 82530 TO 80186
INTRODUCTION
INTEL’s 82530, Serial Communications Controller (SCC), is a dual channel, multi-protocol data communications peripheral. It is designed to interface to high speed communications lines using asynchronous, byte synchronous and bit synchronous protocols. It runs up to 1.5 Mbits/sec, has on-chip baud rate generators and on-chip NRZI encoding and decoding circuits — very useful for SOLC communication. This application note shows how to write 110 drivers for the 82530 to do initialization and data links using asynchronous (ASYNC) and SOLC protocols. The appendix includes sections to show how the on-chip baud rate generators could be programmed, how the modem control pins could be used and how the 82530 could be interfaced to INTEL’s 80186/188 processors.

This article deals with the software for the following:
1. SCC port definition
2. Accessing the SCC registers
3. Initialization for ASYNC communication
4. ASYNC communication in polling mode
5. ASYNC communication in interrupt mode
6. Initialization for SDLC communication
7. SDLC frame reception
8. SDLC frame transmission
9. SDLC interrupt routines

The description is written around illustrations of the actual software written in PLM86 for a 80186 - 82530 system.

I. SCC Port Definition
The Figure 1 shows how the 4 ports (2 per channel) of the SCC can be defined. Note that the sequence of ports in the ascending order of addresses is not the one that is normally expected. In the ascending order it is: command (B), data (B), command (A) and data (A). In an 80186 - 82530 system, the interconnection is as follows:

| PCSn | --- | CS  |
| A1   | --- | D/C |
| A2   | --- | A/B |
| RD   | --- | RD  |
| WR   | --- | WR  |

80186 pins 82530 pins

2. Accessing the SCC Registers
The SCC has 16 registers on each of the channels (A and B). For each channel there is only one port, the command port, to access all the registers. The register #0 can be always accessed directly through the command port. All other registers are accessed indirectly through register #0. First, the number of the register to be accessed is written to the register #0 - see the statement, in Figure 2: 'output (ch_a_command) = reg_no and 0fh'. Then, the desired register is written to or read out of the register #0. The Figure 2 shows 4 procedures: rra and wra, for reading and writing channel A registers; rrb and wrb, for reading and writing channel B registers. The read procedures are of the type 'byte' - they return the contents of the register being read. The write procedures require two parameters - the register number and the value to be written.

```c
declare ch_b_command literally 'pcs5 + 0', // scc channel_b command word#
    ch_b_data literally 'pcs5 + 2',  // scc channel_b data word #/
    ch_a_command literally 'pcs5 + 4',  // scc channel_a command word #/
    ch_a_data literally 'pcs5 + 6'    // scc channel_a data word #/
```

Figure 1. SCC Port Definition

231262-1

231262-001
/*-----------------------------------------------*/

/* read selected scc register */

rra. procedure (reg_no) byte,
    declare reg_no byte;
    if (reg_no and Ofh) <> 0
        then output(ch_a_command) = reg_no and Ofh;
        return input(ch_a_command),
    end rra;

rrb. procedure (reg_no) byte;
    declare reg_no byte;
    if (reg_no and Ofh) <> 0
        then output(ch_b_command) = reg_no and Ofh;
        return input(ch_b_command),
    end rrb;

/* write selected scc register */

wra: procedure (reg_no, value),
    declare reg_no byte;
    declare value byte;
    if (reg_no and Ofh) <> 0
        then output (ch_a_command) = reg_no and Ofh;
        output (ch_a_command) = value,
    end wra;

wrb: procedure (reg_no, value),
    declare reg_no byte;
    declare value byte;
    if (reg_no and Ofh) <> 0
        then output (ch_b_command) = reg_no and Ofh;
        output (ch_b_command) = value,
    end wrb;

/*-----------------------------------------------*/

Figure 2. Accessing the SCC Registers
3. Initialization for ASYNC Operation

Channel B of the SCC is used to perform ASYNC communication. Figure 3 shows how the channel B is initialized and configured for ASYNC operation. This is done by writing the various channel B registers with the proper parameters as shown. The comments in the program show what is achieved by each statement. After a software reset of the channel, register #4 should be written before writing to the other registers. The on-chip Baud Rate Generator is used to generate a 1200 bits/sec clock for both the transmitter and the receiver. The interrupts for transmitter and/or receiver are enabled only for the interrupt mode of operation; for polling, interrupts must be kept disabled.

```plaintext
/*---------------------------------------------*/

scc_init_b: procedure;
/* scc ch B register initialization for ASYNC mode */

  call wrb(09, 01000000b);  /* channel B reset */
  call wrb(04, 11011110b);  /* 2 stop, no parity, bfr = 64x */
  call wrb(02, 00100000b);  /* vector = 20h */
  call wrb(03, 11000000b);  /* rx 8 bits/char, no auto-enable */
  call wrb(05, 01100000b);  /* tx 8 bits/char */
  call wrb(06, 00000000b);
  call wrb(07, 00000000b);
  call wrb(09, 00000011b);
  call wrb(10, 00000000b);
  call wrb(11, 01010110b);  /* rxc = txc = BRG, trxc = BRG out */
  call wrb(12, 00011000b);  /* to generate 1200 baud, x64 @ 4 mhz */
  call wrb(13, 00000000b);
  call wrb(14, 00000111b);  /* BRG source = SYS CLK, enable BRG */
  call wrb(15, 0000000b);   /* all ext status interrupts off */

  /* enables */
  call wrb(03, 11000001b);  /* scc-b receive enable */
  call wrb(05, 11101010b);  /* scc-b transmit enable, dtr on, rts on */

  /* enable interrupts - only for interrupt driven ASYNC I/O */
  call wrb(09, 00000101b);  /* master IE, vector includes status */
  call wrb(01, 00010011b);  /* tx, rx, ext interrupts enable */

end scc_init_b;

/*---------------------------------------------*/
```

Figure 3. Initialization for ASYNC Communication

4. ASYNC Communication in Polling Mode

Figure 4 shows the procedures for reading in a received character from the 82530 (scc_in) and for writing out a character to the 82530 (scc_out) in the polling mode.

The scc_in procedure returns a byte value which is the character read in. The receiver is polled to find if a character has been received by the SCC. Only when a character has been received, the character is read in from the data port of the SCC channel B.

The scc_out procedure requires a byte parameter which is the character being written out. The transmit-
Figure 4. ASYNC Communication in Polling Mode

Typical calls to these procedures are:

```plaintext
abc_variable = scc_in;
call scc_out (xyz_variable);
```

5. ASYNC Communication in Interrupt Mode

In contrast to polling for the receiver and/or the transmitter to be ready with/for the next character, the 82530 can be made to interrupt when it is ready to do receive or transmit.

The on-chip interrupt controller of the SCC can be made to operate in the vectored mode. In this mode, it generates interrupt vectors that are characteristic of the event causing the interrupt. For the example here, the vector base is programmed at 20h and 'Vector Includes Status' (VIS) mode is set - WR9 = XXX0XXX01. Vectors and the associated events are:

<table>
<thead>
<tr>
<th>Vector</th>
<th>Procedure</th>
<th>Event Causing Interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>20h</td>
<td>bxiin_b</td>
<td>ch_b - transmit buffer empty</td>
</tr>
<tr>
<td>22h</td>
<td>esi_b</td>
<td>ch_b - external/status change</td>
</tr>
<tr>
<td>24h</td>
<td>nxinr_b</td>
<td>ch_b - receive character available</td>
</tr>
<tr>
<td>26h</td>
<td>src_b</td>
<td>ch_b - special receive condition</td>
</tr>
<tr>
<td>28h</td>
<td>bxiin_a</td>
<td>ch_a - transmit buffer empty</td>
</tr>
<tr>
<td>2ah</td>
<td>esi_a</td>
<td>ch_a - external/status change</td>
</tr>
<tr>
<td>2ch</td>
<td>nxinr_a</td>
<td>ch_a - receive character available</td>
</tr>
<tr>
<td>2eh</td>
<td>src_a</td>
<td>ch_a - special receive condition</td>
</tr>
</tbody>
</table>

**NOTE:**
Odd vector numbers do not exist.

Figure 5 shows the interrupt procedures for the channel B operating in ASYNC mode. The transmitter buffer empty interrupt occurs when the transmitter can accept one more character to output. In the interrupt procedure for transmit, the byte char_out_530 is output. Following this, is an epilogue that is common to all the
interrupt procedures; the first statement is an end of interrupt command to the 82530 - note that it is issued to channel A - and the second is an End of Interrupt (EOI) command to the 80186 interrupt controller which is, in fact, receiving the interrupt from the 82530.

The receive buffer full interrupt occurs when the receiver has at least one character in its buffer, waiting to be read in by the CPU.

The esi_b is not enabled to occur and src_b cannot occur in the ASYNC mode unless the receiver is overrun or a parity error occurs.

```assembly
/* channel B interrupt procedures */
txintr_b: procedure interrupt 20h:
    output (ch_b_data) = char_out_530;
    call wra(00,38h); /* reset highest IUS */
    output (eoir_186) = 8000h; /* non specific EOI */
    return;
end txintr_b;
esi_b:  procedure interrupt 22h:
    call wrb(00,10h); /* reset ESI */
    call wra(00,38h); /* reset highest IUS */
    output (eoir_186) = 8000h; /* non specific EOI */
    return;
end esi_b;
rxintr_b: procedure interrupt 24h:
    char_in_530 = input (ch_b_data);
    call wra(00,38h); /* reset highest IUS */
    output (eoir_186) = 8000h; /* non specific EOI */
    return;
end rxintr_b;
src_b:  procedure interrupt 26h:
    call wrb(00,30h); /* error reset */
    call wra(00,38h); /* reset highest IUS */
    output (eoir_186) = 8000h; /* non specific EOI */
    return;
end src_b;

Figure 5. ASYNC Communication in Interrupt Mode
```
6. Initialization for SDLC Communication

Channel A of the SCC is programmed for being used for SDLC operation. It uses the DMA channels on the 80186. Figure 6 shows the initialization procedure for channel A. The comments in the software show the effect of each statement. The on-chip Baud Rate Generator is used to generate a clock of 125 KHz both for reception and transmission. This procedure is just to prepare the channel A for SDLC operation. The actual transmission and reception of frames is done using the procedures described further.

7. SDLC Frame Reception

Figure 7 shows the entire set-up necessary to receive a SDLC frame. First the DMA controller is programmed with the receive buffer address (@rx_buff), byte count, mode etc and is also enabled. Then a flag indicating reception of the frame is reset. An Error Reset command is issued to clear up any pending error conditions. The receive interrupt is enabled to occur at the end of frame reception (Special Receive Condition); lastly, the receiver is enabled and put in the Hunt mode (to detect the SDLC flag). When the first flag is detected on the RxDA pin, it goes from the Hunt to the Sync mode. It receives the frame and the end of frame interrupt (src_b, vector = 2eh) occurs.

8. SDLC Frame Transmission

Figure 8 shows the procedure for transmitting a SDLC frame once the channel A is initialized. The DMA controller is initialized with the transmit buffer address (@tx_buff) - note, it is the second byte of the transmit buffer - and the byte count - again one less than the total buffer length. This is done because the first byte in the buffer is output directly using an I/O instruction and not by DMA. Then the flag indicating frame transmitted is reset. The events following are very critical in sequence:

a. Reset external status interrupts
b. Enable the transmitter
c. Reset transmit CRC
d. Enable transmitter underrun interrupt
e. Enable the DMA controller
f. Output first byte of the transmit block to data port
g. Reset Transmit Underrun Latch

```
/*------------------------------------------*/

scc_init_a: procedure;

/* scc_ch A register initialization for SDLC mode */

    call wra(09, 10000000b);  /* channel A reset */
    call wra(04, 00100000b);  /* SDLC mode */
    call wra(01, 01100000b);  /* DMA for Rx */
    call wra(03, 11000000b);  /* @ bit Rx char, Rx disable */
    call wra(05, 01100000b);  /* @ bit Tx char, Tx disable */
    call wra(06, 01010101b);  /* node address */
    call wra(07, 0111110b);  /* SDLC flag */
    call wra(10, 10000000b);  /* preset CRC, NRZ encoding */
    call wra(11, 01010110b);  /* rxc = txc = BRG, trxc = BRG out */
    call wra(12, 00001110b);  /* to generate 125 Kbaud, x1 @ 4 mhz */
    call wra(13, 00000000b);  /* BRG source = SYS CLK, DMA for Tx */
    call wra(14, 00000110b);  /* all ext status interrupts off */
    call wra(15, 00000000b);  /* master IE, vector includes status */

/* enables */

    call wra(14, 00000111b);  /* enable : BRG */
    call wra(01, 11100000b);  /* enable : dreq */
    call wra(09, 00010011b);  /* master IE, vector includes status */

end scc_init_a;

/*------------------------------------------*/
```

Figure 6. Initialization for SDLC Communication
/*----------------------------------------------------*/
rx_init: procedure;
    declare dma_O_mode literally '1010001001000000b';
    /* src=ID, dest=M(inc), sync=src, TC, noint, priority, byte */
    outword(dma_O_dpl) = low16(@rx_buff);
    outword(dma_O_dph) = high16(@rx_buff);
    outword(dma_O_spl) = ch_a_data;
    outword(dma_O_sp) = 0;
    outword(dma_O_tc) = block_length + 2; /* +2 for CRC */
    outword(dma_O_cw) = dma_O_mode or 0006h; /* start DMA channel 0 */
    frame_rec = 0; /* reset frame received flag */
    call wra(00.30h); /* error reset */
    call wra(01.1111001b); /* sp. cond intr only, ext int enable */
    call wra(03.11010001b); /* enable receiver, enter hunt mode */
end rx_init;

/*----------------------------------------------------*/

tx_init: procedure;
    declare dma_1_mode literally '0001011010000000b';
    /* src=ID, dest=ID, sync=dest, TC, noint, noprior, byte */
    outword(dma_1_spl) = low16(@tx_buff(1));
    outword(dma_1_sp) = high16(@tx_buff(1));
    outword(dma_1_dpl) = ch_a_data;
    outword(dma_1_dp) = 0;
    outword(dma_1_tc) = block_length - 1; /* -1 for first byte */
    frame_tx = 0; /* reset frame transmitted flag */
    call wra(00.00010000b); /* reset ESI */
    call wra(05.01101011b); /* enable transmitter */
    call wra(00.10101000b); /* reset tx CRC, TxINT pending */
    call wra(15.01000000b); /* enable : TxU int */
    outword(dma_1_cw) = dma_1_mode or 0006h; /* start DMA channel 1 */
    output(ch_a_data) = tx_buff(0); /* first byte - address field */
    call wra(00.11000000b); /* Reset Tx Underrun latch */
end tx_init;

Figure 7. SDLC Frame Reception

Figure 8. SDLC Frame Transmission
/* channel A interrupt procedures */

txintr_a: procedure interrupt 28h;
  call wra(00,38h); /* reset highest IUS */
  output (eoir_186) = 8000h; /* non specific EOI */
  return;
end txintr_a;

esi_a: procedure interrupt 2ah;
  call wra(00,10h), /* reset ESI */
  tx_stat = rra(0); /* read in status */
  frame_tx = Offh; /* set frame transmitted flag */
  call wra(00,38h); /* reset highest IUS */
  output (eoir_186) = 8000h; /* non specific EOI */
  return;
end esi_a;

rxintr_a: procedure interrupt 2ch;
  call wra(00,38h); /* reset highest IUS */
  output (eoir_186) = 8000h; /* non specific EOI */
  return;
end rxintr_a;

src_a: procedure interrupt 2eh;
  rx_stat = rra(1); /* error reset */
  call wra(00,30h), /* disable rx */
  call wra(03,11000000b); /* set frame received flag */
  frame_rec = Offh;
  call wra(00,38h); /* reset highest IUS */
  output (eoir_186) = 8000h; /* non specific EOI */
  return;
end src_a;

Figure 9. SDLC Interrupt Routines
The frame gets transmitted out with all bytes, except the first one, being fetched by the SCC using the DMA controller. At the end of the block the DMA controller stops supplying bytes to the SCC. This makes the transmitter underrun. Since the Transmitter Underrun Latch is in the reset state at this moment, the CRC bytes are appended by the SCC at the end of the transmit block going out. An External Status Change interrupt (esi_a, vector = 2ah) is generated with the bit for transmitter underrun set in RRO register. This interrupt occurs when the CRC is being transmitted out and not when the frame is completely transmitted out.

9. SDLC Interrupt Routines

Figure 9 shows all the interrupt procedures for channel A when operating in the SDLC mode. The procedures of significance here are esi_a and src_a.

The end of frame reception results in the src_a procedure getting executed. Here the status in register RR1 is stored in a variable rx_stat for future examination. Any error bits set in status are reset, receiver is disabled and the flag indicating reception of a new frame is set.

The esi_a procedure is executed when CRC of the transmitted frame is just going out of the SCC. Reset External Status Interrupt command is executed, the external status is stored in a variable tx_stat for future examination and the flag indicating transmission of the frame is set.

End of frame processing is required after both of these interrupt procedures. It involves looking at rx_stat and tx_stat and checking if the desired operation was successful. The buffers used, may have to be recovered or new ones obtained to start another frame transmission or reception.

CONCLUSIONS

This article should ease the process of writing a complete data link driver for ASYNC and SDLC modes since most of the hardware dependent procedures are illustrated here. It was a conscious decision to make the procedures as small and easy to understand as possible. This had to be done at the expense of making the procedures general and not dealing with various exception conditions that can occur.

REFERENCES

1. 82530 Data Sheet, Order #230834-001
2. 82530 SCC Technical Manual, Order #230925-001
The 82530 has two Baud Rate Generators (BRG) on chip—one for each channel. They are used to provide the baud rate or serial clock for receive and transmit operations. This article describes how the BRG can be programmed and used.

The BRG for each channel is totally independent of each other and have to be programmed separately for each channel. This article describes how any one of the two BRGs can be programmed for operation. To use the BRG, four steps have to be performed:

1. Determine the Baud Rate Time Constant (BRTC) to be programmed into registers WR12 (LSB) and WR13 (MSB).
2. Program in register WR11, to specify where the output of the BRG must go to.
3. Program the clock source to the BRG in register WR14.
4. Enable the BRG.

**Step 1: Baud Rate Time Constant (BRTC)**

The BRTC is determined by a simple formula:

\[
BRTC = \frac{\text{Serial Clock Frequency}}{2 \times (\text{Baud Rate} \times \text{Baud Rate Factor})} - 2
\]

**Example:**

For Serial Clock Frequency = 4 MHz

<table>
<thead>
<tr>
<th>Baud Rate</th>
<th>Baud Rate Factor</th>
<th>BRTC Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>9600</td>
<td>16</td>
<td>( \frac{400000}{2 \times (9600 \times 16)} - 2 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( 13.021 - 2 = 11.021 )</td>
</tr>
</tbody>
</table>

---

**Figure 1. Write Register 11**
Table 1. BRTC: Baud Rate Time Constant

<table>
<thead>
<tr>
<th>Baud Rate Factor</th>
<th>1</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baud Rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9600</td>
<td>206.333</td>
<td>11.021</td>
<td>4.510</td>
</tr>
<tr>
<td>2</td>
<td>4800</td>
<td>414.667</td>
<td>24.042</td>
<td>11.021</td>
</tr>
<tr>
<td>3</td>
<td>2400</td>
<td>831.333</td>
<td>50.083</td>
<td>24.042</td>
</tr>
<tr>
<td>4</td>
<td>1200</td>
<td>1664.667</td>
<td>102.167</td>
<td>50.083</td>
</tr>
<tr>
<td>5</td>
<td>600</td>
<td>3331.333</td>
<td>206.333</td>
<td>102.167</td>
</tr>
<tr>
<td>6</td>
<td>300</td>
<td>6664.667</td>
<td>414.667</td>
<td>206.333</td>
</tr>
</tbody>
</table>

Since only integers can be written into the registers WR12/WR13 this will have to be rounded off to 11 and it will result in an error of:

\[
\text{fraction} \times 100 = \frac{0.021}{11.021} \times 100 = 0.19\%
\]

This error indicates that the baud rate signal generated by the BRG does not provide the exact frequency required by the system. This error is more serious for smaller baud rate factors. For asynchronous systems, errors up to 5% are considered acceptable.

Note that for BRTC = 0, BRG output frequency = \(1/4 \times \text{Serial Clock Freq.}\)

Table 1 shows the BRTC for a 4 MHz serial clock with various baud rates on the Y-axis and baud rate factors on the X-axis. The constant that is really programmed into registers WR12/WR13 is the integer closest to the BRTC value shown in the table.

Step 2: BRG Output

The output of the BRG can be directed to the Receiver, Transmitter and the TRxC output. This is programmed by setting bits D6 D5, bits D4 D3, and bits D1 D0 in register WRI1 to 10. See Figure 1. The output of the BRG can also be directed to the Digital Phase Locked Loop (DPLL) for the on-chip decoding of the NRZI encoded received data signal. This is done by writing 100 into bits D7 D6 D5 of register WR14 as shown in Figure 2.

Step 3: BRG Source Clock

Register WR14 is used to select the input clock to the BRG. See Figure 2.

Step 4: BRG Enable

This is the last step where bit D0 of WR14 is set to start the BRG. The BRG can also be disabled by resetting this bit.
APPENDIX B
MODEM CONTROL PINS ON THE 82530

Introduction

This article describes how the CTS/ and CD/ pins on the 82530 behave and how to write software to service these pins. The article explains when the External Status Interrupt occurs and how and when to issue the Reset External Status Interrupt command to reliably determine the state of these pins.

Bits D3 and D5 of register RR0 show the inverted state of logic levels on CD/ and CTS/ pins respectively. It is important to note that the register RR0 does not always reflect the current state of the CD/ and CTS/ pins. Whenever a Reset External Status Interrupt (RESI) command is issued, the (inverted) states of the CD/ and the CTS/ pins get updated and latched into the RR0 register and the register RR0 then reflect the inverted state of the CD/ and CTS/ pins at the time of the write operation to the chip. On channel or chip reset, the inverted state of CD/ and CTS/ pins get latched into RR0 register.

Normally, a transition on any of the pins does not necessarily change the corresponding bit(s) in RR0. In certain situations it does and in some cases it does not. A sure way of knowing the current state of the pins is to read the register RR0 after a RESI command.

There are two cases:
I. External Status Interrupt (ESI) enabled.
II. Polling (ESI disabled).

Case I: External Status Interrupt (ESI) Enabled

Whenever ESI is enabled, an interrupt can occur whenever there is a transition on CD/ or CTS/ pins - the IE bits for CD/ and/or CTS/ must also be set in WR15 for the interrupt to be enabled.

In this case, the first transition on any of these pins will cause an interrupt to occur and the corresponding bit in RR0 to change (even without the RESI command). A RESI command resets the interrupt line and also latches in the current state of both the CD/ and the CTS/ pins. If there was just one transition the RESI does not really change the contents of RR0.

If there are more than one transitions, either on the same pin or one each on both pins or multiple on both pins, the interrupt would get activated on the first transition and stay active. The bit in RR0 corresponding only to the very first transition is changed. All subsequent transitions have no effect on RR0. The first transition, in effect, freezes all changes in RR0. The first RESI command, as could be expected, latches the final (inverted) state of the CD/ and CTS/ pins into the RR0 register. Note that all the intermediate transitions on the pins are lost (because the response to the interrupt was not fast enough). The interrupt line gets reset for only a brief moment following the first RESI command. This brief moment is approximately 500 ns for the 82530. After that the interrupt becomes active again. A second RESI command is necessary to reset the interrupt. Two RESI commands resets the interrupt line independent of the number of transitions occurred.

Whenever operating with ESI enabled, it is recommendable to issue two back-to-back RESI commands and then read the RR0 register to reliably determine the state of the CD/ and CTS/ pins and also to reset the interrupt line in case multiple transitions may have occurred.

State Diagram
Case II: Polling RR0 for CD/ and CTS/ Pins

If RR0 is polled for determining the state of the CD/ and CTS/ pins, then the External Status Interrupt (ESI) is kept disabled. In this case the bits in RR0 may not change even for the first transition. The best way to handle this case is to always issue a RESI command before reading in the RR0 register to determine the state of CD/ and CTS/ pins. Note, however, if two back-to-back RESI commands were to be issued every time before reading in the RR0 register, the first subsequent transition will change the corresponding bit in RR0.

The state diagram above illustrates how each transition on CD/ and CTS/ pins affect the 82530 and what effect the RESI command has.

State 0
It is entered on reset. No ESI due to CTS/ or CD/ are pending in this state. Any transition on CTS/ or CD/ pins lead to the state 1 accompanied by an immediate change in the RR0 register.

State 1
Interrupt is active (if enabled). If a RESI command is issued, state 0 is reached where interrupt is again inactive. However, a further transition on CTS/ or CD/ pin leads to state 2 without an immediate change in RR0 register.

State 2
Interrupt is active (if enabled). Any further transitions have no effect. A RESI command leads to state 1, temporarily making the interrupt inactive.

CONCLUSIONS

Register RR0 does not always reflect the current (inverted) state of the CD/ and CTS/ pins. The most reliable way to determine the state of the pins in interrupt or polling mode is to issue two back-to-back RESI commands and then read RR0. While polling, the second RESI is redundant but harmless. When issuing the back-to-back RESI commands to 82530 note that the separation between the two write cycles should be at least 6 CLK + 200 ns; otherwise the second RESI will be ignored.
APPENDIX C. Interfacing 82530 to 80186

INTRODUCTION
The 82530 is Intel's new sophisticated dual channel multiprotocol serial communications controller. It can run up to 1.5 Mb/s in synchronous mode. It has useful features like on-chip baud rate generators and oscillators. It can be operated in polled, interrupt, half-duplex DMA, or full-duplex DMA modes. It is also capable of supplying its own interrupt vector during INTA cycles (like the 8274).

Interfacing the 82530 to the 8086/88 and 80186/188 processors requires the external logic shown in Figure 1.

FOUR TTL PACKAGE INTERFACE
A method of interfacing the 82530 to the 80186 CPU with four 14-pin TTL packages is described in this application note. The circuitry is shown in Figure 2. The TTLs are 74LS04, 74LS74, and 74LS08.

The interface supports the following operational modes:

1) Polled
2) Interrupt in vectored mode
3) Interrupt in non-vectored mode
4) Half-duplex - DMA on both channels
5) Full-duplex - DMA on one channel

Figure 1. 80186/82530 Interface
NOTES:
1. H = PULLED HIGH THROUGH 1K OHM
2. U1 = 74LS74
3. U2 = 74LS08
4. U3 = 74LS04
5. U4 = 74LS74

Channel A
- DB7
- DB6
- DB5
- DB4
- DB3
- DB2
- DB1
- DB0
- CDA
- DTR/REQA

Channel B
- 25
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40

Figure 2.4 TTL 82530 - 80186 Interface Circuit
PRINCIPLES AND CIRCUIT DESCRIPTION

The principles shown can be used easily to extend full duplex DMA to both channels. This can in fact be done using the same 4 TTL packages if an 8288 were also used in the system—more of that later. The reason why TTL interfacing is necessary and how it is done is now described.

A) Reset

The 82530 does not have an explicit hardware reset input; however, simultaneous activation of RD and \( \text{WR} \) signals, as shown in Figure 3, is equivalent to a hardware reset of the 82530. This requires ORing of \( \text{RESET} \) with RD and \( \text{WR} \) signals to the 82530.

B) Write

The falling edge of \( \text{WR} \) should not occur before the data (to be written to the 82530) is valid (see Figure 4). Nor should the rising edge of \( \text{WR} \) occur after the data becomes invalid. This means that the \( \text{WR} \) active phase should occur entirely during the time when the data is valid. The \( \text{WR} \) signal from 8086/88/186/188 goes active before the data is valid. A D flip-flop and two inverters are used to delay the \( \text{WR} \) going to 82530 so that it becomes active after

---

**Figure 3. RESET Timing**

**Figure 4. WR Signal Timing**
the data is valid. Note that if an 8288 is used to generate the IOWR signal (as in all big systems), then the flip-flop and inverters are not required since IOWR from the 8288 is compatible with the 82530 timing requirements.

C) DMA

The 82530 has two types of DMA request outputs; also, it has no DACK inputs. This means that the 82530 requires either a two cycle type of DMA transfer (a la 80186/88 or 8089), or DACK from the DMA controller (e.g. 8237A) has to be used to generate CS, A/B, and D/C signals.

The first type of DMA request is RDY/REQ. It can be programmed to function as RDY or DMAREQ (WR1: Bit 6). It can further be programmed as DMAREQ for transmit or for receive (WR1: Bit 5). This enables using just one signal for both the receive and transmit functions—ideal for half-duplex operation. This signal needs just an inversion to be fed into the DRQ input of the 80186.

The second DMA request signal is DTR/REQ. It can be programmed to function as DTR (Data Terminal Ready) or as DMAREQ for transmitter (active on transmitter buffer empty) in WR14: Bit 2. Thus, full-duplex DMA is possible by using DTR/REQ as TxDRQ and RDY/REQ as RxDRQ. DTR/REQ requires a little over 5 CLK cycles to become inactive. This would cause the DMA controller to run multiple DMA cycles, causing loss of data. A flip-flop is set by DTR/REQ whose output is DRQ1 to the 80186. The response of the 80186 to DRQ1 is a read or write at PCS5 address to do the DMA TRANSFER. This resets the flip-flop cutting off the DMA request to the 80186 which prevents false DMA transfer.

The DMA configurations supported by the interface are:
- Half-duplex on Channel A and Channel B
- Full-duplex on Channel A and no DMA on Channel B

D) INTA Processing

80186 generates 2 back-to-back INTA cycles in response to an interrupt and expects to read the interrupt vector on the second cycle. Two flip-flops (U1) are used to convert these two cycles to one INTA cycle and a RD pulse as required by the SCC. See timing diagram in Figure 5. SCC requires that the RD pulse is contained within the INTA pulse. This, along with the pulse width requirements for INTA and RD signals are easily met.
WAIT STATE REQUIREMENTS

The 82530 requires wait states in a normal single buffered system, as shown in Figure 6. They arise primarily due to the WR pulse width (= 390 ns) and its timing with respect to data valid as shown in Figure 4.

<table>
<thead>
<tr>
<th></th>
<th>SCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>82530</td>
<td></td>
</tr>
<tr>
<td>82530-6</td>
<td></td>
</tr>
<tr>
<td>(4 MHz)</td>
<td>2</td>
</tr>
<tr>
<td>(6 MHz)</td>
<td>1</td>
</tr>
</tbody>
</table>

80186-6 (6 MHz)

Processor

80186 (8 MHz)

2

<table>
<thead>
<tr>
<th>80186-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(6 MHz)</td>
</tr>
</tbody>
</table>

2

<table>
<thead>
<tr>
<th>80186</th>
</tr>
</thead>
<tbody>
<tr>
<td>(8 MHz)</td>
</tr>
</tbody>
</table>

3

Figure 6. Wait State Requirements

It is assumed in this interface design that the 80186 generates the chip selects and the appropriate number of wait states. In an 8086/88 system, chip select and wait states must be generated externally just as for all other peripheral components attached to the CPU.

The PCS6 chip select output from the 80186 is used to select the 82530 for all operations except to service DMA on Channel 1 of the 80186 when PCS5 is used. Note that it is necessary to pulse PCS5 signal before enabling the DMA Channel 1. This resets the DRQ1 flip-flop. A block for clock generator is also shown—although it is not considered a part of the CPU interface. It may be easily derived from CLKOUT.

The 4 TTL pack interface presented here covers all features of the SCC usage. In many cases the interface need not be as extensive as shown here and results in saving board space. Two cases where considerable saving is achieved are:

Case 1: System Using 8288

If the system uses an 8288 bus controller for 80186, pre-processing of WR input is not necessary and the IOWC output of 8288 can be fed directly to pin 5 of U2 (74LS08). This is because IOWC signal meets the timing requirements of the SCC. Also note, that the interface circuit is then totally independent of the 80186 clock.

Case 2: System Using Non-Vectored Interrupt Mode for SCC

Such a system will not need the component U1 (74LS74) nor the AND gate U2 (pins 11, 12, 13). Pin 3 of U2 can be fed directly to the RD input of SCC.

CONCLUSION

This four TTL package interface solution is low cost and compact (1.2 sq. inch). It should satisfy 82530 interfacing for almost all applications. In fact, as already mentioned, many applications may require only 2–3 TTL packages for interfacing the 82530 to 80186 or to other INTEL processors.
The 82501 Ethernet Serial Interface (ESI) chip is designed to work directly with the 82586 LAN Coprocessor in IEEE 802.3/Ethernet and non-Ethernet 10-MBps local-area network applications. The major functions of the 82501 are to generate the 10 MHz transmit clock for the 82586, perform Manchester encoding/decoding of the transmitted/received frames, and provide the electrical interface to the Ethernet transceiver cable. Diagnostic loopback control enables the 82501 to route the signal to be transmitted from the 82586 through its Manchester encoding and decoding circuitry and back to the 82586. The combined loopback capabilities of the 82586 and 82501 result in efficient fault detection and isolation by providing sequential testing of the communications interface. An on-chip fail-safe watchdog timer circuit prevents the station from locking up in a continuous transmit mode.
Transmit Clock: A 10-MHz clock output with 5 nsec rise and fall times. This clock is provided to the 82586 for serial transmission.

Transmit Enable: An active low, TTL-level signal synchronous to TXC that enables data transmission to the transceiver cable. TEN can be driven by RTS from the 82586.

Transmit Data: A TTL-level input signal that is directly connected to the serial data output, TXD, of the 82586.

Receive Clock: Clock output with 5 nsec rise and fall times and 50% duty cycle. This output is connected to the 82586 receive clock input RXC. There is a maximum 14 µsec discontinuity at the beginning of a frame reception when the phase-locked loop switches from the on-chip oscillator to the incoming data. During idle (no incoming frames) the clock frequency will be half that of the 20 MHz crystal frequency.

Carrier Sense: A TTL-level, active low input to notify the 82586 that there is activity on the coaxial cable. This signal is asserted when valid data or a collision signal from the transceiver is present. It is asserted at the end of a frame synchronous with RXC, or when the end of the collision-presence signal (CLSN and CLSN) is detected, whichever occurs later.

Receive Data: An MOS-level output tied directly to the RXD input of the 82586 controller and sampled by the 82586 at the negative edge of RXC. The bit stream received from the transceiver cable is Manchester decoded prior to being transferred to the controller. This output remains high during idle.

Collision Detect: A TTL, active low signal which drives the CDT input of the 82586 controller. It is asserted as long as there is activity on the collision-presence pair (CLSN and CLSN), and during SQE test in loopback.

Loopback: A TTL-level control signal to enable the loopback mode. In this mode, serial data on the TXD input is routed through the 82501 internal circuits and back to the RXD output without driving the TRMT output pair to the transceiver cable. When LPBK is asserted, the collision circuit will also be turned on at the end of each transmission to simulate the collision test. The on-chip watchdog timer can be disabled by applying a 12V level through a 4k ohm resistor to this pin. LPBK must not be asserted at power up to ensure proper CDT and CRS signals at 82586 startup.

Transmit Pair: An output driver pair which generates the differential signal for the transmit pair of the Ethernet transceiver cable. Following the last transition, which is always positive at TRMT, the differential voltage is slowly reduced to zero volts. The output stream is Manchester encoded.

Receive Pair: A differentially driven input pair which is tied to the receive pair of the Ethernet transceiver cable. The first transition on RCV will be negative-going to indicate the beginning of a frame. The last transition should be positive-going, indicating the end of a frame. The received bit stream is assumed to be Manchester encoded.

Collision Pair: A differentially driven input pair tied to the collision-presence pair of the Ethernet transceiver cable. The collision presence signal is a 10 MHz ± 15% square wave. The first transition at CLSN is negative-going to indicate the beginning of the signal, the last transition is positive-going to indicate the end of the signal.

PLL Capacitor: Phase-locked-loop capacitor inputs.

Clock Crystal: 20-MHz crystal inputs.

Power: 5 ± 10% volts.

Ground: Reference.
FUNCTIONAL DESCRIPTION

Clock Generation

A 20 MHz crystal-controlled oscillator is provided as the basic clock source. This 20 MHz signal is then divided by 2 to generate a 10 MHz ±0.01% clock as required in the IEEE 802.3 specification. The oscillator requires an external parallel resonant fundamental mode, 20 MHz crystal.

Manchester Encoder and Transceiver Cable Driver

The 20 MHz clock is used to Manchester encode data on the TXD input line. The clock is also divided by 2 to produce the 10 MHz clock required by the 82586 for synchronizing its RTS and TXD signals. See Figure 3. (Note that the 82586 RTS is tied to the 82501 TEN input as shown in Figure 4.)

Data encoding and transmission begins with TEN going low. Since the first bit is a '1', the first transition on the transmit output TRMT is always negative. Transmission ends with the TEN going high. The last transition is always positive at TRMT and may occur at the center of the bit cell (last bit = 1) or at the boundary of the bit cell (last bit = 0). A one-bit delay is introduced by the 82501 between its TXD input and TRMT/RTMT output as shown in Figure 3. Following the last transition, the output TRMT is slowly brought to its high state so that zero differential voltage exists between TRMT and RTMT. The undershoot for return to idle is less than 100 mV. This will eliminate DC currents in the primary of the transceiver's coupling transformer. See Figure 4.

An internal watchdog timer is started at the beginning of the frame. The duration of the watchdog timer is 25 msec ±15%. If the transmission terminates (by deasserting the TEN) before the timer expires, the timer is reset (and ready for the next transmission). If the timer expires before the transmission ends, the frame is aborted. This is accomplished by disabling the output driver for the TRMT/RTMT pair and deasserting CRS, RXD and RXC are not affected. The watchdog timer is reset only when the TEN is deasserted.

The cable driver is a differential gate requiring external resistors or a current sink of 20 mA (on both terminals). In addition, high-voltage protection of +16 volts maximum and short circuit to ground is provided.

Receive Section

CABLE INTERFACE AND NOISE FILTER

The 82501 input circuits can be driven directly from the Ethernet transceiver cable receive pair. In this case the cable is terminated with a 78-ohm resistor for proper impedance matching. The 82501 has internal resistors that establish the common mode voltage. See Figure 4.

The input circuits can also be driven with ECL voltage levels. In either case, the input common mode voltage must be in the range of 0 – VCC volts to allow for a wide driver supply variation at the transceiver. The input terminals have a 16-volt maximum protection and additional clamping of low-energy, high-voltage noise signals.

A noise filter is provided at the RCV/RCV input pair to prevent spurious signals from improperly triggering the receiver circuitry. The noise filter has the following characteristics:

A negative pulse which is narrower than 15 ns or is less than –150 mV in amplitude is rejected during idle.

At the beginning of a reception, the filter is turned off by the first negative pulse which is more negative than –275 mV and is wider than 30 ns.

As soon as the first valid negative pulse is recognized by the noise filter, the data threshold is lowered to 160 mV. The CRS signal is asserted to inform the 82586 controller of the beginning of a transmission, and the RXC will be held low for 1.4 μsec maximum while the internal phase-locked-loop is acquiring lock.

The frame is ended if no negative transition occurs within 160 ns from the last positive transition.

Immediately after the end of a transmission, the filter blocks all the signals for 5 μsec minimum, 7 μsec maximum. This dead time is required to block-off spurious transitions which may occur on the coaxial cable at the end of a transmission and are not filtered out by the transceiver.

MANCHESTER DECODER AND CLOCK RECOVERY

The filtered data enters the clock recovery and decoder circuits. An analog phase-locked-loop (PLL) technique is used to extract the received clock from the data, beginning from the third-negative transition of the incoming data. The PLL will acquire lock within the first 14 bit times, as seen from the RCV/RCV inputs. During that period of time, the RXC is held low. Bit cell timing distortion which can be tolerated in the incoming signal is ±15 nsec for the preamble and ±18 nsec for data. This distortion must have less than ±5 ns bias distortion. The voltage-controlled oscillator (VCO) of the PLL corrects its frequency to match the incoming signal transitions.
Its VCO cycle time stays within 5% of the RXD bit cell time regardless of the time distortion allowed at the RCV/RCV input. The RCV/RCV input is decoded from Manchester to NRZ and transferred synchronously with the receive clock to the 82586 controller.

At the end of a frame, the receive clock is used to detect the absence of RCV/RCV transitions and report it to the 82586 by deasserting CRS while RXD is held high.

**Collision-Presence Section**

The CLSN/CLSN input signal is a 10 MHz ±15% square wave generated by the transceiver whenever two or more data frames are superimposed on the coaxial cable. The maximum asymmetry in the CLSN/CLSN signal is 60%/40% for low-to-high or high-to-low levels. This signal is filtered for noise rejection in the same manner as RCV/RCV. The noise filter rejects signals which are less negative than -150 mV and narrower than 15 ns during idle. It turns on at the first negative pulse which is more negative than -275 mV and wider than 30 ns. After the initial turn-on, the filter remains active indicating that a valid collision signal is present, as long as the negative CLSN/CLSN signal pulses are more negative than -275 mV. The filter returns to the "off" state if the signal becomes less negative than -150 mV, or if no negative transition occurs within 160 ns from the last positive transition. Immediately after turn-off, the collision filter is ready to be reactivated.

The common mode voltage and external termination are identical to the RCV/RCV input (See Figure 4.) The CLSN/CLSN input also has a 16-volt maximum protection and additional clamping against low-energy, high-voltage noise signals.

A valid collision-presence signal will assert the 82501 CDT output which can be directly tied to the CDT input of the 82586 controller.

During the time that valid collision-presence transitions are present on the CLSN/CLSN input, invalid data transitions will be present on the receive data pair due to the superposition of signals from two or more stations transmitting simultaneously. It is possible for RCV/RCV to lose transitions for a few bit times due to perfect cancellation of the signals, which may cause the 82501 to abort the reception.

When a valid collision-presence signal is present the CRS signal is asserted (along with CDT). However, if this collision-presence signal arrives within 6.0 ± 1.0 μs from the last transmission only CDT is generated.

**Internal Loopback**

When asserted, LPBK causes the 82501 to route serial data from its TXD input, through its transmit logic (retiming and Manchester encoding), returning it through the receive logic (Manchester decoding and receive clock generation) to RXD output. The internal routing prevents the data from passing through the output drivers and onto the transmit output pair, TRMT/TRMT. When in loopback mode, all of the transmit and receive circuits, including the noise filter, are tested except for the transceiver cable output driver and input receivers. Also, at the end of each frame transmitted in loopback mode, the 82501 generates the SOE test (heartbeat) signal within 1 μsec after the end of the frame. Thus, the collision circuits, including the noise filter, are also tested in loopback mode.

The watchdog timer remains enabled in loopback mode, terminating test frames that exceed its time-out period. The watchdog can be inhibited by placing the LPBK to a 4K resistor connected to 12V ± 3V. The loopback feature can still be used to test the integrity of the 82501 by using the circuit shown in Figure 5.

In the normal mode (LPBK not asserted), the 82501 operates as a full duplex device, being able to transmit and receive simultaneously. This is similar to the external loopback mode of the 82586. Combining the internal and external loopback modes of the 82586 and the internal loopback and normal modes of the 82501, incremental testing of an 82586/82501-based interface can be performed under program control for systematic fault detection and fault isolation. LPBK must not be asserted at power up to ensure proper CDT and CRS signals to 82586 at start of operation.

**Interface Example**

The 82501 is designed to work directly with the 82586 controller in Ethernet as well as non-Ethernet 10 Mbps LAN applications. The control and data signals connect directly between the two devices without the need for additional external logic. The complete 82586/82501/Ethernet Transceiver cable interface is shown in FIGURE 4. The 82501 provides the driver and receivers needed to directly connect to the transceiver cable, requiring only terminating resistors on each input signal pair.
Figure 3. Start of Transmission and Manchester Encoding

Figure 4. 82586/82501 Transceiver Cable Interface

NOTE:
C1 = 0.022 μF ± 10%
C2 = C3 = 30 – 35 pF (including trace capacitance).
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ........ 0°C to 70°C
Storage Temperature .................. -65°C to +150°C
All Output and Supply Voltages ........ -0.5V to +7V
All Input Voltages .................. -1.0V to +5.5V
Power Dissipation .................. 1.5 Watt

“NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0-70°C, V_CC = 5V ± 10%)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_IL</td>
<td>Input Low Voltage (TTL)</td>
<td>-0.5</td>
<td>+0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_IH</td>
<td>Input High Voltage (TTL)</td>
<td>2.0</td>
<td>V_CC + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_IDF</td>
<td>Input Differential Voltage</td>
<td>± 300</td>
<td>±1500</td>
<td>mV</td>
<td>RCV and CLSN</td>
</tr>
<tr>
<td>V_CM</td>
<td>Input Common Mode Voltage</td>
<td>0</td>
<td>V_CC</td>
<td>V</td>
<td>RCV and CLSN</td>
</tr>
<tr>
<td>V_CL1</td>
<td>Output Low Voltage TTL or MOS</td>
<td>0.45</td>
<td>V</td>
<td>I_CL = 4 mA</td>
<td></td>
</tr>
<tr>
<td>V_CL2</td>
<td>Output Low Voltage TXC, RXC</td>
<td>0.6</td>
<td>V</td>
<td>I_CL = 4 mA</td>
<td></td>
</tr>
<tr>
<td>V_CM</td>
<td>Common Mode Output</td>
<td>1.0</td>
<td>4.5</td>
<td>V</td>
<td>R_L = 78 Ohms Differential Termination and 120Ω pulldown</td>
</tr>
<tr>
<td>V_COH1</td>
<td>Output High Voltage TTL</td>
<td>2.4</td>
<td>V</td>
<td>I_COH = -1.0 mA</td>
<td></td>
</tr>
<tr>
<td>V_COH2</td>
<td>Output High Voltage MOS</td>
<td>3.9</td>
<td>V</td>
<td>I_COH = -400μA</td>
<td></td>
</tr>
<tr>
<td>V_COH3</td>
<td>Output High Voltage RXC, TXC</td>
<td>3.3</td>
<td>V</td>
<td>I_COH = -400μA</td>
<td></td>
</tr>
<tr>
<td>V_ODF</td>
<td>Differential Output Swing</td>
<td>.6</td>
<td>1.1</td>
<td>V</td>
<td>R_L = 78 Ohms Differential Termination and 120Ω pulldown (TRMT)</td>
</tr>
<tr>
<td>I_CL1</td>
<td>Input Leakage Current (TTL)</td>
<td>+200</td>
<td>μA</td>
<td>V_IN = V_CC</td>
<td></td>
</tr>
<tr>
<td>C_IN</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td>f = 1 MHz</td>
<td></td>
</tr>
<tr>
<td>C_OUT</td>
<td>Output Capacitance</td>
<td>20</td>
<td>pF</td>
<td>f = 1 MHz</td>
<td></td>
</tr>
<tr>
<td>I_CC</td>
<td></td>
<td>225</td>
<td>mA</td>
<td>120Ω Pulldowns</td>
<td></td>
</tr>
<tr>
<td>I_P</td>
<td>Input Current (TTL)</td>
<td>-500</td>
<td>μA</td>
<td>V_F = .45V</td>
<td></td>
</tr>
</tbody>
</table>

Please Note: All specifications are preliminary values and are subject to change without notice. Contact your local Intel Sales Office for the latest specifications.

A.C. CHARACTERISTICS

A.C. Measurement Conditions

I) T_A = 0°C to 70°C, V_CC = 5V ± 10%

II) The AC measurements are done at the following voltage levels for the various kinds of inputs and outputs

   a) TTL inputs and outputs: 0.8V and 2.0V
      The input voltage swing is at least 0.4 to 2.4V
      with 3-10 ns rise and fall times.

   b) Clock outputs: The rise and fall times are measured between 0.6V and 3.0V points. The high time is measured between 3.0V points and the low time is measured between 0.6V points.

   c) Differential inputs and outputs:
      The 50% points of the total swing are used for delay measurements. The rise and fall times of outputs are measured at the 20 to 80% points. The differential voltage swing at the inputs is at least ±275mV with rise and fall times of 3-15 ns measured at ±2 volts. Once the squelch threshold has been exceeded the inputs will detect less than ±160mV signals.

III) The AC loads for the various kind of outputs are as follows:

   a) TTL and MOS: A 15-pF Capacitance to GND including test fixture and probe.

   b) Differential: A 10-pF Capacitance from each terminal to GND and a termination load resistor of 78 ohms in parallel with a 27 micro-henries inductor between the two terminals.
### TRANSMIT TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>TXC Cycle Time</td>
<td>99.99</td>
<td>100.01</td>
<td>ns</td>
</tr>
<tr>
<td>t2</td>
<td>TXC Fall Time</td>
<td></td>
<td>5</td>
<td>ns</td>
</tr>
<tr>
<td>t3</td>
<td>TXC Rise Time</td>
<td></td>
<td>5</td>
<td>ns</td>
</tr>
<tr>
<td>t4</td>
<td>TXC Low Time (at 0.9V)</td>
<td>40</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t5</td>
<td>TXC High Time (at 3.0V)</td>
<td>40</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t6</td>
<td>Transmit Enable/Disable to TXC Low</td>
<td>50</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t7</td>
<td>TXD Stable to TXC Low</td>
<td>50</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t8</td>
<td>Bit Cell Center to Bit Cell Center of Transmit Pair Data</td>
<td>99.5</td>
<td>100.5</td>
<td>ns</td>
</tr>
<tr>
<td>t9</td>
<td>Transmit Pair Data Fall Time [1]</td>
<td>1.0</td>
<td>5.0</td>
<td>ns</td>
</tr>
<tr>
<td>t10</td>
<td>Transmit Pair Data Rise Time [1]</td>
<td>1.0</td>
<td>5.0</td>
<td>ns</td>
</tr>
<tr>
<td>t11</td>
<td>Bit Cell Center to Bit Cell Boundary of Transmit Pair Data</td>
<td>49.5</td>
<td>50.5</td>
<td>ns</td>
</tr>
<tr>
<td>t12</td>
<td>TRMT held low from Last Positive Transition of Transmit Pair Data during idle:</td>
<td>200</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t12A</td>
<td>From Last Positive Transition of Transmit Pair Differential Output Approaches Within 40mV of zero volts.</td>
<td>8000</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

**Note:**

1. Measured per 802.3 Para 6.511
RECEIVE TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Parameter Description</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{13}</td>
<td>Receive Pair Signal Pulse Width (at -0.275V) differential signal of First Negative Pulse for a) Signal Rejection by Noise filter, b) Noise Filter Turn-on in order to Begin Reception</td>
<td></td>
<td>30 ns</td>
<td>15 ns</td>
<td>ns</td>
</tr>
<tr>
<td>t_{14}</td>
<td>Duration which the RXC is held at low state</td>
<td></td>
<td>1400 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{15}</td>
<td>Receive Pair Signal Rise/Fall Time at ± .2 volt</td>
<td></td>
<td>20 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{16}^{11}</td>
<td>Receive Pair Bit Cell Center from crossover timing distortion: In preamble In data</td>
<td></td>
<td>± 15 ns</td>
<td>± 18 ns</td>
<td>ns</td>
</tr>
<tr>
<td>t_{17}^{11}</td>
<td>Receive Pair Bit Cell Boundary allowing for timing distortion: In data</td>
<td></td>
<td>± 18 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{18}</td>
<td>Receive Idle Time Before the Next Reception can Begin in a transmitting station (as measured from the deassertion of CRS)</td>
<td></td>
<td>8 μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{19}</td>
<td>Receive Pair Signal Return to Zero Level from Last valid Positive Transition</td>
<td></td>
<td>160 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{20}</td>
<td>CRS Assertion delay from the First received valid Negative Transition of Receive Pair Signal</td>
<td></td>
<td>100 ns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Parameter Description</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{21}</td>
<td>CRS Deassertion delay from the Last valid positive transition received (when no Collision-Presence signal exists on the transceiver cable)</td>
<td></td>
<td>300^{2} ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{24}</td>
<td>RXC Jitter</td>
<td></td>
<td>± 5.0 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{25}</td>
<td>RXC Rise/Fall time</td>
<td></td>
<td>5.0 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{26}</td>
<td>RXC Low Time (at 0.9V)</td>
<td></td>
<td>40 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{26A}</td>
<td>RXC High Time (at 3.0V)</td>
<td></td>
<td>36 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{27}</td>
<td>Receive Data Stable before the Negative Edge of RXC</td>
<td></td>
<td>30 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{28}</td>
<td>Receive Data Held valid past the Negative Edge of RXC</td>
<td></td>
<td>30 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{29}</td>
<td>Carrier Sense Inactive Setup Time to RXC High</td>
<td></td>
<td>60 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{29A}</td>
<td>Carrier Sense Active Hold Time from RXC High</td>
<td></td>
<td>10 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{30}</td>
<td>Receive data Rise/Fall time</td>
<td></td>
<td>10 ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{31}</td>
<td>From the time CRS is deasserted until the time for a transmitting station it can be asserted again</td>
<td></td>
<td>5 μs to 7 μs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTES:
1. ± 5 ns of bias distortion—the remainder is random distortion.
2. CRS is deasserted synchronously with the RXC. This condition is not specified in the IEEE 802.3 specification.

Figure 5. Watchdog Timer Disable

<table>
<thead>
<tr>
<th>LPBK</th>
<th>WDTD</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>LPBK mode</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Normal mode</td>
</tr>
<tr>
<td>0</td>
<td>*1</td>
<td>Normal mode with watchdog timer disabled</td>
</tr>
</tbody>
</table>

* = Open Collector
RECEIVE TIMING: START OF FRAME

```
+ 1 0 1 0 1 0 1 1 +
```

```
13
15
15
16
17
```

**CRS**

**RXC**

**RXD**

*This clock pulse may not be a valid clock pulse*

---

RECEIVE TIMING: END OF FRAME

```
0 0 + + 2 2
```

```
0 1 + + 2 2
```

```
19
18
```

**CRS**

**RXC**

**RXD**

*Note: CRS can be triggered on again by the collision-presence signal*
### COLLISION TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_32</td>
<td>t_32 CLSN/CLSN Signal Pulse Width (at –30V differential signal) of first Negative Pulse for Noise Filter Turn-on</td>
<td>30</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_33</td>
<td>t_33 CLSN/CLSN Cycle Time</td>
<td>86</td>
<td>118</td>
<td>ns</td>
</tr>
<tr>
<td>t_34</td>
<td>t_34 CLSN/CLSN Rise/Fall Time at ±.2 volts</td>
<td>15</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_35</td>
<td>t_35 CLSN/CLSN Transition Time</td>
<td>35</td>
<td>70</td>
<td>ns</td>
</tr>
<tr>
<td>t_36</td>
<td>t_36 CDT Assertion from the First Valid Negative Edge of Collision Pair Signal</td>
<td>75</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_37</td>
<td>t_37 CDT Deassertion from the Last Positive Edge of CLSN/CLSN Signal</td>
<td>200</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_38</td>
<td>t_38 CRS Deassertion from the Last Positive Edge of CLSN/CLSN signal (If no post-collision signal remains on the receive pair.)</td>
<td>450</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_39</td>
<td>t_39 CRS Inactive after Collision Setup Time to RXC High</td>
<td>60</td>
<td>35</td>
<td>ns</td>
</tr>
<tr>
<td>t_39_A</td>
<td>t_39_A CRS Active Hold Time from RXC High after Collision</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

### NOTES:
1. CRS WILL BE DEASSERTED FOR A PERIOD UP TO 7 \( \mu \)SEC MAXIMUM WHEN RCV/RCV OR CLSN/CLSN TERMINATES, WHICHEVER OCCURS LATER
2. CRS WILL REMAIN ASSERTED AFTER THE CLSN/CLSN SIGNAL TERMINATES IF RCV/RCV SIGNALS CONTINUE

### COLLISION TIMING

![Diagram](image)

### LOOPBACK TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_40</td>
<td>t_40 LPBK asserted before the first attempted transmission</td>
<td>500</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_41</td>
<td>t_41 Simulated collision test delay from the end of each attempted transmission</td>
<td>.5</td>
<td>1.5</td>
<td>( \mu )S</td>
</tr>
<tr>
<td>t_42</td>
<td>t_42 Simulated collision test duration</td>
<td>.5</td>
<td>1.0</td>
<td>( \mu )S</td>
</tr>
<tr>
<td>t_43</td>
<td>t_43 LPBK deasserted after the last attempted transmission</td>
<td>5</td>
<td></td>
<td>( \mu )S</td>
</tr>
</tbody>
</table>

### NOTE:
In Loopback mode, RXC, RXD and CRS function in the same manner as a normal Receive.
LOOPBACK TIMING

NOTE:
1. DURING LOOPBACK, THE 82501 RECEIVE CIRCUITRY USES 12 BIT TIMES WHILE THE PLL LOCKS ON THE DATA. AS A RESULT, THE FIRST 12 BITS ARE LOST.

TESTABILITY

NOTES:
1. All AC parameters become valid after the PLL has stabilized: 100μs after the application of power.
82C502 Ethernet Transceiver Chip

- Conforms to IEEE 802.3, Ethernet Rev. 2, and Cheapernet Standards
  - Anti-jabber function
  - Receiver based collision detection
  - Signal Quality Error (heartbeat) test
  - Supports redundant jabber timer
- Requires minimum boardspace
  - On-chip voltage reference
  - 16 pin DIP
- No external adjustments required
- Reliable CHMOS technology

The 82C502 Ethernet Transceiver Chip is a CHMOS LSI device that provides the complete set of transmit, receive, and collision detection functions as specified by the IEEE 802.3/Ethernet and Cheapernet 10 Mbps standards. The 82C502 along with Intel's 82586 LAN Coprocessor and 82501 Ethernet Serial Interface realize highly integrated IEEE 802.3/Ethernet and Cheapernet systems.

The device consists of 3 sections: transmit, collision detection and receive. The transmit section transmits data received from the transceiver (AUI) cable to the 50Ω coax cable. The collision detect section detects collisions (simultaneous transmission of two or more stations) on the coax and indicates a collision by transmitting a 10 MHz signal on the collision presence pair of the transceiver cable. The receiver section receives data from the coax, and transmits the data onto the receive pair of the transceiver cable. The 82C502 can drive transceiver cables up to 50 meters in length.

The 82C502 is fabricated in Intel's reliable 10V CHMOS II process.
Applications

The 82C502 is intended to be used in high performance (10 Mbps) LAN applications such as IEEE 802.3/Ethernet and Cheapernet. IEEE 802.3/Ethernet require that the 82C502 transceiver chip be located in a tap box attached directly to the coax cable. A drop cable up to 50 meters in length connects the transceiver tap box to the data terminal equipment (DTE), see Figure 3.

In Cheapernet applications, the 82C502 would be located inside the DTE, and transformer coupled to the 82501, see Figure 4. In both applications, the IEEE specifications require that a DC isolated power supply power the 82C502.

Figure 3. IEEE 802.3/Ethernet configuration supports 100 users per segment, each segment is 500 meters long
Figure 4. IEEE 802.3 Cheapernet configuration supports 30 users per segment, each segment is 185 meters long
82586
LOCAL AREA NETWORK COPROCESSOR

- Performs Complete CSMA/CD Data Link Functions without CPU Overhead
  - High level command interface

- Supports Established and Emerging LAN Standards
  - IEEE 802.3/Ethernet
  - IEEE 802.3/Chevapernet
  - IBM PC Network (2 Mbps Broadband)
  - 1 Mbps Networks

- On-Chip Memory Management
  - Automatic buffer chaining saves memory
  - Reclaim of buffers after receipt of bad frames
  - Save bad frames

- Interfaces to 8-bit and 16-bit Microprocessors

- Supports Minimum Component Systems
  - Shared bus configuration
  - No TTL interface to iAPX 186 and 188 microprocessors

- Supports High Performance Systems
  - Bus master, with on-chip DMA
  - 4 MBytes/second bus bandwidth
  - Compatible with dual port memory
  - Back to back frame reception at 10 Mbps

- Network Diagnostics:
  - Frame CRC error tally
  - Frame alignment error tally
  - Location of cable opens/shorts
  - Collision tally

- Self Test Diagnostics
  - Internal loopback
  - External loopback
  - Internal register dump
  - Backoff timer check

---

**Figure 1. 82586 Functional Block Diagram**

**Figure 2. 82586 Pinout**

* IBM is a trademark of International Business Machines Corp.

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.

© INTEL CORPORATION, 1982

September, 1984

6-302

Order Number: 231246-001
The 82586 is an intelligent, high performance Local Area Network coprocessor, implementing the CSMA/CD link access method (Carrier Sense Multiple Access with Collision Detection).

The 82586 performs a large range of link management and channel interface functions including: CSMA/CD link access, framing, preamble generation and stripping, source address generation, destination address checking, CRC generation and checking. Any data rate up to 10 Mb/s can be used.

The 82586 features a powerful host system interface. It automatically manages memory structures with command chaining and bidirectional data chaining. An on-chip DMA controller manages 4 channels transparently to the user. Buffers containing errored or collided frames can be automatically recovered. The 82586 can be configured for 8-bit or 16-bit data path, with maximum burst transfer rate of 2 or 4 Mbyte/sec, respectively. Memory address space is 16 Mbyte maximum.

The 82586 provides two independent 16 byte FIFO’s, one for receiving and one for transmitting. The threshold for block transfer to/from memory is programmable, enabling the user to optimize bus overhead for a given worst case bus latency.

The 82586 provides a rich set of diagnostic and network management functions including: internal and external loopbacks, exception condition tallies, channel activity indicators, optimal capture of all frames regardless of destination address, optional capture of errored or collided frames, and time domain reflectometry for locating fault points in the cable.

The 82586 can be used in conjunction with either baseband or broadband networks. The controller can be configured for maximum network efficiency (minimum contention overhead) for any length network operating at any data rate within the 82586’s range. The controller supports address field lengths of 1, 2, 3, 4, 5, or 6 bytes. It can be configured for either the IEEE 802.3/ Ethernet or HDLC method of frame delineation. Both 16-bit and 32-bit CRC are supported.

The 82586 is packaged in a 48 pin DIP and fabricated in Intel’s reliable HMOS II 5 volt technology.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC, VCC</td>
<td>48, 36</td>
<td></td>
<td>System Power: +5 volt power supply.</td>
</tr>
<tr>
<td>VSS, VSS</td>
<td>12, 24</td>
<td></td>
<td>System Ground.</td>
</tr>
<tr>
<td>RESET</td>
<td>34</td>
<td>I</td>
<td>RESET is an active HIGH internally synchronized signal, causing the 82586 to terminate present activity immediately. The signal must be HIGH for at least four clock cycles. The 82586 will execute RESET within ten system clock cycles starting from RESET HIGH. When RESET returns LOW, the 82586 waits for the first CA to begin the initialization sequence.</td>
</tr>
<tr>
<td>TxD</td>
<td>27</td>
<td>0</td>
<td>Transmitted Serial Data output signal. This signal is HIGH when not transmitting.</td>
</tr>
<tr>
<td>TxC</td>
<td>26</td>
<td>I</td>
<td>Transmit Data Clock. This signal provides timing information to the internal serial logic, depending upon the mode of data transfer. For NRZ mode of operation, data is transferred to the TxD pin on the HIGH to LOW clock transition.</td>
</tr>
<tr>
<td>RxD</td>
<td>25</td>
<td>I</td>
<td>Received Data input signal.</td>
</tr>
<tr>
<td>RxC</td>
<td>23</td>
<td>I</td>
<td>Received Data Clock. This signal provides timing information to the internal shifting logic depending upon the mode of data transfer. For NRZ data, the state of the RxD pin is sampled on the HIGH to LOW clock transition.</td>
</tr>
<tr>
<td>RTS</td>
<td>28</td>
<td>0</td>
<td>Request To Send signal. When LOW, notifies an external interface that the 82586 has data to transmit. It is forced HIGH after a Reset and while the Transmit Serial Unit is not sending data.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Pin No.</td>
<td>Type</td>
<td>Name and Function</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>------</td>
<td>-------------------</td>
</tr>
<tr>
<td>CTS</td>
<td>29</td>
<td>I</td>
<td>Active LOW Clear To Send input enables the 82586 transmitter to actually send data. It is normally used as an interface handshake to RTS. This signal going inactive stops transmission. It is internally synchronized. If CTS goes inactive, meeting the setup time to TXC negative edge, transmission is stopped and RTS goes inactive within, at most, two TXC cycles.</td>
</tr>
<tr>
<td>CRS</td>
<td>31</td>
<td>I</td>
<td>Active LOW Carrier Sense input used to notify the 82586 that there is traffic on the serial link. It is used only if the 82586 is configured for external Carrier Sense. When so configured, external circuitry is required for detecting serial link traffic. It is internally synchronized. To be accepted, the signal must stay active for at least two serial clock cycles.</td>
</tr>
<tr>
<td>CDT</td>
<td>30</td>
<td>I</td>
<td>Active LOW Collision Detect input is used to notify the 82586 that a collision has occurred. It is used only if the 82586 is configured for external Collision Detect. External circuitry is required for detecting the collision. It is internally synchronized. To be accepted, the signal must stay active for at least two serial clock cycles. During transmission, the 82586 is able to recognize a collision one bit time after preamble transmission has begun.</td>
</tr>
<tr>
<td>INT</td>
<td>38</td>
<td>0</td>
<td>Active HIGH Interrupt request signal.</td>
</tr>
<tr>
<td>CLK</td>
<td>32</td>
<td>I</td>
<td>The system clock input from the 80186 or another symmetric clock generator.</td>
</tr>
<tr>
<td>MN/MX</td>
<td>33</td>
<td>I</td>
<td>When HIGH, MN/MX selects RD, WR, ALE, DEN, DT/R (Minimum Mode). When LOW, MN/MX selects A22, A23, READY, SO, ST (Maximum Mode). Note: This pin should be static during 82586 operation.</td>
</tr>
<tr>
<td>AD0 - AD15</td>
<td>6-11, 13-22</td>
<td>I/O</td>
<td>These lines form the time multiplexed memory address (t1) and data (t2, t3, tW, t4) bus. When operating with an 8-bit bus, the high byte will output the address during the entire cycle. AD0-AD15 are floated after a RESET or when the bus is not acquired.</td>
</tr>
<tr>
<td>A16-A18, A20-A23</td>
<td>1, 3-5, 45-47</td>
<td>0</td>
<td>Used maximum mode only. These lines constitute 7 out of 8 most significant address bits for memory operation. They switch during t1 and stay valid during the entire memory cycle. The lines are floated after RESET or when the bus is not acquired.</td>
</tr>
<tr>
<td>A19/S6</td>
<td>2</td>
<td>0</td>
<td>During t1 it forms line 19 of the memory address. During t2 through t4 it is used as a status indicating that this is a Master peripheral cycle, and is HIGH. Its timing is identical to that of AD0 - AD15 during write operation.</td>
</tr>
<tr>
<td>HOLD</td>
<td>43</td>
<td>0</td>
<td>HOLD is an active HIGH signal used by the 82586 to request local bus mastership at the end of the current CPU bus transfer cycle, or at the end of the current DMA burst transfer cycle. In normal operation, HOLD goes inactive before HLDA. The 82586 can be forced off the bus by HLDA going inactive. In this case, HOLD goes inactive, at most, three bus cycles after HLDA goes inactive.</td>
</tr>
<tr>
<td>HLDA</td>
<td>42</td>
<td>I</td>
<td>HLDA is an active HIGH Hold Acknowledge signal indicating that the CPU has received the HOLD request and that bus control has been relinquished to the 82586. It is internally synchronized. After HOLD is detected as LOW, the processor drives HLDA LOW. Note, CONNECTING VCC TO HLDA IS NOT ALLOWED because it will cause a deadlock. Users wanting to give permanent bus access to the 82586 should connect HLDA with HOLD. If HLDA goes inactive before HOLD, the 82586 will release the bus (by HOLD going inactive), within three bus cycles at most.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Pin No.</td>
<td>Type</td>
<td>Name and Function</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>------</td>
<td>------------------</td>
</tr>
<tr>
<td>CA 35</td>
<td>I</td>
<td></td>
<td>The CA pin is a Channel Attention input used by the CPU to initiate the 82586 execution of memory resident Command Blocks. The CA signal is synchronized internally. The signal must be HIGH for at least one system clock period. It is latched internally on HIGH to LOW edge and then detected by the 82586.</td>
</tr>
<tr>
<td>BHE 44</td>
<td>0</td>
<td></td>
<td>The Bus High Enable signal (BHE) is used to enable data onto the most significant half of the data bus. Its timing is identical to that of A16-A23. With a 16-bit bus it is LOW and with an 8-bit bus it is HIGH. Note: after RESET, the 82586 is configured to 8-bit bus.</td>
</tr>
<tr>
<td>READY 39</td>
<td>I</td>
<td></td>
<td>This active HIGH signal is the acknowledgement from the addressed memory that the transfer cycle can be completed. While LOW, it causes wait states to be inserted. This signal must be externally synchronized with the system clock. The Ready signal internal to the 82586 is a logical OR between READY and SRDY/ARDY.</td>
</tr>
<tr>
<td>SRDY/ARDY 37</td>
<td>I</td>
<td></td>
<td>This active HIGH signal performs the same function as READY. If it is programmed at configure time to SRDY, it is identical to READY. If it is programmed to ARDY, the positive edge of the Ready signal is internally synchronized. Note, the negative edge must still meet setup and hold time specifications, when in ARDY mode. The ARDY signal must be active for at least one system clock HIGH period for proper strobing. The Ready signal internal to the 82586 is a logical OR between READY (in Maximum Mode only) and SRDY/ARDY. Note that following RESET, this pin assumes ARDY mode.</td>
</tr>
</tbody>
</table>
| S0, S1 40,41 | 0 |      | Maximum mode only. These status pins define the type of DMA transfer during the current memory cycle. They are encoded as follows:  

| S1 | S0 | Status  
|----|----|-------|
| 0  | 0  | Not Used  
| 0  | 1  | Read Memory  
| 1  | 0  | Write Memory  
| 1  | 1  | Passive  

Status is active from the middle of t4 to the end of t2. They return to the passive state during t3 or during tW when READY or ARDY is HIGH. These signals can be used by the 8288 Bus Controller to generate all memory control and timing signals. Any change from the passive state signals the 8288 to start the next t1 to t4 bus cycle. These pins are pulled HIGH and floated after a system RESET and when the bus is not acquired. |
| RD 46 | 0 |      | Used in minimum mode only. The read strobe indicates that the 82586 is performing a memory read cycle. RD is active LOW during t2, t3 and tW of any read cycle. This signal is pulled HIGH and floated after a RESET and when the bus is not acquired. |
| WR 45 | 0 |      | Used in minimum mode only. The write strobe indicates that the 82586 is performing a write memory cycle. WR is active LOW during t2, t3 and tW of any write cycle. It is pulled HIGH and floats after RESET and when the bus is not acquired. |
| ALE 39 | 0 |      | Used in minimum mode only. Address Latch Enable is provided by the 82586 to latch the address into the 8282/8283 address latch. It is a HIGH pulse, during t1 ('clock low') of any bus cycle. Note that ALE is never floated. |
| DEN 40 | 0 |      | Used in minimum mode only. Data ENable is provided as output enable for the 8286/8287 transceivers in a stand-alone (no 8288) system. DEN is active LOW during each memory access. For a read cycle, it is active from the middle of t2 until the beginning of t4. For a write cycle, it is active from the beginning of t2 until the middle of t4. It is pulled HIGH and floats after a system RESET or when the bus is not acquired. |
Table 1. 82586 Pin Description (Cont'd.)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT/R</td>
<td>41</td>
<td>0</td>
<td>Used in minimum mode only. DT/R is used in non-8288 systems using an 8286/8287 data bus transceiver. It controls the direction of data flow through the Transceiver. Logically, DT/R is equivalent to S1. It becomes valid in the t4 preceding a bus cycle and remains valid until the final t4 of the cycle. This signal is pulled HIGH and floated after a RESET or when the bus is not acquired.</td>
</tr>
</tbody>
</table>

82586/HOST CPU INTERACTION

Communication between the 82586 and the host is carried out via shared memory. The 82586's direct access to memory capability allows autonomous transfer of data blocks (buffers, frames) and relieves the CPU of byte transfer overhead. The 82586 is optimized for operating with the iAPX 186, but due to the small number of hardware signals between the 82586 and the CPU, the 82586 can operate easily with other processors. In discussing 82586/Host interaction, the logical interface and the hardware bus interface are referred to separately.

The 82586 consists of two independent units: Command Unit (CU) and Receive Unit (RU). The CU executes commands from shared memory. The RU handles all activities related to frame reception. The CU and RU enable the 82586 to engage in the two activities simultaneously: the CU may be fetching and executing commands out of memory, and the RU may be storing received frames in memory. CPU intervention is only required after the CU executes a sequence of commands or the RU stores a sequence of frames.

The only hardware signals that connect the CPU and the 82586, are the INTERRUPT and CHANNEL ATTENTION, see Figure 3. Interrupt is used by the 82586 to draw the CPU’s attention to a change in the SCB. The Channel Attention is used by the CPU to draw the 82586’s attention.

82586 SYSTEM MEMORY STRUCTURE

The Shared Memory structure is composed of four parts: Initialization Root, System Control Block (SCB), Command List, and Receive Frame Area (RFA), see Figure 4.

The Initialization Root is at a predetermined location in the memory space, (OFFFFF6H), known to both the host the CPU and the 82586. The root is accessed at initialization and points to the System Control Block.

The System Control Block (SCB) serves as a bidirectional mailbox between the host the CPU, CU and RU. It is the central element through which the CPU and the 82586 exchange control and status information.
The SCB is composed of two parts. First, instructions from the CPU to the 82586. These include: control of the CU and RU (START, ABORT, SUSPEND, RESUME), a pointer to the list of commands for the CU, a pointer to the receive frame area, and a set of interrupt acknowledge bits. Second, information from the 82586 to the CPU that includes: state of the CU and RU (e.g. IDLE, ACTIVE READY, SUSPENDED, NO RECEIVE RESOURCES), interrupt bits (command completed, frame received, CU gone not ready, RU gone not ready), and statistics. See Figure 4.

The Command List serves as a program for the CU. Individual commands are placed in memory units called a Command Block, or CB. CB's contain command specific parameters and command specific statuses. Specifically, these high level commands are called Action Commands (e.g. Transmit, Configure).

A specific command, Transmit, causes transmission of a frame by the 82586. The Transmit command block includes Destination Address, Type Field, and a pointer to a list of linked buffers that holds the frame to be constructed from several buffers scattered in memory. The Command Unit performs,
without the CPU intervention, the DMA of each buffer and the prefetching of references to new buffers in parallel. The CPU is notified only after successful transmission or retransmission.

The Receive Frame Area is a list of Free Frame Descriptors (Descriptors not yet used) and a list of buffers prepared by the user. It is conceptually distinct from the Command List. Frames arrive without being solicited by the 82586. The 82586 must be prepared to receive them even if it is engaged in other activities and to store them in the Free Frame Area. The Receive Unit fills the buffers upon frame reception and reformats the Free Buffer List into received frame structures. The frame structure is virtually identical to the format of the frame to be transmitted. The first frame descriptor is referenced by SCB, and the reclaimed and returned to the Free Buffer List, unless the chip is configured to Save Bad Frames.

Receive buffer chaining (i.e. storing incoming frames in a linked list of buffers) improves memory utilization significantly. Without buffer chaining, the user must allocate consecutive blocks of the maximum frame size (1518 bytes in Ethernet) for each frame. Taking into account that a typical frame size may be about 100 bytes, this practice is very inefficient. With buffer chaining, the user can allocate small buffers and the 82586 uses only as many as needed.

In the past, the drawback of buffer chaining was the CPU processing overhead and the time involved in the buffer switching (especially at 10 Mb/s). The 82586 overcomes this drawback by performing buffer management on its own (completely transparent to the user).

The 82586 has a 22-bit memory address range in minimum mode and 24-bit memory address range in maximum mode. All memory structures, the System Control Block, Command List, Receive Descriptor List, and all buffer descriptors must reside within one 64K-byte memory segment. The Data Buffers can be located anywhere in the memory space.

**TRANSMITTING FRAMES**

The 82586 executes high level or action commands from the Command List in external memory. Action commands are fetched and executed in parallel with the host CPU's operation, thereby significantly improving system performance. The general action commands format is shown in Figure 5.

Message transmission is accomplished by using the Transmit command. A single Transmit command contains, as part of the command-specific parameters, the destination address and type field for the transmitted frame along with a pointer to a buffer area in memory containing the data portion of the frame. (See Figure 15.) The data field is contained in a memory data structure consisting of a Buffer Descriptor (BD) and Data Buffer (or a linked list of buffer descriptors and buffers) as shown in Figure 6. The BD contains a Link Field which points to the next BD on the list and a 24-bit address pointing to the Data Buffer itself. The length of the Data Buffer is specified by the Actual Count field of the BD.
Using the BD's and Data Buffers, multiple Data Buffers can be 'chained' together. Thus, a frame with a long Data Field can be transmitted using multiple (shorter) Data Buffers chained together. This chaining technique allows the system designer to develop efficient buffer management policies.

When transmitting a frame as shown below in Figure 7:

![Figure 7. Frame Format](image)

The 82586 automatically generates the preamble (alternating 1's and 0's) and start frame delimiter, fetches the destination address and type field from the Transmit command, inserts its unique address as the source address, fetches the data field from buffers pointed to by the Transmit command, and computes and appends the CRC at the end of the frame.

The 82586 can be configured to generate either the Ethernet or HDLC start and end frame delimiters. In the Ethernet mode, the start frame delimiter is two consecutive 1 bits and the end frame delimiter indicated by the lack of a signal after transmitting the last bit of the frame-check sequence field. When in the HDLC mode, the 82586 will generate the 01111110 'flag' for the start and end frame delimiters and perform the standard 'bit stuffing/stripping.' In addition, the 82586 will optionally pad frames that are shorter than the specified minimum frame length by appending the appropriate number of flags to the end of the frame.

In the event of a collision (or collisions), the 82586 manages the entire jam, random wait and retry process, reinitializing DMA pointers without CPU intervention. Multiple frames can be sent by linking the appropriate number of Transmit commands together. This is particularly useful when transmitting a message that is larger than the maximum frame size (1518 bytes for Ethernet).

**RECEIVING FRAMES**

In order to minimize CPU overhead, the 82586 is designed to receive frames without CPU supervision. The host CPU first sets aside an adequate amount of receive buffer space and then enables the
82586's Receive Unit. Once enabled, the RU 'watches' for any of its frames which it automatically stores in the Receive Frame Area (RFA). The RFA consists of a Receive Descriptor List (RDL) and a list of free buffers called the Free Buffer List (FBL) as shown in Figure 8. The individual Receive Frame Descriptors that make up the RDL are used by the 82586 to store the destination and source address, type field and status of each frame that is received. (Figure 9.)

![Receive Frame Descriptor](image)

Figure 9. Receive Frame Descriptor

The 82586, once enabled, checks each passing frame for an address match. The 82586 will recognize its own unique address, one or more multicast addresses or the broadcast address. If a match occurs, it stores the destination and source address and type field in the next available RFD. It then begins filling the next free Data Buffer on the FBL (which is pointed to by the current RFD) with the data portion of the incoming frame. As one DB is filled, the 82586 automatically fetches the next DB on the FBL until the entire frame is received. This buffer chaining technique is particularly memory efficient because it allows the system designer to set aside buffers that fit a frame size that may be much shorter than the maximum allowable frame.

Once the entire frame is received without error, the 82586 performs the following housekeeping tasks:

- Updates the Actual Count field of the last Buffer Descriptor used to hold the frame just received with the number of bytes stored in its associated Data Buffer.
- Fetches the address of the next free Receive Frame Descriptor.
- Writes the address of the next free Buffer Descriptor into the next free Receive Frame Descriptor.
- Posts a 'Frame Received' interrupt status bit in the SCB.
- Interrupts the CPU.

In the event of a frame error, such as a CRC error, the 82586 automatically reinitializes its DMA pointers and reclaims any data buffers containing the bad frame. As long as Receive Frame Descriptors and data buffers are available, the 82586 will continue to receive frames without further CPU help.

82586 NETWORK MANAGEMENT AND DIAGNOSTIC FUNCTIONS

The behavior of data communication networks is typically very complex due to their distributed and asynchronous nature. It is particularly difficult to pin-point a failure when it occurs. The 82586 was designed in anticipation of these problems and includes a set of features for improving reliability and testability.

The 82586 reports on the following events after each frame transmitted:

- Transmission successful.
- Transmission unsuccessful; lost Carrier Sense.
- Transmission unsuccessful; lost Clear-to-Send.
- Transmission unsuccessful; DMA underrun because the system bus did not keep up with the transmission.
- Transmission unsuccessful; number of collisions exceeded the maximum allowed.

The 82586 checks each incoming frame and reports on the following errors, (if configured to 'Save Bad Frame'):

- CRC error: incorrect CRC in a well aligned frame.
- Alignment error: incorrect CRC in a misaligned frame.
- Frame too short: the frame is shorter than the configured value for minimum frame length.
- Overrun: the frame was not completely placed in memory because the system bus did not keep up with incoming data.
- Out of buffers: no memory resources to store the frame, so part of the frame was discarded.

NETWORK PLANNING AND MAINTENANCE

To perform proper planning, operation, and maintenance of a communication network, the network management entity must accumulate information on network behavior. The 82586 provides a rich set of network-wide diagnostics that can serve as the basis for a network management entity.
Network Activity information is provided in the status of each frame transmitted. The activity indicators are:

- Number of collisions: number of collisions the 82586 experienced in attempting to transmit this frame.
- Deferred transmission: indicates if the 82586 had to defer to traffic on the link during the first transmission attempt.

Statistics registers are updated after each received frame that passes the address filtering, and is longer than the Minimum Frame Length configuration parameter.

- CRC errors: number of frames that experienced a CRC error and were properly aligned.
- Alignment errors: number of frames that experienced a CRC error and were misaligned.
- No-resources: number of correct frames lost due to lack of memory resources.
- Overrun errors: number of frame sequences lost due to DMA overrun.

The 82586 can be configured to Promiscuous Mode. In this mode it captures all frames transmitted on the Network without checking the Destination Address. This is useful in implementing a monitoring station to capture all frames for analysis.

The 82586 is capable of determining if there is a short or open circuit anywhere in the Network using the built-in Time Domain Reflectometer (TDR) mechanism.

**STATION DIAGNOSTICS**

The chip can be configured to External Loopback. The transmitter to receiver interconnection can be placed anywhere between the 82586 and the link to locate faults, for example: the 82586 output pins, the Serial Interface Unit, the Transceiver cable, or in the Transceiver.

The 82586 has a mechanism recognizing the transceiver 'heart beat' signal for verifying the correct operation of the Transceiver's collision detection circuitry.

**82586 SELF TESTING**

The 82586 can be configured to Internal Loopback. It disconnects itself from the Serial Interface Unit, and any frame transmitted is received immediately. The 82586 connects the Transmit Data to the Receive Data signal and the Transmit Clock to the Receive Clock.

The Dump Command causes the chip to write over 100 bytes of its internal registers to memory.

The Diagnose command checks the exponential Backoff random number generator internal to the 82586.

**CONTROLLING THE 82586**

The CPU controls operation of the 82586's Command Unit (CU) and Receive Unit (RU) of the 82586 via the System Control Block.

**THE COMMAND UNIT (CU)**

The Command Unit is the logical unit that executes Action Commands from a list of commands very similar to a CPU program. A Command Block (CB) is associated with each Action Command.

The CU can be modeled as a logical machine that takes, at any given time, one of the following states:

- **IDLE** - CU is not executing a command and is not associated with a CB on the list. This is the initial state.
- **SUSPENDED** - CU is not executing a command but (different from IDLE) is associated with a CB on the list.
- **ACTIVE** - CU is currently executing an Action Command, and points to its CB.

The CPU may affect the CU operation in two ways: issuing a CU control Command or setting bits in the COMMAND word of the Action Command.

**THE RECEIVE UNIT (RU)**

The Receive Unit is the logical unit that receives frames and stores them in memory.

The RU is modeled as a logical machine that takes, at any given time, one of the following states:

- **IDLE** - RU has no memory resources and is discarding incoming frames. This is the initial RU state.
- **NO-RESOURCES** - RU has no memory resources and is discarding incoming frames. This state differs from the IDLE state in that RU accumulates statistics on the number of frames it had to discard.
- **SUSPENDED** - RU has free memory resources to store incoming frames but discards them anyway.
- **READY** - RU has free memory resources and stores incoming frames.

The CPU may affect RU operation in three ways: issuing an RU Control Command, setting bits in Frame Descriptor, FD, COMMAND word of the frame currently being received, or setting EL bit of Buffer Descriptor, BD, of the buffer currently being filled.
SYSTEM CONTROL BLOCK (SCB)

The System Control Block is the communication mail-box between the 82586 and the host CPU. The SCB format is shown in Figure 10.

The host CPU for issuing Control Commands to the 82586 via the SCB. These commands may appear at any time during routine operation, as determined by the host CPU. After the required Control Command is setup, the CPU sends a CA signal to the 82586.

SCB is also used by the 82586 to return status information to the host CPU. After inserting the required status bits into SCB, the 82586 issues an Interrupt to the CPU.

The format is as follows:

**STATUS word:** Indicates the status of the 82586. This word is modified only by the 82586. Defined bits are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>CX (Command in the CBL having its 'I' (interrupt) bit set has been executed.</td>
</tr>
<tr>
<td>14</td>
<td>FR (A frame has been received.</td>
</tr>
<tr>
<td>13</td>
<td>CNR (The Command Unit left the Active state.</td>
</tr>
</tbody>
</table>

**COMMAND word:** Specifies the action to be performed as a result of the CA. This word is set by the CPU and cleared by the 82586. Defined bits are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>ACK-CX (Acknowledges the command executed event.</td>
</tr>
</tbody>
</table>

---

Figure 10. System Control Block (SCB) Format
<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK-FR (Bit 14)</td>
<td>2</td>
<td>Acknowledges the frame received event.</td>
</tr>
<tr>
<td>ACK-CNA (Bit 13)</td>
<td>3</td>
<td>Acknowledges that the Command Unit became not ready.</td>
</tr>
<tr>
<td>ACK-RNR (Bit 12)</td>
<td>4</td>
<td>Acknowledges that the Receive Unit became not ready.</td>
</tr>
<tr>
<td>CUC (Bits 8-10)</td>
<td>5-7</td>
<td>This field contains the command to the Command Unit. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 - NOP (doesn't affect current state of the unit).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - Start execution of the first command on the CBL. If a command is in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>execution, then complete it before starting the new CBL. The beginning of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the CBL is in CBL OFFSET.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - Resume the operation of the command unit by executing the next command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This operation assumes that the command unit has been previously suspended.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 - Suspend execution of commands on CBL after current command is complete.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 - Abort execution of commands immediately</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5-7 - Reserved, illegal for use.</td>
</tr>
<tr>
<td>RESET (Bit 7)</td>
<td>2</td>
<td>Reset chip (logically the same as hardware RESET).</td>
</tr>
</tbody>
</table>

**CBL-OFFSET:**
gives the 16-bit offset address of the first command (Action Command) in the command list to be executed following CU-START. Thus, the 82586 reads this word only if the CUC field contained a CU-START Control Command.

**RFA-OFFSET:**
Points to the first Receive Frame Descriptor in the Receive Frame Area

**CRCERRS:**
CRC Errors - contains the number of properly aligned frames received with a CRC error.

**ALNERRS:**
Alignment Errors - contains the number of misaligned frames received with a CRC error.

**RSCERRS:**
Resource Errors - records the number of correct incoming frames discarded due to lack of memory resources (buffer space or received frame descriptors).

**OVRNERRS:**
Overrun Errors - counts the number of received frame sequences lost because the memory bus was not available in time to transfer them.

**ACTION COMMANDS**
The 82586 executes a 'program' that is made up of action commands in the Command List. As shown in Figure 5, each command contains the command field, status and control fields, link to the next action command in the CL, and any command-specific parameters. This command format is called the Command Block.
The 82586 has a repertoire of 8 commands:

- NOP
- Setup Individual Address
- Configure
- Setup Multicast Address
- Transmit
- TDR
- Diagnose
- Dump

NOP

This command results in no action by the 82586, except as performed in normal command processing. It is present to aid in Command List manipulation.

NOP command includes the following fields:

### STATUS word (written by 82586):

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Command completed</td>
</tr>
<tr>
<td>14</td>
<td>Busy executing command</td>
</tr>
<tr>
<td>13</td>
<td>Error free completion</td>
</tr>
</tbody>
</table>

### COMMAND word:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>End of command list</td>
</tr>
<tr>
<td>14</td>
<td>Suspend after completion</td>
</tr>
<tr>
<td>13</td>
<td>Interrupt after completion</td>
</tr>
<tr>
<td>0-2</td>
<td>NOP = 0</td>
</tr>
</tbody>
</table>

### LINK OFFSET: Address of next Command Block

IA-SETUP

This command loads the 82586 with the Individual Address. This address is used by the 82586 for recognition of Destination Address during reception and insertion of Source Address during transmission.

The IA-SETUP command includes the following fields:

### 82586 Block Diagram

- **NOP Command Block**
- **IA-SETUP Command Block**
STATUS word (written by 82586):

<table>
<thead>
<tr>
<th>C</th>
<th>(Bit 15)</th>
<th>Command completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>(Bit 14)</td>
<td>Busy executing</td>
</tr>
<tr>
<td>OK</td>
<td>(Bit 13)</td>
<td>Error free</td>
</tr>
<tr>
<td>A</td>
<td>(Bit 12)</td>
<td>Command aborted</td>
</tr>
</tbody>
</table>

COMMAND word:

<table>
<thead>
<tr>
<th>EL</th>
<th>(Bit 15)</th>
<th>End of command list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>(Bit 14)</td>
<td>Suspend after</td>
</tr>
<tr>
<td>I</td>
<td>(Bit 13)</td>
<td>Interrupt after</td>
</tr>
<tr>
<td>CMD</td>
<td>(Bits 0-2)</td>
<td>IA-SETUP = 1</td>
</tr>
</tbody>
</table>

**LINK OFFSET**: Address of next Command Block

**INDIVIDUAL ADDRESS**: Individual Address parameter

The least significant bit of the Individual Address parameter must be zero for IEEE 802.3/Ethernet. However, no enforcement of 0 is provided by the 82586. Thus, an Individual Address with least significant bit 1, is possible.

### CONFIGURE

The CONFIGURE command is used to update the 82586 operating parameters.

The CONFIGURE command includes the following fields.

**STATUS word (written by 82586):**

<table>
<thead>
<tr>
<th>C</th>
<th>(Bit 15)</th>
<th>Command completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>(Bit 14)</td>
<td>Busy executing</td>
</tr>
<tr>
<td>OK</td>
<td>(Bit 13)</td>
<td>Error free</td>
</tr>
<tr>
<td>A</td>
<td>(Bit 12)</td>
<td>Command aborted</td>
</tr>
</tbody>
</table>

**COMMAND word:**

<table>
<thead>
<tr>
<th>EL</th>
<th>(Bit 15)</th>
<th>End of command list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>(Bit 14)</td>
<td>Suspend after</td>
</tr>
<tr>
<td>I</td>
<td>(Bit 13)</td>
<td>Interrupt after</td>
</tr>
<tr>
<td>CMD</td>
<td>(Bits 0-2)</td>
<td>Configure = 2</td>
</tr>
</tbody>
</table>

**LINK OFFSET**: Address of next Command Block

**Byte 6-7:**

| BYTE CNT| (Bits 0-3) | Byte Count, Number of bytes including this one, holding the parameters to be configured. A number smaller than 4 is interpreted as 4. A number greater than 12 is interpreted as 12. |

Figure 13. The CONFIGURE Command Block
**82586**

**Preliminary**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FIFO-LIM</strong></td>
<td>(Bits 8-11) - Value of FIFO Threshold.</td>
</tr>
<tr>
<td><strong>Byte 8-9:</strong></td>
<td></td>
</tr>
<tr>
<td>SRDY/ARDY</td>
<td>(Bit 6)</td>
</tr>
<tr>
<td>0</td>
<td>- SRDY/ARDY pin operates as ARDY (internal synchronization).</td>
</tr>
<tr>
<td>1</td>
<td>- SRDY/ARDY pin operates as SRDY (external synchronization).</td>
</tr>
<tr>
<td>SAV-BF</td>
<td>(Bit 7)</td>
</tr>
<tr>
<td>0</td>
<td>- Received bad frames are not saved in memory.</td>
</tr>
<tr>
<td>1</td>
<td>- Received bad frames are saved in memory.</td>
</tr>
<tr>
<td>ADDR-LEN</td>
<td>(Bits 8-10) - Number of address bytes. NOTE: 7 is interpreted as 0.</td>
</tr>
<tr>
<td>AT-LOC</td>
<td>(Bit 11)</td>
</tr>
<tr>
<td>0</td>
<td>- Address and Type Fields separated from data and associated with Transmit Command Block or Receive Frame Descriptor. For transmitted frame, Source Address is inserted by the 82586.</td>
</tr>
<tr>
<td>1</td>
<td>- Address and Type Fields are part of the Transmit/Receive data buffers, including Source Address (which is not inserted by the 82586).</td>
</tr>
<tr>
<td>PREAM-LEN</td>
<td>(Bits 12-13) - Preamble Length including Beginning of Frame indicator: 00 - 2 bytes, 01 - 4 bytes, 10 - 8 bytes, 11 - 16 bytes</td>
</tr>
<tr>
<td>INT-LPBCK</td>
<td>(Bit 14) - Internal Loopback</td>
</tr>
<tr>
<td>EXT-LPBCK</td>
<td>(Bit 15) - External Loopback. NOTE: Bits 14 and 15 configured to 1, cause Internal Loopback.</td>
</tr>
<tr>
<td><strong>Bytes 10-11:</strong></td>
<td></td>
</tr>
<tr>
<td>LIN-PRIO</td>
<td>(Bits 0-2) - Linear Priority</td>
</tr>
<tr>
<td><strong>Bytes 12-13:</strong></td>
<td></td>
</tr>
<tr>
<td>SLOT-TIME</td>
<td>(L) (Bits 0-7) - Slot Time number, low byte</td>
</tr>
<tr>
<td>SLT-TM</td>
<td>(H) (Bits 8-10) - Slot Time number, high bits</td>
</tr>
<tr>
<td>RETRY-NUM</td>
<td>(Bits 12-15) - Maximum number of transmission retries on collisions</td>
</tr>
<tr>
<td><strong>Byte 14-15:</strong></td>
<td></td>
</tr>
<tr>
<td>PRM</td>
<td>(Bit 0) - Promiscuous Mode</td>
</tr>
<tr>
<td>BC-DIS</td>
<td>(Bit 1) - Broadcast Disable</td>
</tr>
<tr>
<td>MANCH/NRZ</td>
<td>(Bit 2) - Manchester or NRZ encoding/decoding</td>
</tr>
<tr>
<td>0</td>
<td>- NRZ</td>
</tr>
<tr>
<td>1</td>
<td>- Manchester</td>
</tr>
<tr>
<td>TONO-CRS</td>
<td>(Bit 3) - Transmit on No Carrier Sense</td>
</tr>
<tr>
<td>0</td>
<td>- Cease transmission if CRS goes inactive during frame transmission</td>
</tr>
<tr>
<td>1</td>
<td>- Continue transmission even if no Carrier Sense</td>
</tr>
<tr>
<td>NCRC-INS</td>
<td>(Bit 4) - No CRC Insertion</td>
</tr>
<tr>
<td>CRC-16</td>
<td>(Bit 5) - CRC Type:</td>
</tr>
<tr>
<td>0</td>
<td>- 32 bit Autodin II CRC polynomial</td>
</tr>
<tr>
<td>1</td>
<td>- 16 bit CCITT CRC polynomial</td>
</tr>
<tr>
<td>BT-STF</td>
<td>(Bit 6) - Bitstuffing:</td>
</tr>
<tr>
<td>0</td>
<td>- End of Carrier mode (Ethernet)</td>
</tr>
<tr>
<td>1</td>
<td>- HDLC like Bitstuffing mode</td>
</tr>
<tr>
<td>PAD</td>
<td>(Bit 7) - Padding</td>
</tr>
<tr>
<td>0</td>
<td>- No Padding</td>
</tr>
</tbody>
</table>

---

**Notes:**
- Accelerated Contention Resolution (Exponential Priority)
- Exponential Backoff Method
- Number indicating the Interframe Spacing in TxC period units
- Internal Loopback
- External Loopback
- No CRC Insertion
- No Padding
The default values of the configuration parameters are compatible with the IEEE 802.3/Ethernet Standards. RESET configures the 82586 according to the defaults shown in Table 2.

Table 2. 82586 Default Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preamble Length</td>
<td>2</td>
</tr>
<tr>
<td>Address Length</td>
<td>6</td>
</tr>
<tr>
<td>Broadcast Disable</td>
<td>0</td>
</tr>
<tr>
<td>CRC-16/CRC-32</td>
<td>0</td>
</tr>
<tr>
<td>No CRC Insertion</td>
<td>0</td>
</tr>
<tr>
<td>Bitstuffing/EOC</td>
<td>0</td>
</tr>
<tr>
<td>Padding</td>
<td>0</td>
</tr>
<tr>
<td>Min-Frame-Length</td>
<td>64</td>
</tr>
<tr>
<td>Interframe Spacing</td>
<td>96</td>
</tr>
<tr>
<td>Slot Time</td>
<td>512</td>
</tr>
<tr>
<td>Number of Retries</td>
<td>15</td>
</tr>
<tr>
<td>Linear Priority</td>
<td>0</td>
</tr>
<tr>
<td>Accelerated Contention Resolution</td>
<td>0</td>
</tr>
<tr>
<td>Exponential Backoff Method</td>
<td>0</td>
</tr>
<tr>
<td>Manchester/NRZ</td>
<td>0</td>
</tr>
<tr>
<td>Internal CRS</td>
<td>0</td>
</tr>
<tr>
<td>CRS Filter</td>
<td>0</td>
</tr>
<tr>
<td>Internal CDT</td>
<td>0</td>
</tr>
<tr>
<td>CDT Filter</td>
<td>0</td>
</tr>
<tr>
<td>Transmit On No CRS</td>
<td>0</td>
</tr>
<tr>
<td>FIFO THRESHOLD</td>
<td>8</td>
</tr>
<tr>
<td>SRDY/ARDY</td>
<td>0</td>
</tr>
<tr>
<td>Save Bad Frame</td>
<td>0</td>
</tr>
<tr>
<td>Address/Type Location</td>
<td>0</td>
</tr>
<tr>
<td>INT Loopback</td>
<td>0</td>
</tr>
<tr>
<td>EXT Loopback</td>
<td>0</td>
</tr>
<tr>
<td>Promiscuous Mode</td>
<td>0</td>
</tr>
</tbody>
</table>

MC-SETUP

This command sets up the 82586 with a set of Multicast Addresses. Subsequently, incoming frames with Destination Addresses from this set are accepted.

Figure 14. The MC-SETUP Command Block
The MC-SETUP command includes the following fields:

**STATUS word (written by 82586):**

| C | (Bit 15) | Command completed |
| B | (Bit 14) | Busy executing command |
| OK | (Bit 13) | Error free completion |
| A | (Bit 12) | Command aborted |

**COMMAND word:**

| EL | (Bit 15) | End of command list |
| S | (Bit 14) | Suspend after completion |
| I | (Bit 13) | Interrupt after completion |
| CMD | (Bits 0-2) | MC-SETUP = 3 |

**LINK OFFSET:** Address of next Command Block

**MC-CNT:** A 14-bit field indicating the number of bytes in the MC-LIST field. MC-CNT is truncated to the nearest multiple of Address Length (in bytes). Issuing a MC-SETUP command with MC-CNT=0 disables reception of any incoming frame with a Multicast Address.

**MC-LIST:** A list of Multicast Addresses to be accepted by the 82586. Note that the most significant byte of an address is followed immediately by the least significant byte of the next address. Note also that the least significant bit of each Multicast Address in the set must be a one.

The Transmit-Byte-Machine maintains a 64-bit HASH table used for checking Multicast Addresses during reception.

An incoming frame is accepted if it has a Destination Address whose least significant bit is a one, and after hashing points to a bit in the HASH table whose value is one. The hash function is selecting bits 2 to 7 of the CRC register. RESET causes the HASH table to become all zeros.

After the Transmit-Byte-Machine reads a MC-SETUP command from TX-FIFO, it clears the HASH table and reads the bytes in groups whose length is determined by the ADDRESS length. Each group is hashed using CRC logic and the bit in the HASH table to which bits 2-7 of the CRC register point is set to one. A group that is not complete has no effect on the HASH table. Transmit-Byte-Machine notifies CU after completion.

**TRANSMIT**

The TRANSMIT command causes transmission (and if necessary retransmission) of a frame.

TRANSMIT CB includes the following fields:

**STATUS word (written by 82586):**

| C | (Bit 15) | Command completed |
| B | (Bit 14) | Busy executing command |
| OK | (Bit 13) | Error free completion |
| A | (Bit 12) | Command aborted |
| S10 | (Bit 10) | No Carrier Sequence signal during transmission (between beginning of Destination Address and end of Frame Check Sequence). |
| S9 | (Bit 9) | Transmission unsuccessful (stopped) due to loss of Clear-to-Send signal. |

![Figure 15. The Transmit Command Block](image)

6-318
**DESTINATION ADDRESS**: Destination Address of the frame.

**TYPE FIELD**: Type Field of the frame.

**STATUS word**:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOF</td>
<td>Indicates that this is the Buffer Descriptor of the last buffer of this frame's Information Field.</td>
</tr>
<tr>
<td>ACT-COUNT</td>
<td>Number of bits indicating the amount of data bytes in buffer (can be even or odd).</td>
</tr>
</tbody>
</table>

**NEXT BD OFFSET**: points to next Buffer Descriptor in list. If EOF is set, this field is meaningless.

**BUFFER ADDRESS**: 24-bit absolute address of buffer.

**TIME DOMAIN REFLECTOMETER - TDR**

This command performs a Time Domain Reflectometer test on the serial link. By performing the command, the user is able to identify shorts or opens and their location. Along with transmission of 'All Ones,' the 82586 triggers an internal timer. The timer measures the time elapsed from transmission start until 'echo' is obtained. 'Echo' is indicated by Collision Detect going active or Carrier Sense signal drop.

TDR command includes the following fields:

**STATUS word (written by 82586)**:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>(Bit 15) - Command completed</td>
</tr>
<tr>
<td>B</td>
<td>(Bit 14) - Busy executing command</td>
</tr>
<tr>
<td>OK</td>
<td>(Bit 13) - Error free completion</td>
</tr>
</tbody>
</table>

**Figure 16. The Transmit Buffer Descriptor**
**Figure 17. The TDR Command Block**

<table>
<thead>
<tr>
<th>COMMAND word:</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL (Bit 15)</td>
</tr>
<tr>
<td>S  (Bit 14)</td>
</tr>
<tr>
<td>I  (Bit 13)</td>
</tr>
<tr>
<td>CMD (Bits 0-2)</td>
</tr>
</tbody>
</table>

**LINK OFFSET:** Address of next Command Block

<table>
<thead>
<tr>
<th>RESULT word:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNK-OK (Bit 15) - No link problem identified</td>
</tr>
<tr>
<td>XCVR-PRB (Bit 14) - Transceiver Cable Problem identified (valid only in the case of a Transceiver that does not return Carrier Sense during transmission).</td>
</tr>
<tr>
<td>ET-OPN (Bit 13) - Open on the link identified (valid only in the case of a Transceiver that returns Carrier Sense during transmission).</td>
</tr>
</tbody>
</table>

**DUMP**

This command causes the contents of over a hundred bytes of internal registers to be placed in memory. It is supplied as a self diagnostic tool, as well as to supply registers of interest to the user.

**DUMP command includes the following fields:**

**STATUS word (written by 82586):**

- C (Bit 15) - Command completed
- B (Bit 14) - Busy executing command
- OK (Bit 13) - Error free completion

**Figure 18. The DUMP Command Block**
**COMMAND word:**

<table>
<thead>
<tr>
<th>EL</th>
<th>(Bit 15)</th>
<th>- End of command list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>(Bit 14)</td>
<td>- Suspend after completion</td>
</tr>
<tr>
<td>I</td>
<td>(Bit 13)</td>
<td>- Interrupt after completion</td>
</tr>
<tr>
<td>CMD</td>
<td>(Bits 0-2)</td>
<td>- DUMP = 6</td>
</tr>
</tbody>
</table>

**LINK OFFSET:** Address of next Command Block

**BUFFER OFFSET:** This word specifies the offset portion of the memory address which points to the top of the buffer allocated for the dumped registers contents. The length of the buffer is 170 bytes.

**DUMP AREA FORMAT**

Figure 18 shows the format of the DUMP area. The fields are as follows:

- **Bytes 00H to 0AH:** These bytes correspond to the 82586 CONFIGURE command field (except bit 6 of the first word).
- **Bytes 0CH to 11H:** The Individual Address Register content. IARO is the Individual Address least significant byte.
- **Bytes 12H to 13H:** Status word of last command block (only bits 0-13).
- **Bytes 14H to 17H:** Content of the Transmit CRC generator. TXCRCRO is the least significant byte. The contents are dependent on the activity before the DUMP command:
  - After RESET - 'All Ones.'
  - After successful transmission - 'All Zeros.'
  - After MC-SETUP command - Generated CRC value of the last MC address, on MC-LIST.
  - After unsuccessful transmission, depends on where it stopped.

**NOTE**

For 16-bit CRC only TXCRCRO and TXCRCR1 are valid.

**Bytes 18H to 1BH:** Contents of Receive CRC Checker. RXCRCRO is the least significant byte. The contents are dependent on the activity performed before the DUMP command:

- After RESET - 'All Ones.'
- After good frame reception -
  1. For CRC-CCITT - 0D1F0H.
  2. For CRC-Autodin-II - C704DD7BH
- After Bad Frame reception - corresponds to the received information.
- After reception attempt, i.e. unsuccessful check for address match, corresponds to the CRC performed on the frame address.

**NOTE**

Any frame on the serial link modifies this register contents.

---

**Figure 19. The DUMP Area**
### Bytes 1CH to 21H: Temporary Registers.

### Bytes 22H to 23H: Receive Status Register. Bits 6,7,8,10,11 and 13 assume the same meaning as corresponding bits in the Receive Frame Descriptor Status field.

### Bytes 24H to 2BH: HASH TABLE.

### Bytes 2CH to 2DH: Status bits of the last time TDR command that was performed.

### NXT-RB-SIZE: Let N be the last buffer of the last received frame, then NXT-RB-SIZE is the number of bytes of available in the N+1 buffer. EL - The EL bit of the Receive Buffer Descriptor.

### NXT-RB-ADR: Let N be the last Receive Buffer used, then NXT-RB-ADR is the BUFFER-ADDRESS field in the N+1 Receive-Buffer Descriptor, i.e. the pointer to the N+1 Receive Buffer.

### CUR-RB-SIZE: The number of bytes in the last buffer of the last received frame. EL - The EL bit of the last buffer in the last received frame.

### LA-RBD-ADR: Look Ahead Buffer Descriptor, i.e. the pointer to N+2 Receive Buffer Descriptor.

### NXT-RBD-ADR: Next Receive Buffer Descriptor Address. Similar to LA-RBD-ADR but points to N+1 Receive Buffer Descriptor.

### CUR-RBD-ADR: Current Receive Buffer Descriptor Address. Similar to LA-RBD-ADR, but points to Nth Receive Buffer Descriptor.

### CUR-RB-EBC: Current Receive Buffer Empty Byte Count. Let N be the currently used Receive Buffer. Then CUR-RB-EBC indicates the Empty part of the buffer, i.e. the ACT-COUNT of buffer N is given by the difference between its SIZE and the CUR-RB-EBC.

### NXT-FD-ADR: Next Frame Descriptor Address. Define N as the last Receive Frame Descriptor with bits C=1 and B=0, then NXT-FD-ADR is the address of N+2 Receive Frame Descriptor (with B=C=0) and is equal to the LINK-ADDRESS field in N+1 Receive Frame Descriptor.

### CUR-FD-ADR: Current Frame Descriptor Address. Similar to next NXT-FD-ADR but refers to N+1 Receive Frame Descriptor (with B=1, C=0).

### Bytes 54H to 55H: Temporary register.

---

**Figure 19. DUMP Area (con’t)**

---

#### Table Example

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value 1</td>
<td>Value 2</td>
<td>Value 3</td>
</tr>
</tbody>
</table>

---

**Diagram Example**

![Diagram of DUMP Area](image-url)
NXT-TB-CNT: Next Transmit Buffer Count. Let \( N \) be the last transmitted buffer of the TRANSMIT command executed recently, the NXT-TB-CNT is the ACT-COUNT field in the \( N \)th Transmit Buffer Descriptor. EOF - Corresponds to the EOF bit of the \( N \)th Transmit Buffer Descriptor. EOF=1 indicates that the last buffer accessed by the 82586 during Transmit was the last Transmit Buffer in the data buffer chain associated with the Transmit Command.

BUF-ADR: Buffer Address. The BUF-PTR field in the DUMP-STATUS Command Block.

NXT-TB-AD-L: Next Transmit Buffer Address Low. Let \( N \) be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then NXT-TB-AD-L are the two least significant bytes of the \( N \)th buffer address.

LA-TBD-ADR: Look Ahead Transmit Buffer Descriptor Address. Let \( N \) be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then LA-TBD-ADR is the NEXT-BD-ADDRESS field of the \( N \)th Buffer Descriptor.

NXT-TBD-ADR: Next Transmit Buffer Descriptor Address. Similar in function to LA-TBD-ADR but related to Transmit Buffer Descriptor \( N-1 \). Actually, it is the address of Transmit Buffer Descriptor \( N \).

Bytes 60H, 61H: This is a copy of the 2nd word in the DUMP-STATUS command presently executing.

NXT-CB-ADR: Next Command Block Address. The LINK-ADDRESS field in the DUMP Command Block presently executing. Points to the next command.

CUR-CB-ADR: Current Command Block Address. The address of the DUMP Command Block currently executing.

SCB-ADR: Offset of the System Control Block (SCB).

Bytes 7EH, 7FH:

RU-SUS-RQ (Bit 4) - Receive Unit Suspend Request.

Bytes 80H, 81H:

CU-SUS-RQ (Bit 4) - Command Unit Suspend Request
END-OF-CBL (Bit 5) - End of Command Block List. If '1' indicates that DUMP-STATUS is the last command in the command chain.
ABRT-IN-PROG (Bit 6) - Command Unit Abort Request.
RU-SUS-FD (Bit 12) - Receive Unit Suspend Frame Descriptor Bit. Assume \( N \) is the Receive Frame Descriptor used recently, then RU-SUS-FD is equivalent to the S bit of \( N+1 \) Receive Frame Descriptor.

Bytes 82H, 83H:

RU-SUS (Bit 4) - Receive Unit in SUSPENDED state.
RU-NRSRC (Bit 5) - Receive Unit in NO RESOURCES state.
RU-RDY (Bit 6) - Receive Unit in READY state.
RU-IDL (Bit 7) - Receive Unit in IDLE state.
RNR (Bit 12) - RNR Interrupt In Service bit.
CNA (Bit 13) - CNA Interrupt In Service bit.
FR (Bit 14) - FR Interrupt In Service bit.
CX (Bit 15) - CX Interrupt In Service bit.

Bytes 90H to 93H:

BUF-ADR-PTR - Buffer pointer is the absolute address of the bytes following the DUMP Command block.

Bytes 94H to 95H:

RCV-DMA-BC - Receive DMA Byte Count. This field contains number of bytes to be transferred during the next Receive DMA operation. The value depends on AT-LOCation configuration bit.

1. If AT-LOCation = 0 then RCV-DMA-BC = (2 times ADDR-LEN plus 2) if the next Receive Frame Descriptor has already been fetched.
2. If AT-LOCation = 1 then it contains the size of the next Receive Buffer.

BR+BUF-PTR+96H - Sum of Base Address plus BUF-PTR field and 96H.

RCV-DMA-ADR - Receive DMA absolute Address. This is the next RCV-DMA start address. The value depends on AT-LOCation configuration bit.

1. If AT-LOCation = 0, then RCV-DMA-ADR is the Destination Address field located in the next Receive Frame Descriptor.
2. If AT-LOCation = 1, then RCV-DMA-ADR is the next Receive Data Buffer Address.
The following nomenclature has been used in the DUMP table:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The 82586 writes zero in this location.</td>
</tr>
<tr>
<td>1</td>
<td>The 82586 writes one in this location.</td>
</tr>
<tr>
<td>X</td>
<td>The 82586 writes zero or one in this location.</td>
</tr>
<tr>
<td>///</td>
<td>The 82586 copies this location from the corresponding position in the memory structure.</td>
</tr>
</tbody>
</table>

**DIAGNOSE**

The DIAGNOSE Command triggers an internal self test procedure of backoff related registers and counters.

The DIAGNOSE command includes the following:

**STATUS word (written by 82586):**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Command completed</td>
</tr>
<tr>
<td>B</td>
<td>Busy executing command</td>
</tr>
<tr>
<td>OK</td>
<td>Error free completion</td>
</tr>
<tr>
<td>FAIL</td>
<td>Indicates that the self test procedure failed</td>
</tr>
</tbody>
</table>

**COMMAND word:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL</td>
<td>End of command list</td>
</tr>
<tr>
<td>S</td>
<td>Suspend after completion</td>
</tr>
<tr>
<td>I</td>
<td>Interrupt after completion</td>
</tr>
<tr>
<td>CMD</td>
<td>DIAGNOSE = 7</td>
</tr>
</tbody>
</table>

**LINK OFFSET:** Address of next Command Block

---

**RECEIVE FRAME AREA (RFA)**

The Receive Frame Area, RFA, is prepared by the host CPU, data is placed into the RFA by the 82586 as frames are received. RFA consists of a list of Receive Frame Descriptors (FD), each of which is associated with a frame. RFA-OFFSET field of SCB points to the first FD of the chain; the last FD is identified by the End-of-List flag (EL). See Figure 21.

**FRAME DESCRIPTOR (FD) FORMAT**

The FD includes the following fields:

**STATUS word (set by the 82586):**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Completed storing frame.</td>
</tr>
<tr>
<td>B</td>
<td>FD was consumed by RU.</td>
</tr>
<tr>
<td>OK</td>
<td>Frame received successfully. If this bit is set, then all others will be reset; if it is reset, then the other bits will indicate the nature of the error.</td>
</tr>
<tr>
<td>S11</td>
<td>Received frame experienced CRC error.</td>
</tr>
<tr>
<td>S10</td>
<td>Received frame experienced an alignment error.</td>
</tr>
<tr>
<td>S9</td>
<td>RU ran out of resources during reception of this frame.</td>
</tr>
<tr>
<td>S8</td>
<td>RCV-DMA overrun.</td>
</tr>
<tr>
<td>S7</td>
<td>Received frame had fewer bits than configured Minimum Frame Length.</td>
</tr>
<tr>
<td>S6</td>
<td>No EOF flag detected (only when configured to Bitstuffing).</td>
</tr>
</tbody>
</table>

**COMMAND word:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL</td>
<td>Last FD in the list.</td>
</tr>
<tr>
<td>S</td>
<td>RU should be suspended after receiving this frame.</td>
</tr>
</tbody>
</table>
**LINK OFFSET:** Address of next FD in list.

**RBD-OFFSET** (initially prepared by the CPU and later may be updated by 82586): Address of the first RBD that represents the Information Field. RBD-OFFSET = 0FFFFH means there is no Information Field.

**DESTINATION ADDRESS** (written by 82586): Contains Destination Address of received frame. The length in bytes, it is determined by the Address Length configuration parameter.

---

**Figure 21. The Receive Frame Area**
Figure 22. The Frame Descriptor (FD) Format

**SOURCE ADDRESS** (written by 82586): Contains Source Address of received frame. Its length is the same as DESTINATION ADDRESS.

**TYPE FIELD** (written by 82586): Contains the 2 byte Type Field of received frame.

**RECEIVE BUFFER DESCRIPTOR FORMAT**

The Receive Buffer Descriptor (RBD) holds information about a buffer: size and location, and the means for forming a chain of RBDs, (forward pointer and end-of-frame indication).

The Buffer Descriptor contains the following fields:

**STATUS word (written by the 82586):**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOF</td>
<td>(bit 15) - Last buffer in received frame.</td>
</tr>
<tr>
<td>F</td>
<td>(bit 14) - ACT COUNT field is valid.</td>
</tr>
<tr>
<td>ACT COUNT</td>
<td>(bits 0-13) - Number of bytes in the buffer that are actually occupied.</td>
</tr>
</tbody>
</table>

**NEXT RBD OFFSET:** Address of next BD in list of BD's.

**BUFFER ADDRESS:** 24-bit absolute address of buffer.

**EL/SIZE:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL</td>
<td>(bit 15) - Last BD in list.</td>
</tr>
<tr>
<td>SIZE</td>
<td>(bits 0-13) - Number of bytes the buffer is capable of holding.</td>
</tr>
</tbody>
</table>

Figure 23. The Receive Buffer Descriptor (RBD) Format
ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias .......... 0°C to 70°C
Storage Temperature ...................... -65°C to 150°C
Voltage on Any Pin With Respect to Ground -1.0V to +7V
Power Dissipation .......................... 3.0 Watts

*NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended period may affect device reliability.

D.C. CHARACTERISTICS

$T_A = 0-70^\circ C$, $V_{CC} = 5V \pm 10\%$ CLK, $TxD$, $TxC$, $RxD$, $RxC$ have MOS levels (see $V_{MIL}$, $V_{MIH}$, $V_{MOL}$, $V_{MOH}$). All other signals have TTL levels (see $V_{IL}$, $V_{IH}$, $V_{OL}$, $V_{OH}$).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$VI_L$</td>
<td>Input Low Voltage (TTL)</td>
<td>-0.5</td>
<td>+0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>$VI_H$</td>
<td>Input high Voltage (TTL)</td>
<td>2.0</td>
<td>$V_{CC}$+0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>$VO_L$</td>
<td>Output Low Voltage (TTL)</td>
<td>0.45</td>
<td>V</td>
<td>$I_{OL}$=2.5mA</td>
<td></td>
</tr>
<tr>
<td>$VO_H$</td>
<td>Output High Voltage (TTL)</td>
<td>2.4</td>
<td>V</td>
<td>$I_{OH}$=-400uA</td>
<td></td>
</tr>
<tr>
<td>$V_{MIL}$</td>
<td>Input Low Voltage (MOS)</td>
<td>-0.5</td>
<td>0.6</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>$V_{MIH}$</td>
<td>Input High Voltage (MOS)</td>
<td>3.9</td>
<td>$V_{CC}$+0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>$V_{MOL}$</td>
<td>Output Low Voltage (MOS)</td>
<td>0.45</td>
<td>V</td>
<td>$I_{OL}$=2.5mA</td>
<td></td>
</tr>
<tr>
<td>$V_{MOH}$</td>
<td>Output High Voltage (MOS)</td>
<td>$V_{CC}$-0.5</td>
<td>V</td>
<td>$I_{OH}$=-400uA</td>
<td></td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>Input Leakage Current</td>
<td>$\pm 10$</td>
<td>uA</td>
<td>$0 \leq V_{IN} \leq V_{CC}$</td>
<td></td>
</tr>
<tr>
<td>$I_{OL}$</td>
<td>Output Leakage Current</td>
<td>$\pm 10$</td>
<td>uA</td>
<td>$0.45 \leq V_{OUT} \leq V_{CC}$</td>
<td></td>
</tr>
<tr>
<td>$C_{IN}$</td>
<td>Capacitance of Input Buffer</td>
<td>10</td>
<td>pF</td>
<td>FC=1MHz</td>
<td></td>
</tr>
<tr>
<td>$C_{OUT}$</td>
<td>Capacitance of Output Buffer</td>
<td>20</td>
<td>pF</td>
<td>FC=1MHz</td>
<td></td>
</tr>
<tr>
<td>$I_{CC}$</td>
<td>Power Supply Current</td>
<td>550</td>
<td>450</td>
<td>mA</td>
<td>$T_A = 0^\circ C$, $T_A = 70^\circ C$</td>
</tr>
</tbody>
</table>

SYSTEM INTERFACE A.C. TIMING CHARACTERISTICS

$T_A=0-70^\circ C$, $V_{CC}=5V \pm 10\%$ Figure 24 and Figure 25 define how the measurements should be done:

![Figure 24. TTL Input/Output Voltage Levels For Timing Measurements](image-url)
Figure 25. System Clock MOS Input Voltage Levels for Timing Measurements

INPUT TIMING REQUIREMENTS (8MHz)*

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>CLK cycle period</td>
<td>125</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>CLK low time at 1.5V</td>
<td>55</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>CLK low time at 0.6V</td>
<td>42.5</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>CLK high time at 1.5V</td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>CLK high time at 3.8V</td>
<td>42.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>CLK rise time</td>
<td></td>
<td>15</td>
<td>Note 1</td>
</tr>
<tr>
<td>T7</td>
<td>CLK fall time</td>
<td></td>
<td>15</td>
<td>Note 2</td>
</tr>
<tr>
<td>T8</td>
<td>Data in setup time</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td>Data in hold time</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td>Async RDY active setup time</td>
<td>20</td>
<td></td>
<td>Note 3</td>
</tr>
<tr>
<td>T11</td>
<td>Async RDY inactive setup time</td>
<td>35</td>
<td></td>
<td>Note 3</td>
</tr>
<tr>
<td>T12</td>
<td>Async RDY hold time</td>
<td>15</td>
<td></td>
<td>Note 3</td>
</tr>
<tr>
<td>T13</td>
<td>Synchronous ready/active setup</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T14</td>
<td>Synchronous ready hold time</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T15</td>
<td>HLDA setup time</td>
<td>20</td>
<td></td>
<td>Note 3</td>
</tr>
<tr>
<td>T16</td>
<td>HLDA hold time</td>
<td>10</td>
<td></td>
<td>Note 3</td>
</tr>
<tr>
<td>T17</td>
<td>Reset setup time</td>
<td>20</td>
<td></td>
<td>Note 3</td>
</tr>
<tr>
<td>T18</td>
<td>Reset hold time</td>
<td>10</td>
<td></td>
<td>Note 3</td>
</tr>
<tr>
<td>T19</td>
<td>CA pulse width</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T20</td>
<td>CA setup time</td>
<td>20</td>
<td></td>
<td>Note 3</td>
</tr>
<tr>
<td>T21</td>
<td>CA hold time</td>
<td>10</td>
<td></td>
<td>Note 3</td>
</tr>
</tbody>
</table>
### OUTPUT TIMINGS (8 MHz)*

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T22</td>
<td>DT/R valid delay</td>
<td>0</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>T23</td>
<td>WR, DEN active delay</td>
<td>0</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>T24</td>
<td>WR, DEN inactive delay</td>
<td>0</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>T25</td>
<td>Int. active delay</td>
<td>0</td>
<td>85</td>
<td>Note 4</td>
</tr>
<tr>
<td>T26</td>
<td>Int. inactive delay</td>
<td>0</td>
<td>85</td>
<td>Note 4</td>
</tr>
<tr>
<td>T27</td>
<td>Hold active delay</td>
<td>0</td>
<td>85</td>
<td>Note 4</td>
</tr>
<tr>
<td>T28</td>
<td>Hold inactive delay</td>
<td>0</td>
<td>85</td>
<td>Note 4</td>
</tr>
<tr>
<td>T29</td>
<td>Address valid delay</td>
<td>0</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>T30</td>
<td>Address float delay</td>
<td>0</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>T31</td>
<td>Data valid delay</td>
<td>0</td>
<td>60</td>
<td>Note 7</td>
</tr>
<tr>
<td>T32</td>
<td>Data hold Time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T33</td>
<td>Status active delay</td>
<td>10</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>T34</td>
<td>Status inactive delay</td>
<td>10</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>T35</td>
<td>ALE active delay</td>
<td>0</td>
<td>45</td>
<td>Note 5</td>
</tr>
<tr>
<td>T36</td>
<td>ALE inactive delay</td>
<td>0</td>
<td>45</td>
<td>Note 5</td>
</tr>
<tr>
<td>T37</td>
<td>ALE width</td>
<td></td>
<td>T2-10</td>
<td>Note 5</td>
</tr>
<tr>
<td>T38</td>
<td>Address valid to ALE low</td>
<td></td>
<td>T2-30</td>
<td></td>
</tr>
<tr>
<td>T39</td>
<td>Address hold to ALE inactive</td>
<td></td>
<td>T4-10</td>
<td></td>
</tr>
<tr>
<td>T40</td>
<td>RD active delay</td>
<td>0</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>T41</td>
<td>RD inactive delay</td>
<td>0</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>T42</td>
<td>RD width</td>
<td></td>
<td>2T1-50</td>
<td></td>
</tr>
<tr>
<td>T43</td>
<td>Address float to RD active</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T44</td>
<td>RD inactive to Address active</td>
<td></td>
<td>T1-40</td>
<td></td>
</tr>
<tr>
<td>T45</td>
<td>WR width</td>
<td></td>
<td>2T1-40</td>
<td></td>
</tr>
<tr>
<td>T46</td>
<td>Data hold after WR</td>
<td></td>
<td></td>
<td>T2-25</td>
</tr>
<tr>
<td>T47</td>
<td>Control inactive after reset</td>
<td>0</td>
<td>60</td>
<td>Note 6</td>
</tr>
</tbody>
</table>

*All units are in ns

**CL on all outputs is 20-200 pF unless otherwise specified.

### NOTE LIST:

1. 1.0V to 3.5V
2. 3.5V to 1.0V
3. to guarantee recognition at next clock
4. CL = 50 pF
5. CL = 100 pF
6. Affects:
   - MIN MODE: RD, WR, DT/R, DEN
   - MAX MODE: S0, S1
7. High address lines (A16-A24, BHE) become valid one clock before T1 only on first memory cycle after the 82586 acquired the bus.

---

**Figure 26. INT Output Timing**

**Figure 27. CA Input Timing**
SERIAL INTERFACE A.C. TIMING CHARACTERISTICS

CLOCK SPECIFICATION

<table>
<thead>
<tr>
<th>Applies for TxC, RxC</th>
<th>for Manchester, symmetry is needed:</th>
</tr>
</thead>
<tbody>
<tr>
<td>for NRZ:</td>
<td>T51, T52 = \frac{1}{2f} ±5%</td>
</tr>
<tr>
<td>f min = 100 kHz ± 100 ppm</td>
<td></td>
</tr>
<tr>
<td>f max = 10 MHz ± 100 ppm</td>
<td></td>
</tr>
<tr>
<td>for Manchester:</td>
<td></td>
</tr>
<tr>
<td>f min = 500 kHz ± 100 ppm</td>
<td></td>
</tr>
<tr>
<td>f max = 10 MHz ± 100 ppm</td>
<td></td>
</tr>
</tbody>
</table>

6-331
### A.C. CHARACTERISTICS

#### TRANSMIT AND RECEIVE TIMING PARAMETER SPECIFICATION*

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter Description</th>
<th>Min.</th>
<th>Max.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T48</td>
<td>( \text{TxC Cycle} )</td>
<td>100</td>
<td>1000</td>
<td>Notes 1, 2</td>
</tr>
<tr>
<td>T48</td>
<td>( \text{TxC Cycle} )</td>
<td>100</td>
<td></td>
<td>Notes 1, 3</td>
</tr>
<tr>
<td>T49</td>
<td>( \text{TxC Rise Time} )</td>
<td>5</td>
<td></td>
<td>Note 1</td>
</tr>
<tr>
<td>T50</td>
<td>( \text{TxC Fall Time} )</td>
<td>5</td>
<td></td>
<td>Note 1</td>
</tr>
<tr>
<td>T51</td>
<td>( \text{TxC High Time} )</td>
<td>40</td>
<td>1000</td>
<td>Note 1</td>
</tr>
<tr>
<td>T52</td>
<td>( \text{TxC Low Time} )</td>
<td>40</td>
<td></td>
<td>Notes 1, 4</td>
</tr>
</tbody>
</table>

#### TRANSMIT DATA PARAMETERS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter Description</th>
<th>Min.</th>
<th>Max.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T53</td>
<td>( \text{TxD Rise Time} )</td>
<td>10</td>
<td></td>
<td>Notes 5, 13</td>
</tr>
<tr>
<td>T54</td>
<td>( \text{TxD Fall Time} )</td>
<td>10</td>
<td></td>
<td>Notes 5, 13</td>
</tr>
<tr>
<td>T55</td>
<td>( \text{TxD Transition - Transition} )</td>
<td>Min(T51, T52)-7</td>
<td></td>
<td>Notes 1, 2, 5</td>
</tr>
<tr>
<td>T56</td>
<td>( \text{TxC Low to TxD Valid} )</td>
<td>40</td>
<td></td>
<td>Notes 1, 3, 5</td>
</tr>
<tr>
<td>T57</td>
<td>( \text{TxC Low to TxD Transition} )</td>
<td>40</td>
<td></td>
<td>Notes 1, 2, 5</td>
</tr>
<tr>
<td>T58</td>
<td>( \text{TxC High to TxD Transition} )</td>
<td>40</td>
<td></td>
<td>Notes 1, 2, 5</td>
</tr>
<tr>
<td>T59</td>
<td>( \text{TxC Low to TxD High at the Transmission end} )</td>
<td>40</td>
<td></td>
<td>Notes 1, 5</td>
</tr>
</tbody>
</table>

#### REQUEST TO SEND/CLEAR TO SEND PARAMETERS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter Description</th>
<th>Min.</th>
<th>Max.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T60</td>
<td>( \text{TxC Low to RTS Low. Time to Activate RTS} )</td>
<td>40</td>
<td></td>
<td>Note 6</td>
</tr>
<tr>
<td>T61</td>
<td>( \text{CTS Valid to TxC Low. CTS Set-Up Time} )</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T62</td>
<td>( \text{TxC Low to CTS Invalid. CTS Hold Time} )</td>
<td>20</td>
<td></td>
<td>Note 7</td>
</tr>
<tr>
<td>T63</td>
<td>( \text{TxC Low to RTS High. time to deactivate RTS} )</td>
<td>40</td>
<td></td>
<td>Note 6</td>
</tr>
</tbody>
</table>

#### RECEIVE CLOCK PARAMETERS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter Description</th>
<th>Min.</th>
<th>Max.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T64</td>
<td>( \text{RxC Clock Cycle} )</td>
<td>100</td>
<td></td>
<td>Notes 1, 3</td>
</tr>
<tr>
<td>T65</td>
<td>( \text{RxC Rise Time} )</td>
<td>5</td>
<td></td>
<td>Note 1</td>
</tr>
<tr>
<td>T66</td>
<td>( \text{RxC Fall Time} )</td>
<td>5</td>
<td></td>
<td>Note 1</td>
</tr>
<tr>
<td>T67</td>
<td>( \text{RxC High Time} )</td>
<td>36</td>
<td>1000</td>
<td>Note 1</td>
</tr>
<tr>
<td>T68</td>
<td>( \text{RxC Low Time} )</td>
<td>40</td>
<td></td>
<td>Note 1</td>
</tr>
</tbody>
</table>

#### RECEIVE DATA PARAMETERS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter Description</th>
<th>Min.</th>
<th>Max.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T69</td>
<td>( \text{RxD Setup Time} )</td>
<td>30</td>
<td></td>
<td>Note 1</td>
</tr>
<tr>
<td>T70</td>
<td>( \text{RxD Hold Time} )</td>
<td>30</td>
<td></td>
<td>Note 1</td>
</tr>
<tr>
<td>T71</td>
<td>( \text{RxD Rise Time} )</td>
<td>10</td>
<td></td>
<td>Note 1</td>
</tr>
<tr>
<td>T72</td>
<td>( \text{RxD Fall Time} )</td>
<td>10</td>
<td></td>
<td>Note 1</td>
</tr>
</tbody>
</table>

*All units are in ns.
TRANSMIT AND RECEIVE TIMING PARAMETER SPECIFICATION* (cont'd.)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARRIER SENSE/COLLISION DETECT PARAMETERS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T73</td>
<td>CDT Valid to TxC Low Ext. Collision Detect Setup Time</td>
<td>30</td>
<td></td>
<td>Note 12</td>
</tr>
<tr>
<td>T74</td>
<td>TxC Low to CDT Inactive. CDT Hold Time</td>
<td>20</td>
<td></td>
<td>Note 12</td>
</tr>
<tr>
<td>T75</td>
<td>CDT Low to Jamming Start</td>
<td></td>
<td></td>
<td>Note 8</td>
</tr>
<tr>
<td>T76</td>
<td>CRS Valid to TxC Low Ext. Carrier Sense Setup time</td>
<td>30</td>
<td></td>
<td>Note 12</td>
</tr>
<tr>
<td>T77</td>
<td>TxC Low to CRS Inactive, CRS Hold Time</td>
<td>20</td>
<td></td>
<td>Note 12</td>
</tr>
<tr>
<td>T78</td>
<td>CRS Low to Jamming Start</td>
<td></td>
<td></td>
<td>Note 9</td>
</tr>
<tr>
<td>T79</td>
<td>Jamming Period</td>
<td></td>
<td></td>
<td>Note 10</td>
</tr>
<tr>
<td>T80</td>
<td>CRS Inactive Setup Time to RxC High. End of Receive Frame</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T81</td>
<td>CRS Active Hold Time From RxC High</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

INTERFRAME SPACING PARAMETERS

| T82 | Inter Frame Delay | | Note 11 |

*All units are in ns.

NOTES:

1 — MOS levels.
2 — Manchester only.
3 — NRZ only.
4 — Manchester requires 50% Duty Cycle.
5 — 1 TTL Load + 50 pF.
6 — 1 TTL Load + 100 pF.
7 — Abnormal End of Transmission. CTS Expires Before RTS.
8 — Programmable value: T75 = NCDF × T48 + (12.5 to 23.5) × T48 if collision occurs after preamble. NCDF — The Collision Detection Filter Configuration Value.
9 — Programmable value: T78 = NCSF × T48 + (12.5 to 23.5) × T48. NCSF — The Carrier Sense Filter Configuration Value. TBD is a function of Internal/External Carrier Sense Bit.
10 — T79 = 32 × T48.
11 — Programmable value: T88 = NIFS × T48. NIFS — the IFS Configuration Value.
   If NIFS is less than 31, then NIFS is enforced to 32.
12 — To guarantee recognition on the next clock.
13 — Applies to TTL Levels.
A.C. TIMING CHARACTERISTICS

INPUT AND OUTPUT WAVEFORMS FOR AC TESTS

AC TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC 1 AND 0.45 FOR A LOGIC 0. TRIMMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC 1 AND 0.

Figure 33. TTL Input/Output Voltage Levels for Timing Measurements

Figure 34. Serial Clock Input Voltage Levels for Timing Measurements

Figure 35. Transmit and Control and Data Timing
Figure 35. Transmit and Control and Data Timing (cont.)

Figure 36. RxD Timing Relative to RxC

Figure 37. CRS Timing Relative to RxC
82588
Single Chip LAN Controller

82588: High Integration Mode
82588-5: High Speed Mode

- Integrates ISO Layers 1 and 2
  - CSMA/CD Data Link Controller
  - On-Chip Manchester, NRZI Encoding/Decoding
  - On-Chip Logic based Collision Detect and Carrier Sense
- Supports Emerging IEEE 802.3 Standards
  - 2 Mbps Broadband
  - 1 Mbps Baseband
- High level command interface offloads the CPU
- Efficient memory use via Multiple Buffer Reception

- User Configurable
  - Up to 2 Mbps Bit rates with on-chip Encoder/Decoder (High Integration Mode)
  - Up to 5 Mbps with External Encoder/Decoder (High Speed Mode)
- No TTL Glue required with IAPX 186 and 188 microprocessors
- Network Management and Diagnostics
  - Short or Open Circuit localization
  - Station Diagnostics (External loopback)
  - Self test Diagnostics
    - Internal loopback
    - User readable registers

The 82588 is a highly integrated device designed for realizing cost sensitive Local Area Network applications such as Personal Workstations.

At data rates of up to 2 Mbps, it provides a highly integrated interface and performs: CSMA/CD Data Link Control, Manchester, Differential Manchester or NRZI encoding/decoding, clock recovery, Carrier Sense, and Collision Detection. This mode is called "High Integration Mode." In the 82588 "High Speed Mode", the user can transfer data at a rate of up to 5 Mbps. In this mode the physical link functions are done external to the 82588.

The 82588 is packaged in a 28 pin DIP and fabricated in Intel's reliable HMOS II 5 volt technology.
### Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>6</td>
<td>I/O</td>
<td>DATA BUS: The Data Bus lines are bi-directional three state lines connected to the system's Data Bus for the transfer of data, commands, status and parameters.</td>
</tr>
<tr>
<td>D6</td>
<td>7</td>
<td>I</td>
<td>READ: Together with CS, DACK0 or DACK1, Read controls data or status transfers out of the 82588 registers.</td>
</tr>
<tr>
<td>D5</td>
<td>8</td>
<td>I</td>
<td>WRITE: Together with CS, DACK0 or DACK1, Write controls data or command transfers into the 82588 registers.</td>
</tr>
<tr>
<td>D4</td>
<td>9</td>
<td>I</td>
<td>CHIP SELECT: When this signal is LOW, the 82588 is selected by the CPU for transfer of command or status. The direction of data flow is determined by the RD or WR inputs.</td>
</tr>
<tr>
<td>D3</td>
<td>10</td>
<td>I</td>
<td>CLOCK: System clock. MOS compatible signal.</td>
</tr>
<tr>
<td>D2</td>
<td>11</td>
<td>I</td>
<td>RESET: A HIGH signal on this pin will cause the 82588 to terminate current activity. This signal is internally synchronized and must be held HIGH for at least four Clock cycles.</td>
</tr>
<tr>
<td>D1</td>
<td>12</td>
<td>O</td>
<td>INTERRUPT: Active HIGH signal indicates to the CPU that the 82588 is requesting an interrupt.</td>
</tr>
<tr>
<td>D0</td>
<td>13</td>
<td>I</td>
<td>DMA REQUEST (CHANNEL 0): This pin is used by the 82588 to request a DMA transfer. DRQ0 remains HIGH as long as 82588 requires data transfers. Burst transfers are done by having the signal active for multiple transfers.</td>
</tr>
<tr>
<td>R6</td>
<td>1</td>
<td>I</td>
<td>DMA ACKNOWLEDGE (CHANNEL 0): When LOW, this input signal from the DMA Controller notifies the 82588 that the requested DMA cycle is in progress. This signal acts like chip select for data and parameter transfer using DMA channel 0.</td>
</tr>
<tr>
<td>R7</td>
<td>2</td>
<td>I</td>
<td>DMA REQUEST (CHANNEL 1): This pin is used by the 82588 to request a DMA transfer. DRQ1 remains HIGH as long as 82588 requires data transfers. Burst transfers are done by having the signal active for multiple transfers.</td>
</tr>
<tr>
<td>WR</td>
<td>3</td>
<td>I</td>
<td>DMA ACKNOWLEDGE (CHANNEL 1): When LOW, this input signal from the DMA controller notifies the 82588 that the requested DMA cycle is in progress. This signal acts like chip select for data and parameter transfer using DMA channel 1.</td>
</tr>
<tr>
<td>CS</td>
<td>2</td>
<td>I</td>
<td>DMA REQUEST (CHANNEL 1): This pin is used by the 82588 to request a DMA transfer. DRQ1 remains HIGH as long as 82588 requires data transfers. Burst transfers are done by having the signal active for multiple transfers.</td>
</tr>
<tr>
<td>CLK</td>
<td>4</td>
<td>I</td>
<td>DMA REQUEST (CHANNEL 0): This pin is used by the 82588 to request a DMA transfer. DRQ0 remains HIGH as long as 82588 requires data transfers. Burst transfers are done by having the signal active for multiple transfers.</td>
</tr>
<tr>
<td>INT</td>
<td>26</td>
<td>O</td>
<td>DMA ACKNOWLEDGE (CHANNEL 0): When LOW, this input signal from the DMA Controller notifies the 82588 that the requested DMA cycle is in progress. This signal acts like chip select for data and parameter transfer using DMA channel 0.</td>
</tr>
<tr>
<td>DRQ0</td>
<td>17</td>
<td>O</td>
<td>DMA REQUEST (CHANNEL 1): This pin is used by the 82588 to request a DMA transfer. DRQ1 remains HIGH as long as 82588 requires data transfers. Burst transfers are done by having the signal active for multiple transfers.</td>
</tr>
<tr>
<td>DRQ1</td>
<td>18</td>
<td>O</td>
<td>DMA ACKNOWLEDGE (CHANNEL 1): When LOW, this input signal from the DMA controller notifies the 82588 that the requested DMA cycle is in progress. This signal acts like chip select for data and parameter transfer using DMA channel 1.</td>
</tr>
</tbody>
</table>
Table 1. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1/X2</td>
<td>15/16</td>
<td>I</td>
<td>High Integration Mode</td>
</tr>
<tr>
<td>OSCI</td>
<td></td>
<td></td>
<td>OSCILLATOR INPUTS: These inputs may be used to connect a quartz crystal that controls the internal clock generator for the serial unit. X1 may also be driven by a MOS level clock whose frequency is 8 or 16 times the bit rate of Transmit/Receive data. X2 must be left floating if X1 has an external MOS clock.</td>
</tr>
<tr>
<td>TxC</td>
<td>15</td>
<td>I</td>
<td>TRANSMIT CLOCK: This signal provides timing information to the internal serial logic, depending upon the mode of data transfer. For NRZ encoding, data is transferred to the TxD pin on the HIGH to LOW clock transition. For Manchester encoding the transmitted bit center is aligned with the TxC LOW to HIGH transition.</td>
</tr>
<tr>
<td>RxC</td>
<td>16</td>
<td>I</td>
<td>RECEIVE CLOCK: This signal provides timing information to the internal serial logic. NRZ data should be provided for reception (RxD). The state of the RxD pin is sampled on the HIGH to LOW transition of RxC.</td>
</tr>
<tr>
<td>TCLK/CRS</td>
<td>24</td>
<td>I</td>
<td>In High Speed Mode, this pin is Carrier Sense, input CRS, and is used to notify the 82588 that there is activity on the serial link. In High Integration Mode, this pin is Transmit Clock, TCLK, and is used to output the transmit clock.</td>
</tr>
<tr>
<td>CDT</td>
<td>23</td>
<td>I</td>
<td>COLLISION DETECT: This input notifies the 82588 that a collision has occurred. It is sensed only if the 82588 is configured for external Collision Detect (external circuitry is then required for detecting the collision).</td>
</tr>
<tr>
<td>RxD</td>
<td>19</td>
<td>I</td>
<td>RECEIVE DATA: This pin receives serial data. Requires MOS level signal.</td>
</tr>
<tr>
<td>TxD</td>
<td>20</td>
<td>O</td>
<td>TRANSMIT DATA: This pin transmits data to the Serial Link. This signal is HIGH when not transmitting. It generates a MOS level signal.</td>
</tr>
<tr>
<td>RTS</td>
<td>21</td>
<td>O</td>
<td>REQUEST TO SEND: When this signal is LOW, the 82588 notifies an external interface that it has data to transmit. It is forced HIGH after a reset and when transmission is stopped.</td>
</tr>
<tr>
<td>CTS</td>
<td>22</td>
<td>I</td>
<td>CLEAR TO SEND: CTS enables the 82588 to start transmitting data. Raising this signal to HIGH stops the transmission.</td>
</tr>
<tr>
<td>VCC</td>
<td>28</td>
<td></td>
<td>POWER: +5V Supply</td>
</tr>
<tr>
<td>VSS</td>
<td>14</td>
<td></td>
<td>Ground</td>
</tr>
</tbody>
</table>

6-338
FUNCTIONAL DESCRIPTION

High Integration Mode

The 82588 Single Chip LAN Controller is a highly integrated device designed for Cost Sensitive LAN applications such as LANS for personal computers. Included on the chip is a programmable CSMA/CD controller, a NRZI and Manchester and encoder/decoder with clock recovery, and two collision detection mechanisms. With the addition of simple transceiver line drivers or RF Modem, the 82588 takes care of all the major functions of the ISO Physical and Data Link Layers.

CSMA/CD Controller

The 82588 on-chip CSMA/CD controller is programmable which allows it to operate in a variety of LAN environments including emerging IEEE 802.3 standards such as 1 Mbps baseband and 2 Mbps baseband (IBM PC Network). Programmable parameters include:

- Framing (End of Carrier or SDLC)
- Address field length
- Station priority
- Interframe spacing
- Slot time
- CRC-32 OR CRC-16

Encoder/Decoder

The on-chip NRZI and Manchester encoder/decoder supports data rates up to 2 Mbps. Manchester encoding is often times used in baseband applications and NRZI is used in broadband applications.

Collision Detection

A major innovation with the 82588 is its on-chip logic based collision detection. To ensure high probability of collision detection two mechanisms are provided. The Code Violation method defines a collision when a transition edge occurs outside the area of normal transitions as specified by either the Manchester or NRZI encoding methods. The Bit Comparison method compares the signature of the transmitted frame to the receive frame signature (re-calculated by the 82588 while listening to itself). If the signatures are identical the frame is assumed to have been transmitted without a collision.

System Interface

The 82588 goes beyond providing the designer with the functions necessary for interfacing to the LAN. It has an extremely friendly system interface that makes it easy to design with. First, the 82588 has a high level command interface, that is the CPU sends the 82588 commands such as Transmit or Configure. This means the designer does not have to write low level software to perform these tasks, and it offloads the CPU in the application. Second, the 82588 supports an efficient memory structure called Multiple Buffer Reception in which buffers are chained together to receive frames. This is an important feature in applications with limited memory such as personal computers. Third, the 82588 has two independent sixteen byte FIFO's, one for reception and one for transmission. The FIFO's allow the 82588 to tolerate bus latency. Finally the 82588 provides an eight byte data path that supports up to 4 Mbytes/second using external DMA.

Network Management & Diagnostics

The 82588 provides a rich set of diagnostic and network management functions including: internal and external loopbacks, channel activity indicators, optional capture of all frames regardless of destination address (Promiscuous Mode), capture of collided frames, (if address matches), and time domain reflectometry for locating fault points in the network cable. The 82588 Register Dump Command ensures reliable software by dumping the content of the 82588 registers into the system memory.

The next section will describe the 82588 system bus interface, the 82588 network interface, and the 82588 internal architecture.

82588/Host CPU Interaction

The CPU communicates with the 82588 through the system's memory and 82588's on-chip registers. The CPU creates a data structure in the memory, programs the external DMA controller with the start address and byte count of the block, and issues the command to the 82588.

The 82588 is optimized for operating with the iAPX 186/188, but due to the small number of hardware signals between the 82588 and the CPU, the 82588 can operate easily with other processors. The data bus is 8 bits wide and there is no address bus.

Chip select and Interrupt lines are used to communicate between the 82588 and the host as shown in the Figure 3. Interrupt is used by the 82588 to draw the CPU's attention. The Chip Select is used by the CPU to draw the 82588's attention.
There are two kinds of transfer over the bus: Command/Status and data transfers. Command/Status transfers are always performed by the CPU. Data transfers are requested by the 82588, and are typically performed by a DMA controller. The table given in Figure 4 shows the Command/Status and data transfer control signals.

The CPU writes to 82588 using CS and WR signals. The CPU reads the 82588 status register using CS and RD signals.

To initiate an operation like Transmit or Configure (see Figure 5), a Write operation command from CPU to 82588 is issued by the CPU. A Read operation from CPU gives the status of the 82588. Although there are four status registers they're read using the same port in a round robin fashion (Figure 6).

Any parameters or data associated with command are transferred between the memory and 82588 using DMA. The 82588 has two data channels, each having Request and Acknowledge line. Typically one channel is used to receive data and other to transmit data and to do all the other initialization and maintenance operations like Configure, Address Set-Up, Diagnose etc. The channels are identical and can be used interchangeably.

When 82588 requires access to the memory for parameter or data transfer it activates the DMA request lines and uses the DMA controller to achieve the data transfer. Upon the completion of an operation, the 82588 interrupts the CPU. The CPU then reads results of the operation (or the status of the 82588).

---

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS*</td>
<td>RD</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>DACK0[DACK1]*</td>
<td>RD</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

* Only one of CS, DACK0 and DACK1 may be active at any time.

---

Figure 3. 82588/HOST CPU Interaction

Figure 4. DATABUS CONTROL SIGNALS AND THEIR FUNCTIONS
<table>
<thead>
<tr>
<th>COMMANDS</th>
<th>VALUE</th>
<th>COMMANDS</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>0</td>
<td>ABORT</td>
<td>13</td>
</tr>
<tr>
<td>IA-SETUP</td>
<td>1</td>
<td>RECEIVER-ENABLE</td>
<td>8</td>
</tr>
<tr>
<td>CONFIGURE</td>
<td>2</td>
<td>ASSIGN NEXT BUF</td>
<td>9</td>
</tr>
<tr>
<td>MC-SETUP</td>
<td>3</td>
<td>RECEIVE-DISABLE</td>
<td>10</td>
</tr>
<tr>
<td>TRANSMIT</td>
<td>4</td>
<td>STOP-RECEPTION</td>
<td>11</td>
</tr>
<tr>
<td>TDR</td>
<td>5</td>
<td>RESET</td>
<td>14</td>
</tr>
<tr>
<td>DUMP</td>
<td>6</td>
<td>FIX PTR</td>
<td>15 (CHNL = 1)</td>
</tr>
<tr>
<td>DIAGNOSE</td>
<td>7</td>
<td>RLS PTR</td>
<td>15 (CHNL = 0)</td>
</tr>
<tr>
<td>RETRANSMIT</td>
<td>12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. Command Format and Operation Values

<table>
<thead>
<tr>
<th>Status 0</th>
<th>INT.</th>
<th>RCV</th>
<th>EXEC</th>
<th>CHNL</th>
<th>EVENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RESULT 1</td>
</tr>
<tr>
<td>Status 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RESULT 2</td>
</tr>
<tr>
<td>Status 3</td>
<td>RCV</td>
<td>CHNL</td>
<td>RCV STATE</td>
<td>BUFF</td>
<td>CHNL</td>
</tr>
</tbody>
</table>

EVENTS

<table>
<thead>
<tr>
<th>VALUE (STATUS 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA-SETUP-DONE</td>
</tr>
<tr>
<td>CONFIGURE-DONE</td>
</tr>
<tr>
<td>MC-SETUP-DONE</td>
</tr>
<tr>
<td>TRANSMIT-DONE</td>
</tr>
<tr>
<td>TDR-DONE</td>
</tr>
<tr>
<td>DUMP-DONE</td>
</tr>
<tr>
<td>DIAGNOSE-PASSED</td>
</tr>
<tr>
<td>END OF FRAME</td>
</tr>
<tr>
<td>REQUEST NEXT BUFFER</td>
</tr>
<tr>
<td>RECESSION ABORTED</td>
</tr>
<tr>
<td>RETRANSMIT-DONE</td>
</tr>
<tr>
<td>EXECUTION-ABORTED</td>
</tr>
<tr>
<td>DIAGNOSE-FAILED</td>
</tr>
</tbody>
</table>

Figure 6. Status Registers and Event Values
Transmitting a Frame

To transmit a frame, the CPU prepares a Transmit Data Block in memory as shown in Figure 7. Its first two bytes specify the length of the rest of the block. The next few bytes (Up to 6 bytes long) contain the destination address of the node it is being sent to. The rest of the block is the data field. The CPU programs the DMA controller with the start address of the block, length of the block and other control information and then issues the Transmit command to the 82588.

Upon receiving the command, the 82588 fetches the first two bytes of the block to determine the length of the block. If the link is free, and the first data byte was fetched, the 82588 begins transmitting the preamble and concurrently fetches the bytes from the Transmit Data Block and loading them into a 16 byte FIFO to keep them ready for transmitting. The FIFO is a buffer between the serial and parallel part of the 82588. The on-chip FIFOs help the 82588 to tolerate system bus latency as well as provides efficient usage of system bandwidth.

The destination address is sent out after the preamble. This is followed by the source or the station individual address, which is stored earlier on the 82588 using the IA-SETUP command. After that, the entire information field is transmitted followed by a CRC field calculated by the 82588. If during the transmission of the frame, a collision is encountered, then the transmission is aborted and a jam pattern is sent out after completion of the preamble. The 82588 generates an Interrupt indicating the experience of a collision and the frame has to be re-transmitted. Re-transmission is done by the CPU exactly as the Transmit command except for using the Re-Transmit command which keeps track of the number of collisions encountered. When the 82588 gets the Retransmit command and the exponential back-off time is expired, the 82588 transmits the frame again. The transmitted frame can be coded to either Manchester, Differential Manchester or NRZI methods.

Collision Detection

The 82588 eliminates the need for external collision detection logic, in most applications, while easing or eliminating the need for complex transceivers. Two algorithms are used for collision detection: Bit Comparison and Code Violation. The Bit Comparison Method is useful in Broadband networks where there are separate transmit and receive channels. Bit Comparison compares the “signature” of the transmitted data and received data at the end of the collision window in any network configuration. This algorithm calculates the CRC after a programmable number of transmitted bits, holds this CRC in a register, and compares it with received data’s CRC. A CRC or “signature” difference indicates a collision. The code violation is detected if the encoding of the received data has any bit that does not fit the encoding rules. The code violation method is useful in short bus topology and serial backplane applications where bit attenuation over the bus is negligible.

![Figure 7. The 82588 Frame Structure and location of Data element in System Memory](image-url)
Receiving a Frame

The 82588 can receive a frame when its receiver has been enabled. The received frame is decoded by either on-chip Manchester, Differential Manchester or NRZI decoders in High Integration Mode and NRZI in High Speed Mode. The 82588 checks for an address match for an Individual address, a Multicast address or a Broadcast address. In the Promiscuous mode the 82588 receives all frames. Only when the address match is successful does the 82588 transfer the frame to the memory using the DMA controller. Before enabling the receiver, the CPU makes a memory buffer area available to the Receive Unit, and programs the starting address of the DMA controller. The received frame is transferred to the memory buffer in the format shown in Figure 8. This method of reception is called "Single Buffer" reception. The entire frame is contained in one continuous buffer. Upon completion of reception the total number of bytes written into the memory buffer is loaded into status registers 1 and 2 and the status of the reception itself is appended to the received frame. An interrupt to CPU follows.

If the frame size is unknown then to optimize the memory usage "Multiple buffer" reception is used. This way the user does not have to allocate large memory space for short frames. Instead, the 82588 can dynamically allocate memory space as it receives frames. This method requires both DMA channels alternately to receive the frame. As the frame reception starts, the 82588 interrupts the CPU and automatically requests assignment of sequentially the next buffer. The CPU does this and loads the second DMA channel with the next buffer information so that the 82588 can immediately switch to the other channel as soon as the current buffer is full. When the 82588 switches from the first to the second buffer it again interrupts the CPU requesting it to allocate another buffer on the other (previous) channel in advance. This process continues until the entire frame is received. The received frame is spread over a multiple memory buffer. The link between the buffers is easily maintained by the CPU using a buffer chain descriptor structure in memory (see Figure 9).

This dynamic (pre) allocation of memory buffers results in efficient use of available storage when handling frames of widely differing sizes. Since the buffers are pre-allocated one block in advance, the system is not time critical.
80188 Based System

Figure 10 shows a high-performance, high-integration configuration of the 82588 with the 80188 in a typical iAPX188-based microcomputer. The 80188 controls the 82588, as well as providing DMA control services for data transfer, using its ‘on chip’ two channel DMA controller.

Link Interface

The Serial Interface Mode configuration parameter selects either a highly integrated Direct Link Interface (High Integration Mode) or a highly flexible Transceiver Interface (High Speed Mode).

Application

In this mode it is possible to connect the 82588 on a very short ‘Wired OR’ link, on a longer twisted pair cable, or a broadband connection.

Twisted Pair Connection

The link consists of a twisted pair that interconnects the 82588 (See Figure 11). The transmit data pin is connected via a driver and the receive data pin is connected via a buffer. The twisted pair must be properly terminated to prevent reflections.

In the minimum configuration, TxD and RxD are connected to the twisted pair and CTS is grounded. The 82588 may control the driver with the RTS pin. It is also possible to use external circuitry for performing collision detection, and feeding it to the 82588 through the CDT pin.

Broadband Connection

The 82588 supports data communications over a broadband link in both its modes. Proper MODEM interface should be provided. Collision Detection by Bit Comparison, in High Integration Mode, fits transmission over broadband links.
Figure 10. 80188 Based System
Figure 11. Twisted Pair Connection
**Absolute Maximum Ratings**

Ambient Temperature Under Bias ........ 0°C to 70°C
Storage Temperature .................. −65°C to 150°C
Voltage on Any Pin With Respect to Ground ............ −1.0V to +7V
Power Dissipation ..................... 1.7 Watts

*Notice: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended period may affect device reliability.*

**D.C. Characteristics** ($T_A = 0^\circ C\ to\ 70^\circ C;\ VCC = +5V\ \pm\ 10\%$)

CLK, TxD, TxC, RxD, Rx\overline{C} have MOS levels (See VMIL, VMIH, VMOL, VHOH). All other signals have TTL levels (See VI\overline{L}, VI\overline{H}, VOL, VOH).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage (TTL)</td>
<td>−0.5</td>
<td>+0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage (TTL)</td>
<td>2.0</td>
<td>VCC + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage (TTL)</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 2.0 mA</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage (TTL)</td>
<td>2.4</td>
<td>V</td>
<td>IOH = −400 μA</td>
<td></td>
</tr>
<tr>
<td>VMIL</td>
<td>Input Low Voltage (MOS)</td>
<td>−0.5</td>
<td>0.6</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VMIH</td>
<td>Input High Voltage (MOS)</td>
<td>3.9</td>
<td>VCC + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VMOL</td>
<td>Output Low Voltage (MOS)</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 2.0 mA</td>
<td></td>
</tr>
<tr>
<td>VMOH</td>
<td>Output High Voltage (MOS)</td>
<td>VCC − 0.5</td>
<td>V</td>
<td>IOH = −400 μA</td>
<td></td>
</tr>
<tr>
<td>ILI</td>
<td>Input Leakage Current</td>
<td>+10</td>
<td>μA</td>
<td>0 = VIN = VCC</td>
<td></td>
</tr>
<tr>
<td>ILO</td>
<td>Output Leakage Current</td>
<td>±10</td>
<td>μA</td>
<td>0.45 = VOUT = VCC</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>Power Supply Current</td>
<td>400 mA</td>
<td>300 mA</td>
<td>IA = 0°C</td>
<td></td>
</tr>
</tbody>
</table>

**A.C. Characteristics** ($T_A = 0^\circ C\ to\ 70^\circ C;\ VCC = +5V\ \pm\ 10\%$)

**System Clock Parameters**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>CLK Cycle Period</td>
<td>125 ns</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>CLK Low Time</td>
<td>53 ns</td>
<td>1000 ns</td>
<td>*5</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>CLK High Time</td>
<td>53 ns</td>
<td></td>
<td>*6</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>CLK Rise Time</td>
<td>15 ns</td>
<td></td>
<td>*1</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>CLK Fall Time</td>
<td>15 ns</td>
<td></td>
<td>*2</td>
<td></td>
</tr>
</tbody>
</table>
### A.C. Characteristics (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reset Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>Reset Active to Clock Low</td>
<td>20</td>
<td></td>
<td>ns</td>
<td>*3</td>
</tr>
<tr>
<td>T7</td>
<td>Clock Low to Reset Inactive</td>
<td>20</td>
<td></td>
<td>ns</td>
<td>*3</td>
</tr>
<tr>
<td>T8</td>
<td>Reset Pulse Width</td>
<td>4T1</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td>Control Inactive After Reset</td>
<td>T1</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td><strong>Interrupt Timing Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td>CLK High to Interrupt Active</td>
<td>85</td>
<td></td>
<td>ns</td>
<td>*4</td>
</tr>
<tr>
<td>T11</td>
<td>WR Idle to Interrupt Idle</td>
<td>85</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td><strong>Write Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T12</td>
<td>CS or DACK0 or DACK1 Setup to WR Low</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T13</td>
<td>WR Pulse Width</td>
<td>95</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T14</td>
<td>CS or DACK0 or DACK1 Hold After WR High</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T15</td>
<td>Data Setup to WR High</td>
<td>75</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T16</td>
<td>Data Hold After WR High</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td><strong>Read Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T17</td>
<td>CS or DACK0 or DACK1 Setup to RD Low</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T18</td>
<td>RD Pulse Width</td>
<td>95</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T19</td>
<td>CS or DACK0 or DACK1 Address Valid After RD High</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T20</td>
<td>RD Low to Data Valid</td>
<td>80</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T21</td>
<td>Data Float After RD High</td>
<td>55</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td><strong>DMA Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T22</td>
<td>CLK Low to DRQ0 or BRQS Active</td>
<td>85</td>
<td></td>
<td>ns</td>
<td>*4</td>
</tr>
<tr>
<td>T23</td>
<td>WR or RD Low to ARQS or DRQ1 Inactive</td>
<td>60</td>
<td></td>
<td>ns</td>
<td>*4</td>
</tr>
</tbody>
</table>

**NOTE:**
- *1 — 1.0V to 3.5V
- *2 — 3.5V to 1.0V
- *3 — to guarantee recognition at next clock
- *4 — CL = 50 pF
- *5 — measured at 1.5V
- 42.5 ns measured at 0.6V
- *6 — measured at 1.5V
- 42.5 ns measured at 3.8V
A.C. TESTING INPUT, OUTPUT WAVEFORM

2.4

1.5 ← TEST POINTS → 1.5

0.45

AC TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC 1 AND 0.45V FOR A LOGIC 0. TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC 1 AND 0.

MOS I/O - 0.1 AND 0.9 OF THE VOLTAGE SWING

TTL Input/Output Voltage Levels for Timing Measurements

T4

T3

T5

3.0V
2.5V
1.5V
1.0V
0.6V

HIGH LEVEL MAY VARY WITH VCC

MOS I/O MEASUREMENTS ARE TAKEN AT 0.1 AND 0.9 OF THE VOLTAGE SWING

System Clock MOS Output Voltage Levels for Timing Measurements

CLK

INT

T10

Interrupt Timing (Going Active)
Interrupt Timing (Going Inactive)

Reset Timing
Serial Interface A.C. Timing Characteristics

High Integration Mode

TFC is the crystal clock of the serial clock at pin 15 (TCLK).

**TFC Frequency Range:**

<table>
<thead>
<tr>
<th>For Oscillator Frequency = 1 to 16 MHz (HIGH)</th>
<th>x 8 Sampling</th>
<th>x 16 Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLK Frequency</td>
<td>0.125–2 MHz</td>
<td>62.5 kHz–1 MHz</td>
</tr>
<tr>
<td>T29 = TCLK Cycle Time</td>
<td>8 x T24</td>
<td>16 x T24</td>
</tr>
<tr>
<td>T30 = TCLK High Time</td>
<td>T24</td>
<td>T24</td>
</tr>
<tr>
<td>T31 = TCLK Low Time</td>
<td>7 x T24–10 ns</td>
<td>15 x T24–10 ns</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>For Oscillator Frequency = 0 to 1 MHz (LOW)*</th>
<th>x 8 Sampling</th>
<th>x 16 Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLK Frequency</td>
<td>0–0.125 MHz</td>
<td>0–62.5 kHz</td>
</tr>
<tr>
<td>T29 = TCLK Cycle Time</td>
<td>8 x T24</td>
<td>16 x T24</td>
</tr>
<tr>
<td>T30 = TCLK High Time</td>
<td>T25</td>
<td>T25</td>
</tr>
<tr>
<td>T31 = TCLK Low Time</td>
<td>7 x T24 + T26–10 ns</td>
<td>15 x T24 + T26–10 ns</td>
</tr>
</tbody>
</table>

* A non-symmetrical clock should be provided so that T25 is less than 1000 ns.

T24 = Serial Clock Period
T25 = Serial Clock High Time
T26 = Serial Clock Low Time

**High Speed Mode**

- Applies for TxC, RxC
- f max = 5 MHz ± 100 ppm
- For Manchester, symmetry is required: T2, T3 = \( \frac{1}{2f} \) ± 5%

**High Integration Mode**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T24</td>
<td>Fast Clock Cycle</td>
<td>62.5</td>
<td>ns</td>
<td></td>
<td>*1</td>
</tr>
<tr>
<td>T25</td>
<td>TFC High Time</td>
<td>20</td>
<td>1000</td>
<td>ns</td>
<td>*1, *14</td>
</tr>
<tr>
<td>T26</td>
<td>TFC Low Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td>*1</td>
</tr>
<tr>
<td>T27</td>
<td>TFC Rise Time</td>
<td>5</td>
<td></td>
<td>ns</td>
<td>*1</td>
</tr>
<tr>
<td>T28</td>
<td>TFC Fall Time</td>
<td>5</td>
<td></td>
<td>ns</td>
<td>*1</td>
</tr>
</tbody>
</table>

**Transmit Clock Parameters**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T29</td>
<td>Transmit Clock Cycle</td>
<td>500</td>
<td>ns</td>
<td></td>
<td>*3, *12</td>
</tr>
<tr>
<td>T30</td>
<td>TCLK High Time</td>
<td>62.5</td>
<td>1000</td>
<td>ns</td>
<td>*3, *8</td>
</tr>
<tr>
<td>T31</td>
<td>TCLK Low Time</td>
<td>*9</td>
<td></td>
<td>ns</td>
<td>*3</td>
</tr>
<tr>
<td>T32</td>
<td>TCLK Rise Time</td>
<td>15</td>
<td></td>
<td>ns</td>
<td>*3</td>
</tr>
<tr>
<td>T33</td>
<td>TCLK Fall Time</td>
<td>15</td>
<td></td>
<td>ns</td>
<td>*3</td>
</tr>
</tbody>
</table>
### High Integration Mode (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T34</td>
<td>TxD Transition-Transition</td>
<td>250</td>
<td>ns</td>
<td></td>
<td>*12</td>
</tr>
<tr>
<td>T35</td>
<td>TCLK Low to TxD Transition Half Bit Cell</td>
<td></td>
<td>*11 ns</td>
<td>*2, *12</td>
<td></td>
</tr>
<tr>
<td>T36</td>
<td>TCLK Low to TxD Transition Full Bit Cell</td>
<td></td>
<td>*10 ns</td>
<td>*2, *12</td>
<td></td>
</tr>
<tr>
<td>T37</td>
<td>TxD Rise Time</td>
<td>15</td>
<td>ns</td>
<td></td>
<td>*2</td>
</tr>
<tr>
<td>T38</td>
<td>TxD Fall Time</td>
<td>15</td>
<td>ns</td>
<td></td>
<td>*2</td>
</tr>
</tbody>
</table>

### Transmit Data Parameters (NRZI)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T39</td>
<td>TxD Transition -Transition</td>
<td>8T24-10</td>
<td></td>
<td>ns</td>
<td>*12</td>
</tr>
<tr>
<td>T40</td>
<td>TCLK Low to TxD Transition</td>
<td></td>
<td>*10 ns</td>
<td>*2, *12</td>
<td></td>
</tr>
<tr>
<td>T41</td>
<td>TxD Rise Time</td>
<td>15</td>
<td>ns</td>
<td></td>
<td>*2</td>
</tr>
<tr>
<td>T42</td>
<td>TxD Fall Time</td>
<td>15</td>
<td>ns</td>
<td></td>
<td>*2</td>
</tr>
</tbody>
</table>

### RTS, CTS, Parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T43</td>
<td>TCLK Low to RTS Low</td>
<td></td>
<td>*10 ns</td>
<td></td>
<td>*3, *12</td>
</tr>
<tr>
<td>T44</td>
<td>CTS Low to TCLK Low</td>
<td>65</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T45</td>
<td>TCLK Low to RTS High</td>
<td></td>
<td>*10 ns</td>
<td></td>
<td>*3, *12</td>
</tr>
<tr>
<td>T46</td>
<td>TCLK Low to CTS Invalid. CTS Hold Time</td>
<td>20</td>
<td>ns</td>
<td></td>
<td>*4, *13</td>
</tr>
<tr>
<td>T47</td>
<td>CTS High to TCLK Low. CTS Setup Time to Stop Transmission.</td>
<td>65</td>
<td>ns</td>
<td></td>
<td>*4</td>
</tr>
</tbody>
</table>

### IFS Parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T48</td>
<td>Interframe Delay</td>
<td></td>
<td>*5</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### Collision Detect Parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T49</td>
<td>CDT Low to TCLK High. External Collision Detect Setup Time</td>
<td>50</td>
<td>ns</td>
<td></td>
<td>*13</td>
</tr>
<tr>
<td>T50</td>
<td>CDT High to TCLK Low</td>
<td>50</td>
<td>ns</td>
<td></td>
<td>*13</td>
</tr>
<tr>
<td>T51</td>
<td>TCLK High to CDT Inactive. CDT Hold Time</td>
<td>20</td>
<td>ns</td>
<td></td>
<td>*13</td>
</tr>
</tbody>
</table>
High Integration Mode (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
</table>

Collision Detect Parameters (Continued)

<table>
<thead>
<tr>
<th>T52</th>
<th>CDT Low to Jamming Start</th>
<th>*6</th>
<th>ns</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T53</td>
<td>Jamming Period</td>
<td>*7</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Received Data Parameters (Manchester)

<table>
<thead>
<tr>
<th>T54</th>
<th>RxD Transition - Transition</th>
<th>4T24</th>
<th>ns</th>
<th></th>
<th>*12</th>
</tr>
</thead>
</table>

Received Data Parameters (Manchester)

<table>
<thead>
<tr>
<th>T55</th>
<th>RxD Rise Time</th>
<th>10</th>
<th>ns</th>
<th></th>
<th>*1</th>
</tr>
</thead>
<tbody>
<tr>
<td>T56</td>
<td>RxD Fall Time</td>
<td>10</td>
<td>ns</td>
<td></td>
<td>*1</td>
</tr>
</tbody>
</table>

Received Data Parameters (NRZI)

<table>
<thead>
<tr>
<th>T57</th>
<th>RxD Transition - Transition</th>
<th>8T24-10</th>
<th>ns</th>
<th></th>
<th>*12</th>
</tr>
</thead>
<tbody>
<tr>
<td>T58</td>
<td>RxD Rise Time</td>
<td>10</td>
<td>ns</td>
<td></td>
<td>*1</td>
</tr>
<tr>
<td>T59</td>
<td>RxD Fall Time</td>
<td>10</td>
<td>ns</td>
<td></td>
<td>*1</td>
</tr>
</tbody>
</table>

NOTES:
* 1 — MOS levels.
* 2 — 1 TTL load + 50 pF.
* 3 — 1 TTL load + 100 pF.
* 4 — Abnormal end to transmission: CTS expires before RTS.
* 5 — Programmable value: T48 + NIFS x T29 (ns)
  NIFS - the IFS configuration value.
  If NIFS is less than 32, then it is enforced to 32.
* 6 — Programmable value:
  T52 = NCDF x T29 + (12 to 15) x T29
  (if collision occurs after preamble).
* 7 — T53 = 32 x T29
* 8 — Depends on T24 frequency range:
  High Range: T24
  Low Range: T25
* 9 — T31 - T29 - T30 - T32 - T33
*10 — 2T24 + 40 ns
*11 — 6T24 + 40 ns
*12 — For x16 sampling clock parameter minimum value should be multiplied by a factor of 2.
*13 — To guarantee recognition on the next clock.
*14 — 62.5 ns minimum in Low Range.
Read Timing

DMA Request (Going Active)

DMA Request (Going Inactive)
Transmit Timings: Clocks RTS and CTS

Transmit Timings — Manchester Data Encoding

Transmit Timings — Lost CTS

6-355
Transmit Timings — NRZI Data Encoding

Transmit Timings — Lost CTS

Receive Data Timings (Manchester)
Receive Data Timings (NRZI)

Transmit Timings — Interframe Spacing

Transmit Timings — Collision Detect and Jamming
### High Speed Mode

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Transmit/Receive Clock Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T60</td>
<td>$T_{xC}$, $T_{xCC}$ Cycle</td>
<td>200</td>
<td>*13</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T61</td>
<td>$T_{xCC}$ Rise Time</td>
<td>10</td>
<td></td>
<td>ns</td>
<td>*1</td>
</tr>
<tr>
<td>T62</td>
<td>$T_{xCC}$ Fall Time</td>
<td>10</td>
<td></td>
<td>ns</td>
<td>*1</td>
</tr>
<tr>
<td>T63</td>
<td>$T_{xCC}$ High</td>
<td>80</td>
<td>1000</td>
<td>ns</td>
<td>*1, *3</td>
</tr>
<tr>
<td>T64</td>
<td>$T_{xCC}$ Low</td>
<td>80</td>
<td></td>
<td>ns</td>
<td>*1, *3</td>
</tr>
<tr>
<td></td>
<td><strong>Transmit Data Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T65</td>
<td>$T_{xD}$ Rise Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td>*1, *4</td>
</tr>
<tr>
<td>T66</td>
<td>$T_{xD}$ Fall Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td>*1, *4</td>
</tr>
<tr>
<td>T67</td>
<td>$T_{xCC}$ Low to $T_{xD}$ Valid</td>
<td>60</td>
<td></td>
<td>ns</td>
<td>*1, *4, *6</td>
</tr>
<tr>
<td>T68</td>
<td>$T_{xCC}$ Low to $T_{xD}$ Transition</td>
<td>60</td>
<td></td>
<td>ns</td>
<td>*1, *2, *4</td>
</tr>
<tr>
<td>T69</td>
<td>$T_{xCC}$ High to $T_{xD}$ Transition</td>
<td>60</td>
<td></td>
<td>ns</td>
<td>*1, *2, *4</td>
</tr>
<tr>
<td>T70</td>
<td>$T_{xD}$ Transition—Transition</td>
<td>70</td>
<td></td>
<td></td>
<td>*1, *2, *4</td>
</tr>
<tr>
<td>T71</td>
<td>$T_{xCC}$ Low to $T_{xD}$ High. (At the Transmission End.)</td>
<td>60</td>
<td></td>
<td>ns</td>
<td>*1, *4</td>
</tr>
<tr>
<td></td>
<td><strong>RTS, CTS Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T72</td>
<td>$T_{xCC}$, Low to RTS Low. Time to Activate RTS.</td>
<td>60</td>
<td></td>
<td>ns</td>
<td>*5</td>
</tr>
<tr>
<td>T73</td>
<td>CTS Low to $T_{xCC}$ Low. CTS Setup Time.</td>
<td>65</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T74</td>
<td>$T_{xCC}$ Low to RTS High.</td>
<td>60</td>
<td></td>
<td>ns</td>
<td>*5</td>
</tr>
<tr>
<td>T75</td>
<td>$T_{xCC}$ Low to CTS Invalid</td>
<td>20</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T75A</td>
<td>CTS High to $T_{xCC}$ Low CTS Setup Time to Stop Transmission</td>
<td>65</td>
<td></td>
<td>ns</td>
<td>*7</td>
</tr>
<tr>
<td></td>
<td><strong>Interframe Spacing Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T76</td>
<td>Inter Frame Delay</td>
<td>*9</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CRS, CDT, Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T77</td>
<td>CDT Low to $T_{xCC}$ High External Collision Detect Setup Time</td>
<td>45</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T78</td>
<td>$T_{xCC}$ High to CDT Inactive CDT Hold Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td>*14</td>
</tr>
<tr>
<td>T79</td>
<td>CDT Low to Jamming Start</td>
<td>*10</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T80</td>
<td>Jamming Period</td>
<td>*11</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T81</td>
<td>CRS Low to $T_{xCC}$ High. Carrier Sense Setup Time.</td>
<td>45</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T82</td>
<td>$T_{xCC}$ High to CRS Inactive. CRS Hold Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td>*14</td>
</tr>
</tbody>
</table>
### High Speed Mode (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>CRS, CDT, Parameters (Continued)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T83</td>
<td>CRS High to Jamming</td>
<td>*12</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Internal Collision Detect)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T84</td>
<td>CRS High to RxC High. End of Receive Packet</td>
<td>80</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T85</td>
<td>RxC High to CRS High. End of Receive Packet.</td>
<td>20</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Receive Clock Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T86</td>
<td>RxC Rise Time</td>
<td>10</td>
<td>ns</td>
<td>*1</td>
<td></td>
</tr>
<tr>
<td>T87</td>
<td>RxC Fall Time</td>
<td>10</td>
<td>ns</td>
<td>*1</td>
<td></td>
</tr>
<tr>
<td>T88</td>
<td>RxC High Time</td>
<td>80</td>
<td>ns</td>
<td>*1</td>
<td></td>
</tr>
<tr>
<td>T89</td>
<td>RxC Low Time</td>
<td>80</td>
<td>ns</td>
<td>*1</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Received Data Parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T90</td>
<td>RxD Setup Time</td>
<td>45</td>
<td>ns</td>
<td>*1</td>
<td></td>
</tr>
<tr>
<td>T91</td>
<td>RxD Hold Time</td>
<td>45</td>
<td>ns</td>
<td>*1</td>
<td></td>
</tr>
<tr>
<td>T92</td>
<td>RxD Rise Time</td>
<td>20</td>
<td>ns</td>
<td>*1</td>
<td></td>
</tr>
<tr>
<td>T93</td>
<td>RxD Fall Time</td>
<td>20</td>
<td>ns</td>
<td>*1</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**

- 1 — MOS levels.
- 2 — Manchester only.
- 3 — Manchester. Needs 50% duty cycle.
- 4 — 1 TTL load + 50pF.
- 5 — 1 TTL load + 100pF.
- 6 — NRZ only.
- 7 — Abnormal end to transmissions: CTS expires before RTS.
- 8 — Normal end to transmission.
- 9 — Programmable value.

\[ T76 = \text{NIFS} \times T60 \text{ (ns)} \]
\[ \text{NIFS} - \text{the IFS configuration value.} \]
If NIFS is less than 32, then NIFS is enforced to 32.

\[ *10 \] — Programmable value:
\[ T79 = \text{NCDF} \times T60 + (12 \text{ to } 15) \times T60 \text{ (ns)} \] (if collision occurs after preamble).
\[ *11 \] — T80 = 32 x T60
\[ *12 \] — Programmable value:
\[ \text{NCSF} \times \text{TTRC} + (12 \text{ to } 15) \times \text{TTRC} \]
\[ T83 = \text{NCSF} \times T60 + (12 \text{ to } 15) \times T60 \]
\[ \text{NCDF} - \text{collision detect filter configuration value.} \]

\[ *13 \] — 2000 ns if configured for Manchester encoding.
\[ *14 \] — To guarantee recognition on the next clock.
Receive Data Waveforms (NRZ)

Receive Data Waveforms
Build a VLSI-based workstation for the Ethernet environment

In a shared-resources, distributed-system environment, you can design a compact computer system using the latest chips to keep network node component count low and operations at hard-disk speeds.

Michael Webb, Intel Corp

Early distributed minicomputer systems used a variety of dissimilar networking methods, services and expansion provisions. As \( \mu \)C-based workstations invade the business environment, the need arises for standardized local networking that can quickly transfer information and make optimum use of shared resources. You can build an Ethernet workstation that's relatively simple and inexpensive by configuring a network using the system, hardware and software considerations described in this article.

Choosing Ethernet as the network environment makes good sense. For example, Ethernet makes it possible to share large, high-speed, remote file facilities and thus minimize or even eliminate the need for disk storage at the workstation. Similarly, you can eliminate printers at most stations by sharing higher quality printers that provide automatic print spooling and such special features as graphics with text and electronic typesetting.

These remote high-quality services are possible because Ethernet permits rapid, 10M-bps data transmission with even a large number of network users, and sharing makes the services cost effective. This shared approach to overall system configuration allows you to build a compact, high-performance workstation that is optimal for a local-area-network (LAN) environment.

Although most major components used in this workstation are Intel parts, many are available from second sources. You can substitute other parts with some design adjustment. For example, you can use a high-speed 68000 CPU. Its system interface is more complex, and the total parts count would increase. In the design described, the total component count can be fewer than 75 ICs, even with 256k bytes of 64k-bit dynamic RAM (32 ICs).

A system overview

Fig 1 shows an Ethernet workstation configured without disk drives or a printer. The two JEDEC 28-pin EPROMs that reside on the system bus store bootstrap, diagnostic and utility programs. The bootstrap program locates the correct file server on power-up and downloads the operating system over the net. File accesses to remote file servers depend on Ethernet speed to keep performance at the level experienced when hard disks are dedicated to a single workstation. With large system memory, say, 256k bytes with 64k RAMs or 1M byte with 256k RAMs, large applications programs and data files can reside in the workstation once loaded over the network.

Of course, to connect to the network, the workstation needs an Ethernet interface. This formerly required a board of MSI devices and one or more DMA channels. Today, a dedicated LAN coprocessor combined with an Ethernet serial interface chip can provide the complete Ethernet interface. An 82586 LAN coprocessor, which
Available shared LAN resources provide lower cost workstations

implements the full Ethernet specification and is compatible with the IEEE 802.3 LAN standard, provides buffer management both concurrently and in real time so that you can take full advantage of Ethernet performance.

Using a dual-port memory system permits the various system processors to share as much as 1M byte of 150-nsec dynamic RAM, which runs at 8 MHz, without any wait states. One memory port is tied to the system bus, while the other port is tied to a display system through a multiplexed bus. Using this dual-bus arrangement offloads the system bus from such tasks as screen refresh.

The dual-channel communications controller has two serial I/O channels, each of which can be configured for a different protocol; one of its ports is used for the workstation’s keyboard, and the other channel supports a local modem interface. The keyboard contains a battery-backed CMOS single-chip microcontroller (80C51) that controls the keyboard, has an interface for a mouse or digitizing pad and maintains semipermanent system configuration and real-time clock information even when the system is powered down.

The logic needed to read and execute high-level Ethernet communications tasks is in the LAN coprocessor. It accesses the Ethernet through a serial interface chip and a transceiver (Fig 2). The interface between the serial interface chip and the transceiver requires a crystal clock and a few capacitors and resistors.

The hardware interface between the LAN coprocessor and CPU (Fig 3) is even more straightforward, requiring only an inverter and transistor. The LAN coprocessor, which has the same pin configuration as the CPU, resides on the multiplexed CPU bus. To gain control of this bus, the LAN coprocessor uses its Hold/Hold Acknowledge (HOLD/HLDA) lines, which are directly wired to the CPU. The LAN coprocessor and CPU share a set of octal noninverting address latches and transceivers, through which they access the system bus and communicate with system memory. All commands and status concerning the Ethernet link are exchanged between the two through this memory.

The dual-port controller, through which system memory is accessed (Fig 4), gives priority to Port A, the port shared by the CPU and LAN coprocessor, thereby minimizing latency when receiving an Ethernet packet. This arrangement allows time for the

![Diagram](image-url)
VLSI reduces a board of components to a single coprocessor chip

workstation to receive a maximum-size Ethernet packet, while the text coprocessor, which resides on Port B of the memory controller, simultaneously fills a display buffer. By using a dual-port memory controller, you also shift the overhead incurred in filling a display buffer away from the main system bus, freeing it for other application tasks, such as communicating via the modem.

Because high-quality, low-cost printing is to be a network resource, your workstation should let you generate documents with different type fonts and graphics. A text display of 25 lines x 132 characters and proportional-spacing capability is suitable for most business applications. Integrating a text coprocessor into the display system (Fig 5) makes sense because it can generate the proportionally spaced text using an LSI video interface component. Add a character generator and three latches for synchronization, and the display interface is complete.

Concurrent display of data from different files in separate areas of a CRT screen, called windows, has proven desirable in business applications and should be part of any modern workstation. In the past, hardware support for multiple windows was available only in $10,000-and-up workstations. A text coprocessor with on-chip DMA simplifies list-based manipulation of multiple overlaid text windows, as used with the Xerox Star and Apple Lisa workstations.

Looking at bandwidth realities

Considering the memory-intensive nature of this design, sufficient bandwidth in the dual-port memory system is a key factor. There must be sufficient bandwidth to support the data-rate requirements for display refresh. At the same time, the LAN coprocessor must be able to access memory frequently enough during packet reception or transmission to avoid overrun or underrun in its on-chip FIFO buffers.

Fig 2.—A serial interface chip (82501) provides a path from LAN controller to cable transceiver. Its six outputs correspond with the six major signal lines of the 8-wire Ethernet cable. The remaining two lines are for dc power and return.
An Ethernet workstation should support windows

To refresh a video display, you must fill the row buffer once for each character row displayed. If you use a display with 14 scan lines per character row, you can generate characters in a 9×14-character cell, which provides good character definition. Assuming that it takes 49 μsec to write a line (this timing depends on monitor selection), it would take 686 μsec to display a single row of characters.

If the text coprocessor must wait during every memory access for the LAN coprocessor or CPU to complete a cycle, and Port A of the dual-port controller has priority, then the worst-case memory-access time would be approximately 800 nsec per word. With 132 words per row and 20% overhead for string and pointer manipulation, 126.72 μsec are needed to fill the next row buffer. This is easily within the display refresh requirements, because 686 μsec are required to display a row. Because the text processor has dual row buffers, it makes no bus accesses 81% of the time and still provides continuous screen refresh.

When receiving packets from the Ethernet, the LAN coprocessor takes charge of the bus for the time required to store two maximum-size packets (1518 bytes each) that can arrive with a minimum interframe spacing of 9.6 μsec (the worst-case situation defined in the Ethernet specification). The LAN processor uses the 9.6-μsec gap between frames to set up pointers to the next free buffer. Without an intelligent controller, this time would be insufficient to prepare for the next frame.

Data arrives over the Ethernet at 800 nsec per byte. At 800 nsec per byte multiplied by 3036 (2 × 1518) bytes, it takes 2.4288 msec to receive these frames. During this time, for 126.72 μsec of every 686 μsec, the text coprocessor is contending with the LAN coprocessor for data to load its row buffers. Under worst-case conditions, the LAN processor can write a word every 800 nsec during this contention interval; without contention, it can write a word every 400 to 450 nsec. Either way, there is plenty of time to store data coming in at 1600 nsec per word.

In fact, the LAN processor in this system can continuously store incoming packets with the minimum interframe spacing as long as receive-buffer space is

Fig 3—For the core of an Ethernet workstation, an 80186 is teamed with an 82586 LAN controller to provide computational power and network communication, respectively. The units share the local CPU bus. No random logic is needed to interface them because they have identical interface structures and timing requirements.
Fig 4—The memory system has two ports tied to the system bus and the memory bus, which communicate with the display. Offloading display-memory accesses keeps the system bus open for Ethernet operations and user applications.

Fig 5—For the CRT subsystem, the 82730 and 82731 convert bytes from the workstation bus to characters on a 25-line x 132-character CRT screen. The text coprocessor also handles multiple windows.
Providing dual-ported memory offloads the system bus available in memory. At the same time, ¼ to ½ of the bus bandwidth is still available for the CPU to continue program execution. In previous systems, program execution virtually stopped while high-bandwidth peripherals are using the bus. Because both peripherals are coprocessors, they run asynchronously and concurrently with other system activity.

With plenty of leftover bandwidth, the two DMA channels on the CPU can be used to add efficiency and performance to the dual-channel communications controller. Data from the keyboard-input buffer is transferred to the CPU via DMA, and the other DMA channel makes possible a 64k-baud synchronous or HDLC modem link on the communications controller's other port. Baud-rate timing for the two channels is generated using two of the CPU's three on-chip timers.

The CPU also directly generates chip selects, channel attentions and wait states for the system peripherals. The CPU's on-chip interrupt controller services interrupt inputs from the LAN coprocessor, text coprocessor, communications controller and other peripherals.

The software relationship

Most of the system's hardware relationships are easily grasped, but a firm understanding of the software architecture is essential to building an optimum workstation. Fig 6 shows how the LAN coprocessor interacts with the system from a software point of view. Receive and Command units are software constructs rather than physical segments of the LAN coprocessor; in reality, the same hardware performs both functions.

As previously noted, all communications between the CPU and the LAN coprocessor occur in system memory. The CPU builds a command block, stores it in memory, updates the command-block list and then activates channel attention to get the LAN coprocessor to look at the command-block list for one or more new commands. If requested by the command block, the LAN coprocessor interrupts the CPU on completion of one or more commands.

The focal point for all interaction is the system control block (Fig 7). This data structure contains chip status, pointers to the command-block list and receive-frame areas, and universal statistics on faults such as CRC errors and alignment errors.

Both the command-block list and receive-frame area use the same concept, or model, to manage data buffers for either transmission or reception. The buffer management model employs one or more arbitrarily sized buffers to construct each data frame. Pointers control and access the buffers, and linked lists manipulate them.

This model offers distinct advantages over more primitive approaches. Allowing the physical buffers to be arbitrary sizes gives the system designer maximum flexibility in selecting the buffer size and in allocation methods. Because you can locate the memory for these buffers anywhere in the 16M-byte address space, this buffer management support simplifies the task for the

![Diagram](image_url)
The CPU and LAN coprocessor communicate through shared memory

operating system's dynamic memory-allocation scheme. Communication buffers for the LAN coprocessor are dynamic by definition.

A buffer needn't be the size of the largest frame ever expected; it can be any convenient size. If a frame larger than the selected size arrives, the LAN coprocessor automatically allocates as many buffers as necessary to contain the frame, and updates the pointers and links to indicate where the frame starts and which buffers are occupied by the frame.

This flexible buffer size avoids the waste of large, dedicated buffers for receive frames when most of the frames actually received are much smaller than the maximum size (i.e., are control frames rather than data frames). Also, this automatic buffer chaining of receive data increases communications performance and efficiency; even if several frames arrive before the CPU is free to examine incoming data, the workstation seldom, if ever, misses a frame addressed to it.

The effectiveness of this buffer management scheme is perhaps best understood by examining what happens when a packet comes in from Ethernet. The LAN coprocessor's Receive section handles all frame reception activities. It manages a pool of memory space—the receive-frame area (Fig 8)—using the receive-frame and free-frame lists.

Within each list are receive-frame descriptors (RFDs) that contain status data and pointers. The RFDs in the receive-frame area point both to the first buffer that has been filled with received data, and to the first RFD of the free-frame area. The CPU can organize the pointers in linear, random or circular fashion. Once a pointer structure is adopted, the LAN coprocessor allocates buffers and maintains the proper linkage automatically.

When the LAN coprocessor begins receiving a frame, it uses the first RFD in the free-frame list to hold status and information concerning the frame, and then allocates and links in as many free buffers as necessary to contain the frame data. The linking

Fig 8—Received packets are stored in a receive-frame area consisting of two linked lists—the receive-frame list and free-frame list. The LAN coprocessor pulls into the receive-frame list as many buffers of any size as needed to store an incoming packet. Buffers used are pointed to by receive-frame descriptor (RFD) and receive-buffer descriptor (RBD) blocks. The variable-size message block saves memory, with packets stored according to actual size rather than maximum size.
process is accomplished using a receive-buffer descriptor (RBD) to point to the next buffer containing contiguous data.

Once frame data is complete, the LAN coprocessor writes the frame status into the associated RFD and the actual count of valid data into each RBD used by the frame. It then flags the last RBD used to contain the frame and updates the first RFD on the free list to point to the first free RBD. All this is done in time to catch a second frame sent to the workstation address, if one is transmitted immediately following completion of the previous frame using Ethernet's 9.6-μs minimum frame spacing.

In effect, the LAN coprocessor can receive continuous data packets from the network as long as buffer space remains available. This is achieved by dedicating a complete microcoded machine in the LAN coprocessor to generating buffer management primitives, and giving this machine DMA control to speed its bus requests. No DMA setup or control is needed from the CPU, reducing overhead and simplifying the system.

Data transmission is accomplished in a similar fashion. To transmit a frame over the Ethernet via the LAN coprocessor, the CPU constructs a command block (Fig 9). Included in this command block is a pointer to a buffer descriptor, which points to one or more buffers containing the data to transmit. If more than one buffer is used, the LAN coprocessor automatically links the buffers together as it transmits the frame. In addition, the LAN coprocessor automatically inserts the frame preamble, source and destination addresses, type field, and CRC during the transmission process. The CPU can choose to be interrupted following the transmission of one frame, or it can link together several transmission requests and be interrupted following the final frame transmission.

References

Author's biography
Michael Webb is a regional applications specialist for Intel Corp (Dallas, TX), where his primary job is to help customers define architectures and designs for distributed systems, such as the Ethernet system described in this article. No stranger to Ethernet, Mike previously worked at Xerox Corp in the Office Products Div where he was engineering manager for Ethernet communications software. Mike's hobbies include devising game software and chess.
VLSI Solutions for Tiered Office Networks

BOB DAHLBERG AND CHARLES GOPEN
CONTENTS

Introduction ................................ 1
The Tiered Network Model ....................... 1
Applications and Tiers ......................... 2
Evolution Scenarios ........................... 2
VLSI and the LAN Backbone ..................... 4
VLSI for the Human Interface Tier ............. 5
Cheapernet ..................................... 5
1 Mbps CSMA/CD LAN ......................... 5
Voice/Data PBX ................................. 6
Conclusion ...................................... 6
References ..................................... 6

APPENDIX A

INTEL LAN SOLUTIONS .................. A-1
The 82586 LAN Coprocessor .................. A-1
The 82501 Ethernet Serial Interface .......... A-1
iNA Transport Software ....................... A-2
Transport Services ............................ A-2
Network Management Services ................. A-2
User Environment ............................. A-2
Introduction

Local area networks, or LANs, were developed as a response to the development of distributed intelligence. In the past decade the performance/price ratio of microprocessors has increased well over 1000 fold. It is these low cost microprocessors that have enabled computational capability formerly residing in a centralized computer to be placed on users' desks. However, the cost of computer peripherals (such as letter quality printers, disk memories, and communication servers) has not dropped in a similar fashion (because of high electromechanical content). Also, there is an increasing need to share timely and accurate information among users in a business setting. LAN technology is the solution to these problems by allowing users to share the cost of peripherals and access common data bases.

As LANs begin to proliferate, it is becoming clear that no single network type can cost effectively meet all office users' requirements. Some applications require high data rates; for example, real time graphic display information. Other applications require the lowest cost per connection; for example, data entry terminals. This fundamental tradeoff between performance and cost drives the evolution of a tiered network architecture for the office. A model based on tiered network architecture predicts that user workstations within a department will be clustered together, and that these clusters will be interconnected through a LAN Backbone network.

Today these two types of networks (cluster and LAN Backbone) can be realized by using available VLSI technology. Intel's 82586 LAN Coprocessor supports LAN Backbone technologies such as IEEE 802.3/Ethernet. The 82586 also supports the cluster networks by realizing 1 Mbps CSMA/CD networks. 1 Mbps networks are significantly cheaper than LAN Backbone networks because lower cost cabling and electronics can be used, and fewer repeaters are required between cable segments. In the future, PBXs will play an important role in this clustering tier as true two wire voice/data communication becomes a reality.

The Tiered Network Model

An office network can be thought of as consisting of three performance tiers. End users can optimize their network cost/performance ratio by building up networks with different performance attributes.

The Three Tier Network Model is shown in Figure 1. Tier 1, the highest performance tier, is referred to as the Computer-to-Computer tier. A network in this tier is characterized by a very high data rate, 50 to 100 Mbps. Solutions for this tier take the form of loops or rings and even fiber optics. An example of this type of network is Network Systems' Hyperchannel.

Tier 2, the LAN Backbone, is a high performance tier generally operating in the 10 Mbps data rate range and cover a distance sufficient for a single building. An example of this kind of network is the IEEE 802.3/Ethernet. This tier is the main highway over which information travels throughout a building connecting expensive peripherals (e.g. laser printers and file servers) to end users located in the clustered tier.

Tier 3, or the Human Interface tier, is characterized by the clustering of end user workstations. Networking capability in this tier exhibits the most cost sensitivity because the workstations themselves are numerous and low cost ($500 for a terminal to $3000 for a personal computer). Fortunately humans can tolerate display screen latencies of 0.5 to 1 second that lower bit rates provide. These lower bit rates enable low cost networks to be realized. The need for low cost is the reason why data rates in this tier are generally 1 Mbps or less. Examples of Tier 3 networks are personal computer networks such as Corvus Systems' Omninet, Orchid Technology's PCnet, Nestar's Plan Series and IEEE 802.3 Star LAN.

Voice/data PBXs will play an important role in Tier 3. Telecommunication suppliers have a big advantage in the office in that almost everyone has a phone on his desk. Today users take advantage of this installed network capability through modems. PBX manufacturers have already begun to make cluster products available in the form of voice/data PBXs. The data rates offered are 19.2 to 64 kbps in addition to voice, which is sufficient for terminal applications. These manufacturers are already reducing the terminal/station apparatus footprint size by offering teleterminal (combined terminal and phone) products.

![A Three Tier Network Office/Commercial](image-url)

**A Three Tier Network Office/Commercial**

**A TIERED NETWORK OPTIMIZES COST AND PERFORMANCE**

Tier 1: Computer-to-Computer (20-100 Mbps)

Tier 2: LAN Backbone (2-5 Mbps)

Tier 3: Workstation Cluster (64 kbps to 2 Mbps)

A Three Tier Network Model

Figure 1. Three Tier Network Model

6-372 231194-002
The tiered network model is analogous to a road system. Tier I is the 10 lane freeway in a major metropolitan area. This highway is responsible for moving very large volumes of traffic. This type of highway is very expensive to build, but the traffic volume warrants it. Tier 2 can be thought of as an Interstate Highway in which a large amount of traffic can be transported over long distances. Tier 3, or the Human Interface tier, can be thought of as the streets within a city which interconnect onto the interstate highway. In this scheme, no single user has a freeway butting up to his driveway; in a similar fashion no user is connected to a Tier 1 network. This tiered approach maximizes the performance of Tier 2, because most of the Tier 3 traffic stays within Tier 3; just as farm tractors primarily stay on dirt roads, not super highways.

Another way to view the model is to draw an analogy to microprocessors. To meet the requirements of diverse applications there are 4-bit, 8-bit, 16-bit and 32-bit processors available. Nobody questions that a 32-bit microprocessor is overkill for a microwave oven. In a similar fashion no single network can cost effectively solve the problems of each networking need. It is through this tiered approach that users achieve the best cost/performance ratio for moving vehicle traffic.

Applications and Tiers

This model can be mapped into application performance requirements found in the office. Figure 2 shows a graph of cost and performance for various applications. Experience has shown that end users are willing to pay no more than 10 to 15 percent of their system cost in order to obtain data communication capability; this percentage is an important assumption.

Application data rate requirements can be placed into three groups analogous to the three tiers of Figure 1. At the very high end is the computer-to-computer communication requirements, in which end users will spend $50k to $60k per connection.

At the high end is the CAD/CAM user requirements in which very expensive peripherals such as electrostatic plotters and disks need to be shared. The cost of an Ethernet connection, $1k to 1.5k, is very affordable at this tier.

At the lowest end is the terminal and personal computer requirements. This application space spans a wide spectrum of performance requirements. At the low end, data entry can tolerate very low data rates with not much performance degradation, and consequently is the most cost sensitive. Using modems as a benchmark, users are willing to pay upwards to $450 for a 1200 bps serial connection. At the higher end, resource sharing and graphic requirements for PCs require in the range of 1 Mbps. Popular personal computer LANs cost $500 to $1000 per connection (not including wiring cost).

Networking at Tier 3 provides an overall lower cost solution because the cost of the network is less than the cost of each user having his own peripheral. This observation is validated in that a major trend in the market place today is diskless workstations.

It is the wide range of personal computer and terminal data rate requirements that make the Tier 3 the most interesting. It is possible for a user to spend too much for performance he will not use. In fact, for personal computer networks, bit rate is not the major limitation, rather it is the restrictions of electro-mechanical peripherals such as Winchester disks and software overhead on the local CPU that cannot keep up with 1 Mbps continuous (as opposed to bursty) data rates.

Evolution Scenarios

Two scenarios have been developed to explain how a tiered network will be realized in the real world. Scenario 1, the local optimization scenario, assumes that departments within an organization will make their networking decision in isolation.

In this scenario the particular application requirements of a department are very well known. For example, an Engineering department has very high data rate requirements to support its CAD environment; whereas Sales has low data rate requirements for their order entry and order inquiry needs. Because the applications are well known, a decision can be made quickly on which network to purchase. Departmental budgets usually can cover the costs of these networks, so approval of a higher authority is not required. The result is that each department will develop its own cluster network (Tier 3).
However, over time many departments will develop their own cluster tier; each department will realize they have a need to interconnect among each other. For example, the Marketing department may have to access cost information from the Finance department as well as last month’s order rate from Sales. When cluster-to-cluster communication requirements become important, the company will make a conscious decision to provide interconnect capability. This interconnect capability is realized through the LAN Backbone. A growing concern is whether gateways/bridges will exist. This concern leads to Scenario 2.

Scenario 2, the global optimization scenario, occurs when the users make a conscious decision to solve their networking requirements at one time. In this scenario, the decision is centralized because it impacts the entire operation or company. The advantage of this approach is built-in capability to interconnect the users. However, at this time the decision can be very difficult because the technology is not stable, and user requirements are not fully understood.

In this scenario the Tiered Network model predicts that clustering will occur as well. For example, it will not be cost effective for each user to connect onto an Ethernet cable ($1500 cost). Thus, each department will have a cluster optimized for its particular application interconnecting through a LAN Backbone.
To summarize, we can see that there is no single network that solves all user problems. Whether a user optimizes locally or globally, clustering is likely to occur. Each end user group will have a cluster that is optimized for its particular application requirements. It is through this clustering with interconnection through a LAN Backbone that end users will realize the most cost effective network.

VLSI and the LAN Backbone

The IEEE 802.3/Ethernet standard has gained wide acceptance by a number of system suppliers. IEEE 802.3’s popularity has been driven primarily by its acceptance by major minicomputer manufacturers, the approved IEEE specification itself, and the availability of low cost VLSI controller chips. From a technical viewpoint, the IEEE 802.3 shares the benefits of Carrier Sense Multiple Access/Collision Detection, CSMA/CD, technology. These benefits are:

1. Proven technology. Ethernet has been in use since 1975 by Xerox. The technology is well-understood, and has resulted in the IEEE standard.
2. Performance. Elimination of the centralized (or hierarchical) control network communications results in greater efficiency and bandwidth utilization and shorter delay in getting the message to its destination.
3. Reliability. The CSMA/CD media access method enables the network to operate without central control or switching logic. If a station on the network malfunctions, it does not affect the ability of other stations to intercommunicate.
4. Easy expansion. The passive, distributed nature of a CSMA/CD network permits easy expansion. Stations can be added to the existing network without reinitialization of all the other stations. Such capability supports future growth requirements through simple expansion of the network.

Figure 3 shows the basic building blocks for an IEEE 802.3/Ethernet system and how it relates to the International Standards Organization (ISO) Open Systems Interconnect model for networking. Basic components consist of a coaxial cable for transmission media, a transceiver to transmit and receive signals that come over the media and detect collisions, a transceiver cable to connect the data terminal equipment to the transceiver which allows flexibility of the location of the terminal, and a controller board.
Today, Intel supplies VLSI for the controller board function. Intel's 82586 LAN Coprocessor performs the IEEE 802.3 data link functions without any CPU involvement:

* frame assembly/disassembly
* handling of source and destination addressing
* detection of physical channel transmission errors
* CSMA/CD network link management
  —collision detection
  —backoff and retransmission after a collision

In addition the 82586 supports the designer with diagnostic capability to make system design easier. For example DMA underrun and overrun errors, frames that are received in error, and number of deferrals are reported. Loopback capability is allowed to facilitate self diagnostics. These capabilities are performed without any involvement from the host CPU.

Intel's 82501, Ethernet Serial Interface, performs Manchester encoding and decoding of the data and timing information.

More details on operation and design support capabilities of the 82586 are included as an appendix to this paper.

### VLSI For The Human Interface Tier

From a technology viewpoint, the Human Interface Tier is an interesting one. Traditional computer manufacturers and PBX manufacturers are providing solutions that leverage their traditional strengths. Computer manufacturers are providing solutions via LANs based on their data communications expertise. PBX manufacturers, on the other hand, are beginning to offer voice/data PBXs. While these are two competing technologies, both suppliers realize they do not have the complete solution. Minicomputer and PBX manufacturers have cooperated in developing the standard "Computer-to-PBX" interface. These technologies are discussed in greater detail below:

### Cheaperernet

Within the IEEE 802.3 committee is a subgroup defining a lower cost version of Ethernet called Cheaperernet (also known as Thin Ethernet or Skinny Ethernet). Cheaperernet maintains Ethernet's 10 Mbps data rate, but cost is reduced through a lower cost cabling scheme. Ethernet's yellow cable, cable tap box, and transceiver drop cable are replaced by low cost RG58 CATV coaxial cable. The Ethernet transceiver function is located within the terminal itself. The coax cable is attached directly to the terminal through a T-connector. Installation does not require a specialized craft person to install.

While this approach is lower in cost than Ethernet, it has two limitations. First, the segment length is restricted to 185 meters. For the office this distance limitation requires the use of repeaters that increase the cost and reduce system reliability. Second, the cable/terminal (ground) isolation scheme is the same as for Ethernet which requires D.C. isolation between the transceiver and the terminal (because of ground). This isolation scheme limits the potential cost reduction because it does not allow integrating the transceiver, encoder/decoder and controller functions into a single chip. Ethernet/Cheaperernet require DC/DC converters to the transceiver.

### 1 Mbps CSMA/CD LAN

Today there are a number of personal computer network products that are unique to a single vendor. These networks lack the ability to electronically (physical link) interconnect, much less have compatible software link among other vendors. These networks are characterized by bit rates in the 1 Mbps area and are generally of the CSMA variety. In an effort to see a standard emerge in this area, Intel is working with AT&T, Wang, Tandem, Toshiba, and others to arrive at a 1 Mbps standard within the IEEE 802.3 committee.

1 Mbps networks offer a lower cost of connection than do 10 Mbps networks. First, cabling cost can be reduced by using low cost CATV coax, or twisted pair wire. Second, the length of cable segments can be much greater for 1 Mbps than in 10 Mbps technologies: going from less than 200 meters in Cheaperernet, to 500 meters for Ethernet to over 1000 meters for 1 Mbps CSMA/CD. Longer cable segments mean few repeaters are needed on the network. Third, is that 1 Mbps networks allow VLSI interface costs to be reduced significantly. For 1 Mbps networks, it is possible with available technology to cost effectively integrate the controller function with the serial interface function and the transceivers into one chip. This level of integration is not achievable in Ethernet/Cheaperernet networks because the transceiver chip and serial interface chip are electrically isolated through transformers as mentioned above.

A concern is that 1 Mbps may not offer adequate performance for personal computer applications. The performance of 1 Mbps networks, such as Omnimet and PCnet, is not limited by the serial bit rate, but rather electro-mechanical peripherals, particularly Winchester disk access time. Network performance (as measured by the time required for many users to download a common file) can be significantly (3-4 x) improved by using "RAM Disks" within the file server. RAM Disks are really extensions of the file server's local RAM memory that can hold commonly accessed files (such as a spreadsheet program or BASIC language). Several personal computer network vendors already have these products available.
1 Mbps CSMA/CD networks can be cost effectively realized using the 82586 LAN Coprocessor from Intel. The 82586 is unique among present LAN controllers in that data rates and CSMA/CD network parameters (slot time, back-off priority, framing, etc.) are programmable. This programmability allows the 82586 to be used as a 1 Mbps controller. The advantage of this approach is that software developed for Ethernet workstations can be immediately transferred to 1 Mbps networks because the system interfaced to the 82586 remains the same. Available 1 Mbps Manchester encoders/decoders and a low cost discrete transceiver complete the 1 Mbps physical interface. Future cost reductions can be realized by integrating the controller and Manchester encoder/decoder and transceiver functions onto a single chip.

Voice/Data PBX

Many PBX manufacturers are touting voice/data capability. This capability usually takes the form of four wire systems in which voice and data are carried over separate twisted wire pairs. The data rates generally are 19.2 kbps or 56 kbps, depending on the asynchronous and synchronous nature of the data. Fourth generation PBXs, some using two wires, are beginning to enter the market now and will continue through the 1980's. Even these products have data rates ranging from 64 to 128 kbps, although 256 kbps for data is talked about. These data rates are adequate only for the Human Interface Tier.

Presently PBX manufacturers are focusing on the terminal application market as indicated by the numerous IBM 3270 interfaces offered. A 19.2 kbps data rate is more than adequate for data entry, data inquiry and editing applications. It is not clear whether this data rate is adequate for personal computers. Certainly for a personal computer working in an editing type environment, this performance is adequate. The PBX may not be adequate for applications that require heavy use of file access, file transfers and graphics.

Voice/Data PBX

Many PBX manufacturers are touting voice/data capability. This capability usually takes the form of four wire systems in which voice and data are carried over separate twisted wire pairs. The data rates generally are 19.2 kbps or 56 kbps, depending on the asynchronous and synchronous nature of the data. Fourth generation PBXs, some using two wires, are beginning to enter the market now and will continue through the 1980's. Even these products have data rates ranging from 64 to 128 kbps, although 256 kbps for data is talked about. These data rates are adequate only for the Human Interface Tier.

Presently PBX manufacturers are focusing on the terminal application market as indicated by the numerous IBM 3270 interfaces offered. A 19.2 kbps data rate is more than adequate for data entry, data inquiry and editing applications. It is not clear whether this data rate is adequate for personal computers. Certainly for a personal computer working in an editing type environment, this performance is adequate. The PBX may not be adequate for applications that require heavy use of file access, file transfers and graphics.

Intel currently offers a family of components specifically designed to facilitate the design of voice/data PBXs. At the heart of the system is the 2952 Integrated Line Card Controller. This device supports 8 analog or digital subscribers simultaneously. It includes an interface to 2 PCM highways and 1 HDLC control highway. Analog subscribers interface to the 2952 through the 29C51 high feature CHMOS combo. The combo embodies both PCM codec and anti-alias filter functions on chip. In addition, integrated signaling test and line balancing are performed by the 29C51. Future products will allow PBX manufacturers to easily upgrade their 2952 based products to include true two wire voice/data subscribers.

Conclusion

There is no single local area network that meets every user's needs cost effectively. IEEE 802.3/Ethernet offers users a high performance Local Area Network suitable for a LAN Backbone, but it is too costly for personal computer and terminal networking. 1 Mbps networks and voice/data PBXs solve this problem. At present, Intel's 82586 LAN Coprocessor is the only VLSI chip that solves both Ethernet and 1 Mbps LAN requirements while simultaneously maintaining software compatibility from the system point of view. In the future it can be expected that LAN controllers optimized for 1 Mbps networks that include on chip encoder/decoder and transceiver functions will appear. Intel also offers a family of components to facilitate the realization of voice/data PBXs.

In the long run, office networks will be structured into department clusters that will be interconnected through a LAN Backbone or PBX. The ultimate choice will be related to application performance requirements.

References:
Intel offers a broad range of products to realize LANs. These products are in the form of components (82586 and 82501), boards (ISBC 186/51), and network software (INA 960). See Figure A. A functional summary of components and software solutions is below:

**The 82586 LAN Coprocessor**

The 82586 is an intelligent peripheral that completely manages the processes of transmitting and receiving frames over a network. It offloads the host CPU of the tasks related to managing communication activities. More importantly, it does not depend on the host CPU for time critical functions (e.g. transmission and reception of frames) because it contains its own processor allowing it to be a coprocessor along with the host CPU.

The 82586 interfaces easily to available microprocessors. Systems requiring minimum component count can take advantage of its direct interface (no 'TTL glue') to Intel's 80188 (8-bit bus) and 80186 (16-bit bus) microprocessors.

The 82586 efficiently uses memory through data chaining. System memory is not wasted because short frame (75% of network traffic is less than 100 bytes) can be saved in minimal size buffers, while long frames are stored by successively chaining buffers together. It manages this chaining process without CPU intervention, thereby maintaining high system performance.

The 82586 facilitates network management by maintaining error tallies in system memory to count:

- Number of frames incorrectly received due to CRC errors
- Number of frames incorrectly received due to misaligned frames

The 82586 counts number of collisions that occurred while attempting to transmit a specific frame which is an indicator of traffic loading. It also monitors the transceiver's collision detection failure reporting mechanism.

The 82586 assists in developing and maintaining LAN systems by maintaining tallies that count the:

- Number of frames lost due to lack of receive buffers
- Number of frames lost due to DMA overrun while receiving frames

---

**Figure A. Intel LAN Solution**

- **Solutions**
  - Network Management
  - INTEL INA 960, TRANSPORT SOFTWARE/NETWORK MANAGER
  - INTEL 82586, LAN CO-PROCESSOR
  - INTEL 82501, ETHERNET SERIAL INTERFACE

- **Standards**
  - ISO 8073
  - IEEE 802.3/ETHERNET
The 82586 provides diagnostic capability via internal and external loopback service. Distance to cable breaks and shorts is provided by on-chip time domain reflectometry.

The 82586's network parameters are programmable so that LANs optimized to specific applications can be realized; for example: broadband networks, short topology networks that require higher throughput than IEEE 802.3 and low cost (1 Mbps) networks.

The 82501 Ethernet Serial Interface

The 82501 is designed to work directly with the 82586 in 10 Mbps LAN applications. The primary function of the 82501 is to perform Manchester encoding/decoding, provide 10 MHz transmit and receive clocks to the 82586, and to drive the transceiver cable. The 82501 provides for fault isolation via an internal loopback. Continuous transmission (babbling) is prevented by an on-chip watchdog timer.

iNA 960 Transport Software

iNA 960 is a general purpose Local Area Network software package that provides the user with guaranteed end to end message delivery. iNA 960 conforms to the International Standards Organization's 8073 specification including up to Class 4 transport layer services. iNA 960 also provides network management functions, and 82586 device drivers.

Transport Services

The iNA 960 transport layer implements two kinds of message delivery services: virtual circuits and datagram. Virtual circuits provide a reliable point-to-point message delivery service ensuring maximum data integrity and are fully compatible with the ISO 8073 Class 4 protocol. In addition to guaranteeing message integrity, iNA 960:

- Provides flow control (data rate matching between sender and receiver)
- Supports multiple simultaneous connections (process multiplexing)
- Handles variable length messages (independently of physical frame size)
- Supports expedited delivery (to transmit urgent data)

The datagram option provides 'best effort' delivery service for non-critical messages. The datagram service does not guarantee message integrity but requires less channel overhead than virtual circuits.

Network Management Services

The Network Management facility supports the users of the network in planning, operating and maintaining the network by providing network usage statistics, by allowing the monitoring of network functions and by detecting, isolating and correcting network faults.

The Network Management facility also supports up-line dumping and down-line loading of data bases or to boot systems without a local mass storage.

User Environment

In the iRMX (Intel’s real time, multitasking operating system) environment, both the user programs and iNA 960 run under iRMX 86. The communications software is implemented as an iRMX 86 job requiring the nucleus only for most operations. The only exception is the boot server option, which also needs the Basic I/O System. iNA 960 will run in any iRMX environment including configurations based on the 80130 software on silicon component.

In those systems where iRMX 86 is not the primary operating system, or where off-loading the host of the communication tasks is necessary for performance reasons, the user may wish to dedicate a processor for communication purposes. iNA 960 can be configured to support such implementations by providing network services on an 8086, 8088, or 80186 microprocessor.
Chips Support Two Local Area Networks

BOB DAHLBERG
LAN Component Product Line Manager
CHIPS SUPPORT TWO LOCAL AREA NETWORKS

Data communication ICs permit easy implementation of Ethernet and high level data-link control networks.

by Bob Dahlberg

The main rationale for local networks is resource sharing. Today, small, powerful computers using VLSI components sell for less than $2000. Under the circumstances, companies intending to use several such systems are reluctant to equip each one with a disk drive and printer that could more than double the price per station. Rather, they prefer to share disks and printers among several systems in order to spread the cost of peripherals across several users.

By connecting these small computers to a local area network (LAN), resource sharing with little degradation in overall system performance becomes practical. However, if the network interface costs $1000 or more per computer, the economic advantage of resource sharing wanes. Thus, network interface cost is a primary criterion in selection, particularly for low cost computers.

Access methodologies represent another important factor in network selection. And, although an equal access, first-come, first-served method might be appropriate for an office system environment, it could be the curse of a process control system. In the latter case, a priority-based (or controlled) access method might be the only realistic choice.

All else being equal, networks supported by available LSI and VLSI components exhibit cost and development speed advantages over board-based LANs. Now, available chips support both priority-based and equal access schemes. One such network is based on the IEEE 802.3 specification, while another uses a variety of physical interface schemes overlaid by high level data-link control/synchronous data-link control (HDLC/SDLC) protocols.

Costly copper

In short distance networks, one can choose a serial, two-wire scheme or a parallel, multiwire interface. Parallel bus structures are implicitly faster than serial structures but tend to be more expensive and less reliable. The amount and cost of the copper
The networks described are both serial, two-wire types. The data after learning that no match existed between the request for a larger group of data. Thus, the networks described are both serial, two-wire types.

A fundamental assumption in data communications is that noise will corrupt the transmitted data. Error detection schemes can be used to determine the validity of received data. One common data error detection method applies a numerical algorithm to the message bit pattern and produces a unique sum. This sum is appended to the end of the message and is used by a receiving system as a quick check for the proper bit pattern. Called a cyclic redundancy check (CRC), this process permits the validity of received data.

As needs grow, users may want to add more workstations and intelligent peripherals to a network. It would be ideal to attach each station to the network by simply connecting the station directly to the serial network bus cable. This is called a multidrop configuration and it resembles a party line telephone circuit [Fig 1(a)]. As a party line, each station attached to the cable receives all the data transmitted on the cable. In order to route messages to their intended recipients, the messages are logic switched, or specifically addressed, to one or more receiving stations. All others will ignore the data after learning that no match existed between their addresses and those of the data being sent.

Each data packet or frame contains a set of address bits that determines which stations receive the data. In a sense, address bits constitute overhead because they are not part of the information being sent between stations. Any loss in data transfer efficiency, however, is made up by the simplicity of the network expansion interconnect scheme.

The Ethernet specification (a modified version of which was recently accepted as IEEE standard 802.3) describes its physical link characteristics in detail. Coaxial cable is used as the network cable bus, and each station is connected to that cable via a transceiver and transceiver cable. Minimum distance between station transceivers is 2.5 m, and a network segment can extend to 500 m (and contain up to 200 nodes). Because up to five segments can be joined using active repeaters between each segment, the overall Ethernet network can be 2500 m long and support up to 1000 nodes. Individual nodes can connect to more than one station, and the number of stations connected to an Ethernet network can exceed 1000.

Data is sent at a 10-Mbit/s rate using a self-clocking Manchester encoding format. Only one data packet can be sent at a time using Ethernet, and access is on a first-come, first-served basis. Carrier sense multiple access/collision detection (CSMA/CD) methodology is used. The maximum and minimum distances between transceivers are derived from the CSMA/CD requirements based on interframe-spacing and the collision detection procedures.

A second alternative requires no specific physical link. Speed, distance, and cost parameters dictate actual implementation. The simplest and least expensive method is to drive a twisted-pair cable with off-the-shelf transceiver chips.

Choosing protocols
Both the IEEE 802.3/CSMA/CD and the HDLC/SDLC protocols provide logic-switched messaging and frame-by-frame error detection. IEEE 802.3/Ethernet treats each station equally and does not permit priority network access, whereas HDLC/SDLC enforces a primary/secondary hierarchy [Fig 1(b)]. A primary station controls the overall network by issuing commands to the secondary stations. Secondary stations comply with the primary station's commands and access the bus for retransmitting data only in response to those commands. Unlike Ethernet, which is based on probabilistic network access, HDLC/SDLC provides deterministic (or controlled) access.

SDLC is an IBM standard communication protocol and a subset of HDLC, a standard communication link control established by the International Standards Organization (ISO). HDLC and its subset are data-transparent protocols, which means the arbitrary data streams can be sent without concern that some of the data might be mistaken for control characters. Thus, unlike the BISYNC protocol and its controller, an HDLC/SDLC controller need not detect special characters except for the unique opening/closing flag bytes. Moreover, unlike an
Asynchronous protocol and its controller, the HDLC/SDLC need not provide start and stop bits.

Both HDLC/SDLC and Ethernet protocols specify particular message formats (or frames). The HDLC/SDLC protocol consists of five basic fields—flag, address, control, data, and error detection. Each frame is enclosed by an opening and closing flag. Both the opening and closing flags form a similar bit sequence—0111110—that is an individual character in SDL/HDL. Inserting a 0 in the information data flow whenever a sequence of five 1s occurs achieves flag character individuality in SDL/HDL. These inserted 0 bits are automatically stripped out upon reception. For SDLC, the address field is 8 bits wide, but can be 2 (or more) bytes long in HDLC. Similarly, the control field in SDLC is 8 bits wide, but can also be longer in HDLC. The SDLC data or information field can contain any number of bytes. However, the same is true for HDLC in certain instances where the data field must end on an 8-bit boundary. Finally, the frame check sequence field contains the 16-bit CRC result for all of the bits between flags (Fig 2).

Three types of frames are used in HDLC and SDLC. A nonsequenced frame establishes initialization and control of the secondary stations. A supervisory frame handles control, and an information frame is used for data transfers.

The SDL protocol appears in low cost asynchronous modems using nonreturn to zero inverted (NRZI) coding and decoding. NRZI coding is used at the transmitter to enable clock recovery from data at the receiver terminal. Clock recovery is accomplished using a digital phase locked loop technique. NRZI coding specifies that the signal condition does not change for transmitting a 1, but changes state whenever a 0 is transmitted. Hence, NRZI coding ensures that an active data line will have a transition at least every 6 bits time (by virtue of the 0-bit insertion requirement). Both 0-bit insertion and NRZI coding/decoding maintain the data transparency characteristics of the HDLC protocol and its SDLC subset.

Like HDLC/SDLC, Ethernet specifies a frame format (Fig 3). It contains a destination field, source field, frame type field, data field, and a frame check sequence. The destination and source fields both contain 6 octets (8 serial bits), for a total of 48 bits. The type field contains 2 octets. The data field can have as few as 46 octets or as many as 1,500. Finally, the frame check sequence consists of 4 octets, allowing a 32-bit CRC code to be calculated and appended to the rest of the frame. The first transmitted Ethernet frame is preceded by a 64-bit preamble, made up of seven groups of 10101010 followed by an eighth group of 10101011. The next bit that follows is the first bit of the first destination octet.

In the CSMA/CD scheme, a "collision" occurs when two stations attempt to gain access to the bus at the same time. Thus, it is important that all stations on the network are notified of the collision. This way, any transmitted data can be flagged as invalid. To solve this problem IEEE 802.3/Ethernet specifies that, after collision detection, transmitting stations send a jam signal to ensure that stations on the network recognize the collision. At the end of the jam interval, each station delays bus access according to an individually calculated random backoff time interval. Should a collision occur again when bus access attempts are renewed, the next backoff interval increases in length. Up to 16 repeated attempts can occur before a system fault is automatically assumed. Thus, even during periods of high bus demand, ample bandwidth should be available and delays relatively short.

**It's in the chips**

Any of the working LANs can be implemented using various components. If there is enough time and a large budget, custom VLSI chips can be
developed and an elegant solution forged. Most engineers, however, have neither luxury. For this reason, the two networks selected are supported by off-the-shelf VLSI components.

Intel's 8273 and 8274 data communication controller ICs offer HDLC/SDLC capabilities. Teamed with a microprocessor and some random logic ICs, a capable network data-link controller could be built. The 8051 single-chip microcontroller has become a popular component for many terminal applications because of its high performance 8-bit CPU, large internal program and data memory capacity, plus onchip counter timers and interrupt controllers. In addition, Intel has combined an intelligent HDLC/SDLC controller and 8051 core processor onto a single chip, the 8044. The resulting single-chip microcontroller with onchip serial communication controller allows low cost network terminal and peripheral design.

Each station would contain an 8044 (with its programmable I/O ports to provide local control) and serial HDLC/SDLC interface. Thus, to manage the network interface, 8044-based stations would be capable of acting as a secondary station within an HDLC/SDLC network (Fig 4). Since data transfer speed and electrical characteristics are not specified for these protocols, the designer has a wide choice in tailoring the physical link to the application. The single VLSI device provides local intelligence and network management, thus permitting low cost network development.

Various Ethernet controllers have been announced, with several already sampled and available. Among these is the 82586 general-purpose CSMA/CD controller. It is designed to come up in the Ethernet mode on power up, but can be programmed for other parameters as well. A companion chip (the 82501) provides the Manchester encoding/decoding function between the 82586 and a transceiver.

This chip pair operates in conjunction with the IA686 microprocessor family, and is most cost-effectively used with the 80186 microprocessor. The 80186 and 82586 have identical bus interface and control signal requirements. Hence, they can be linked without adding random logic ICs. Essentially, these three ICs—the 80186, 82586, and 82501—provide the basis for an Ethernet interface. Therefore, only some buffer memory and bus interface chips are additionally required (Fig 5).

A subsystem built using these components provides an intelligent Ethernet interface that can continuously operate at the full 10-Mbit/s network speed. Moreover, these components can implement a complete computer and communication system. It is therefore possible to create an appropriate and usable Ethernet workstation out of these few VLSI components.

**Different strokes**

The HDLC/SDLC-based network is intended for non-Ethernet applications. HDLC/SDLC has become an accepted standard supported by a variety of hardware and software products. There is no specified standard for physical link implementation or for the software layers beyond the data-link level. Therefore, networks based on these protocols are usually "closed." That is, the vendor provides all the pieces to the network. Vendors, of course, are familiar with their own network architecture and are free to provide compatible systems. But such networks do not encourage others to develop compatible systems unless the vendor's market share is large enough and vulnerable enough to attract competition. The IBM SNA is an example.

HDLC/SDLC-based LANS are suitable for system clusters where distances are less than those of Ethernet, and where priority access is important. Networks within a box (eg, a copier), and networks on table-tops (eg, an instrumentation cluster), are examples. Although there is a parallel bus interface standard (IEEE 488), an instrumentation manufacturer may want to provide for longer distances using two-wire cables and simpler protocols.

An HDLC/SDLC LAN cluster could also be used for process control applications and data acquisition systems. An example is Intel's recent distributed control module products for the factory. Again, a priority bus access capability would be important in these applications. Office system applications where Ethernet offers too much performance at too high a cost (eg, an electronic typewriter networked to a file server) might use this network as well.

The concept of open-system compatibility comes from the ISO's Open System Interconnection (OSI) model. This provides a seven-layer model in which each layer is characterized by a unique set of functions and a specific interface to adjacent layers. The goal is to eventually arrive at a set of standards that would permit systems from several vendors to communicate with one another through common physical, data-link, and software layer protocols.

Xerox Corp developed Ethernet as a local network for its systems, but the company later joined with Digital Equipment Corp and Intel to develop a set of specifications for Ethernet that would allow it to
map into the first two layers of the OSI model—physical and data-link. The IEEE adopted its 802.3 specification as a result of these efforts. Efforts to develop standards for the other layers continue. An example is the ISO transport layer protocol, 8073, which provides "return receipt" quality communication services.

Today, Ethernet supports OSI physical and data packet level protocols. It is an emerging technology that is still closer to the top than to the bottom of the learning (and pricing) curve. Nevertheless, many vendors support Ethernet and will no doubt manufacture products equipped to swap data with other Ethernet systems.

**Open and closed**

Office automation constitutes the biggest apparent application area for Ethernet. The office has traditionally been a multivendor site in which the computer, copier, and printer are likely to come from different vendors. An open system appeals to users seeking vendor independence. When the LAN concept was first proposed, it was described as an all-encompassing network, connecting all the intelligent subsystems throughout a facility. In fact, that is not the way local network installations have progressed. Instead, clusters of user stations (typically 10 or so) are cropping up in various places within a facility. Most analysts expect local networking to occur in tiers. The cluster tier provides the lowest cost per connection. An example is a 1-Mbit/s CSMA/CD LAN used for personal computers. Clusters would be interconnected through a longer and faster data highway (called a LAN backbone) such as Ethernet.

Will closed and open networks be able to cooperate and coexist? Quite simply, they have to. Economics will determine the network types used for connecting the systems within a cluster, and standardization will drive the methods by which clusters are ultimately joined.

Closed systems, such as microcontrollers connecting the HDLC/SDLC-based network, represent the least expensive and most flexible LAN configuration. Open systems, because of the push for standardization and subsequently larger user base, are more likely to benefit from future cost reduction through multiple-sourced VLSI components than closed systems. Similarly, open systems probably attract more third-party suppliers and enjoy greater variety and lower cost software.

Gateways will join closed and open systems. These hardware/software intermediaries will pave the way for data transfer between formerly incompatible networks. By such means, a closed engineering workstation network will gain access to information stored in the corporate data base and be available on the Ethernet data highway.
8291A
GPIB TALKER/LISTENER

- Designed to Interface Microprocessors (e.g., 8048/49, 8051, 8080/85, 8086/88) to an IEEE Standard 488 Digital Interface Bus
- Programmable Data Transfer Rate
- Complete Source and Acceptor Handshake
- Complete Talker and Listener Functions with Extended Addressing
- Service Request, Parallel Poll, Device Clear, Device Trigger, Remote/Local Functions
- Selectable Interrupts
- On-Chip Primary and Secondary Address Recognition
- Automatic Handling of Addressing and Handshake Protocol
- Provision for Software Implementation of Additional Features
- 1–8 MHz Clock Range
- 16 Registers (8 Read, 8 Write), 2 for Data Transfer, the Rest for Interface Function Control, Status, etc.
- Directly Interfaces to External Non-Inverting Transceivers for Connection to the GPIB
- Provides Three Addressing Modes, Allowing the Chip to be Addressed Either as a Major or a Minor Talker/Listener with Primary or Secondary Addressing
- DMA Handshake Provision Allows for Bus Transfers without CPU Intervention
- Trigger Output Pin
- On-Chip EOS (End of Sequence) Message Recognition Facilitates Handling of Multi-Byte Transfers

The 8291A is an enhanced version of the 8291 GPIB Talker/Listener designed to interface microprocessors to an IEEE Standard 488 Instrumentation Interface Bus. It implements all of the Standard's interface functions except for the controller. The controller function can be added with the 8292 GPIB Controller, and the 8293 GPIB Transceiver performs the electrical interface for Talker/Listener and Talker/Listener/Controller configurations.

Figure 1. Block Diagram

Figure 2. Pin Configuration
8291A FEATURES AND IMPROVEMENTS

The 8291A offers the following improvements to the 8291:

1. **EOI** is active with the data as a ninth data bit rather than as a control bit. This is to comply with some additions to the 1975 IEEE-488 Standard incorporated in the 1978 Standard.

2. The BO interrupt is not asserted until RFD is true. If the Controller asserts **ATN** synchronously, the data is guaranteed to be transmitted. If the Controller asserts **ATN** asynchronously, the SH (Source Handshake) will return to SIDS (Source Idle State), and the output data will be cleared. The, if **ATN** is released while the 8291A is addressed to talk, a new BO interrupt will be generated. This change fixes 8291 problems which caused data to be lost or repeated and a problem with the RQS bit (sometimes cannot be asserted while talking).

3. LLOC and REMC interrupts are setting flipflops rather than toggling flipflops in the interrupt backup register. This ensures that the CPU knows that these state changes have occurred. The actual state can be determined by checking the LLO and REM status bits in the upper nibble of the Interrupt Status 2 Register.

4. DREQ is cleared by DACK (RD + WR). DREQ on the 8291 was cleared only by DACK which is not compatible with the 8089 I/O Processor.

5. The INT bit in Interrupt Status 2 Register is duplicated in bit 7 of the Address 0 Register. If software polling is used to check for an interrupt, INT in the Address 0 Register should be polled rather than the Interrupt Status 2 Register. This ensures that no interrupts are lost due to asynchronous status reads and interrupts.

6. The 8291A’s Send **EOI** Auxiliary Command works on any byte including the first byte of a message. The 8291 did not assert **EOI** after this command for a one byte message nor on two consecutive bytes.

7. To avoid confusion between holdoff on DAV versus RFD if a device is readdressed from a talker to a listener role or vice-versa during a holdoff, the “Holdoff on Source Handshake” has been eliminated. Only “Holdoff on Acceptor Handshake” is available.

8. The rsv local message is cleared automatically upon exit from SPAS if (APRS:STRS:SPAS) occurred. The automatic resetting of the bit after the serial poll is complete simplifies the service request software.

9. The SPSC interrupt on the 8291 has been replaced by the SPC (Serial Poll Complete) interrupt on the 8291A. SPC interrupt is set on exit from SPAS if APRS:STRS:SPAS occurred, indicating that the controller has read the bus status byte after the 8291A requested service. The SPSC interrupt was ambiguous because a controller could enter SPAS and exit SPAS generating two SPSC interrupts without reading the serial poll status byte. The SPC interrupt also simplifies the CPU’s software by eliminating the interrupt when the serial poll is half way done.

10. The rtl Auxiliary Command in the 8291 has been replaced by Set and Clear rtl Commands in the 8291A. Using the new commands, the CPU has the flexibility to extend the length of local mode or leave it as a short pulse as in the 8291.

11. A holdoff RFD on GET, SDC, and DCL feature has been added to prevent additional bus activity while the CPU is responding to any of these commands. The feature is enabled by a new bit (B4) in the Auxiliary Register B.

12. On the 8291, BO could cease to occur upon IFC going false if IFC occurred asynchronously. On the 8291A, BO continues to occur after IFC has gone false even if it arrived asynchronously.

13. User’s software can distinguish between the 8291 and the 8291A as follows:

   a) pon (00H to register 5)
   b) RESET (02H to register 5)
   c) Read Interrupt Status 1 Register. If BO interrupt is set, the device is the 8291. If BO is clear, it is the 8291A.

   This can be used to set a flag in the user’s software which will permit special routines to be executed for each device. It could be included as part of a normal initialization procedure as the first step after a chip reset.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>D&lt;sub&gt;0&lt;/sub&gt;–D&lt;sub&gt;7&lt;/sub&gt;</td>
<td>12–19</td>
<td>I/O</td>
<td>Data Bus Port: To be connected to microprocessor data bus.</td>
</tr>
<tr>
<td>RS&lt;sub&gt;0&lt;/sub&gt;–RS&lt;sub&gt;2&lt;/sub&gt;</td>
<td>21–23</td>
<td>I</td>
<td>Register Select: Inputs, to be connected to three non-multiplexed microprocessor address bus lines. Select which of the 8 internal read (write) registers will be read from (written into) with the execution of RD (WR).</td>
</tr>
<tr>
<td>CS</td>
<td>8</td>
<td>I</td>
<td>Chip Select: When low, enables reading from or writing into the register selected by RS&lt;sub&gt;0&lt;/sub&gt;–RS&lt;sub&gt;2&lt;/sub&gt;.</td>
</tr>
<tr>
<td>RD</td>
<td>9</td>
<td>I</td>
<td>Read Strobe: When low with CS or DACK low, selected register contents are read.</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>I</td>
<td>Write Strobe: When low with CS or DACK low, data is written into the selected register.</td>
</tr>
<tr>
<td>INT (INT)</td>
<td>11</td>
<td>O</td>
<td>Interrupt Request: To the microprocessor, set high for request and cleared when the appropriate register is accessed by the CPU. May be software configured to be active low.</td>
</tr>
<tr>
<td>DREQ</td>
<td>6</td>
<td>O</td>
<td>DMA Request: Normally low, set high to indicate byte output or byte input in DMA mode; reset by DACK.</td>
</tr>
<tr>
<td>DACK</td>
<td>7</td>
<td>I</td>
<td>DMA Acknowledge: When low, resets DREQ and selects data in/data out register for DMA data transfer (actual transfer done by RD/WR pulse). Must be high if DMA is not used.</td>
</tr>
<tr>
<td>TRIG</td>
<td>5</td>
<td>O</td>
<td>Trigger Output: Normally low; generates a triggering pulse with 1 μsec min. width in response to the GET bus command or Trigger auxiliary command.</td>
</tr>
<tr>
<td>CLOCK</td>
<td>3</td>
<td>I</td>
<td>External Clock: Input, used only for T&lt;sub&gt;1&lt;/sub&gt; delay generator. May be any speed in 1–8 MHz range.</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>4</td>
<td>I</td>
<td>Reset Input: When high, forces the device into an &quot;idle&quot; (initialization) mode. The device will remain at &quot;idle&quot; until released by the microprocessor, with the &quot;Immediate Execute pon&quot; local message.</td>
</tr>
<tr>
<td>DIO&lt;sub&gt;0&lt;/sub&gt;–DIO&lt;sub&gt;4&lt;/sub&gt;</td>
<td>28–35</td>
<td>I/O</td>
<td>8-Bit GPIB Data Port: Used for bidirectional data byte transfer between 8291A and GPIB via non-inverting external line transceivers.</td>
</tr>
<tr>
<td>DAV</td>
<td>36</td>
<td>I/O</td>
<td>Data Valid: GPIB handshake control line. Indicates the availability and validity of information on the DIO&lt;sub&gt;0&lt;/sub&gt;–DIO&lt;sub&gt;4&lt;/sub&gt; and EOI lines.</td>
</tr>
<tr>
<td>NRFD</td>
<td>37</td>
<td>I/O</td>
<td>Not Ready for Data: GPIB handshake control line. Indicates the condition of readiness of device(s) connected to the bus to accept data.</td>
</tr>
<tr>
<td>NDAC</td>
<td>38</td>
<td>I/O</td>
<td>Not Data Accepted: GPIB handshake control line. Indicates the condition of acceptance of data by the device(s) connected to the bus.</td>
</tr>
<tr>
<td>ATN</td>
<td>26</td>
<td>I</td>
<td>Attention: GPIB command line. Specified how data on DIO lines are to be interpreted.</td>
</tr>
<tr>
<td>IFC</td>
<td>24</td>
<td>I</td>
<td>Interface Clear: GPIB command line. Places the interface functions in a known quiescent state.</td>
</tr>
<tr>
<td>SRQ</td>
<td>27</td>
<td>O</td>
<td>Service Request: GPIB command line. Indicates the need for attention and requests an interruption of the current sequence of events on the GPIB.</td>
</tr>
<tr>
<td>REN</td>
<td>25</td>
<td>I</td>
<td>Remote Enable: GPIB command line. Selects (in conjunction with other messages) remote or local control of the device.</td>
</tr>
<tr>
<td>EOI</td>
<td>39</td>
<td>I/O</td>
<td>End or Identify: GPIB command line. Indicates the end of a multiple byte transfer sequence or, in conjunction with ATN, addresses the device during a polling sequence.</td>
</tr>
</tbody>
</table>
Table 1. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>T/R1</td>
<td>1</td>
<td>O</td>
<td><strong>External Transceivers Control Line:</strong> Set high to indicate output data/signals on the $D_{IO}<em>1$–$D</em>{IO}<em>8$ and DAV lines and input signals on the NRFD and NDAC lines (active source handshake). Set low to indicate input data/signals on the $D</em>{IO}<em>1$–$D</em>{IO}_8$ and DAV lines and output signals on the NRFD and NDAC lines (active acceptor handshake).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>T/R2</td>
<td>2</td>
<td>O</td>
<td><strong>External Transceivers Control Line:</strong> Set to indicate output signals on the EOI line. Set low to indicate expected input signal on the EOI line during parallel poll.</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>40</td>
<td>P.S.</td>
<td>Positive Power Supply: (5V ±10%).</td>
</tr>
<tr>
<td>GND</td>
<td>20</td>
<td>P.S.</td>
<td>Circuit Ground Potential.</td>
</tr>
</tbody>
</table>

**NOTE:** All signals on the 8291A pins are specified with positive logic. However, IEEE 488 specifies negative logic on its 16 signal lines. Thus, the data is inverted once from $0_0$–$0_1$ to $0_0$–$0_1$, and non-inverting bus transceivers should be used.

---

**THE GENERAL PURPOSE INTERFACE BUS (GPIB)**

The General Purpose Interface Bus (GPIB) is defined in the IEEE Standard 488-1978 "Digital Interface for Programmable Instrumentation." Although a knowledge of this standard is assumed, Figure 4 provides the bus structure for quick reference. Also, Tables 2 and 3 reference the interface state mnemonics and the interface messages respectively. Modified state diagrams for the 8291A are presented in Appendix A.

**General Description**

The 8291A is a microprocessor-controlled device designed to interface microprocessors, e.g., 8048/49, 8051, 8080/85, 8086/88 to the GPIB. It implements all of the interface functions defined in the IEEE-488 Standard except for the controller function. If an implementation of the Standard’s Controller is desired, it can be connected with an Intel® 8292 to form a complete interface.

The 8291A handles communication between a microprocessor-controlled device and the GPIB. Its capabilities include data transfer, handshake protocol, talker/listener addressing procedures, device clearing and triggering, service request, and both serial and parallel polling. In most procedures, it does not disturb the microprocessor unless a byte has arrived (input buffer full) or has to be sent out (output buffer empty).

The 8291A architecture includes 16 registers. Eight of these registers may be written into by the microprocessor. The other eight registers may be read by the microprocessor. One each of these read and...
write registers is for direct data transfers. The rest of
the write registers control the various features of the
chip, while the rest of the read registers provide the
microprocessor with a monitor of GPIB states, vari-
ous bus conditions, and device conditions.

GPIB Addressing

Each device connected to the GPIB must have at
least one address whereby the controller device in
charge of the bus can configure it to talk, listen, or
send status. An 8291A implementation of the GPIB
offers the user three alternative addressing modes
for which the device can be initialized for each appli-
cation. The first of these modes allows for the device
to have two separate primary addresses. The second
mode allows the user to implement a single
talker/listener with a two byte address (primary ad-
dress + secondary address). The third mode again
allows for two distinct addresses but in this instance,
they can each have a ten-bit address (5 low-order
bits of each of two bytes). However, this mode re-
quires that the secondary addresses be passed to
the microprocessor for verification. These three
addressing schemes are described in more detail in
the discussion of the Address Registers.

Table 2. IEEE 488 Interface State Mnemonics

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>State Represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDS</td>
<td>Accept Data State</td>
</tr>
<tr>
<td>ACRS</td>
<td>Acceptor Ready State</td>
</tr>
<tr>
<td>AIDS</td>
<td>Acceptor Idle State</td>
</tr>
<tr>
<td>ANRS</td>
<td>Acceptor Not Ready State</td>
</tr>
<tr>
<td>APRS</td>
<td>Affirmative Poll Response State</td>
</tr>
<tr>
<td>AWNS</td>
<td>Acceptor Wait for New Cycle State</td>
</tr>
<tr>
<td>CACS</td>
<td>Controller Active State</td>
</tr>
<tr>
<td>CADS</td>
<td>Controller Addressed State</td>
</tr>
<tr>
<td>CAWS</td>
<td>Controller Active Wait State</td>
</tr>
<tr>
<td>CIDS</td>
<td>Controller Idle State</td>
</tr>
<tr>
<td>CPPS</td>
<td>Controller Parallel Poll State</td>
</tr>
<tr>
<td>CPWS</td>
<td>Controller Parallel Poll Wait State</td>
</tr>
<tr>
<td>CSBS</td>
<td>Controller Standby State</td>
</tr>
<tr>
<td>CSNS</td>
<td>Controller Service Not Requested State</td>
</tr>
<tr>
<td>CSRS</td>
<td>Controller Service Requested State</td>
</tr>
<tr>
<td>CSWS</td>
<td>Controller Synchronous Wait State</td>
</tr>
<tr>
<td>CTRS</td>
<td>Controller Transfer State</td>
</tr>
<tr>
<td>DCAS</td>
<td>Device Clear Active State</td>
</tr>
<tr>
<td>DCIS</td>
<td>Device Clear Idle State</td>
</tr>
<tr>
<td>DTAS</td>
<td>Device Trigger Active State</td>
</tr>
<tr>
<td>DTIS</td>
<td>Device Trigger Idle State</td>
</tr>
<tr>
<td>LACS</td>
<td>Listener Active State</td>
</tr>
<tr>
<td>LADS</td>
<td>Listener Addressed State</td>
</tr>
<tr>
<td>LIDS</td>
<td>Listener Idle State</td>
</tr>
<tr>
<td>LCS</td>
<td>Local State</td>
</tr>
<tr>
<td>LPAS</td>
<td>Listener Primary Addressed State</td>
</tr>
<tr>
<td>LPIS</td>
<td>Listener Primary Idle State</td>
</tr>
<tr>
<td>LWLS</td>
<td>Local With Lockout State</td>
</tr>
<tr>
<td>NPRS</td>
<td>Negative Poll Response State</td>
</tr>
<tr>
<td>PACS</td>
<td>Parallel Poll Addressed to Configure State</td>
</tr>
<tr>
<td>PPAS</td>
<td>Parallel Poll Active State</td>
</tr>
<tr>
<td>PPIS</td>
<td>Parallel Poll Idle State</td>
</tr>
<tr>
<td>PPSS</td>
<td>Parallel Poll Standby State</td>
</tr>
<tr>
<td>PUCS</td>
<td>Parallel Poll Unaddressed to Configure State</td>
</tr>
<tr>
<td>REMS</td>
<td>Remote State</td>
</tr>
<tr>
<td>RWLS</td>
<td>Remote With Lockout State</td>
</tr>
<tr>
<td>SACS</td>
<td>System Control Active State</td>
</tr>
<tr>
<td>SDYS</td>
<td>Source Delay State</td>
</tr>
<tr>
<td>SGNS</td>
<td>Source Generate State</td>
</tr>
<tr>
<td>SIAS</td>
<td>System Control Interface Clear Active State</td>
</tr>
<tr>
<td>SIDS</td>
<td>Source Idle State</td>
</tr>
<tr>
<td>SIIS</td>
<td>System Control Interface Clear Idle State</td>
</tr>
<tr>
<td>SINS</td>
<td>System Control Interface Clear Not Active State</td>
</tr>
<tr>
<td>SIWS</td>
<td>Source Idle Wait State</td>
</tr>
<tr>
<td>SNAS</td>
<td>System Control Not Active State</td>
</tr>
<tr>
<td>SPAS</td>
<td>Serial Poll Active State</td>
</tr>
<tr>
<td>SPIS</td>
<td>Serial Poll Idle State</td>
</tr>
<tr>
<td>SPMS</td>
<td>Serial Poll Mode State</td>
</tr>
<tr>
<td>SRAS</td>
<td>System Control Remote Enable Active State</td>
</tr>
<tr>
<td>SRIS</td>
<td>System Control Remote Enable Idle State</td>
</tr>
<tr>
<td>SRNS</td>
<td>System Control Remote Enable Not Active State</td>
</tr>
<tr>
<td>SRSQ</td>
<td>Service Request State</td>
</tr>
<tr>
<td>STRS</td>
<td>Source Transfer State</td>
</tr>
<tr>
<td>SWNS</td>
<td>Source Wait for New Cycle State</td>
</tr>
<tr>
<td>TACS</td>
<td>Talker Active State</td>
</tr>
<tr>
<td>TADS</td>
<td>Talker Addressed State</td>
</tr>
<tr>
<td>TIDS</td>
<td>Talker Idle State</td>
</tr>
<tr>
<td>TPIIS</td>
<td>Talker Primary Idle State</td>
</tr>
</tbody>
</table>
Table 3. IEEE 488 Interface Message Reference List

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Message</th>
<th>Interface Function(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCAL MESSAGES RECEIVED (By Interface Functions)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'gts</td>
<td>go to standby</td>
<td>C</td>
</tr>
<tr>
<td>'ist</td>
<td>individual status</td>
<td>PP</td>
</tr>
<tr>
<td>'lon</td>
<td>listen only</td>
<td>L, LE</td>
</tr>
<tr>
<td>'lpe</td>
<td>local poll enable</td>
<td>PP</td>
</tr>
<tr>
<td>'nba</td>
<td>new byte available</td>
<td>SH</td>
</tr>
<tr>
<td>'pon</td>
<td>power on</td>
<td>SH, AH, T, TE, L, LE, SR, RL, PP, C</td>
</tr>
<tr>
<td>'rdy</td>
<td>ready</td>
<td>AH</td>
</tr>
<tr>
<td>'rpp</td>
<td>request parallel poll</td>
<td>C</td>
</tr>
<tr>
<td>r'src</td>
<td>request system control</td>
<td>C</td>
</tr>
<tr>
<td>'rsv</td>
<td>request service</td>
<td>SR</td>
</tr>
<tr>
<td>'rtl</td>
<td>return to local</td>
<td>RL</td>
</tr>
<tr>
<td>'sic</td>
<td>send interface clear</td>
<td>C</td>
</tr>
<tr>
<td>'sre</td>
<td>send remote enable</td>
<td>C</td>
</tr>
<tr>
<td>'tca</td>
<td>take control asynchronously</td>
<td>C</td>
</tr>
<tr>
<td>'tcs</td>
<td>take control synchronously</td>
<td>AH, C</td>
</tr>
<tr>
<td>ton</td>
<td>talk only</td>
<td>T, TE</td>
</tr>
<tr>
<td>REMOTE MESSAGES RECEIVED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATN</td>
<td>Attention</td>
<td>SH, AH, T, TE, L, LE, PP, C</td>
</tr>
<tr>
<td>DAB</td>
<td>Data Byte</td>
<td>(Via L, LE)</td>
</tr>
<tr>
<td>DAC</td>
<td>Data Accepted</td>
<td>SH</td>
</tr>
<tr>
<td>DAV</td>
<td>Data Valid</td>
<td>AH</td>
</tr>
<tr>
<td>DCL</td>
<td>Device Clear</td>
<td>DC</td>
</tr>
<tr>
<td>END</td>
<td>End</td>
<td>(via L, LE)</td>
</tr>
<tr>
<td>GET</td>
<td>Group Execute Trigger</td>
<td>DT</td>
</tr>
<tr>
<td>GTL</td>
<td>Go to Local</td>
<td>RL</td>
</tr>
<tr>
<td>IDY</td>
<td>Identify</td>
<td>L, LE, PP</td>
</tr>
<tr>
<td>'IFC</td>
<td>Interface Clear</td>
<td>T, TE, L, LE, C</td>
</tr>
<tr>
<td>LLO</td>
<td>Local Lockout</td>
<td>RL</td>
</tr>
<tr>
<td>MLA</td>
<td>My Listen Address</td>
<td>L, LE, RL, T, TE</td>
</tr>
<tr>
<td>MSA</td>
<td>My Secondary Address</td>
<td>TE, LE, RL</td>
</tr>
<tr>
<td>MTA</td>
<td>My Talk Address</td>
<td>T, TE, L, LE</td>
</tr>
<tr>
<td>OSA</td>
<td>Other Secondary Address</td>
<td>TE</td>
</tr>
<tr>
<td>OTA</td>
<td>Other Talk Address</td>
<td>T, TE</td>
</tr>
<tr>
<td>PCG</td>
<td>Primary Command Group</td>
<td>TE, LE, PP</td>
</tr>
<tr>
<td>PPC</td>
<td>Parallel Poll Configure</td>
<td>PP</td>
</tr>
<tr>
<td>[PPD]</td>
<td>Parallel Poll Disable</td>
<td>PP</td>
</tr>
<tr>
<td>[PPE]</td>
<td>Parallel Poll Enable</td>
<td>PP</td>
</tr>
<tr>
<td>'PPRN</td>
<td>Parallel Poll Response N</td>
<td>(via C)</td>
</tr>
<tr>
<td>'PPU</td>
<td>Parallel Poll Unconfigure</td>
<td>PP</td>
</tr>
<tr>
<td>REN</td>
<td>Remote Enable</td>
<td>RL</td>
</tr>
<tr>
<td>RDF</td>
<td>Ready for Data</td>
<td>SH</td>
</tr>
<tr>
<td>RQS</td>
<td>Request Service</td>
<td>(via L, LE)</td>
</tr>
<tr>
<td>[SDC]</td>
<td>Select Device Clear</td>
<td>DC</td>
</tr>
<tr>
<td>SPD</td>
<td>Serial Poll Disable</td>
<td>T, TE</td>
</tr>
<tr>
<td>SPE</td>
<td>Serial Poll Enable</td>
<td>T, TE</td>
</tr>
<tr>
<td>'SOR</td>
<td>Service Request</td>
<td>(via C)</td>
</tr>
<tr>
<td>STB</td>
<td>Status Byte</td>
<td>(via L, LE)</td>
</tr>
<tr>
<td>'TCT or [TCT]</td>
<td>Take Control</td>
<td>C</td>
</tr>
<tr>
<td>UNL</td>
<td>Unlisten</td>
<td>L, LE</td>
</tr>
</tbody>
</table>

NOTE:
1. These messages are handled only by Intel's 8292.
2. Undefined commands which may be passed to the microprocessor
### Table 3. (Cont’d)
IEEE 488 Interface Message Reference List

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Message</th>
<th>Interface Function(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE MESSAGES SENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATN</td>
<td>Attention</td>
<td>C</td>
</tr>
<tr>
<td>DAB</td>
<td>Data Byte</td>
<td>(via T, TE)</td>
</tr>
<tr>
<td>DAC</td>
<td>Data Accepted</td>
<td>AH</td>
</tr>
<tr>
<td>DAV</td>
<td>Data Valid</td>
<td>SH</td>
</tr>
<tr>
<td>DCL</td>
<td>Device Clear</td>
<td>(via C)</td>
</tr>
<tr>
<td>END</td>
<td>End</td>
<td>(via T)</td>
</tr>
<tr>
<td>GET</td>
<td>Group Execute Trigger</td>
<td>(via C)</td>
</tr>
<tr>
<td>GTL</td>
<td>Go to Local</td>
<td>(via C)</td>
</tr>
<tr>
<td>IDY</td>
<td>Identify</td>
<td>C</td>
</tr>
<tr>
<td>IFC</td>
<td>Interface Clear</td>
<td>C</td>
</tr>
<tr>
<td>LLO</td>
<td>Local Lockout</td>
<td>(via C)</td>
</tr>
<tr>
<td>MLA or</td>
<td>MLA</td>
<td>My Listen Address</td>
</tr>
<tr>
<td>MSA or</td>
<td>MSA</td>
<td>My Secondary Address</td>
</tr>
<tr>
<td>MTA or</td>
<td>MTA</td>
<td>My Talk Address</td>
</tr>
<tr>
<td>OSA</td>
<td>Other Secondary Address</td>
<td>(via C)</td>
</tr>
<tr>
<td>OTA</td>
<td>Other Talk Address</td>
<td>(via C)</td>
</tr>
<tr>
<td>PCG</td>
<td>Primary Command Group</td>
<td>(via C)</td>
</tr>
<tr>
<td>PPC</td>
<td>Parallel Poll Configure</td>
<td>(via C)</td>
</tr>
<tr>
<td>PPD</td>
<td>Parallel Poll Disable</td>
<td>(via C)</td>
</tr>
<tr>
<td>PPE</td>
<td>Parallel Poll Enable</td>
<td>(via C)</td>
</tr>
<tr>
<td>PPRn</td>
<td>Parallel Poll Response N</td>
<td>PP</td>
</tr>
<tr>
<td>PPU</td>
<td>Parallel Poll Unconfigure</td>
<td>(via C)</td>
</tr>
<tr>
<td>REN</td>
<td>Remote Enable</td>
<td>C</td>
</tr>
<tr>
<td>RDF</td>
<td>Ready for Data</td>
<td>AH</td>
</tr>
<tr>
<td>RQS</td>
<td>Request Service</td>
<td>T, TE</td>
</tr>
<tr>
<td>[SDC]</td>
<td>Selected Device Clear</td>
<td>(via C)</td>
</tr>
<tr>
<td>SPD</td>
<td>Serial Poll Disable</td>
<td>(via C)</td>
</tr>
<tr>
<td>SPE</td>
<td>Serial Poll Enable</td>
<td>(via C)</td>
</tr>
<tr>
<td>SRQ</td>
<td>Service Request</td>
<td>SR</td>
</tr>
<tr>
<td>STB</td>
<td>Status Byte</td>
<td>(via T, TE)</td>
</tr>
<tr>
<td>TCT</td>
<td>Take Control</td>
<td>(via C)</td>
</tr>
<tr>
<td>UNL</td>
<td>Unlisten</td>
<td>(via C)</td>
</tr>
</tbody>
</table>

**NOTE:**
3. All Controller messages must be sent via Intel's 8292.

### 8291A Registers

A bit-by-bit map of the 16 registers on the 8291A is presented in Figure 5. A more detailed explanation of each of these registers and their functions follows. The access of these registers by the microprocessor is accomplished by using the CS, RD, WR, and RS₀-RS₂ pins.

<table>
<thead>
<tr>
<th>Register</th>
<th>CS</th>
<th>RD</th>
<th>WR</th>
<th>RS₀-RS₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Read Registers</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>CCC</td>
</tr>
<tr>
<td>All Write Registers</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>CCC</td>
</tr>
<tr>
<td>High Impedance</td>
<td>1</td>
<td>d</td>
<td>d</td>
<td>ddd</td>
</tr>
</tbody>
</table>

### Data Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>CS</th>
<th>RD</th>
<th>WR</th>
<th>RS₀-RS₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA-IN REGISTER (0R)</td>
<td>D17</td>
<td>D16</td>
<td>D15</td>
<td>D14</td>
</tr>
<tr>
<td>DATA-OUT REGISTER (0W)</td>
<td>D07</td>
<td>D06</td>
<td>D05</td>
<td>D04</td>
</tr>
</tbody>
</table>

The Data-In Register is used to move data from the GPIB to the microprocessor or to memory when the 8291A is addressed to listen. Incoming information is separately latched by this register, and its contents are not destroyed by a write to the data-out
The RFD (Ready for Data) message is held false until the byte is removed from the data in register, either by the microprocessor or by DMA. The 8291A then completes the handshake automatically. In RFD holdoff mode (see Auxiliary Register A), the handshake is not finished until a command is sent telling the 8291A to release the holdoff. In this way, the same byte may be read several times, or an over anxious talker may be held off until all available data has been processed.

When the 8291A is addressed to talk, it uses the data-out register to move data onto the GPIB. After the BO interrupt is received and a byte is written to this register, the 8291A initiates and completes the handshake while sending the byte out over the bus. In the BO interrupt disable mode, the user should wait until BO is active before writing to the register. (In the DMA mode, this will happen automatically.) A read of the Data-In Register does not destroy the information in the Data-Out Register.

### Figure 5. 8291A Registers

<table>
<thead>
<tr>
<th>READ REGISTERS</th>
<th>REGISTER SELECT CODE</th>
<th>WRITE REGISTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RS2 RS1 RS0</td>
<td>Data In</td>
</tr>
<tr>
<td>D[7:0]</td>
<td>0 0 0</td>
<td>D[7:0]</td>
</tr>
<tr>
<td>CPT APT GET</td>
<td>0 0 0</td>
<td>CPT APT GET</td>
</tr>
<tr>
<td>END DEC ERR</td>
<td>0 0 1</td>
<td>END DEC ERR</td>
</tr>
<tr>
<td>BO BI</td>
<td>0 0 0</td>
<td>BO BI</td>
</tr>
<tr>
<td>Interrupt Status 1</td>
<td>INT SPAS LLO REM SP C LLOC REMC ADSC</td>
<td></td>
</tr>
<tr>
<td>S[7:0]</td>
<td>1 1 1</td>
<td>S[7:0]</td>
</tr>
<tr>
<td>SERIAL POLL STATUS</td>
<td>ton lon EOI LPAS TPAS LA TA MJMN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 0 0</td>
<td>AD[7:0]</td>
</tr>
<tr>
<td>ADDRESS STATUS</td>
<td>1 1 1</td>
<td>ADDRESS MODE</td>
</tr>
<tr>
<td>CPT7 CPT6 CPT5</td>
<td>1 0 1</td>
<td>AUX MODE</td>
</tr>
<tr>
<td>CPT4 CPT3 CPT2</td>
<td>1 0 1</td>
<td>ADDRESS 0</td>
</tr>
<tr>
<td>CPT1 CPT0</td>
<td>1 0 1</td>
<td>ADDRESS 1</td>
</tr>
<tr>
<td>ADDRESS 0</td>
<td>1 1 1</td>
<td>EOS</td>
</tr>
</tbody>
</table>

### Interrupt Registers

<table>
<thead>
<tr>
<th>Interrupt Status 1 (1R)</th>
<th>Interrupt Status 2 (2R)</th>
<th>Interrupt Enable 1 (1W)</th>
<th>Interrupt Enable 2 (2W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT SPAS LLO REM SP C LLOC REMC ADSC</td>
<td>INT SPAS LLO REM SP C LLOC REMC ADSC</td>
<td>0 0 DMAO DMA1 SP C LLOC REMC ADSC</td>
<td>INT DT0 DL0 AD5-0 AD4-0 AD3-0 AD2-0 AD1-0</td>
</tr>
</tbody>
</table>

ADDRESS 0 REGISTER
The 8291A can be configured to generate an interrupt to the microprocessor upon the occurrence of any of 12 conditions or events on the GPIB. Upon receipt of an interrupt, the microprocessor must read the Interrupt Status Registers to determine which event has occurred, and then execute the appropriate service routine (if necessary). Each of the 12 interrupt status bits has a matching enable bit in the interrupt enable registers. These enable bits are used to select the events that will cause the INT pin to be asserted. Writing a logic "1" into any of these bits enables the corresponding interrupt status bits to generate an interrupt. Bits in the Interrupt Status Registers are set regardless of the states of the enable bits. The Interrupt Status Registers are then cleared upon being read or when a local pon (power-on) message is executed. If an event occurs while one of the Interrupt Status Registers is being read, the event is held until after its register is cleared and then placed in the register.

The mnemonics for each of the bits in these registers and a brief description of their respective functions appears in Table 4. This table also indicates how each of the interrupt bits is set.

NOTE: The INT bit in the Address 0 Register is a duplicate of the INT bit in the Interrupt Status 2 Register. It is only a status bit. It does not generate interrupts and thus does not have a corresponding enable bit.

The BO and BI interrupts enable the user to perform data transfer cycles. BO indicates that a data byte should be written to the Data Out Register. It is set by TACS · (SWNS + SGNS) · RFD. It is reset when the data byte is written, ATN is asserted, or the 8291A exits TACS. Data should never be written to the Data Out Register before BO is set. Similarly, BI is set when an input byte is accepted into the 8291A and reset when the microprocessor reads the Data In Register. BO and BI are also reset by pon (power-on local message) and by a read of the Interrupt

---

### Table 4. Interrupt Bits

<table>
<thead>
<tr>
<th>Indicates Undefin&lt;i&gt;d Commands</th>
<th>CFT</th>
<th>An undefined command has been received.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set by (TPAS + LPAS) · SCG · ACDS · MODE 3</td>
<td>APT</td>
<td>A secondary address must be passed through to the microprocessor for recognition.</td>
</tr>
<tr>
<td>Set by DTAS</td>
<td>GET</td>
<td>A group execute trigger has occurred.</td>
</tr>
<tr>
<td>Set by (EOS + EOI) · LACS</td>
<td>END</td>
<td>An EOS or EOI message has been received.</td>
</tr>
<tr>
<td>Set by DCAS</td>
<td>DEC</td>
<td>Device Clear Active State has occurred.</td>
</tr>
<tr>
<td>Set by TACS · (SWNS + SGNS)</td>
<td>ERR</td>
<td>Interface error has occurred, no listeners are active.</td>
</tr>
<tr>
<td>Set by LACS · ACDS</td>
<td>BO</td>
<td>A byte should be output.</td>
</tr>
<tr>
<td>-shows status of the INT pin</td>
<td>BI</td>
<td>A byte has been input.</td>
</tr>
</tbody>
</table>

These are status only. They will not generate interrupts, nor do they have corresponding mask bits.

NOTE: In ton (talk-only) and ion (listen-only) modes, no ADSC interrupt is generated.

---

6-394

205248-002
Status 1 Register. However, if it is so desired, data transfer cycles may be performed without reading the Interrupt Status 1 Register if all interrupts except for BO or BI are disabled; BO and BI will automatically reset after each byte is transferred.

If the 8291A is used in the interrupt mode, the INT and DREQ pins can be dedicated to data input and output interrupts respectively by enabling BI and DMAO, provided that no other interrupts are enabled. This eliminates the need to read the interrupt status registers if a byte is received or transmitted.

The ERR bit is set to indicate the bus error condition when the 8291A is an active talker and tries to send a byte to the GPIB, but there are no active listeners (e.g., all devices on the GPIB are in AIDS). The logical equivalent of (nba · TACS · DAC · RFD) will set this bit.

The DEC bit is set whenever DCAS has occurred. The user must define a known state to which all device functions will return in DCAS. Typically this state will be a power-on state. However, the state of the device functions at DCAS is at the designer’s discretion. It should be noted that DCAS has no effect on the interface functions which are returned to a known state by the IFC (interface clear) message or the pon local message.

The END interrupt bit may be used by the microprocessor to detect that a multi-byte transfer has been completed. The bit will be set when the 8291A is an active listener (LACS) and either EOS (provided the End on EOS Received feature is enabled in the Auxiliary Register A) or E0I is received. EOS will generate an interrupt when the byte in the Data In Register matches the byte in the EOS register. Otherwise the interrupt will be generated when a true input is detected on E0I.

The GET interrupt bit is used by the microprocessor to detect that DTAS has occurred. It is set by the 8291A when the GET message is received while it is addressed to listen. The TRIG output pin of the 8291A fires when the GET message is received. Thus, the basic operation of device trigger may be started without microprocessor software intervention.

The APT interrupt bit indicates to the processor that a secondary address is available in the CPT register for validation. This interrupt will only occur if Mode 3 addressing is in effect. (Refer to the section on addressing.) In Mode 2, secondary addresses will be recognized automatically on the 8291A. They will be ignored in Mode 1.

The CPT interrupt bit flags the occurrence of an undefined command and of all secondary commands following an undefined command. The Command Pass Through feature is enabled by the B0 bit of Auxiliary Register B. Any message not decoded by the 8291A (not included in the state diagrams in Appendix B) becomes an undefined command. Note that any addressed command is automatically ignored when the 8291A is not addressed.

Undefined commands are read by the CPU from the Command Pass Through register of the 8291A. This register reflects the logic levels present on the data lines at the time it is read. If the CPT feature is enabled, the 8291A will hold off the handshake until this register is read.

An especially useful feature of the 8291A is its ability to generate interrupts from state transitions in the interface functions. In particular, the lower 3 bits of the Interrupt Status 2 Register, if enabled by the corresponding enable bits, will cause an interrupt upon changes in the following states as defined in the IEEE 488 Standard:

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>ADSC</th>
<th>change in LIDS or TIDS or MJMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 1</td>
<td>REMC</td>
<td>change in LOCS or REMS</td>
</tr>
<tr>
<td>Bit 2</td>
<td>LLOC</td>
<td>change in LWLS or RWLS</td>
</tr>
</tbody>
</table>

The upper 4 bits of the Interrupt Status 2 Register are available to the processor as status bits. Thus, if one of the bits 0–2 generates an interrupt indicating a state change has taken place, the corresponding status bit (bits 3–5) may be read to determine what the new state is. To determine the nature of a change in addressed status (bit 0) the Address Status Register is available to be read. The SPC interrupt (bit 3 in Interrupt Status 2) is set upon exit from SPAS if APRS:STRS:SPAS occurred which indicates that the GPIB controller has read the bus serial poll status byte after the 8291A requested service (asserted SRQ). The SPC interrupt occurs once after the controller reads the status byte if service was requested.

6-395

205248-002
The controller may read the status byte later, and the byte will contain the last status the 8291A’s CPU wrote to the Serial Poll Mode Register, but the SRQS bit will not be set and no interrupt will be generated. Finally, bit 7 monitors the state of the 8291A INT pin. Logically, it is an OR of all enabled interrupt status bits. One should note that bits 3–6 of the Interrupt Status 2 Register do not generate interrupts, but are available only to be read as status bits by the processor. Bit 7 in Interrupt Status 2 is duplicated in Address 0 Register, and the latter should be used when polling for interrupts to avoid losing one of the interrupts in Interrupt Status 2 Register.

Bits 4 and 5 (DMAI, DMAO) of the Interrupt Mask 2 Register are available to enable direct data transfers between memory and the GPIB; DMAI (DMA in) enables the DREQ (DMA request) pin of the 8291A to be asserted upon the occurrence of BI. Similarly, DMAO (DMA out) enables the DREQ pin to be asserted upon the occurrence of BO. One might note that the DREQ pin may be used as a second interrupt output pin, monitoring BI and/or BO and enabled by DMAI and DMAO. One should note that the DREQ pin is not affected by a read of the Interrupt Status 1 Register. It is reset whenever a byte is written to the Data Out Register or read from the Data In Register.

To ensure that an interrupt status bit will not be cleared without being read, and will not remain uncleared after being read, the 8291A implements a special interrupt handling procedure. When an enabled interrupt bit is set in either of the Interrupt Status Registers, the input of the registers is blocked until the set bit is read and reset by the microprocessor. Thus, potential problems arise when interrupt status changes while the register is being blocked. However, the 8291A stores all new interrupts in a temporary register and transfers them to the appropriate Interrupt Status Register after the interrupt has been reset. This transfer takes place only if the corresponding bits were read as zeroes.

The Serial Poll Mode Register determines the status byte that the 8291A sends out on the GPIB data lines when it receives the SPE (Serial Poll Enable) message. Bit 6 of this register is reserved for the rsv (request service) local message. Setting this bit to 1 causes the 8291A to assert its SRQ line, indicating its need for attention from the controller-in-charge of the GPIB. The other bits of this register are available for sending status information over the GPIB. Sometime after the microprocessor initiates a request for service by setting bit 6, the controller of the GPIB sends the SPE message and then addresses the 8291A to talk. At this point, one byte of status is returned by the 8291A via the Serial Poll Mode Register. The other bits of the register are read by the controller, rsv is automatically cleared by the 8291A and an SPC interrupt is generated. The CPU may request service again by writing another byte to the Serial Poll Mode Register with the rsv bit set. If the controller performs a serial poll when the rsv bit is clear, the last status byte written will be read, but the SRQ line will not be driven by the 8291A and the SRQS bit will be clear in the status byte.

The Serial Poll Status Register is available for reading the status byte in the Serial Poll Mode Register. The processor may check the status of a request for service by polling bit 6 of this register, which corresponds to SRQS (Service Request State). When a Serial Poll is conducted and the controller-in-charge reads the status byte, the SRQS bit is cleared. The SRQ line and the rsv bit are tied together.

### Address Registers

<table>
<thead>
<tr>
<th>Address Status (4R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address 0 (6R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address 1 (7R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address Mode (4W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address 0/1 (6W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-396</td>
</tr>
</tbody>
</table>
The Address Mode Register is used to select one of the five modes of addressing available on the 8291A. It determines the way in which the 8291A uses the information in the Address 0 and Address 1 Registers.

In Mode 1, the contents of the Address 0 Register constitute the “Major” talker/listener address while the Address 1 Register represents the “Minor” talker/listener address. In applications where only one address is needed, the major talker/listener is used, and the minor talker/listener should be disabled. Loading an address via the Address 0/1 Register into Address Registers 0 and 1 enables the major and minor talker/listener functions respectively.

In Mode 2, the 8291A recognizes two sequential address bytes: a primary followed by a secondary. Both address bytes must be received in order to enable the device to talk or listen. In this manner, Mode 2 addressing implements the extended talker and listener functions as defined in IEEE-488.

To use Mode 2 addressing the primary address must be loaded into the Address 0 Register, and the Secondary Address is placed in the Address 1 Register. With both primary and secondary addresses residing on chip, the 8291A can handle all addressing sequences without processor intervention.

In Mode 3, the 8291A handles addressing just as it does in Mode 1, except that each Major or Minor primary address must be followed by a secondary address. All secondary addresses must be verified by the microprocessor when Mode 3 is used. When the 8291A is in TPAS or LPAS (talker/listener primary addressed state), and it does not recognize the byte on the DIO lines, an APT interrupt is generated (see section on Interrupt Registers) and the byte is available in the CPT (Command Pass-Through) Register. As part of its interrupt service routine, the microprocessor must read the CPT Register and write one of the following responses to the Auxiliary Mode Register:

1. 07H implies a non-valid secondary address
2. 0FH implies a valid secondary address

Setting the TO bit generates the local ion (listen-only) message and sets the 8291A to a listen-only mode. This mode allows the device to operate as a listener in an interface system without a controller. The above bits may also be used by a controller-in-charge to set itself up for remote command or data communication.

The mode of addressing implemented by the 8291A may be selected by writing one of the following bytes to the Address Mode Register.

<table>
<thead>
<tr>
<th>Register Contents</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000</td>
<td>Enable talk only mode (ton)</td>
</tr>
<tr>
<td>01000000</td>
<td>Enable listen only mode (ion)</td>
</tr>
<tr>
<td>11000000</td>
<td>The 8291 may talk to itself</td>
</tr>
<tr>
<td>00000001</td>
<td>Mode 1, (Primary-Primary)</td>
</tr>
<tr>
<td>00000010</td>
<td>Mode 2 (Primary-Secondary)</td>
</tr>
<tr>
<td>00000011</td>
<td>Mode 3 (Primary/APT-Primary/APT)</td>
</tr>
</tbody>
</table>

The Address Status Register contains information used by the microprocessor to handle its own addressing. This information includes status bits that monitor the address state of each talker/listener, “ton” and “ion” flags which indicate the talk and listen only states, and an EOI bit which, when set, signifies that the END message came with the last data byte. LPAS and TPAS indicate that the listener or talker primary address has been received. The microprocessor can use these bits when the secondary address is passed through to determine whether the 8291A is addressed to talk or listen. The LA (listener addressed) bit will be set when the 8291A is in LACS (Listener Active State) or in LADS (Listener Addressed State). Similarly, the TA (Talker Addressed bit) will be set to indicate TACS or TADS, but also to indicate SPAS (Serial Poll Active State). The MJMN bit is used to determine whether the information in the other bits applies to the Major or Minor talker/listener. It is set to “1” when the Minor talker/listener is addressed. It should be noted that only one talker/listener may be active at any one time. Thus, the MJMN bit will indicate which, if either, of the talker/listeners is addressed or active.

The Address 0/1 Register is used for specifying the device’s addresses according to the format selected in the Address Mode Register. Five bit addresses may be loaded into the Address 0 and Address 1 Registers by writing into the Address 0/1 Register. The ARS bit is used to select which of these registers the other seven bits will be loaded into. The DT and DL bits may be used to disable the talker or listener function at the address signified by the other five
bits. When Mode 1 addressing is used and only one primary address is desired, both the talker and the listener should be disabled at the Minor address.

As an example of how the Address 0/1 Register might be used, consider an example where two primary addresses are needed in the device. The Major primary address will be selectable only as a talker and the Minor primary address will be selectable only as a listener. This configuration of the 8291A is formed by the following sequence of writes by the microprocessor.

<table>
<thead>
<tr>
<th>Operation</th>
<th>CS</th>
<th>RD</th>
<th>WR</th>
<th>Data</th>
<th>RS2-RS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Select addressing Mode 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>00000001</td>
<td>100</td>
</tr>
<tr>
<td>2 Load major address into Address 0 Register with listener function disabled.</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>001AAAAA</td>
<td>110</td>
</tr>
<tr>
<td>3 Load minor address into Address 1 Register with talker function disabled</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>110BBBBB</td>
<td>110</td>
</tr>
</tbody>
</table>

At this point, the addresses AAAAA and BBBBB are stored in the Address 0 and Address 1 Registers respectively, and are available to be read by the microprocessor. Thus, it is not necessary to store any address information elsewhere. Also, with the information stored in the Address 0 and Address 1 Registers, processor intervention is not required to recognize addressing by the controller. Only in Mode 3, where secondary addresses are passed through, must the processor intervene in the addressing sequence.

The Address 0 Register contains a copy of bit 7 of the Interrupt Status 2 Register (INT). This is to be used when polling for interrupts. Software should poll register 6 checking for INT (bit 7) to be set. When INT is set, the Interrupt Status Register should be read to determine which interrupt was received.

**Command Pass Through Register**

<table>
<thead>
<tr>
<th>CPT7</th>
<th>CPT6</th>
<th>CPT5</th>
<th>CPT4</th>
<th>CPT3</th>
<th>CPT2</th>
<th>CPT1</th>
<th>CPT0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COMMAND PASS THROUGH (5R)

The Command Pass Through Register is used to transfer undefined 8-bit remote message codes from the GPIB to the microprocessor. When the CPT feature is enabled (bit B0 in Auxiliary Register B), any message not decoded by the 8291A becomes an undefined command. When Mode 3 addressing is used secondary addresses are also passed through the CPT Register. In either case, the 8291A will hold-off the handshake until the microprocessor reads this register and issues the VSCMD auxiliary command.

The CPT and APT interrupts flag the availability of undefined commands and secondary addresses in the CPT Register. The details of these interrupts are explained in the section on Interrupt Registers.

An added feature of the 8291A is its ability to handle undefined secondary commands following undefined primaries. Thus, the number of available commands for future IEEE-488 definition is increased; one undefined primary command followed by a sequence of as many as 32 secondary commands can be processed. The IEEE-488 Standard does not permit users to define their own commands, but upgrades of the standard are thus provided for.

The recommended use of the 8291A's undefined command capabilities is for a controller-configured Parallel Poll. The PPC message is an undefined primary command typically followed by PPE, an undefined secondary command. For details on this procedure, refer to the section on Parallel Poll Protocol.

**Auxiliary Mode Register**

<table>
<thead>
<tr>
<th>CNT2</th>
<th>CNT1</th>
<th>CNT0</th>
<th>COM4</th>
<th>COM3</th>
<th>COM2</th>
<th>COM1</th>
<th>COM0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AUX MODE (5W)

CNT0-2:CONTROL BITS
COM0-4:COMMAND BITS

The Auxiliary Mode Register contains a three-bit control field and a five-bit command field. It is used for several purposes on the 8291A:

1. To load "hidden" auxiliary registers on the 8291A.
2. To issue commands from the microprocessor to the 8291A.
3. To preset an internal counter used to generate T1, delay in the Source Handshake function, as defined in IEEE-488.

Table 5 summarizes how these tasks are performed with the Auxiliary Mode Register. Note that the three control bits determine how the five command bits are interpreted.
<table>
<thead>
<tr>
<th>CODE</th>
<th>CONTROL BITS</th>
<th>COMMAND BITS</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0CCC</td>
<td>CCC</td>
<td>Execute auxiliary command CCC</td>
</tr>
<tr>
<td>001</td>
<td>ODDDD</td>
<td></td>
<td>Preset internal counter to match external clock frequency of DDDD MHz (DDDD binary representation of 1 to 8 MHz)</td>
</tr>
<tr>
<td>100</td>
<td>DDDDD</td>
<td></td>
<td>Write DDDDD into auxiliary register A</td>
</tr>
<tr>
<td>101</td>
<td>DDDDD</td>
<td></td>
<td>Write DDDDD into auxiliary register B</td>
</tr>
<tr>
<td>011</td>
<td>USP/P2P1</td>
<td></td>
<td>Enable/disable parallel poll either in response to remote messages (PPC followed by PPE or PPD) or as a local IPE message. (Enable if U = 0, disable if U = 1.)</td>
</tr>
</tbody>
</table>

0100—Trigger: A “Group Execute Trigger” is forced by this command. It has the same effect as a GET command issued by the controller-in-charge of the GPIB, but does not cause a GET interrupt.

0101, 1101—Clear/Set rtl: These commands correspond to the local rtl message as defined by the IEEE-488. The 8291A will go into local mode when a Set rtl Auxiliary Command is received if local lockout is not in effect. The 8291A will exit local mode after receiving a Clear rtl Auxiliary Command if the 8291A is addressed to listen.

0110—Send EOI: The EOI line of the 8291A may be asserted with this command. The command causes EOI to go true with the next byte transmitted. The EOI line is then cleared upon completion of the handshake for that byte.

0111, 1111—Non Valid/Valid Secondary Address or Command (VSCMD): This command informs the 8291A that the secondary address received by the microprocessor was valid or invalid (0111 = invalid, 1111 = valid). If Mode 3 addressing is used, the processor must field each extended address and respond to it, or the GPIB will hang up. Note that the COM3 bit is the invalid/valid flag.

The valid (1111) command is also used to tell the 8291A to continue from the command-pass-through-state, or from RFD holdoff on GET, SDC or DCL.

1000—pon: This command puts the 8291A into the pon (power on) state and holds it there. It is similar to a Chip Reset except none of the Auxiliary Mode Registers are cleared. In this state, the 8291A does not participate in any bus activity. An Immediate Execute pon releases the 8291A from the pon state and permits the device to participate in the bus activity again.

0001, 1001—Parallel Poll Flag (local "ist" message): This command sets (1001) or clears (0001) the parallel poll flag. A "1" is sent over the assigned data line (PRR = Parallel Poll Response true) only if the parallel poll flag matches the sense bit from the lpe local message (or indirectly from the PPE message). For a more complete description of the Parallel Poll features and procedures refer to the section on Parallel Poll Protocol.

**INTERNAL COUNTER**

The internal counter determines the delay time allowed for the setting of data on the DIO lines. This delay time is defined as T, in IEEE-488 and appears in the Source Handshake state diagram between the
SDYS and STRS. As such, DAV is asserted T, after
the DIO lines are driven. Consequently, T, is a major
factor in determining the data transfer rate of the
8291A over the GPIB (T, = TWRDV2-TWRD15).

When open-collector transceivers are used for con-
nection to the GPIB, T, is defined by IEEE-488 to be
2μsec. By writing 0010DDDD into the Auxiliary Mode
Register, the counter is preset to match a f, MHz
clock input, where DDDD is the binary representa-
tion of Nf [1≦Nf≦8, Nf=(DDDD)2]. When Nf = f, a
2μsec T, delay will be generated before each DAV
 asserted.

\[ T_{(μsec)} = \frac{2N_f}{f_c} + t_{SYNC}, \ 1≦N_f≦8 \]

\( t_{SYNC} \) is a synchronization error, greater than zero
and smaller than the larger of T clock high and T
clock low. (For a 50% duty cycle clock, tSYNC is less
than half the clock cycle).

If it is necessary that T, be different from 2μsec, Nf
may be set to a value other than f, In this manner,
data transfer rates may be programmed for a given
system. In small systems, for example, where trans-
fer rates exceeding GPIB specifications are re-
quired, one may set Nf<f, and decrease T,.

When tri-state transceivers are used, IEEE-488 al-
 lows a higher transfer rate (lower T,). Use of the
8291A with such transceivers is enabled by setting
B, in Auxiliary Register B. In this case, setting Nf=f, causes a T, delay of 2μsec to be generated for the
first byte transmitted — all subsequent bytes will
have a delay of 500 nsec.

\[ T_{(High \ Speed \ μsec)} = \frac{N_f}{2f_c} + t_{SYNC} \]

Thus, the shortest T, is achieved by setting Nf=1
using an 8 MHz clock with a 50% duty cycle clock
(tSYNC<63 nsec):

\[ T_{(HS)} = \frac{1}{2x8} + 0.063 = 125 \text{ nsec max.} \]

**AUXILIARY REGISTER A**

Auxiliary Register A is a “hidden” 5-bit register
which is used to enable some of the 8291A features.
Whenever a 100 A,A,A,A,A byte is written into the
Auxiliary Register, it is loaded with the data
A,A,A,A,A. Setting the respective bits to “1”
enables the following features.

A0 — RFD Holdoff on all Data: If the 8291A is listen-
ing, RFD will not be sent true until the “finish hand-
shake” auxiliary command is issued by the
microprocessor. The holdoff will be in effect for each
data byte.

A1—RFD Holdoff on End: This feature enables the
holdoff on EOI or EOS (if enabled). However, no
holdoff will be in effect on any other data bytes.

A2—End on EOS Received: Whenever the byte in the
Data In Register matches the byte in the EOS Regis-
ter, the END interrupt bit will be set in the Interrupt
Status 1 Register.

A3—Output EOI on EOS Sent: Any occurrence of
data in the Data Out Register matching the EOS
Register causes the EOI line to be sent true along
with the data.

A4—EOS Binary Compare: Setting this bit causes
the EOS Register to function as a full 8-bit word.
When it is not set, the EOS Register is a 7-bit word
(for ASCII characters).

If A0=A1=1, a special “continuous Acceptor Hand-
shake cycling” mode is enabled. This mode should
be used only in a controller system configuration,
where both the 8291A and the 8292 are used. It
provides a continuous cycling through the Acceptor
Handshake state diagram, requiring no local mes-
gages from the microprocessor; the rdy local mes-
sage is automatically generated when in ANRS. As
such, the 8291A Acceptor Handshake serves as the
controller Acceptor Handshake. Thus, the controller
cycles through the Acceptor Handshake without de-
laying the data transfer in progress. When the tcs
local message is executed, the 8291A should be
taken out of the “continuous AH cycling” mode, the
GPIB will hang up in ANRS, and a BI interrupt will be
generated to indicate that control may be taken. A
simpler procedure may be used when a “tcs on end
of block” is executed; the 8291A may stay in “con-
tinuous AH cycling”. Upon the end of a block (EOI or
EOS received), a holdoff is generated, the GPIB
hangs up in ANRS, and control may be taken.
AUXILIARY REGISTER B

Auxiliary Register B is a "hidden" 4-bit register which is used to enable some of the features of the 8291A. Whenever a 101 B, B3, B2, B1, B0 is written into the Auxiliary Mode Register, it is loaded with the data B, B3, B2, B1. Setting the respective bits to "1" enables the following features:

B0—Enable Undefined Command Pass Through: This feature allows any commands not recognized by the 8291A to be handled in software. If enabled, this feature will cause the 8291A to holdoff the handshake when an undefined command is received. The microprocessor must then read the command from the Command Pass Through Register and send the VSCMD auxiliary command. Until the VSCMD command is sent, the handshake holdoff will be in effect.

B1—Send EOI in SPAS: This bit enables EOI to be sent with the status byte; EOI is sent true in Serial Poll Active State. Otherwise, EOI is sent false in SPAS.

B2—Enable High Speed Data Transfer: This feature may be enabled when tri-state external transceivers are used. The data transfer rate is limited by T, delay time generated in the Source Handshake function, which is defined according to the type of transceivers used. When the "High Speed" feature is enabled, T1 = 2 microseconds is generated for the first byte transmitted after each true to false transition of ATN. For all subsequent bytes, T1 = 500 nanoseconds. Refer to the Internal Counter section for an explanation of T1 duration as a function of B2 and of clock frequency.

B3—Enable Active Low Interrupt: Setting this bit causes the polarity of the INT pin to be reversed, providing an output signal compatible with Intel's MCS-48® Family. Interrupt registers are not affected by this bit.

B4—Enable RFD Holdoff on GET or DEC: Setting this bit causes RFD to be held false until the "VSCMD" auxiliary command is written after GET, SDC, and DCL commands. This allows the device to hold off the bus until it has completed a clear or trigger similar to an unrecognized command.

PARALLEL POLL PROTOCOL

Writing a 011USP,P2,P1 into the Auxiliary Mode Register will enable (U=0) or disable (U=1) the 8291A for a parallel poll. When U=0, this command is the "Ipe" (local poll enable) local message as defined in IEEE-488. The "S" bit is the sense in which the 8291A is enabled; only if the Parallel Poll Flag ("ist" local message) matches this bit will the Parallel Poll Response, PPRn, be sent true (Response=S + ist). The bits P3,P2,P1 specify which of the eight data lines PPRn will be sent over. Thus, once the 8291A has been configured for Parallel Poll, whenever it senses both EOI and ATN true, it will automatically compare its PP flag with the sense bit and send PPRn true or false according to the comparison.

If a PP2* implementation is desired, the "Ipe" and "ist" local messages are all that are needed. Typically, the user will configure the 8291A for Parallel Poll immediately after initialization. During normal operation the microprocessor will set or clear the Parallel Poll Flag (ist) according to the device's need for service. Consequently the 8291A will be set up to give the proper response to IDY (EOI + ATN) without directly involving the microprocessor.

If a PP1* implementation is desired, the undefined command features of the 8291A must be used. In PP1, the 8291A is indirectly configured for Parallel Poll by the active controller on the GPIB. The sequence at the 8291A being enabled or disabled remotely is as follows:

1. The PPC message is received and is loaded into the Command Pass Through Register as an undefined command. A CPT Interrupt is sent to the microprocessor; the handshake is automatically held off.

2. The microprocessor reads the CPT Register and sends VSCMD to the 8291A, releasing the handshake.

3. Having received an undefined primary command, the 8291A is set up to receive an undefined secondary command (the PPE or PPD message). This message is also received into the CPT Register, the handshake is held off, and the CPT interrupt is generated.

NOTE: *As defined in IEEE Standard 488.
4. The microprocessor reads the PPE or PPD message and writes the command into the Auxiliary Mode Register (bit 7 should be cleared first). Finally, the microprocessor sends VSCMD and the handshake is released.

End of Sequence (EOS) Register

<table>
<thead>
<tr>
<th>EC7</th>
<th>EC6</th>
<th>EC5</th>
<th>EC4</th>
<th>EC3</th>
<th>EC2</th>
<th>EC1</th>
<th>EC0</th>
</tr>
</thead>
</table>

The EOS Register and its features offer an alternative to the "Send EOI" auxiliary command. A seven or eight bit byte (ASCII or binary) may be placed in the register to flag the end of a block or read. The type of EOS byte to be used is selected in Auxiliary Register bit A.

If the 8291A is a listener, and the "End on EOS Received" is enabled with bit A0, then an END interrupt is generated in the Interrupt Status 1 Register whenever the byte in the Data-In Register matches the byte in the EOS Register.

If the 8291A is a talker, and the "Output EOI on EOS Sent" is enabled with bit A3, then the EOI line is sent true with the next byte whenever the contents of the Data Out Register match the EOS register.

Reset Procedure

The 8291A is reset to an initialization state either by a pulse applied to its Reset pin, or by a reset auxiliary command (02H written into the Auxiliary Command Register). The following conditions are caused by a reset pulse (or local reset command):

1. A "'pon" local message as defined by IEEE-488 is held true until the initialization state is released.
2. The Interrupt Status Registers are cleared (not Interrupt Enable Registers).
3. Auxiliary Registers A and B are cleared.
4. The Serial Poll Mode Register is cleared.
5. The Parallel Poll Flag is cleared.
6. The EOI bit in the Address Status Register is cleared.
7. Np in the Internal Counter is set to 8 MHz. This setting causes the longest possible T, delay to be generated in the Source Handshake (16 μsec for 1 MHz clock).
8. The rdy local message is sent.

The initialization state is released by an "Immediate execute pon" command (00H written into the Auxiliary Command Register).

The suggested initialization sequence is:

1. Apply a reset pulse or send the reset auxiliary command.
2. Set the desired initial conditions by writing into the Interrupt Enable, Serial Poll Mode, Address Mode, Address 0/1, and EOS Registers. Auxiliary Registers A and B, and the internal counter should also be initialized.
3. Send the "Immediate execute pon" auxiliary command to release the initialization state.
4. If a PP2 Parallel Poll implementation is to be used the "Ipe" local message may be sent, enabling the 8291A for a Parallel Poll Response on an assigned line. (Refer to the section on Parallel Poll Protocol.)

Using DMA

The 8291A may be connected to the Intel® 8237 or 8257 DMA Controllers or the 8089 I/O Processor for DMA operation. The 8237 will be used to refer to any DMA controller. The DREQ pin of the 8291A requests a DMA byte transfer from the 8237. It is set by BO or BI flip flops, enabled by the DMAO and DMAI bits in the Interrupt Enable 2 Register. (After reading, the INT1 register BO and BI interrupts will be cleared but not BO and BI in DREQ equation.)

The DACK pin is driven by the 8237 in response to the DMA request. When DACK is true (active low) it sets CS = RS0 = RS1 = RS2 = 0 such that the RD and WR signals sent by the 8237 refer to the Data In and Data Out Registers. Also, the DMA request line is reset by DACK (RD + WR).

DMA input sequence:

1. A data byte is accepted from the GPIB by the 8291A.
2. A BI interrupt is generated and DREQ is set.
3. DACK and RD are driven by the 8237, the contents of the Data In Register are transferred to the system bus, and DREQ is reset.
4. The 8291A sends RFD true on the GPIB and proceeds with the Acceptor Handshake protocol.

DMA output sequence:

1. A BO interrupt is generated (indicating that a byte should be output) and DREQ is asserted.
2. DACK and WR are driven by the 8237, a byte is transferred from the MCS bus into the Data Out Register, and DREQ is reset.

3. The 8291A sends DAV true on the GPIB and proceeds with the Source Handshake protocol. It should be noted that each time the device is addressed (MTA + MLA + ton + Ion), the Address Status Register should be read, and the 8237 should be initialized accordingly. (Refer to the 8237 or 8257 Data Sheets.)

**APPLICATION BRIEF**

**System Configuration**

**MICROPROCESSOR BUS CONNECTION**
The 8291A is 8048/49, 8051, 8080/85, and 8086/88 compatible. The three address pins (RS0, RS1, RS2) should be connected to the non-multiplexed address bus (for example: A0, A9, A10). In case of 8080, any address lines may be used. If the three lowest address bits are used (A0, A1, A2), then they must be demultiplexed first.

**EXTERNAL TRANSCEIVERS CONNECTION**
The 8293 GPIB Transceiver interfaces the 8291A directly to the IEEE-488 bus. The 8291A and two 8293's can be configured as a talker/listener (see Figure 6) or with the 8292 as a talker/listener/controller (see Figure 7). Absolutely no active or passive external components are required to comply with the complete IEEE-488 electrical specification.
Figure 7. 8291A, 8292, and 8293 System Configuration
Start-Up Procedures

The following section describes the steps needed to initialize a typical 8291A system implementing a talker/listener interface and an 8291A/8292 system implementing a talker/listener/controller interface.

TALKER/LISTENER SYSTEM

Assume a general system configuration with the following features: (i) Polled system interface; (ii) Mode 1 addressing; (iii) same address for talker and listener; (iv) ASCII carriage return as the end-of-sequence (EOS) character; (v) EOI sent true with the last byte; and, (vi) 8 MHz clock.

Initialization. Initialization is accomplished with the following steps:

1. Pulse the RESET input or write 02H to the Auxiliary Mode Register.
2. Write 00H to the Interrupt Enable Registers 1 and 2. This disables interrupt and DMA.
3. Write 01H to the Address Mode Register to select Mode 1 addressing.
4. Write 28H to the Auxiliary Mode Register. This loads 8H to the Auxiliary Register A matching the 8 MHz clock input to the internal T1 delay counter to generate the delay meeting the IEEE spec.
5. Write the talker/listener address to the Address 0/1 register. The three most significant bits are zero.
6. Write an ASCII carriage return (0DH) to the EOS register.
7. Write 84H to the Auxiliary Mode Register to allow EOI to be sent true when the EOS character is sent.
8. Write 00H to the Auxiliary Mode Register. This writes the "Immediate Execute pno" message and takes the 8291A from the initialization state into the idle state. The 8291A will remain idle until the controller initiates some activity by driving ATN true.

Communication. The local CPU now polls the 8291A to determine which controller command has been received.

The controller addresses the 8291A by driving ATN, placing MLA (My Listen Address) on the bus and driving DAT. If the lower five bits of the MLA message match the address programmed into the Address 0/1 register, the 8291A is addressed to listen. It would be addressed to talk if the controller sent the MTA message instead of MLA.

The ADSC bit in the Interrupt Status 2 Register indicates that the 8291A has been addressed or unaddressed. The TA and LA bits in the Address Status Register indicate whether the 8291A is talker (TA=1), listener (LA=1), both (TA=LA=1) or unaddressed (TA=LA=0).

If the 8291A is addressed to listen, the local CPU can read the Data-In Register whenever the BI (Byte In) interrupt occurs in the Interrupt Status 1 Register. If the END bit in the same register is also set, either EOI or a data byte matching the pattern in the EOS register has been received.

In the talker mode, the CPU writes data into the Byte-Out Register on BO (Byte Out) true.

TALKER/LISTENER/CONTROLLER SYSTEM

Combined with the Intel 8292, the 8291A executes a complete IEEE-488-1978 controller function. The 8291A talks and listens via the data and handshake lines (NRFD, NDAC and DAV). The 8292 controls four of the five bus management lines (IFC, SRQ, ATN and REN). EOI, the fifth line, is shared. The 8291A drives and receives EOI when EOI is used as an end-of-block indicator. The 8292 drives EOI along with ATN during a parallel poll command.

Once again, assume a general system configuration with the following features: (i) Polled system interface; (ii) 8292 as the system controller and controller-in-charge; (iii) ASCII carriage return (0DH) as the EOS identifier; (iv) EOI sent with the last character; and, (v) an external buffer (8282) used to monitor the TCI line.

Initialization. In order to send a command across the GPIB, the 8292 has to drive ATN, and the 8291A has to drive the data lines. Both devices therefore need initialization.

To initialize the 8292:

1. Pulse the RESET input. The 8292 will initially drive all outputs high. TCI, SPI, OBFI, IBFI and CLTH will then go low. The Interrupt Status, Interrupt Mask, Error Flag, Error Mask and Timeout registers will be cleared. The interrupt counter will be disabled and loaded with 255. The 8292 will then monitor the status of the SYC pin. If high, the 8292 will pulse IFC true for at least 100μs in compliance with the IEEE-488-1978 standard. It will then take control by asserting ATN.

To initialize the 8291A, the following is necessary:

1. Write 00H to Interrupt Enable registers 1 and 2. This disables interrupt and DMA.
2. With the 8292 as the controller-in-charge, it is impossible to address the 8292 via the GPIB. Therefore, the ton or Ion modes of the 8291A must be used. To send commands, set the 8291A in the ton mode by writing 80H to the Address Mode Register.

3. Write 26H to the Auxiliary Mode Register to match the T1 data settling time to the 6 MHz clock input.

4. Write an ASCII carriage return (0DH) to the EOS Register.

5. Write 84H to the Auxiliary Mode Register in order to enable “Output EOI on EOS sent” and thus send EOI with the last character.

6. Write OOh—Immediate Execute pon—to the Auxiliary Mode Register to put the 8291A in the idle state.

Communication. Since the 8291A is in the ton mode, a BO interrupt is generated as soon as the immediate Execute pon command is written. The CPU writes the command into the Data Out Register, and repeats it on BO becoming true for as many commands as necessary. ATN remains continuously true unless the GTSB (Go To Standby) command is sent to the 8292.

ATN has to be false in order to send data rather than commands from the controller. To do this, the following steps are needed:

1. Enable the TCI interrupt if not already enabled.
2. Wait for IBF (Input Buffer Full) in the 8292 Interrupt Status Register to be reset.
3. Write the GTSB (F6H) command to the 8292 Command Field Register.
4. Read the 8282 and wait for TCI to be true.
5. Write the ton (80H) and pon (00H) command to the 8291A Address Mode Register and Auxiliary Mode Registers respectively.
6. Wait for the BO interrupt to be set in the 8291A.
7. Write the data to the 8291A Data-Out Register.

Identically, the user could command the controller to listen rather than talk. To do that, write Ion (40H) instead of ton into the Address Mode Register. Then wait for BI rather than BO to go true. Read the data Register.
ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias ............ -65°C to 70°C
Storage Temperature ........................ -65°C to +150°C
Voltage on Any Pin With Respect to Ground ........................ -0.5V to +7V
Power Dissipation .......................... 0.65 Watts

"NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS \[V_{cc} = 5V \pm 10\%, T_a = 0°C to 70°C (Commercial)\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter Description</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_{IL}</td>
<td>Low Voltage Input</td>
<td>-0.5</td>
<td>0</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{IH}</td>
<td>High Voltage Input</td>
<td>2</td>
<td>V_{cc}+0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{OL}</td>
<td>Low Voltage Output</td>
<td>0.45</td>
<td>2 \times V_{ee}+0.5</td>
<td>V</td>
<td>I_{OL} = 2mA \text{ (4mA for TR1 pin)}</td>
</tr>
<tr>
<td>V_{OH}</td>
<td>High Voltage Output</td>
<td>2.4</td>
<td>2 \times V_{ee}</td>
<td>V</td>
<td>I_{OH} = -400\mu A \text{ (-150\mu A for SRQ pin)}</td>
</tr>
<tr>
<td>V_{OH-INT}</td>
<td>Interrupt High Voltage</td>
<td>2.4</td>
<td>2 \times V_{ee}</td>
<td>V</td>
<td>I_{OH} = -400\mu A</td>
</tr>
<tr>
<td>I_{IL}</td>
<td>Input Leakage</td>
<td>10</td>
<td>\mu A</td>
<td></td>
<td>Vin=0V to V_{cc}</td>
</tr>
<tr>
<td>I_{OFL}</td>
<td>Output Leakage Current</td>
<td>10</td>
<td>\mu A</td>
<td></td>
<td>V_{OUT} = 0.45V, V_{cc}</td>
</tr>
<tr>
<td>I_{CC}</td>
<td>V_{cc} Supply Current</td>
<td>120</td>
<td>mA</td>
<td></td>
<td>T_{a}=0°C</td>
</tr>
</tbody>
</table>

A.C. CHARACTERISTICS \[V_{cc} = 5V \pm 10\%, T_a = 0°C to 70°C (Commercial)\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter Description</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{AR}</td>
<td>Address Stable Before READ</td>
<td>0</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{RA}</td>
<td>Address Hold After READ</td>
<td>0</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{RR}</td>
<td>READ width</td>
<td>140</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{AD}</td>
<td>Address Stable to Data Valid</td>
<td>250</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{RD}</td>
<td>READ to Data Valid</td>
<td>100</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{RF}</td>
<td>Data Float After READ</td>
<td>0</td>
<td>60</td>
<td>nsec</td>
<td></td>
</tr>
<tr>
<td>t_{AW}</td>
<td>Address Stable Before WRITE</td>
<td>0</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{WA}</td>
<td>Address Hold After WRITE</td>
<td>0</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{WW}</td>
<td>WRITE width</td>
<td>170</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{DW}</td>
<td>Data Set Up Time to the Trailing Edge of WRITE</td>
<td>130</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{WD}</td>
<td>Data Hold Time After WRITE</td>
<td>0</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{DKDR4}</td>
<td>RD\downarrow or WR\downarrow to DREQ\downarrow</td>
<td>130</td>
<td>nsec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{DKDA6}</td>
<td>RD\downarrow to Valid Data (D_0\rightarrow D_7)\</td>
<td>200</td>
<td>nsec \text{ (DACK\downarrow to RD\downarrow, } 0 \leq t \leq 50\text{nsec)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
WAVEFORMS

READ

<table>
<thead>
<tr>
<th>CS/RS</th>
<th>tRR</th>
<th>tRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>tAD</td>
<td></td>
</tr>
<tr>
<td>DATA BUS (DATA OUT)</td>
<td>tAR</td>
<td>tRD</td>
</tr>
<tr>
<td>VALID DATA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WRITE

<table>
<thead>
<tr>
<th>CS/RS</th>
<th>tWW</th>
<th>tWA</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE</td>
<td>tAW</td>
<td>tOW</td>
</tr>
<tr>
<td>DATA BUS (DATA IN)</td>
<td>tAW</td>
<td>tWD</td>
</tr>
<tr>
<td>DATA MAY CHANGE</td>
<td>VALID DATA</td>
<td>DATA MAY CHANGE</td>
</tr>
</tbody>
</table>

DMA

<table>
<thead>
<tr>
<th>DREQ</th>
<th>tCKDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DACK</td>
<td></td>
</tr>
<tr>
<td>RD or WR</td>
<td></td>
</tr>
</tbody>
</table>
## A.C. TIMING MEASUREMENT POINTS AND LOAD CONDITIONS

![Diagram showing test points and load conditions](image.png)

**Notes:**
1. A.C. testing inputs are driven at 2.4V for a logic '1' and 0.45V for a logic '0'. Timing measurements are made at 2.4V for a logic '1' and 0.8V for a logic '0'.

### GPIB Timings

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEO13<code>1</code></td>
<td>EOI↓ to TR1↑</td>
<td>135</td>
<td>nsec</td>
<td>PPSS, ATN=0.45V</td>
</tr>
<tr>
<td>TEO16</td>
<td>EOI↓ to DIO Valid</td>
<td>155</td>
<td>nsec</td>
<td>PPSS, ATN=0.45V</td>
</tr>
<tr>
<td>TEO12</td>
<td>EOI↑ to TR1↓</td>
<td>155</td>
<td>nsec</td>
<td>PPSS, ATN=0.45V</td>
</tr>
<tr>
<td>TATND4</td>
<td>ATN↓ to NDAC↑</td>
<td>155</td>
<td>nsec</td>
<td>TACS, AID, SP</td>
</tr>
<tr>
<td>TATT14</td>
<td>ATN↑ to TR1↓</td>
<td>155</td>
<td>nsec</td>
<td>TACS, AID, SP</td>
</tr>
<tr>
<td>TATT24</td>
<td>ATN↓ to TR2↓</td>
<td>155</td>
<td>nsec</td>
<td>TACS, AID, SP</td>
</tr>
<tr>
<td>TDVND3-C</td>
<td>DV↓ to NDAC↑</td>
<td>650</td>
<td>nsec</td>
<td>AH, CACP, SP</td>
</tr>
<tr>
<td>TNDDV1</td>
<td>NDAC↑ to DV↑</td>
<td>350</td>
<td>nsec</td>
<td>SH, STRS, SP</td>
</tr>
<tr>
<td>TNRDR1</td>
<td>NRFD↑ to DREQ↑</td>
<td>400</td>
<td>nsec</td>
<td>SH</td>
</tr>
<tr>
<td>TDVDR3</td>
<td>DV↓ to DREQ↑</td>
<td>600</td>
<td>nsec</td>
<td>AH, LACS, ATN=2.4V</td>
</tr>
<tr>
<td>TDVND2-C</td>
<td>DV↑ to NDAC↓</td>
<td>350</td>
<td>nsec</td>
<td>AH, LACS, SP</td>
</tr>
<tr>
<td>TDVNR1-C</td>
<td>DV↑ to NRFD↑</td>
<td>350</td>
<td>nsec</td>
<td>AH, LACS, rdy=True</td>
</tr>
<tr>
<td>TRDNR3</td>
<td>R↓ to NRFD↑</td>
<td>500</td>
<td>nsec</td>
<td>AH, LACS, SP</td>
</tr>
<tr>
<td>TWRD15</td>
<td>WR↑ to DIO Valid</td>
<td>280</td>
<td>nsec</td>
<td>SH, TACS, RS=0.4V</td>
</tr>
<tr>
<td>TWRE05</td>
<td>WR↑ to EOI Valid</td>
<td>350</td>
<td>nsec</td>
<td>SH, TACS</td>
</tr>
<tr>
<td>TWRD2V2</td>
<td>WR↑ to DV↓</td>
<td>830</td>
<td>nsec</td>
<td>High Speed Transfers Enabled, (N_F = f _C, \ t_{SYNC} = 1/2f_C)</td>
</tr>
</tbody>
</table>

**Notes:**
1. All GPIB timings are at the pins of the 8291A.
2. The last number in the symbol for any GPIB timing parameter is chosen according to the transition directions of the reference signals. The following table describes the numbering scheme.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑ to ↑</td>
<td>1</td>
</tr>
<tr>
<td>↑ to ↓</td>
<td>2</td>
</tr>
<tr>
<td>↓ to ↑</td>
<td>3</td>
</tr>
<tr>
<td>↓ to ↓</td>
<td>4</td>
</tr>
<tr>
<td>↑ to VALID</td>
<td>5</td>
</tr>
<tr>
<td>↓ to VALID</td>
<td>6</td>
</tr>
</tbody>
</table>
APPENDIX A

MODIFIED STATE DIAGRAMS

Figure A-1 presents the interface function state diagrams. It is derived from IEEE Std. state diagrams, with the following changes:

A. The 8291A supports the complete set of IEEE-488 interface functions except for the controller. These include: SH1, AH1, T5, TE5, L3, LE3, SR1, RL1, PP1, DC1, DT1, and C0.

B. Addressing modes included in T,L state diagrams.

Note that in Mode 3, MSA, OSA are generated only after secondary address validity check by the microprocessor (APT interrupt).

C. In these modified state diagrams, the IEEE-488-1978 convention of negative (low true) logic is followed. This should not be confused with the Intel pin- and signal-naming convention based on positive logic. Thus, while the state diagrams below carry low true logic, the signals described elsewhere in this data sheet are consistent with Intel notation and are based on positive logic.

<table>
<thead>
<tr>
<th>Level</th>
<th>Logic</th>
<th>IEEE-488</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td>DAV</td>
<td>DAV</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>DAV</td>
<td>DAV</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>NDAC</td>
<td>NDAC</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>NDAC</td>
<td>NDAC</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>NRFD</td>
<td>NRFD</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>NRFD</td>
<td>NRFD</td>
</tr>
</tbody>
</table>

Consider the condition when the Not-Ready-For-Data signal (pin 37) is active. Intel indicates this active low signal with the symbol NRFD (V_{OUT} \leq V_{OL} for AH; V_{IN} \leq V_{IL} for SH). The IEEE-488-1978 Standard, in its state diagrams, indicates the active state of this signal (True condition) with NRFD.

D. All remote multiline messages decoded are conditioned by ACDS. The multiplication by ACDS is not drawn to simplify the diagrams.

E. The symbol

![Symbol](image)

indicates:

1. When event X occurs, the function returns to state S.
2. X overrides any other transition condition in the function.

Statement 2 simplifies the diagram, avoiding the explicit use of X to condition all transitions from S to other states.

---

Figure A-1. 8291A State Diagrams (Continued next page)
Figure A-1. 8291A State Diagrams (Continued next page)
Figure A-1. 8291A State Diagrams
### APPENDIX B

#### Table B-1. IEEE 488 Time Values

<table>
<thead>
<tr>
<th>Time Value Identifier</th>
<th>Function (Applies to)</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_1</td>
<td>SH</td>
<td>Settling Time for Multiline Messages</td>
<td>≥ 2μs²</td>
</tr>
<tr>
<td>t_2</td>
<td>LC, IC, SH, AH, T, L</td>
<td>Response to ATN</td>
<td>≤ 200ns</td>
</tr>
<tr>
<td>T_3</td>
<td>AH</td>
<td>Interface Message Accept Time³</td>
<td>&gt; 0*</td>
</tr>
<tr>
<td>T_4</td>
<td>T, TE, L, LE, C, CE</td>
<td>Response to IFC or REN False</td>
<td>&lt; 100μs</td>
</tr>
<tr>
<td>t_5</td>
<td>PP</td>
<td>Response to ATN+EOI</td>
<td>≤ 200ns</td>
</tr>
<tr>
<td>T_6</td>
<td>C</td>
<td>Parallel Poll Execution Time</td>
<td>≥ 2μs</td>
</tr>
<tr>
<td>T_7</td>
<td>C</td>
<td>Controller Delay to Allow Current Talker to see ATN Message</td>
<td>≥ 500 ns</td>
</tr>
<tr>
<td>T_8</td>
<td>C</td>
<td>Length of IFC or REN False</td>
<td>&gt; 100μs</td>
</tr>
<tr>
<td>T_9</td>
<td>C</td>
<td>Delay for EOI⁴</td>
<td>≥ 1.5μs⁴</td>
</tr>
</tbody>
</table>

**NOTES:**

1. Time values specified by a lower case t indicate the maximum time allowed to make a state transition. Time values specified by an upper case T indicate the minimum time that a function must remain in a state before exiting.

2. If three-state drivers are used on the DIO, DAV, and EOI lines, T may be:
   1. ≥ 1100 ns.
   2. Or ≥ 700 ns if it is known that within the controller ATN is driven by a three-state driver.
   3. Or ≥ 500ns for all subsequent bytes following the first sent after each false transition of ATN (the first byte must be sent in accordance with (1) or (2).
   4. Or ≥ 350ns for all subsequent bytes following the first sent after each false transition of ATN under conditions specified in Section 5.2.3 and warning note. See IEEE Standard 488.

³Time required for interface functions to accept, not necessarily respond to interface messages.

⁴Implementation dependent.

⁵Delay required for EOI, NDAC, and NRFD signal lines to indicate valid states.

⁶≥ 600 ns for three-state drivers.
APPENDIX C
THE THREE-WIRE HANDSHAKE

Figure C-1. 3-Wire Handshake Timing at 8291A
The 8292 GPIB Controller is a microprocessor-controlled chip designed to function with the 8291 GPIB Talker/Listener to implement the full IEEE Standard 488 controller function, including transfer control protocol. The 8292 is a pre-programmed Intel® 8041A.
# Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFC_L</td>
<td>1</td>
<td>I</td>
<td>IFC Received (Latched): The 8292 monitors the IFC Line (when not system controller) through this pin.</td>
</tr>
<tr>
<td>X1, X2</td>
<td>2, 3</td>
<td>I</td>
<td>Crystal Inputs: Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.</td>
</tr>
<tr>
<td>RESET</td>
<td>4</td>
<td>I</td>
<td>Reset: Used to initialize the chip to a known state during power on.</td>
</tr>
<tr>
<td>CS</td>
<td>6</td>
<td>I</td>
<td>Chip Select Input: Used to select the 8292 from other devices on the common data bus.</td>
</tr>
<tr>
<td>RD</td>
<td>8</td>
<td>I</td>
<td>Read Enable: Allows the master CPU to read from the 8292.</td>
</tr>
<tr>
<td>A0</td>
<td>9</td>
<td>I</td>
<td>Address Line: Used to select between the data bus and the status register during read operations and to distinguish between data and commands written into the 8292 during write operations.</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>I</td>
<td>Write Enable: Allows the master CPU to write to the 8292.</td>
</tr>
<tr>
<td>SYNC</td>
<td>11</td>
<td>O</td>
<td>Sync: 8041A instruction cycle synchronization signal; it is an output clock with a frequency of XTAL ( \sim 15 ) Hz.</td>
</tr>
<tr>
<td>D0–D7</td>
<td>12-19</td>
<td>I/O</td>
<td>Data: 8 bidirectional lines used for communication between the central processor and the 8292’s data bus buffers and status register.</td>
</tr>
<tr>
<td>VSS</td>
<td>7, 20</td>
<td>P.S.</td>
<td>Ground: Circuit ground potential.</td>
</tr>
<tr>
<td>SRQ</td>
<td>21</td>
<td>I</td>
<td>Service Request: One of the IEEE control lines. Sampled by the 8292 when it is controller in charge. If true, SPI interrupt to the master will be generated.</td>
</tr>
<tr>
<td>ATNI</td>
<td>22</td>
<td>I</td>
<td>Attention In: Used by the 8292 to monitor the GPIB ATN control line. It is used during the transfer control procedure.</td>
</tr>
<tr>
<td>IFC</td>
<td>23</td>
<td>I/O</td>
<td>Interface Clear: One of the GPIB management lines, as defined by IEEE Std. 488-1978, places all devices in a known quiescent state.</td>
</tr>
<tr>
<td>SYC</td>
<td>24</td>
<td>I</td>
<td>System Controller: Monitors the system controller switch.</td>
</tr>
<tr>
<td>CLTH</td>
<td>27</td>
<td>O</td>
<td>Clear Latch: Used to clear the IFC latch after being recognized by the 8292. Usually low (except after hardware Reset), it will be pulsed high when IFC is recognized by the 8292.</td>
</tr>
<tr>
<td>ATNO</td>
<td>29</td>
<td>O</td>
<td>Attention Out: Controls the ATN control line of the bus through external logic for tcs and tca procedures. (ATN is a GPIB control line, as defined by IEEE Std. 488-1978.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>5, 26, 40</td>
<td>P.S.</td>
<td>Voltage: ( +5 ) V supply input ( \pm 10% ).</td>
</tr>
<tr>
<td>COUNT</td>
<td>39</td>
<td>I</td>
<td>Event Count: When enabled by the proper command the internal counter will count external events through this pin. High to low transition will increment the internal counter by one. The pin is sampled once per three internal instruction cycles (7.5 μsec sample period when using 5 MHz XTAL). It can be used for byte counting when connected to NDAC, or for block counting when connected to the EOI.</td>
</tr>
<tr>
<td>REN</td>
<td>38</td>
<td>O</td>
<td>Remote Enable: The Remote Enable bus signal selects remote or local control of the device on the bus. A GPIB bus management line, as defined by IEEE Std. 488-1978.</td>
</tr>
<tr>
<td>DAV</td>
<td>37</td>
<td>I/O</td>
<td>Data Valid: Used during parallel poll to force the 8291 to accept the parallel poll status bits. It is also used during the tcs procedure.</td>
</tr>
<tr>
<td>IBFI</td>
<td>36</td>
<td>O</td>
<td>Input Buffer Not Full: Used to interrupt the central processor while the input buffer of the 8292 is empty. This feature is enabled and disabled by the interrupt mask register.</td>
</tr>
<tr>
<td>OBFI</td>
<td>35</td>
<td>O</td>
<td>Output Buffer Full: Used as an interrupt to the central processor while the output buffer of the 8292 is full. The feature can be enabled and disabled by the interrupt mask register.</td>
</tr>
<tr>
<td>EO12</td>
<td>34</td>
<td>I/O</td>
<td>End Or Identify: One of the GPIB management lines, as defined by IEEE Std. 488-1978. Used with ATN as Identify Message during parallel poll.</td>
</tr>
<tr>
<td>SPI</td>
<td>33</td>
<td>O</td>
<td>Special Interrupt: Used as an interrupt on events not initiated by the central processor.</td>
</tr>
<tr>
<td>TCI</td>
<td>32</td>
<td>O</td>
<td>Task Complete Interrupt: Interrupt to the control processor used to indicate that the task requested was completed by the 8292 and the information requested is ready in the data bus buffer.</td>
</tr>
<tr>
<td>CIC</td>
<td>31</td>
<td>O</td>
<td>Controller In Charge: Controls the S/R input of the SRQ bus transceiver. It can also be used to indicate that the 8292 is in charge of the GPIB bus.</td>
</tr>
</tbody>
</table>

P.S. VDHa98: +5V

IEEE Std. 488-1978.
FUNCTIONAL DESCRIPTION

The 8292 is an Intel 8041A which has been programmed as a GPIB Controller interface element. It is used with the 8291 GPIB Talker/Listener and two 8293 GPIB Transceivers to form a complete IEEE-488 Bus Interface for a microprocessor. The electrical interface is performed by the transceivers, data transfer is done by the talker/listener, and control of the bus is done by the 8292. Figure 3 is a typical controller interface using Intel’s GPIB peripherals.

Figure 3. Talker/Listener/Controller Configuration

The internal RAM in the 8041A is used as a special purpose register bank for the 8292. Most of these registers (except for the interrupt flag) can be accessed through commands to the 8292. Table 2 identifies the registers used by the 8292 and how they are accessed.

Table 2. 8292 Registers

<table>
<thead>
<tr>
<th>READ FROM 8292</th>
<th>INTERRUPT STATUS</th>
<th>WRITE TO 8292</th>
<th>INTERRUPT MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SYC</td>
<td>ERR</td>
<td>SRQ</td>
</tr>
<tr>
<td>ERROR FLAG</td>
<td>X</td>
<td>X</td>
<td>USER</td>
</tr>
<tr>
<td>CONTROLLER STATUS</td>
<td>CSBS</td>
<td>CA</td>
<td>X</td>
</tr>
<tr>
<td>GPIB (BUS) STATUS</td>
<td>REN</td>
<td>DAV</td>
<td>EOI</td>
</tr>
<tr>
<td>EVENT COUNTER STATUS</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>TIME OUT STATUS</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

Note: These registers are accessed by a special utility command, see page 6.

Interrupt Status Register

<table>
<thead>
<tr>
<th>SYC</th>
<th>ERR</th>
<th>SRQ</th>
<th>EV</th>
<th>X</th>
<th>IFC</th>
<th>IBF</th>
<th>OBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>D0</td>
<td>D0</td>
<td>D0</td>
<td>D0</td>
<td>D0</td>
<td>D0</td>
<td>D0</td>
</tr>
</tbody>
</table>

The 8292 can be configured to interrupt the microprocessor on one of several conditions. Upon receipt of the interrupt the microprocessor must read the 8292 interrupt status register to determine which event caused the interrupt, and then the appropriate subroutine can be performed. The interrupt status register is read with A0 high. With the exception of OBF and IBF, these interrupts are enabled or disabled by the SPI interrupt mask. OBF and IBF have their own bits in the interrupt mask (OBF1 and IBF1).

OBF Output Buffer Full. A byte is waiting to be read by the microprocessor. This flag is cleared when the output data bus buffer is read.

IBF Input Buffer Full. The byte previously written by the microprocessor has not been read yet by the 8292. If another byte is written to the 8292 before this flag clears, data will be lost. IBF is cleared when the 8292 reads the data byte.

IFCR Interface Clear Received. The GPIB system controller has set IFC. The 8292 has become idle and is no longer in charge of the bus. The flag is cleared when the IACK command is issued.

EV Event Counter Interrupt. The requested number of blocks or data bytes has been transferred. The EV interrupt flag is cleared by the IACK command.

SRQ Service Request. Notifies the 8292 that a service request (SRQ) message has been received. It is cleared by the IACK command.

ERR Error occurred. The type of error can be determined by reading the error status register. This interrupt flag is cleared by the IACK command.

SYNC System Controller Switch Change. Notifies the processor that the state of the system controller switch has changed. The actual state is contained in the GPIB Status Register. This flag is cleared by the IACK command.
**Interrupt Mask Register**

<table>
<thead>
<tr>
<th>D7</th>
<th>SPI</th>
<th>TCI</th>
<th>SYC</th>
<th>IBSF</th>
<th>IBFI</th>
<th>0</th>
<th>SRQ</th>
</tr>
</thead>
</table>

The Interrupt Mask Register is used to enable features and to mask the SPI and TCI interrupts. The flags in the Interrupt Status Register will be active even when masked out. The Interrupt Mask Register is written when A0 is low and reset by the INM command. When the register is read, D1 and D2 are undefined. An interrupt is enabled by setting the corresponding register bit.

- SRQ Enable interrupts on SRQ received.
- IBFI Enable interrupts on input buffer empty.
- OBFI Enable interrupts on output buffer full.
- SYC Enable interrupts on a change in the system controller switch.
- TCI Enable interrupts on the task completed.
- SPI Enable interrupts on special events.

**NOTE:** The event counter is enabled by the GSEC command, the error interrupt is enabled by the error mask register, and IFC cannot be masked (it will always cause an interrupt).

**Controller Status Register**

<table>
<thead>
<tr>
<th>D7</th>
<th>CA</th>
<th>X</th>
<th>X</th>
<th>SYCS</th>
<th>IFC</th>
<th>REN</th>
<th>SRQ</th>
</tr>
</thead>
</table>

The Controller Status Register is used to determine the status of the controller function. This register is accessed by the RCST command.

- SRQ Service Request line active (CSRS).
- REN Sending Remote Enable.
- IFC Sending or receiving interface clear.
- SYCS System Controller Switch Status (SACS).
- CA Controller Active (CACS + CAWS + CSWS).
- CSBS Controller Stand-by State (CSBS, CA) = (0,0) — Controller Idle

**GPIB Bus Status Register**

<table>
<thead>
<tr>
<th>D7</th>
<th>REN</th>
<th>DAV</th>
<th>EOI</th>
<th>X</th>
<th>SYC</th>
<th>IFC</th>
<th>ATNI</th>
<th>SRQ</th>
</tr>
</thead>
</table>

This register contains GPIB bus status information. It can be used by the microprocessor to monitor and manage the bus. The GPIB Bus Register can be read using the RBST command.

Each of these status bits reflect the current status of the corresponding pin on the 8292.

- SRQ Service Request
- ATNI Attention In
- IFC Interface Clear
- SYC System Controller Switch
- EOI End or Identify
- DAV Data Valid
- REN Remote Enable

**Event Counter Register**

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
</table>

The Event Counter Register contains the initial value for the event counter. The counter can count pulses on pin 39 of the 8292 (COUNT). It can be connected to EOI or NDAC to count blocks or bytes respectively during standby state. A count of zero equals 256. This register cannot be read, and is written using the WEVC command.

**Event Counter Status Register**

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
</table>

This register contains the current value in the event counter. The event counter counts back from the initial value stored in the Event Counter Register to zero and then generates an Event Counter Interrupt. This register cannot be written and can be read using a REVC command.

**Time Out Register**

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
</table>

The Time Out Register is used to store the time used for the time out error function. See the individual timeouts (TOUT1, 2, 3) to determine the units of this counter. This Time Out Register cannot be read, and it is written with the WOUT command.

**Time Out Status Register**

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
</table>

This register contains the current value in the time out counter. The time out counter decrements from the original value stored in the Time Out Register. When zero is reached, the appropriate error interrupt is generated. If the register is read while none of the time out functions are active, the register will contain the last value reached the last time a function was active. The Time Out Status Register cannot be written, and it is read with the RTC command.

**Error Flag Register**

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
</table>

Four errors are flagged by the 8292 with a bit in the Error Flag Register. Each of these errors can be masked by the Error Mask Register. The Error Flag Register cannot be written, and it is read by the IACK command when the error flag in the Interrupt Status Register is set.

**TOUT1 Time Out Error 1** occurs when the current controller has not stopped sending ATN after receiving the TCT message for the time period specified by the Time Out Register. Each count in the Time Out Register is at least 1800 TCY. After flagging the error, the 8292 will remain in a loop trying to take control until the current controller stops sending ATN or a new command is written by the microprocessor. If a new command is written, the 8292 will return to the loop after executing it.
**OPERATION COMMANDS**

Operation commands initiate some action on the GPIB interface bus. It is using these commands that the control functions such as polling, taking and passing control, and system controller functions are performed.

**F0 — SPCNI — Stop Counter Interrupts**

This command disables the internal counter interrupt so that the 8292 will stop interrupting the master on event counter underflows. However, the counter will continue counting and its contents can still be used.

**F1 — GIDL — Go To Idle**

This command is used during the transfer of control procedure while transferring control to another controller. The 8292 will respond to this command only if it is in the active state. ATNO will go high, and CIC will be high so that this 8292 will no longer be driving the ATN line on the GPIB interface bus. TCI will be set upon completion.

**F2 — RST — Reset**

This command has the same effect as asserting the external reset on the 8292. For details, refer to the reset procedure described later.

**F3 — RSTI — Reset Interrupts**

This command resets any pending interrupts and clears the error flags. The 8292 will not return to any loop it was in (such as from the time out interrupts).

**F4 — GSEC — Go To Standby, Enable Counting**

The function causes ATNO to go high and the counter will be enabled. If the 8292 was not the active controller, this command will exit immediately. If the 8292 is the active controller, the counter will be loaded with the value stored in the Event Counter Register, and the internal interrupt will be enabled so that when the counter reaches zero, the SPI interrupt will be generated. SPI will be generated every 256 counts thereafter until the controller exits the standby state or the SPCNI command is written. An initial count of 256 (zero in the Event Counter Register) will be used if the WEVC command is not executed. If the data transmission does not start, a TOUT2 error will be generated.

**F5 — EXPP — Execute Parallel Poll**

This command initiates a parallel poll by asserting EOI when ATN is already active. TCI will be set at the end of the command. The 8291 should be previously configured as a listener. Upon detection of DAV true, the 8291 enters ACDS and latches the parallel poll response (PPR) byte into its data register. The master will be interrupted by the 8291 BI interrupt when the PPR byte is available. No interrupts except the IBI will be generated by the 8292. The 8292 will respond to this command only when it is the active controller.

**F6 — GTSB — Go To Standby**

If the 8292 is the active controller, ATNO will go high then TCI will be generated. If the data transmission does not start, a TOUT2 error will be generated.

**F7 — SLOC — Set Local Mode**

If the 8292 is the system controller, then REN will be asserted false and TCI will be set true. If it is not the system controller, the User Error bit will be set in the Error Flag Register.

**F8 — SREM — Set Interface To Remote Control**

This command will set REN true and TCI true if this 8292 is the system controller. If not, the User Error bit will be set in the Error Flag Register.
ABORT
WTOUT
RCST
RBST
Start

inter
(by an error flag), this
There are two different procedures used to transfer the
TCAS
3.

(cycling) mode (Aux. Reg.
8291 in the system.

3. After the current controller releases ATN, the 8292
command).

Finally, the TCI interrupt is generated to inform the
master that it is in control of the bus.

TCAS
- Take Control Asynchronously

TCAS transfers the 8292 from CSBS to CACS independent
of the handshake lines. If a bus hangup is detected
(by an error flag), this command will force the 8292
to take control (asserting ATN) even if the AH function
is not in ANRS (Acceptor Not Ready State). This command
should be used very carefully since it may cause the
loss of a data byte. Normally, control should be taken
synchronously. After checking the controller function
for being in the CSBS (else it will exit immediately),
ATNO will go low, and a TCI interrupt will be generated.

FC
- TCASY

There are two different procedures used to transfer the
8292 from CSBS to CACS depending on the state of the
8291 in the system. If the 8291 is in "continuous AH
(cycling) mode (Aux. Reg. A0 = A1 = 1), then the
following procedure should be followed:

1. The master microprocessor stops the continuous AH
(cycling) mode in the 8291;
2. The master reads the 8291 Interrupt Status 1
Register;
3. If the END bit is set, the master sends the TCASY
command to the 8292;
4. If the END bit was not set, the master reads the 8291
Data In Register and then waits for another BI
interrupt from the 8291. When it occurs, the master
sends the 8292 the TCASY command.

If the 8291 is not in AH cycling mode, then the master
just waits for a BI interrupt and then sends the TCASY
command. After the TCASY command has been issued,
the 8292 checks for CSBS. If CSBS, then it exits the
routine. Otherwise, it then checks the DAV bit in the
GPIB status. When DAV becomes false, the 8292 will
wait for at least 1.5 μsec. (T10) and then ATNO will go
low. If DAV does not go low, a TOUT3 error will be
generated. If the 8292 successfully takes control, it sets
TCI true.

FE
- STCNI

This command enables the internal counter interrupt.
The counter is enabled by the GSEC command.

UTILITY COMMANDS

All these commands are either Read or Write to registers
in the 8292. Note that writing to the Error Mask Register
and the Interrupt Mask Register are done directly.

E1
- WTOUT

The byte written to the data bus buffer (with A0 = 0)
following this command will determine the time used
for the time out function. Since this function is
implemented in software, this will not be an accurate
time measurement. This feature is enable or disable by the
Error Mask Register. No interrupts except for the IBF1
will be generated upon completion.

E2
- WEVC

The byte written to the data bus buffer (with A0 = 0)
following this command will be loaded into the Event
Counter Register and the Event Counter Status for byte
counting or EOI counting. Only IBF1 will indicate
completion of this command.

E3
- REVC

This command transfers the contents of the Event
Counter into the data bus buffer. A TCI is generated
when the data is available in the data bus buffer.

E4
- RERF

This command transfers the contents of the Error Flag
Register into the data bus buffer. A TCI is generated
when the data is available.

E5
- RINM

This command transfers the contents of the Interrupt
Mask Register into the data bus buffer. This register is
available to the processor so that it does not need to
store this information elsewhere. A TCI is generated
when the data is available in the data bus buffer.

E6
- RCST

This command transfers the contents of the Controller
Status Register into the data bus buffer and a TCI
interrupt is generated.

E7
- RBST

This command transfers the contents of the GPIB Bus
Status Register into the data bus buffer, and a TCI
interrupt is generated when the data is available.
E9 — ROUT — Read Time Out Status Register
This command transfers the contents of the Time Out Status Register into the data bus buffer, and a TCI interrupt is generated when the data is available.

EA — RERM — Read Error Mask Register
This command transfers the contents of the Error Mask Register to the data bus buffer so that the processor does not need to store this information elsewhere. A TCI interrupt is generated when the data is available.

Interrupt Acknowledge

<table>
<thead>
<tr>
<th>SYC</th>
<th>ERR</th>
<th>SRQ</th>
<th>EV</th>
<th>IFCR</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each named bit in an Interrupt Acknowledge (ACK) corresponds to a flag in the Interrupt Status Register. When the 8292 receives this command, it will clear the SPI and the corresponding bits in the Interrupt Status Register. If not all the bits were cleared, then the SPI will be set true again. If the error flag is not acknowledged by the IACK command, then the Error Flag Register will be transferred to the data bus buffer, and a TCI will be generated.

NOTE: XXXX1X11 is an undefined operation or utility command, so no conflict exists between the IACK operation and utility commands.

SYSTEM OPERATION
8292 To Master Processor Interface
Communication between the 8292 and the Master Processor can be either interrupt based communication or based upon polling the interrupt status register in predetermined intervals.

Interrupt Based Communication
Four different interrupts are available from the 8292:

- **OBFI** Output Buffer Full Interrupt
- **IBFI** Input Buffer Not Full Interrupt
- **TCI** Task Completed Interrupt
- **SPI** Special Interrupt

Each of the interrupts is enabled or disabled by a bit in the interrupt mask register. Since OBFI and IBFI are directly connected to the OBFI and IBFI flags, the master can write a new command to the input data bus buffer as soon as the previous command has been read.

The TCI interrupt is useful when the master is sending commands to the 8292. The pending TCI will be cleared with each new command written to the 8292. Commands sent to the 8292 can be divided into two major groups:

1. Commands that require response back from the 8292 to the master, e.g., reading register.
2. Commands that initiate some action or enable features but do not require response back from the 8292, e.g., enable data bus buffer interrupts.

With the first group, the TCI interrupt will be used to indicate that the required response is ready in the data bus buffer and the master may continue and read it. With the second group, the interrupt will be used to indicate completion of the required task, so that the master may send new commands.

The SPI should be used when immediate information or special events is required (see the Interrupt Status Register).

"Polling Status" Based Communication
When interrupt based communication is not desired, all interrupts can be masked by the interrupt mask register. The communication with the 8292 is based upon sequential poll of the interrupt status register. By testing the OBFI and IBFI flags, the data bus buffer status is determined while special events are determined by testing the other bits.

Receiving IFC
The IFC pulse defined by the IEEE-488 standard is at least 100 μsec. In this time, all operation on the bus should be aborted. Most important, the current controller (the one that is in charge at that time) should stop sending ATN or EOI. Thus, IFC must externally gate CIC (controller in charge) and ATNO to ensure that this occurs.

Reset and Power Up Procedure
After the 8292 has been reset either by the external reset pin, the device being powered on; or a RST command, the following sequential events will take place:

1. All outputs to the GPIB interface will go high (SRQ, ATNI, IFC, SYC, CLTH, ATNO, CIC, TCI, SPI, EOI, OBFI, IBFI, DAV, REV).
2. The four interrupt outputs (TCI, SPI, OBFI, IBFI) and CLTH output will go low.
3. The following registers will be cleared:
   - Interrupt Status
   - Interrupt Mask
   - Error Flag
   - Error Mask
   - Time Out
   - Event Counter (= 256), Counter is disabled.
4. If the 8292 is the system controller, an ABORT command will be executed, the 8292 will become the controller in charge, and it will enter the CACS state. If it is not the system controller, it will remain in CIIDS.

System Configuration
The 8291 and 8292 must be interfaced to an IEEE-488 bus meeting a variety of specifications including drive capability and loading characteristics. To interface the 8291 and the 8292 without the 8293's, several external gates are required, using a configuration similar to that used in Figure 5.
NOTES:
1. CONNECT TO NDAC FOR BYTE COUNT OR TO EOI FOR BLOCK COUNT.
2. GATE ENSURES OPEN COLLECTOR OPERATION DURING PARALLEL POLL.

Figure 4. 8291 and 8292 System Configuration
Figure 5. 8291, 8292, and 8293 System Configuration
ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias............0°C to 70°C
Storage Temperature: ..................-65°C to +150°C
Voltage on Any Pin With Respect
  to Ground...........................0.5V to +7V
Power Dissipation.........................1.5 Watt

"NOTICE: Stresses above those listed under ‘‘Absolute Maximum Ratings’’ may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (TA = 0°C to 70°C, VSS = 0V: 8292, VCC = ±5V ±10%)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI1</td>
<td>Input Low Voltage (All Except X1, X2, RESET)</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VI2</td>
<td>Input Low Voltage (X1, X2, RESET)</td>
<td>-0.5</td>
<td>0.6</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VH1</td>
<td>Input High Voltage (All Except X1, X2, RESET)</td>
<td>2.2</td>
<td>VCC</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VH2</td>
<td>Input High Voltage (X1, X2, RESET)</td>
<td>3.8</td>
<td>VCC</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage (D0-D7)</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 2.0 mA</td>
<td></td>
</tr>
<tr>
<td>VOL2</td>
<td>Output Low Voltage (All Other Outputs)</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 1.6 mA</td>
<td></td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage (D0-D7)</td>
<td>2.4</td>
<td>V</td>
<td>IOH = -400 μA</td>
<td></td>
</tr>
<tr>
<td>VOH2</td>
<td>Output High Voltage (All Other Outputs)</td>
<td>2.4</td>
<td>V</td>
<td>IOH = -50 μA</td>
<td></td>
</tr>
<tr>
<td>IL</td>
<td>Input Leakage Current (COUNT, IFCL, RD, WR, CS, A0)</td>
<td>±10</td>
<td>μA</td>
<td>VSS &lt; VIN &lt; VCC</td>
<td></td>
</tr>
<tr>
<td>IOZ</td>
<td>Output Leakage Current (D0-D7, High Z State)</td>
<td>±10</td>
<td>μA</td>
<td>VSS + 0.45 &lt; VIN &lt; VCC</td>
<td></td>
</tr>
<tr>
<td>IL1</td>
<td>Low Input Load Current (Pins 21–24, 27–38)</td>
<td>0.5</td>
<td>mA</td>
<td>VIL = 0.8V</td>
<td></td>
</tr>
<tr>
<td>IL2</td>
<td>Low Input Load Current (RESET)</td>
<td>0.2</td>
<td>mA</td>
<td>VIL = 0.8V</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>Total Supply Current</td>
<td>125</td>
<td>mA</td>
<td>Typical = 65 mA</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>Input High Leakage Current (Pins 21–24, 27–38)</td>
<td>100</td>
<td>μA</td>
<td>VIN = VCC</td>
<td></td>
</tr>
<tr>
<td>CIN</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CI/O</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A.C. CHARACTERISTICS (TA = 0°C to 70°C, VSS = 0V: 8292, VCC = ±5V ±10%)

DBB READ

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tAR</td>
<td>CS, A0 Setup to RD</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tAR</td>
<td>CS, A0 Hold After RD</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tRR</td>
<td>RD Pulse Width</td>
<td>250</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tAD</td>
<td>CS, A0 to Data Out Delay</td>
<td>225</td>
<td>ns</td>
<td>CL = 150 pF</td>
<td></td>
</tr>
<tr>
<td>tRD</td>
<td>RD to Data Out Delay</td>
<td>225</td>
<td>ns</td>
<td>CL = 150 pF</td>
<td></td>
</tr>
<tr>
<td>tDF</td>
<td>RD to Data Float Delay</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tCY</td>
<td>Cycle Time</td>
<td>2.5</td>
<td>15</td>
<td>μs</td>
<td></td>
</tr>
</tbody>
</table>

DBB WRITE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tAW</td>
<td>CS, A0 Setup to WR</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWA</td>
<td>CS, A0 Hold After WR</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWW</td>
<td>WR Pulse Width</td>
<td>250</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tDW</td>
<td>Data Setup to WR</td>
<td>150</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWD</td>
<td>Data Hold After WR</td>
<td>0</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### COMMAND TIMINGS

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Execution Time</th>
<th>IFPI</th>
<th>TC[2]</th>
<th>SPI</th>
<th>ATNO</th>
<th>CIC</th>
<th>IFC</th>
<th>REN</th>
<th>EO1</th>
<th>DAV</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>WOUT</td>
<td>63</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>WEC</td>
<td>63</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>REV</td>
<td>71</td>
<td>24</td>
<td></td>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td>RFR</td>
<td>67</td>
<td>24</td>
<td></td>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td>RINM</td>
<td>69</td>
<td>24</td>
<td></td>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E6</td>
<td>RCST</td>
<td>97</td>
<td>24</td>
<td></td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E7</td>
<td>RBST</td>
<td>92</td>
<td>24</td>
<td></td>
<td>72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E9</td>
<td>RTOUT</td>
<td>69</td>
<td>24</td>
<td></td>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EA</td>
<td>RERM</td>
<td>69</td>
<td>24</td>
<td></td>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F0</td>
<td>SPCTI</td>
<td>53</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Count Stops After 99</td>
</tr>
<tr>
<td>F1</td>
<td>GIOL</td>
<td>88</td>
<td>24</td>
<td></td>
<td>70</td>
<td>161</td>
<td>161</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>RST</td>
<td>94</td>
<td>24</td>
<td></td>
<td>152</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not System Controller</td>
</tr>
<tr>
<td>F3</td>
<td>RST</td>
<td>94</td>
<td>24</td>
<td></td>
<td>192</td>
<td>152</td>
<td>174</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td>System Controller</td>
</tr>
<tr>
<td>F4</td>
<td>RSTI</td>
<td>61</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>GSEC</td>
<td>125</td>
<td>24</td>
<td></td>
<td>198</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>GTSB</td>
<td>118</td>
<td>24</td>
<td></td>
<td>191</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F7</td>
<td>SLOC</td>
<td>73</td>
<td>24</td>
<td></td>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F8</td>
<td>SREM</td>
<td>91</td>
<td>24</td>
<td></td>
<td>73</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>ABORT</td>
<td>155</td>
<td>24</td>
<td></td>
<td>133</td>
<td>120</td>
<td>115</td>
<td>142</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FA</td>
<td>TCNTR</td>
<td>106</td>
<td>24</td>
<td></td>
<td>86</td>
<td>171</td>
<td>168</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC</td>
<td>TCST</td>
<td>92</td>
<td>24</td>
<td></td>
<td>67</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FD</td>
<td>TCSY</td>
<td>115</td>
<td>24</td>
<td></td>
<td>91</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FE</td>
<td>STCN</td>
<td>59</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Starts Count After 43</td>
</tr>
<tr>
<td>PIN</td>
<td>RESET</td>
<td>29</td>
<td>-</td>
<td>17</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not System Controller</td>
</tr>
<tr>
<td>X</td>
<td>IACK</td>
<td>116</td>
<td>-</td>
<td>173</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If Interrupt Pending</td>
</tr>
</tbody>
</table>

Notes:
1. All times are multiples of tCY from the 8041A command interrupt.
2. TCI clears after 7 tCY on all commands.
3. † Indicates a level transition from low to high, ‡ indicates a high to low transition.

### A.C. TESTING INPUT, OUTPUT WAVEFORM

**INPUT/OUTPUT**

2.4

0.45

A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC 1 AND 0.45V FOR A LOGIC 0. TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC 1 AND 0.8V FOR A LOGIC 0.

### A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

C<sub>L</sub> INCLUDES JIG CAPACITANCE
CLOCK DRIVER CIRCUITS

CRystal Oscillator Mode

CRYSTAL SERIES RESISTANCE SHOULD BE <15Ω AT 6 MHz; <18Ω AT 3.6 MHz

Driving from External Source

BOTH XTAL1 AND XTAL2 SHOULD BE DRIVEN RESISTORS TO VCC ARE NEEDED TO ENSURE VIM = 3.3V IF TTL CIRCUITRY IS USED.

LC Oscillator Mode

LC Oscillator Mode - Diagram

EACH C SHOULD BE APPROXIMATELY 20 pF, INCLUDING STRAY CAPACITANCE

Waveforms

READ OPERATION—DATA BUS BUFFER REGISTER

WRITE OPERATION — DATA BUS BUFFER REGISTER
APPENDIX

The following tables and state diagrams were taken from the IEEE Standard Digital Interface for Programmable Instrumentation, IEEE Std. 488-1978. This document is the official standard for the GPIB bus and can be purchased from IEEE, 345 East 47th St., New York, NY 10017.

**CMNEMONICS**

<table>
<thead>
<tr>
<th>Messages</th>
<th>Interface States</th>
</tr>
</thead>
<tbody>
<tr>
<td>pon = power on</td>
<td>CIDS = controller idle state</td>
</tr>
<tr>
<td>rsc = request system control</td>
<td>CADS = controller addressed state</td>
</tr>
<tr>
<td>rpp = request parallel poll</td>
<td>CTRS = controller transfer state</td>
</tr>
<tr>
<td>gts = go to standby</td>
<td>CACS = controller active state</td>
</tr>
<tr>
<td>tca = take control asynchronously</td>
<td>CPWS = controller parallel poll wait state</td>
</tr>
<tr>
<td>tcs = take control synchronously</td>
<td>CPPS = controller parallel poll state</td>
</tr>
<tr>
<td>sic = send interface clear</td>
<td>CSBS = controller standby state</td>
</tr>
<tr>
<td>sre = send remote enable</td>
<td>CSHS = controller standby hold state</td>
</tr>
<tr>
<td>IFC = interface clear</td>
<td>CAWS = controller active wait state</td>
</tr>
<tr>
<td>ATN = attention</td>
<td>CSWS = controller synchronous wait state</td>
</tr>
<tr>
<td>TCT = take control</td>
<td>CSRS = controller service requested state</td>
</tr>
<tr>
<td></td>
<td>CSNS = controller service not requested state</td>
</tr>
<tr>
<td></td>
<td>SNAS = system control not active state</td>
</tr>
<tr>
<td></td>
<td>SACS = system control active state</td>
</tr>
<tr>
<td></td>
<td>SRIS = system control remote enable idle state</td>
</tr>
<tr>
<td></td>
<td>SRNS = system control remote enable not active state</td>
</tr>
<tr>
<td></td>
<td>SRAS = system control remote enable active state</td>
</tr>
<tr>
<td></td>
<td>SIIS = system control interface clear idle state</td>
</tr>
<tr>
<td></td>
<td>SINS = system control interface clear not active state</td>
</tr>
<tr>
<td></td>
<td>SIAS = system control interface clear active state</td>
</tr>
<tr>
<td></td>
<td>(ACDS) = accept data state (AH function)</td>
</tr>
<tr>
<td></td>
<td>(ANRS) = acceptor not ready state (AH function)</td>
</tr>
<tr>
<td></td>
<td>(SDYS) = source delay state (SH function)</td>
</tr>
<tr>
<td></td>
<td>(STRS) = source transfer state (SH function)</td>
</tr>
<tr>
<td></td>
<td>(TADS) = talker addressed state (T function)</td>
</tr>
</tbody>
</table>

* T0 > 1.8 µsec
† THE MICROPROCESSOR MUST WAIT FOR THE 80 INTERRUPT BEFORE WRITING THE OTI8 OR OSEC COMMANDS TO ENSURE THAT (STRS > 0) IS TRUE.

![Figure A.1. C State Diagram](231322-001)

6-427
### REMOTE MESSAGE CODING

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Message Name</th>
<th>C</th>
<th>L</th>
<th>A</th>
<th>I</th>
<th>O</th>
<th>D</th>
<th>NN</th>
<th>I</th>
<th>D</th>
<th>R</th>
<th>A</th>
<th>E</th>
<th>S</th>
<th>I</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACG</td>
<td>Addressed Command Group</td>
<td>M</td>
<td>AC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>XXX</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATN</td>
<td>Attention</td>
<td>U</td>
<td>UC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAB</td>
<td>Data Byte has been received</td>
<td>M</td>
<td>DD</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>DAC</td>
<td>Data Accepted</td>
<td>U</td>
<td>HS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAV</td>
<td>Data Valid</td>
<td>U</td>
<td>HS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCL</td>
<td>Device Clear</td>
<td>M</td>
<td>UC</td>
<td>Y</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td>End of string</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOS</td>
<td>End of String</td>
<td>M</td>
<td>DD</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>GET</td>
<td>Group Execute Trigger</td>
<td>M</td>
<td>AC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTL</td>
<td>Go to Local</td>
<td>M</td>
<td>AC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDY</td>
<td>Identify</td>
<td>U</td>
<td>UC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFC</td>
<td>Interface Clear</td>
<td>U</td>
<td>UC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAG</td>
<td>Listen Address Group</td>
<td>M</td>
<td>AD</td>
<td>Y</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLO</td>
<td>Local Lock Out</td>
<td>M</td>
<td>UC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MLA</td>
<td>My Listen Address</td>
<td>M</td>
<td>AD</td>
<td>Y</td>
<td>0</td>
<td>1</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTA</td>
<td>My Talk Address</td>
<td>M</td>
<td>AD</td>
<td>Y</td>
<td>1</td>
<td>0</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSA</td>
<td>My Secondary Address</td>
<td>M</td>
<td>SE</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUL</td>
<td>Null Byte</td>
<td>M</td>
<td>DD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSA</td>
<td>Other Secondary Address</td>
<td>M</td>
<td>SE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OTA</td>
<td>Other Talk Address</td>
<td>M</td>
<td>AD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCG</td>
<td>Primary Command Group</td>
<td>M</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PPC</td>
<td>Parallel Poll Configure</td>
<td>M</td>
<td>AC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PPE</td>
<td>Parallel Poll Enable</td>
<td>M</td>
<td>SE</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>S</td>
<td>P</td>
<td>P</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PPD</td>
<td>Parallel Poll Disable</td>
<td>M</td>
<td>SE</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PPR1</td>
<td>Parallel Poll Response 1</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PPR2</td>
<td>Parallel Poll Response 2</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PPR3</td>
<td>Parallel Poll Response 3</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PPR4</td>
<td>Parallel Poll Response 4</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PPR5</td>
<td>Parallel Poll Response 5</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PPR6</td>
<td>Parallel Poll Response 6</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PPR7</td>
<td>Parallel Poll Response 7</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PPR8</td>
<td>Parallel Poll Response 8</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PPU</td>
<td>Parallel Poll Unconfigure</td>
<td>M</td>
<td>UC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>REN</td>
<td>Remote Enable</td>
<td>U</td>
<td>UC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDF</td>
<td>Ready for Data</td>
<td>U</td>
<td>HS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQS</td>
<td>Request Service</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCG</td>
<td>Secondary Command Group</td>
<td>M</td>
<td>SE</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDC</td>
<td>Selected Device Clear</td>
<td>M</td>
<td>AC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SPD</td>
<td>Serial Poll Disable</td>
<td>M</td>
<td>UC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SPE</td>
<td>Serial Poll Enable</td>
<td>M</td>
<td>UC</td>
<td>Y</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRQ</td>
<td>Service Request</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STB</td>
<td>Status Byte</td>
<td>M</td>
<td>ST</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>XXX</td>
<td>1</td>
<td>XXX</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

The 1/0 coding on ATN when sent concurrent with multiline messages has been added to this revision for interpretive convenience.
NOTES:
1. D1–D8 specify the device dependent data bits.
2. E1–E8 specify the device dependent code used to indicate the EOS message.
3. L1–L5 specify the device dependent bits of the device’s listen address.
4. T1–T5 specify the device dependent bits of the device’s talk address.
5. S1–S5 specify the device dependent bits of the device’s secondary address.
6. S specifies the sense of the PPR.
   Response = S 0 0 0 0
P1–P3 specify the PPR message to be sent when a parallel poll is executed.

<table>
<thead>
<tr>
<th>P3</th>
<th>P2</th>
<th>P1</th>
<th>PPR Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>PPR1</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>PPR8</td>
</tr>
</tbody>
</table>

7. D1–D4 specify don’t-care bits that shall not be decoded by the receiving device. It is recommended that all zeroes be sent.
8. S1–S6, S8 specify the device dependent status. (DIO7 is used for the RQS message.)
9. The source of the message on the ATN line is always the C function, whereas the messages on the DIO and EOI lines are enabled by the T function.
10. The source of the messages on the ATN and EOI lines is always the C function, whereas the source of the messages on the DIO lines is always the PP function.
11. This code is provided for system use, see 6.3.
The Intel® 8294A Data Encryption Unit (DEU) is a microprocessor peripheral device designed to encrypt and decrypt 64-bit blocks of data using the algorithm specified in the Federal Information Processing Data Encryption Standard. The DEU operates on 64-bit text words using a 56-bit user-specified key to produce 64-bit cipher words. The operation is reversible: if the cipher word is operated upon, the original text word is produced. The algorithm itself is permanently contained in the 8294A; however, the 56-bit key is user-defined and may be changed at any time.

The 56-bit key and 64-bit message data are transferred to and from the 8294A in 8-bit bytes by way of the system data bus. A DMA interface and three interrupt outputs are available to minimize software overhead associated with data transfer. Also, by using the DMA interface two or more DEUs may be operated in parallel to achieve effective system conversion rates which are virtually any multiple of 400 bytes/second. The 8294A also has a 7-bit TTL compatible output port for user-specified functions.

Because the 8294A implements the NBS encryption algorithm it can be used in a variety of Electronic Funds Transfer applications as well as other electronic banking and data handling applications where data must be encrypted.

Figure 1. Block Diagram

Figure 2. Pin Configuration
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC</td>
<td>1</td>
<td>No Connection.</td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td>2</td>
<td>I</td>
<td>Crystal: Inputs for crystal, L-C or external timing signal to determine internal oscillator frequency</td>
</tr>
<tr>
<td>X2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>4</td>
<td>I</td>
<td>Reset: A low signal to this pin resets the 8294A.</td>
</tr>
<tr>
<td>VCC</td>
<td>5</td>
<td></td>
<td>Power: Tied high.</td>
</tr>
<tr>
<td>CS</td>
<td>6</td>
<td>I</td>
<td>Chip Select: A low signal to this pin enables the CPU to read data and status from the internal DEU registers</td>
</tr>
<tr>
<td>GND</td>
<td>7</td>
<td></td>
<td>Ground: This pin must be tied to ground.</td>
</tr>
<tr>
<td>RD</td>
<td>8</td>
<td>I</td>
<td>Read: An active low read strobe at this pin enables the CPU to send data and commands to the DEU.</td>
</tr>
<tr>
<td>A0</td>
<td>9</td>
<td>I</td>
<td>Address: Address input used by the CPU to select DEU registers during read and write operations.</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>I</td>
<td>Write: An active low write strobe at this pin enables the CPU to send data and commands to the DEU.</td>
</tr>
<tr>
<td>SYNC</td>
<td>11</td>
<td>O</td>
<td>Sync: High frequency (Clock ~ 15) output. Can be used as a strobe for external circuitry</td>
</tr>
<tr>
<td>D0</td>
<td>12</td>
<td>I/O</td>
<td>Data Bus: Three-state, bi-directional data bus lines used to transfer data between the CPU and the 8294A.</td>
</tr>
<tr>
<td>D1</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D5</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D6</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D7</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>20</td>
<td></td>
<td>Ground: This pin must be tied to ground.</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td></td>
<td>Power: +5 volt power input. +5V ± 10%.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC</td>
<td>39</td>
<td></td>
<td>No Connection.</td>
</tr>
<tr>
<td>DACK</td>
<td>38</td>
<td>I</td>
<td>DMA Acknowledge: Input signal from the 8257 DMA Controller acknowledging that the requested DMA cycle has been granted</td>
</tr>
<tr>
<td>DRQ</td>
<td>37</td>
<td>O</td>
<td>DMA Request: Output signal to the 8257 DMA Controller requesting a DMA cycle.</td>
</tr>
<tr>
<td>SRQ</td>
<td>36</td>
<td>O</td>
<td>Service Request: Interrupt to the CPU indicating that the 8294A is awaiting data or commands at the input buffer. SRQ=1 implies IBF=0.</td>
</tr>
<tr>
<td>OAV</td>
<td>35</td>
<td>O</td>
<td>Output Available: Interrupt to the CPU indicating that the 8294A has data or status available in its output buffer. OAV=1 implies OBF=1.</td>
</tr>
<tr>
<td>NC</td>
<td>34</td>
<td></td>
<td>No Connection.</td>
</tr>
<tr>
<td>P6</td>
<td>33</td>
<td>O</td>
<td>Output Port: User output port lines</td>
</tr>
<tr>
<td>P5</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P0</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD</td>
<td>26</td>
<td></td>
<td>Power: +5V power input (-5V ±10%) Low power standby pin</td>
</tr>
<tr>
<td>VCC</td>
<td>25</td>
<td></td>
<td>Power: Tied high.</td>
</tr>
<tr>
<td>CCMP</td>
<td>24</td>
<td>O</td>
<td>Conversion Complete: Interrupt to the CPU indicating that the encryption/decryption of an 8-byte block is complete</td>
</tr>
<tr>
<td>NC</td>
<td>23</td>
<td></td>
<td>No Connection.</td>
</tr>
<tr>
<td>NC</td>
<td>22</td>
<td></td>
<td>No Connection.</td>
</tr>
<tr>
<td>NC</td>
<td>21</td>
<td></td>
<td>No Connection.</td>
</tr>
</tbody>
</table>
FUNCTIONAL DESCRIPTION

OPERATION

The data conversion sequence is as follows:
1. A Set Mode command is given, enabling the desired interrupt outputs.
2. An Enter New Key command is issued, followed by 8 data inputs which are retained by the DEU for encryption/decryption. Each byte must have odd parity.
3. An Encrypt Data or Decrypt Data command sets the DEU in the desired mode.

After this, data conversions are made by writing 8 data bytes and then reading back 8 converted data bytes. Any of the above commands may be issued between data conversions to change the basic operation of the DEU; e.g., a Decrypt Data command could be issued to change the DEU from encrypt mode to decrypt mode without changing either the key or the interrupt outputs enabled.

INTERNAL DEU REGISTERS

Four internal registers are addressable by the master processor: 2 for input, and 2 for output. The following table describes how these registers are accessed.

<table>
<thead>
<tr>
<th>RD</th>
<th>WR</th>
<th>CS</th>
<th>A₀</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Data input buffer</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Data output buffer</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Command input buffer</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Status output buffer</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>Don't care</td>
</tr>
</tbody>
</table>

The functions of each of these registers are described below.

Data Input Buffer — Data written to this register is interpreted in one of three ways, depending on the preceding command sequence.
1. Part of a key.
2. Data to be encrypted or decrypted.
3. A DMA block count.

Data Output Buffer — Data read from this register is the output of the encryption/decryption operation.

Command Input Buffer — Commands to the DEU are written into this register. (See command summary below.)

Status Output Buffer — DEU status is available in this register at all times. It is used by the processor for poll-driven command and data transfer operations.

<table>
<thead>
<tr>
<th>STATUS BIT:</th>
<th>FUNCTION:</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>X X X KPE CF DEC IBF OBF</td>
</tr>
</tbody>
</table>

IBF Input Buffer Full; A write to the Data Input Buffer or to the Command Input Buffer sets IBF = 1. The DEU resets this flag when it has accepted the input byte. Nothing should be written when IBF = 1.

DEC Decrypt; indicates whether the DEU is in an encrypt or a decrypt mode. DEC = 1 implies the decrypt mode. DEC = 0 implies the encrypt mode.

After 8294A has accepted a 'Decrypt Data' or 'Encrypt Data' command, 11 cycles are required to update the DEC bit.

CF Completion Flag; This flag may be used to indicate any or all of three events in the data transfer protocol.
1. It may be used in lieu of a counter in the processor routine to flag the end of an 8-byte transfer.
2. It must be used to indicate the validity of the KPE flag.
3. It may be used in lieu of the CCMP interrupt to indicate the completion of a DMA operation.

KPE Key Parity Error; After a new key has been entered, the DEU uses this flag in conjunction with the CF flag to indicate correct or incorrect parity.

COMMAND SUMMARY

1 — Enter New Key

OP CODE:

```
0 1 0 0 0 0 0
```

This command is followed by 8 data byte inputs which are retained in the key buffer (RAM) to be used in encrypting and decrypting data. These data bytes must have odd parity represented by the LSB.

2 — Encrypt Data

OP CODE:

```
0 0 1 1 0 0 0 0
```

This command puts the 8294A into the encrypt mode.

3 — Decrypt Data

OP CODE:

```
0 0 1 0 0 0 0 0
```

This command puts the 8294A into the decrypt mode.

4 — Set Mode

OP CODE:

```
0 0 0 0 A B C D
```

where:
A is the OAV (Output Available) interrupt enable
B is the SRQ (Service Request) interrupt enable
C is the DMA (Direct Memory Access) transfer enable
D is the CCMP (Conversion Complete) interrupt enable
This command determines which interrupt outputs will be enabled. A "1" in bits A, B, or D will enable the OAV, SRQ, or CCMP interrupts respectively. A "1" in bit C will allow DMA transfers. When bit C is set the OAV and SRQ interrupts should also be enabled (bits A, B = 1). Following the command in which bit C, the DMA bit, is set, the 8294 will expect one data byte to specify the number of 8-byte blocks to be converted using DMA.

5 — Write to Output Port

This command causes the 7 least significant bits of the command byte to be latched as output data on the 8294 output port. The initial output data is 1111111. Use of this port is independent of the encryption/decryption function

PROCESSOR/DEU INTERFACE PROTOCOL
ENTERING A NEW KEY

The timing sequence for entering a new key is shown in Figure 3. A flowchart showing the CPU software to accommodate this sequence is given in Figure 4.

After the Enter New Key command is issued, 8 data bytes representing the new key are written to the data input buffer (most significant byte first). After the eighth byte is entered into the DEU, CF goes true (CF=1). The CF bit goes false again when KPE is valid. The CPU can then check the KPE flag. If KPE=1, a parity error has been detected and the DEU has not accepted the key. Each byte is checked for odd parity, where the parity bit is the LSB of each byte.

Since CF=1 only for a short period of time after the last byte is accepted, the CPU which polls the CF flag might miss detecting CF=1 momentarily. Thus, a counter should be used, as in Figure 4, to flag the end of the new key entry. Then CF is used to indicate a valid KPE flag.

Figure 3. Entering a New Key

Figure 4. Flowchart for Entering a New Key
ENCRYPTING OR DECRYPTING DATA

Figure 5 shows the timing sequence for encrypting or decrypting data. The CPU writes 8 data bytes to the DEU's data input buffer for encryption/decryption. CF then goes true (CF = 1) to indicate that the DEU has accepted the 8-byte block. Thus, the CPU may test for IBF = 0 and CF = 1 to terminate the input mode, or it may use a software counter. When the encryption/decryption is complete, the CCMP and OAV interrupts are asserted and the OBF flag is set true (OBF = 1). OAV and OBF are set false again after each of the converted data bytes is read back by the CPU. The CCMP interrupt is set false, and remains false, after the first read. After 8 bytes have been read back by the CPU, CF goes false (CF = 0). Thus, the CPU may test for CF = 0 to terminate the read mode. Also, the CCMP interrupt may be used to initiate a service routine which performs the next series of 8 data reads and 8 data writes.

Figure 5. Encrypting/Decrypting Data

Figure 6 offers two flowcharts outlining the alternative means of implementing the data conversion protocol. Either the CF flag or a software counter may be used to end the read and write modes.

SRO = 1 implies IBF = 0, OAV = 1 implies OBF = 1. This allows interrupt routines to do data transfers without checking status first. However, the OAV service routine must detect and flag the end of a data conversion.

Figure 6. Data Conversion Flowcharts
USING DMA

The timing sequence for data conversions using DMA is shown in Figure 7. This sequence can be better understood when considered in conjunction with the hardware DMA interface in Figure 8. Note that the use of the DMA feature requires 3 external AND gates and 2 DMA channels (one for input, one for output). Since the DEU has only one DMA request pin, the SRQ and OAV outputs are used in conjunction with two of the AND gates to create separate DMA request outputs for the 2 DMA channels. The third AND gate combines the two active-low DACK inputs.

To initiate a DMA transfer, the CPU must first initialize the two DMA channels as shown in the flowchart in Figure 9. It must then issue a Set Mode command to the DEU enabling the OAV, SRQ, and DMA outputs. The CCMP interrupt may be enabled or disabled, depending on whether that output is desired. Following the Set Mode command, there must be a data byte giving the number of 8-byte blocks of data (n<256) to be converted. The DEU then generates the required number of DMA requests to the 2 DMA channels with no further CPU intervention. When the requested number of blocks has been converted, the DEU will set CF and assert the CCMP interrupt (if enabled). CCMP then goes false again with the next write to the DEU (command or data). Upon completion of the conversion, the DMA mode is disabled and the DEU returns to the encrypt/decrypt mode. The enabled interrupt outputs, however, will remain enabled until another Set Mode command is issued.

SINGLE BYTE COMMANDS

Figure 10 shows the timing and protocol for single byte commands. Note that any of the commands is effective as a pacify command in that they may be entered at any time, except during a DMA conversion. The DEU is thus set to a known state. However, if a command is issued out of sequence, an additional protocol is required (Figure 11). The CPU must wait until the command is accepted (IBF = 0). A data read must then be issued to clear anything the preceding command sequence may have left in the Data Output Buffer.
CPU/DEU INTERFACES

Figures 12 through 15 illustrate four interface configurations used in the CPU/DEU data transfers. In all cases SRQ will be true (if enabled) and IBF will be false when the DEU is ready to accept data or commands.

![Diagram of Polling Interface](image)

**Figure 12. Polling Interface**

![Diagram of Single Byte Commands](image)

**Figure 10. Single Byte Commands**

![Diagram of Pacify Protocol](image)

**Figure 11. Pacify Protocol**

![Diagram of Single Interrupt Interface](image)

**Figure 13. Single Interrupt Interface**

![Diagram of Dual Interrupt Interface](image)

**Figure 14. Dual Interrupt Interface**
OSCILLATOR AND TIMING CIRCUITS

The 8294A's internal timing generation is controlled by a self-contained oscillator and timing circuit. A choice of crystal, L-C or external clock can be used to derive the basic oscillator frequency.

The resident timing circuit consists of an oscillator, a state counter and a cycle counter as illustrated in Figure 16.

Figure 15. DMA Interface

Figure 16. Oscillator Configuration
DRIVING FROM EXTERNAL SOURCE—TWO OPTIONS

FOR THE 8294A XTAL2 MUST BE HIGH
35-65% OF THE PERIOD
RISE AND FALL TIMES MUST
NOT EXCEED 10 ns
RESISTOR TO VCC IS NEEDED
TO ENSURE VIN = 3.0V IF TTL
CIRCUITRY IS USED

ABSOLUTE MAXIMUM RATINGS*  

Ambient Temperature Under Bias ...... 0°C to 70°C  
Storage Temperature ...... −65°C to + 150°C  
Voltage on Any Pin With Respect to Ground ...... −0.5V to + 7V  
Power Dissipation ......................... 1.5 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. AND OPERATING CHARACTERISTICS (Ta = 0°C to 70°C, VCC = −5V ± 10%, VSS = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
<td>Typ.</td>
</tr>
<tr>
<td>VIL</td>
<td>Input Low Voltage (All Except X1, X2, RESET)</td>
<td>−0.5</td>
<td>0.8</td>
</tr>
<tr>
<td>VIL1</td>
<td>Input Low Voltage (X1, X2, RESET)</td>
<td>−0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage (All Except X1, RESET)</td>
<td>2.0</td>
<td>VCC</td>
</tr>
<tr>
<td>VIH1</td>
<td>Input High Voltage (X1, RESET)</td>
<td>3.5</td>
<td>VCC</td>
</tr>
<tr>
<td>VIH2</td>
<td>Input High Voltage (X2)</td>
<td>2.2</td>
<td>VCC</td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage (D0-D7)</td>
<td>0.45</td>
<td>V</td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage (All Other Outputs)</td>
<td>0.45</td>
<td>V</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage (D0-D7)</td>
<td>2.4</td>
<td>V</td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage (All Other Outputs)</td>
<td>2.4</td>
<td>V</td>
</tr>
<tr>
<td>IIL</td>
<td>Input Leakage Current (RD, WR, CS, A0)</td>
<td>± 10</td>
<td>μA</td>
</tr>
<tr>
<td>IOFL</td>
<td>Output Leakage Current (D0-D7, High Z State)</td>
<td>± 10</td>
<td>μA</td>
</tr>
<tr>
<td>IDD</td>
<td>VDD Supply Current</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>IDD + ICC</td>
<td>Total Supply Current</td>
<td>60</td>
<td>135</td>
</tr>
<tr>
<td>ILI</td>
<td>Low Input Load Current (Pins 24, 27-38)</td>
<td>0.3</td>
<td>mA</td>
</tr>
<tr>
<td>IIL1</td>
<td>Low Input Load Current (RESET)</td>
<td>0.2</td>
<td>mA</td>
</tr>
<tr>
<td>IH</td>
<td>Input High Leakage Current (Pins 24, 27-38)</td>
<td>100</td>
<td>μA</td>
</tr>
<tr>
<td>CIN</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
</tr>
<tr>
<td>CIO</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
</tr>
</tbody>
</table>

6-438
A.C. CHARACTERISTICS  \((T_A = 0^\circ C \text{ to } 70^\circ C, V_{CC} = V_{DD} = -5V \pm 10\%, V_{SS} = 0V)\)

### DBB READ

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_{AR})</td>
<td>CS, A0 Setup to RD ↓</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{RA})</td>
<td>CS, A0 Hold After RD ↑</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{RR})</td>
<td>RD Pulse Width</td>
<td>160</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{AD})</td>
<td>CS, A0 to Data Out Delay</td>
<td></td>
<td>130</td>
<td>ns</td>
<td>(C_L = 100, pF)</td>
</tr>
<tr>
<td>(t_{RD})</td>
<td>RD ↓ to Data Out Delay</td>
<td></td>
<td>130</td>
<td>ns</td>
<td>(C_L = 100, pF)</td>
</tr>
<tr>
<td>(t_{DF})</td>
<td>RD ↑ to Data Float Delay</td>
<td></td>
<td>85</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{CY})</td>
<td>Cycle Time</td>
<td>1.25</td>
<td>15</td>
<td>(\mu s)</td>
<td>1–12 MHz Crystal</td>
</tr>
</tbody>
</table>

### DBB WRITE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_{AW})</td>
<td>CS, A0 Setup to WR ↓</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{WA})</td>
<td>CS, A0 Hold After WR ↑</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{WW})</td>
<td>WR Pulse Width</td>
<td>160</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{DW})</td>
<td>Data Setup to WR ↑</td>
<td>130</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{WD})</td>
<td>Data Hold to WR ↑</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### DMA AND INTERRUPT TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_{ACC})</td>
<td>DACK Setup to Control</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{CAC})</td>
<td>DACK Hold After Control</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{ACD})</td>
<td>DACK to Data Valid</td>
<td>130</td>
<td></td>
<td>ns</td>
<td>(C_L = 100, pF)</td>
</tr>
<tr>
<td>(t_{CRO})</td>
<td>Control L.E. to DRQ T.E.</td>
<td>110</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{CI})</td>
<td>Control T.E. to Interrupt T.E.</td>
<td>400</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### CLOCK

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8042</th>
<th>8742</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>(t_{CY})</td>
<td>Cycle Time</td>
<td>1.25</td>
<td>9.20</td>
<td>1.25</td>
</tr>
<tr>
<td>(t_{CYC})</td>
<td>Clock Period</td>
<td>83.3</td>
<td>613</td>
<td>83.3</td>
</tr>
<tr>
<td>(t_{FPHH})</td>
<td>Clock High Time</td>
<td>33</td>
<td>38</td>
<td>33</td>
</tr>
<tr>
<td>(t_{FPLL})</td>
<td>Clock Low Time</td>
<td>33</td>
<td>38</td>
<td>33</td>
</tr>
<tr>
<td>(t_{FR})</td>
<td>Clock Rise Time</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>(t_{F})</td>
<td>Clock Fall Time</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

### NOTES:
1. \(t_{CY} = 15/f(XTAL)\)

### A.C. TESTING INPUT, OUTPUT WAVEFORM
CLOCK TIMING
Using The 8292 GPIB Controller

 Peripheral Components Applications
Using the 8292 GPIB Controller

Contents

INTRODUCTION

GPIB/IEEE 488 OVERVIEW

HARDWARE ASPECTS OF THE SYSTEM
  8291 Talker/Listener
  8292 Controller
  8293 Bus Transceivers
  ZT7488/18 GPIB Controller

8292 COMMAND DESCRIPTION

SOFTWARE DRIVER OUTLINE
  Initialization
  Talker/Listener
    Send Data
    Receive Data
    Transfer Data
  Controller
    Trigger
    Device Clear
    Serial Poll
    Parallel Poll
    Pass Control
    Receive Control
    Service Request
  System Controller
    Remote
    Local
    Interface Clear/Abort

INTERRUPT AND DMA CONSIDERATIONS

APPLICATION EXAMPLE

CONCLUSION

APPENDIX A
  Source Listings

APPENDIX B
  Test Cases for the Software Drivers

APPENDIX C
  Remote Message Coding
INTRODUCTION

The Intel® 8292 is a preprogrammed UPI™-41A that implements the Controller function of the IEEE Std 488-1978 (GPIB, HP-IB, IEC Bus, etc.). In order to function the 8292 must be used with the 8291 Talker/Listener and suitable interface and transceiver logic such as a pair of Intel 8293s. In this configuration the system has the potential to be a complete GPIB Controller when driven by the appropriate software. It has the following capabilities: System Controller, send IFC and Take Charge, send REN, Respond to SRQ, send Interface messages, Receive Control, Pass Control, Parallel Poll and Take Control Synchronously.

This application note will explain the 8292 only in the system context of an 8292, 8291, two 8293s and the driver software. If the reader wishes to learn more about the UPI-41A aspects of the 8292, Intel's Application Note AP-41 describes the hardware features and programming characteristics of the device. Additional information on the 8291 may be obtained in the data sheet. The 8293 is detailed in its data sheet. Both chips will be covered here in the details that relate to the GPIB controller.

The next section of this application note presents an overview of the GPIB in a tutorial, but comprehensive nature. The knowledgeable reader may wish to skip this section; however, certain basic semantic concepts introduced there will be used throughout this note.

Additional sections cover the view of the 8292 from the CPU's data bus, the interaction of the 3 chip types (8291, 8292, 8293), the 8292's software protocol and the system level hardware/software protocol. A brief description of interrupts and DMA will be followed by an application example. Appendix A contains the source code for the system driver software.

GPIB/IEEE 488 OVERVIEW

DESIGN OBJECTIVES

What is the IEEE 488 (GPIB)?

The experience of designing systems for a variety of applications in the early 1970's caused Hewlett-Packard to define a standard intercommunication mechanism which would allow them to easily assemble instrumentation systems of varying degrees of complexity. In a typical situation each instrument designer designed his/her own interface from scratch. Each one was inconsistent in terms of electrical levels, pin-outs on a connector, and types of connectors. Every time they built a system they had to invent new cables and new documentation just to specify the cabling and interconnection procedures.

Based on this experience, Hewlett-Packard began to define a new interconnection scheme. They went further than that, however, for they wanted to specify the typical communication protocol for systems of instruments. So in 1972, Hewlett-Packard came out with the first version of the bus which since has been modified and standardized by a committee of several manufacturers, coordinated through the IEEE, to perfect what is now known as the IEEE 488 Interface Bus (also known as the HP-IB, the GPIB and the IEC bus). While this bus specification may not be perfect, it is a good compromise of the various desires and goals of instrumentation and computer peripheral manufacturers to produce a common interconnection mechanism. It fits most instrumentation systems in use today and also fits very well the microcomputer I/O bus requirements. The basic design objectives for the GPIB were to:

1. Specify a system that is easy to use, but has all of the terminology and the definitions related to that system precisely spelled out so that everyone uses the same language when discussing the GPIB.
2. Define all of the mechanical, electrical, and functional interface requirements of a system, yet not define any of the device aspects (they are left up to the instrument designer).
3. Permit a wide range of capabilities of instruments and computer peripherals to use a system simultaneously and not degrade each other's performance.
4. Allow different manufacturers' equipment to be connected together and work together on the same bus.
5. Define a system that is good for limited distance interconnections.
6. Define a system with minimum restrictions on performance of the devices.
7. Define a bus that allows asynchronous communication with a wide range of data rates.
8. Define a low cost system that does not require extensive and elaborate interface logic for the low cost instruments, yet provides higher capability for the higher cost instruments if desired.
9. Allow systems to exist that do not need a central controller; that is, communication directly from one instrument to another is possible.

Although the GPIB was originally designed for instrumentation systems, it became obvious that most of these systems would be controlled by a calculator or computer. With this in mind several modifications were made to the original proposal before its final adoption as an international standard. Figure 1 lists the salient characteristics of the...
APPLICATIONS

GPIB as both an instrumentation bus and as a computer I/O bus.

<table>
<thead>
<tr>
<th>Data Rate</th>
<th>1M bytes/s, max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>250k bytes/s, typ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiple Devices</th>
<th>15 devices, max (electrical limit)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 devices, typ (interrupt flexibility)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bus Length</th>
<th>20 m, max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 m/device, typ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte Oriented</th>
<th>8-bit commands</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8-bit data</td>
</tr>
</tbody>
</table>

| Block Multiplexed | Optimum strategy on GPIB due to setup overhead for commands |

<table>
<thead>
<tr>
<th>Interrupt Driven</th>
<th>Serial poll (slower devices)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Parallel poll (faster devices)</td>
</tr>
</tbody>
</table>

| Direct Memory Access | One DMA facility at controller serves all devices on bus |

<table>
<thead>
<tr>
<th>Asynchronous</th>
<th>One talker</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Multiple listeners</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I/O to I/O Transfers</th>
<th>3-wire handshake</th>
</tr>
</thead>
<tbody>
<tr>
<td>include microcomputer/controller</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 1. Major Characteristics of GPIB as Microcomputer I/O Bus**

The bus can be best understood by examining each of these characteristics from the viewpoint of a general microcomputer I/O bus.

**Data Rate** — Most microcomputer systems utilize peripherals of differing operational rates, such as floppy discs at 31k or 62k bytes/s (single or double density), tape cassettes at 5k to 10k bytes/s, and cartridge tapes at 40k to 80k bytes/s. In general, the only devices that need high speed I/O are 0.5" (1.3-cm) magnetic tapes and hard discs, operational at 30k to 781k bytes/s, respectively. Certainly, the 250k-bytes/s data rate that can be easily achieved by the IEEE 488 bus is sufficient for microcomputers and their peripherals, and is more than needed for typical analog instruments that take only a few readings per second. The 1M-byte/s maximum data rate is not easily achieved on the GPIB and requires special attention to considerations beyond the scope of this note. Although not required, data buffering in each device will improve the overall bus performance and allow utilization of more of the bus bandwidth.

**Multiple Devices** — Many microcomputer systems used as computers (not as components) service from three to seven peripherals. With the GPIB, up to 8 devices can be handled easily by 1 controller; with some slowdown in interrupt handling, up to 15 devices can work together. The limit of 8 is imposed by the number of unique parallel poll responses available; the limit of 15 is set by the electrical drive characteristics of the bus. Logically, the IEEE 488 Standard is capable of accommodating more device addresses (31 primary, each potentially with 31 secondaries).

**Bus Length** — Physically, the majority of microcomputer systems fit easily on a desk top or in a standard 19" (48-cm) rack, eliminating the need for extra long cables. The GPIB is designed typically to have 2 m of length per device, which accommodates most systems. A line printer might require greater cable lengths, but this can be handled at the lower speeds involved by using extra dummy terminations.

**Byte Oriented** — The 8-bit byte is almost universal in I/O applications; even 16-bit and 32-bit computers use byte transfers for most peripherals. The 8-bit byte matches the ASCII code for characters and is an integral submultiple of most computer word sizes. The GPIB has an 8-bit wide data path that may be used to transfer ASCII or binary data, as well as the necessary status and control bytes.

**Block Multiplexed** — Many peripherals are block oriented or are used in a block mode. Bytes are transferred in a fixed or variable length group; then there is a wait before another group is sent to that device, e.g., one sector of a floppy disc, one line on a printer or tape punch, etc. The GPIB is, by nature, a block multiplexed bus due to the overhead involved in addressing various devices to talk and listen. This overhead is less bothersome if it only occurs once for a large number of data bytes (once per block). This mode of operation matches the needs of microcomputers and most of their peripherals. Because of block multiplexing, the bus works best with buffered memory devices.

**Interrupt Driven** — Many types of interrupt systems exist, ranging from complex, fast, vectored/priority networks to simple polling schemes. The main tradeoff is usually cost versus speed of response. The GPIB has two interrupt protocols to help span the range of applications. The first is a single service request (SRQ) line that may be asserted by all interrupting devices. The controller then polls all devices to find out which wants service. The polling mechanism is well defined and can be easily
Automated. For higher performance, the parallel poll capability in the IEEE 488 allows up to eight devices to be polled at once — each device is assigned to one bit of the data bus. This mechanism provides fast recognition of an interrupting device. A drawback is the frequent need for the controller to explicitly conduct a parallel poll, since there is no equivalent of the SRQ line for this mode.

Direct Memory Access (DMA) — In many applications, no immediate processing of I/O data on a byte-by-byte basis is needed or wanted. In fact, programmed transfers slow down the data transfer rate unnecessarily in these cases, and higher speed can be obtained using DMA. With the GPIB, one DMA facility at the controller serves all devices. There is no need to incorporate complex logic in each device.

Asynchronous Transfers — An asynchronous bus is desirable so that each device can transfer at its own rate. However, there is still a strong motivation to buffer the data at each device when used in large systems in order to speed up the aggregate data rate on the bus by allowing each device to transfer at top speed. The GPIB is asynchronous and uses a special 3-wire handshake that allows data transfers from one talker to many listeners.

I/O To I/O Transfers — In practice, I/O to I/O transfers are seldom done due to the need for processing data and changing formats or due to mismatched data rates. However, the GPIB can support this mode of operation where the microcomputer is neither the talker nor one of the listeners.

GPIB SIGNAL LINES

Data Bus

The lines D101 through D108 are used to transfer addresses, control information and data. The formats for addresses and control bytes are defined by the IEEE 488 standard (see Appendix C). Data formats are undefined and may be ASCII (with or without parity) or binary. D101 is the Least Significant Bit (note that this will correspond to bit 0 on most computers).

Management Bus

ATN — Attention This signal is asserted by the Controller to indicate that it is placing an address or control byte on the Data Bus. ATN is de-asserted to allow the assigned Talker to place status or data on the Data Bus. The Controller regains control by re-asserting ATN; this is normally done synchronously with the handshake to avoid confusion between control and data bytes.

EOI — End or Identify This signal has two uses as its name implies. A talker may assert EOI simultaneously with the last byte of data to indicate end of data. The Controller may assert EOI along with ATN to initiate a Parallel Poll. Although many devices do not use Parallel Poll, all devices should use EOI to end transfers (many currently available ones do not).

SRQ — Service Request This line is like an interrupt; it may be asserted by any device to request the Controller to take some action. The Controller must determine which device is asserting SRQ by conducting a Serial Poll at its earliest convenience. The device deasserts SRQ when polled.

IFC — Interface Clear This signal is asserted only by the System Controller in order to initialize all device interfaces to a known state. After deasserting IFC, the System Controller is the active controller of the system.

REN — Remote Enable This signal is asserted only by the System Controller. Its assertion does not place devices into Remote Control mode; REN only enables a device to go remote when addressed to listen. When in Remote, a device should ignore its front panel controls.
APPLICATIONS

Transfer Bus

NRFD — Not Ready For Data  This handshake line is asserted by a listener to indicate it is not yet ready for the next data or control byte. Note that the Controller will not see NRFD deasserted (i.e., ready for data) until all devices have deasserted NRFD.

NDAC — Not Data Accepted  This handshake line is asserted by a Listener to indicate it has not yet accepted the data or control byte on the DIO lines. Note that the Controller will not see NDAC deasserted (i.e., data accepted) until all devices have deasserted NDAC.

DAV — Data Valid  This handshake line is asserted by the Talker to indicate that a data or control byte has been placed on the DIO lines and has had the minimum specified settling time.

GPIB INTERFACE FUNCTIONS

There are ten (10) interface functions specified by the IEEE 488 standard. Not all devices will have all functions and some may only have partial subsets. The ten functions are summarized below with the relevant section number from the IEEE document given at the beginning of each paragraph. For further information please see the IEEE standard.

1. SH — Source Handshake (section 2.3)  This function provides a device with the ability to properly transfer data from a Talker to one or more Listeners using the three handshake lines.

2. AH — Acceptor Handshake (section 2.4)  This function provides a device with the ability to properly receive data from the Talker using the three handshake lines. The AH function may also delay the beginning (NRFD) or end (NDAC) of any transfer.

3. T — Talker (section 2.5)  This function allows a device to send status and data bytes when addressed to talk. An address consists of one (Primary) or two (Primary and Secondary) bytes. The latter is called an extended Talker.

4. L — Listener (section 2.6)  This function allows a device to receive data when addressed to listen. There can be extended Listeners (analogous to extended Talkers above).

5. SR — Service Request (section 2.7)  This function allows a device to request service (interrupt) the Controller. The SRQ line may be asserted asynchronously.

6. RL — Remote Local (section 2.8)  This function allows a device to be operated in two modes: Remote via the GPIB or Local via the manual front panel controls.

7. PP — Parallel Poll (section 2.9)  This function allows a device to present one bit of status to the Controller-in-charge. The device need not be addressed to talk and no handshake is required.

8. DC — Device Clear (section 2.10)  This function allows a device to be cleared (initialized) by the Controller. Note that there is a difference between DC (device clear) and the IFC line (interface clear).

9. DT — Device Trigger (section 2.11)  This function allows a device to have its basic operation started either individually or as part of a group. This capability is often used to synchronize several instruments.

10. C — Controller (section 2.12)  This function allows a device to send addresses, as well as universal and addressed commands to other devices. There may be more than one controller on a system, but only one may be the controller-in-charge at any one time.

At power-on time the controller that is handwired to be the System Controller becomes the active controller-in-charge. The System Controller has several unique capabilities including the ability to send Interface Clear (IFC — clears all device interfaces and returns control to the System Controller) and to send Remote Enable (REN — allows devices to respond to bus data once they are addressed to listen). The System Controller may optionally Pass Control to another controller, if the system software has the capability to do so.

GPIB CONNECTOR

The GPIB connector is a standard 24-pin industrial connector such as Cinch or Amphenol series 57 Micro-Ribbon. The IEEE standard specifies this connector, as well as the signal connections and the mounting hardware.

The cable has 16 signal lines and 8 ground lines. The maximum length is 20 meters with no more than two meters per device.
APPLICATIONS

GPIB SIGNAL LEVELS

The GPIB signals are all TTL compatible, low true signals. A signal is asserted (true) when its electrical voltage is less than 0.5 volts and is deasserted (false) when it is greater than 2.4 volts. Be careful not to become confused with the two handshake signals, NRFD and NDAC which are also low true (i.e. > 0.5 volts implies the device is Not Ready For Data).

The Intel 8293 GPIB transceiver chips ensure that all relevant bus driver/receiver specifications are met. Detailed bus electrical specifications may be found in Section 3 of the IEEE Std 488-1978. The Standard is the ultimate reference for all GPIB questions.

GPIB MESSAGE PROTOCOLS

The GPIB is a very flexible communications medium and as such has many possible variations of protocols. To bring some order to the situation, this section will discuss a protocol similar to the one used by Ziatech's ZT80 GPIB controller for Intel's MULTIBUS™ computers. The ZT80 is a complete high-level interface processor that executes a set of high level instructions that map directly into GPIB actions. The sequences of commands, addresses and data for these instructions provide a good example of how to use the GPIB (additional information is available in the ZT80 Instruction Manual). The 'null' at the end of each instruction is for cosmetic use to remove previous information from the DIO lines.

**DATA** — Transfer a block of data from device A to devices B, C...
1. Device A Primary (Talk) Address
   Device A Secondary Address (if any)
2. Universal Unlisten
3. Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   Device C Primary (Listen) Address
   etc.
4. First Data Byte
   Second Data Byte
   ...
   Last Data Byte (EOI)
5. Null

**TRIGR** — Trigger devices A, B,…to take action
1. Universal Unlisten
2. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3. Group Execute Trigger
4. Null

**PSCTL** — Pass control to device A
1. Device A Primary (Talk) Address
   Device A Secondary Address (if any)
2. Take Control
3. Null

**CLEAR** — Clear all devices
1. Device Clear
2. Null

**REMAL** — Remote Enable
1. Assert REN continuously

**GOREM** — Put devices A, B,…into Remote
1. Assert REN continuously
2. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3. Null

**GOLOC** — Put devices A, B,…into Local
1. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
2. Go To Local
3. Null

**LOCAL** — Reset all devices to Local
1. Stop asserting REN
APPLICATIONS

**LLKAL** — Prevent all devices from returning to Local
1. Local Lock Out
2. Null

**SPOLL** — Conduct a serial poll of devices A, B, ...
1. Serial Poll Enable
2. Universal Unlisten
3. ZT 80 Primary (Listen) Address
   ZT 80 Secondary Address
4. Device Primary (Talk) Address
   Device Secondary Address (if any)
5. Status byte from device
6. Go to Step 4 until all devices on list have been polled
7. Serial Poll Disable
8. Null

**PPUAL** — Unconfigure and disable Parallel Poll response from all devices
1. Parallel Poll Unconfigure
2. Null

**ENAPP** — Enable Parallel Poll response in devices A, B, ...
1. Universal Unlisten
2. Device Primary (Listen) Address
   Device Secondary Address (if any)
3. Parallel Poll Configure
4. Parallel Poll Enable
5. Go to Step 2 until all devices on list have been configured.
6. Null

**DISPP** — Disable Parallel Poll response from devices A, B, ...
1. Universal Unlisten
2. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
3. Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3. Disable Parallel Poll
4. Null

This Ap Note will detail how to implement a useful subset of these controller instructions.

**HARDWARE ASPECTS OF THE SYSTEM**

**8291 GPIB TALKER/LISTENER**
The 8291 is a custom designed chip that implements many of the non-controller GPIB functions. It provides hooks so the user's software can implement additional features to complete the set. This chip is discussed in detail in its data sheet. The major features are summarized here:

- Designed to interface microprocessors to the GPIB
- Complete Source and Acceptor Handshake
- Complete Talker and Listener Functions with extended addressing
- Service Request, Parallel Poll, Device Clear, Device Trigger, Remote/Local functions
- Programmable trigger rate
- Maskable interrupts
- On-chip primary and secondary address recognition
- 1-8 MHz clock range
- 16 registers (8 read, 8 write) for CPU interface
- DMA handshake provision
- Trigger output pin
- On-chip EOS (End of Sequence) recognition

The pinouts and block diagram are shown in Fig. 5. One of eight read registers is for data transfer to the CPU; the other seven allow the microprocessor to monitor the GPIB states and various bus and device conditions. One of the eight write registers is for data transfer from the CPU; the other seven control various features of the 8291.

The 8291 interface functions will be software configured in this application example to the following subsets for use with the 8292 as a controller that does not pass control. The 8291 is used only to provide the handshake logic and to send and receive data bytes. It is not acting as a normal device in this mode, as it never sees ATN asserted.

| SH1 | Source Handshake |
| AH1 | Acceptor Handshake |
| T3  | Basic Talk-only |
| L1  | Basic Listen-only |
| SR0 | No Service Requests |
| RL0 | No Remote/Local |
| PP0 | No Parallel Poll response |
| DC0 | No Device Clear |
| DT0 | No Device Trigger |

If control is passed to another controller, the 8291 must be reconfigured to act as a talker/listener with the following subsets:

| SH1 | Source Handshake |
| AH1 | Acceptor Handshake |
| T5  | Basic Talker and Serial Poll |
| L3  | Basic Listener |
| SR1 | Service Requests |
| RL1 | Remote/Local with Lockout |
| PP2 | Preconfigured Parallel Poll |
| DC1 | Device Clear |
| DT1 | Device Trigger |
| C0  | Not a Controller |

Most applications do not pass control and the controller is always the system controller (see 8292 commands below).

**8292 GPIB CONTROLLER**
The 8292 is a preprogrammed Intel® 8041A that provides the additional functions necessary to
APPLICATIONS

PIN CONFIGURATION

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>T/R2</td>
<td>38</td>
</tr>
<tr>
<td>T/R3</td>
<td>39</td>
</tr>
<tr>
<td>CLOCK</td>
<td>38</td>
</tr>
<tr>
<td>TRIG</td>
<td>37</td>
</tr>
<tr>
<td>TRIG1</td>
<td>36</td>
</tr>
<tr>
<td>D1N0</td>
<td>35</td>
</tr>
<tr>
<td>D1N1</td>
<td>34</td>
</tr>
<tr>
<td>D1N2</td>
<td>33</td>
</tr>
<tr>
<td>D1N3</td>
<td>32</td>
</tr>
<tr>
<td>RD</td>
<td>31</td>
</tr>
<tr>
<td>WR</td>
<td>30</td>
</tr>
<tr>
<td>INT</td>
<td>29</td>
</tr>
<tr>
<td>DC</td>
<td>28</td>
</tr>
<tr>
<td>D0</td>
<td>27</td>
</tr>
<tr>
<td>D1</td>
<td>26</td>
</tr>
<tr>
<td>D2</td>
<td>25</td>
</tr>
<tr>
<td>D3</td>
<td>24</td>
</tr>
<tr>
<td>D4</td>
<td>23</td>
</tr>
<tr>
<td>D5</td>
<td>22</td>
</tr>
<tr>
<td>VCC</td>
<td>21</td>
</tr>
</tbody>
</table>

8291

BLOCK DIAGRAM

Interface Functions

Figure 5. 8291 Pin Configuration and Block Diagram

implement a GPIB controller when used with an 8291 Talker/Listener. The 8041A is documented in both a user's manual and in AP-41. The following description will serve only as an outline to guide the later discussion.

The 8292 acts as an intelligent slave processor to the main system CPU. It contains a processor, memory, I/O and is programmed to perform a variety of tasks associated with GPIB controller operation. The on-chip RAM is used to store information about the state of the Controller function, as well as a variety of local variables, the stack and certain user status information. The timer/counter may be optionally used for several time-out functions or for counting data bytes transferred. The I/O ports provide the GPIB control signals, as well as the ancillary lines necessary to make the 8291, 2, 3 work together.

The 8292 is closely coupled to the main CPU through three on-chip registers that may be independently accessed by both the master and the 8292 (UPI-41A). Figure 6 shows this Register Interface. Also refer to Figure 12.

The status register is used to pass Interrupt Status information to the master CPU (A0 = 1 on a read).

The DBBOUT register is used to pass one of five other status words to the master based on the last command written into DBBIN. DBBOUT is accessed when A0 = 0 on a Read. The five status words are Error Flag, Controller Status, GPIB Status, Event Counter Status or Time Out Status.

DBBIN receives either commands (A0 = 1 on a Write) or command related data (A0 = 0 on a write) from the master. These command related data are Interrupt Mask, Error Mask, Event Counter or Time Out.

8293 GPIB TRANSCIEVERS

The 8293 is a multi-use HMOS chip that implements the IEEE 488 bus transceivers and contains the additional logic required to make the 8291 and 8292 work together. The two option strapping pins are used to internally configure the chip to perform the specialized gating required for use with 8291 as a device or with 8291/92 as a controller.

In this application example the two configurations used are shown in Fig. 7a and 7b. The drivers are set to open collector or three state mode as required and the special logic is enabled as required in the two modes.
Figure 7a. 8293 Mode 2

Figure 7b. 8293 Mode 3

8291/2/3 CHIP SET

Figure 8 shows the four chips interconnected with the special logic explicitly shown.

The 8291 acts only as the mechanism to put commands and addresses on the bus while the 8292 is asserting ATN. The 8291 is tricked into believing that the ATN line is not asserted by the ATN2 output of the ATN transceiver and is placed in Talk-only mode by the CPU. The 8291 then acts as though it is sending data, when in reality it is sending addresses and/or commands. When the 8292 deasserts ATN, the CPU software must place the 8291 in Talk-only, Listen-only or Idle based on the implicit knowledge of how the controller is going to participate in the data transfer. In other words, the 8291 does not respond directly to addresses or commands that it sends on the bus on behalf of the Controller. The user software, through the use of Listen-only or Talk-only, makes the 8291 behave as though it were addressed.

Although it is not a common occurrence, the GPIB specification allows the Controller to set up a data transfer between two devices and not directly participate in the exchange. The controller must know when to go active again and regain control. The chip set accomplishes this through use of the “Continuous Acceptor Handshake cycling mode” and the ability to detect EOI or EOS at the end of the transfer. See XFER in the Software Driver Outline below.

If the 8292 is not the System Controller as determined by the signal on its SYC pin, then it must be able to respond to an IFC within 100usec. This is accomplished by the cross-coupled NORs in Fig. 7a which deassert the 8293’s internal version of CIC (Not Controller-in-Charge). This condition is latched until the 8292’s firmware has received the IFCL (interface clear received latch) signal by testing the IFCL input. The firmware then sets its signals to reflect the inactive condition and clears the 8293’s latch.

In order for the 8292 to conduct a Parallel Poll the 8291 must be able to capture the PP response on the DIO lines. The only way to do this is to fool the 8291 by putting it into Listen-only mode and generating a DAV condition. However, the bus spec does not allow a DAV during Parallel Poll, so the back-to-back 3-state buffers (see Fig. 7b) in the 8293 isolate the bus and allow the 8292 to generate a local DAV for this purpose. Note that the 8291 cannot assert a Parallel Poll response. When the 8292 is not the controller-in-charge the 8291 may respond to PPs and the 8293 guarantees that the DIO drivers are in “open collector” mode through the OR gate (Fig. 7b).
Figure 8. Talker/Listener/Controller
ZIATECH'S GPIB CONTROLLER

Ziatech's GPIB Controller, the ZT7488/18 will be used as the controller hardware in this Application Note. The controller consists of an 8291, 8292, an 8 bit input port and TTL logic equivalent to that shown in Figure 8. Figure 9 shows the card’s block diagram. The ZT7488/18 plugs into the STD bus, a 56 pin 8 bit microprocessor oriented bus. An 8085 CPU card is also available on the STD bus and will be used to execute the driver software.

The 8291 uses I/O Ports 60H to 67H and the 8292 uses I/O Ports 68H and 69H. The five interrupt lines are connected to a three-state buffer at I/O Port 6FH to facilitate polling operation. This is required for the TC1, as it cannot be read internally in the 8292. The other three 8292 lines (SPI, IBF, OBF) and the 8291’s INT line are also connected to minimize the number of I/O reads necessary to poll the devices.

NDAC is connected to COUNT on the 8292 to allow byte counting on data transfers. The example driver software will not use this feature, as the software is simpler and faster if an internal 8085 register is used for counting in software.

---

**Figure 9. ZT7488/18 GPIB Controller**

**Figure 10. 8291 Registers**
Applications

The application example will not use DMA or interrupts; however, the Figure 11 block diagram includes these features for completeness.

The 8257-5 DMA chip can be used to transfer data between the RAM and the 8291 Talker/Listener. This mode allows a faster data rate on the GPIB and typically will depend on the 8291's EOS or EOI detection to terminate the transfer. The 8259-5 interrupt controller is used to vector the five possible interrupts for rapid software handling of the various conditions.

8292 Command Description

This section discusses each command in detail and relates them to a particular GPIB activity. Recall that although the 8041A has only two read registers and one write register, through the magic of on-chip firmware the 8292 appears to have six read registers and five write registers. These are listed in Figure 12. Please see the 8292 data sheet for detailed definitions of each register. Note the two letter mnemonics to be used in later discussions. The CPU must not write into the 8292 while IBF (Input Buffer Full) is a one, as information will be lost.

Direct Commands

Both the Interrupt Mask (IM) and the Error Mask (EM) register may be directly written with the LSB of the address bus (A0) a “0”. The firmware uses the MSB of the data written to differentiate between IM and EM.

Load Interrupt Mask

This command loads the Interrupt Mask with D7–D0. Note that D7 must be a “1” and that interrupts are enabled by a corresponding “1” bit in this register. IFC interrupt cannot be masked off; however, when the 8292 is the System Controller, sending an ABORT command will not cause an IFC interrupt.
## APPLICATIONS

### READ FROM 8292

**Interrupt Status**

<table>
<thead>
<tr>
<th>SYC</th>
<th>ERR</th>
<th>SRQ</th>
<th>EV</th>
<th>X</th>
<th>IFCR</th>
<th>IBF</th>
<th>OBF</th>
<th>D7</th>
<th>D0</th>
</tr>
</thead>
</table>

**Error Flag**

<table>
<thead>
<tr>
<th>X</th>
<th>X</th>
<th>USER</th>
<th>X</th>
<th>X</th>
<th>TOUT3</th>
<th>TOUT2</th>
<th>TOUT1</th>
<th>D7</th>
<th>D0</th>
</tr>
</thead>
</table>

**Controller Status**

<table>
<thead>
<tr>
<th>CSBS</th>
<th>CA</th>
<th>X</th>
<th>X</th>
<th>SYCS</th>
<th>IFC</th>
<th>REN</th>
<th>SRQ</th>
<th>D7</th>
<th>D0</th>
</tr>
</thead>
</table>

**GPIB (Bus) Status**

<table>
<thead>
<tr>
<th>REN</th>
<th>DAV</th>
<th>EOI</th>
<th>X</th>
<th>SYC</th>
<th>IFC</th>
<th>ANT1</th>
<th>SRQ</th>
<th>D7</th>
<th>D0</th>
</tr>
</thead>
</table>

**Event Counter Status**

<table>
<thead>
<tr>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
</tr>
</thead>
</table>

**Time Out Status**

<table>
<thead>
<tr>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
</tr>
</thead>
</table>

### PORT #

<table>
<thead>
<tr>
<th>HD69</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>OP</th>
<th>C</th>
<th>C</th>
<th>C</th>
<th>C</th>
<th>C</th>
</tr>
</thead>
</table>

**Interrupt Mask**

<table>
<thead>
<tr>
<th>HD68</th>
<th>1</th>
<th>SPI</th>
<th>TCI</th>
<th>SYC</th>
<th>OBF</th>
<th>IBF</th>
<th>0</th>
<th>SRQ</th>
<th>D7</th>
<th>D0</th>
</tr>
</thead>
</table>

**Error Mask**

<table>
<thead>
<tr>
<th>HD68</th>
<th>0</th>
<th>0</th>
<th>USER</th>
<th>0</th>
<th>0</th>
<th>TOUT4</th>
<th>TOUT3</th>
<th>TOUT2</th>
<th>TOUT1</th>
<th>D7</th>
<th>D0</th>
</tr>
</thead>
</table>

**Event Counter**

<table>
<thead>
<tr>
<th>HD68</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
</tr>
</thead>
</table>

**Time Out**

<table>
<thead>
<tr>
<th>HD68</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
</tr>
</thead>
</table>

*Note: These registers are accessed by a special utility command.*

### Figure 12. 8292 Registers

#### Load Error Mask

This command loads the Error Mask with D7–D0. Note that D7 must be a zero and that interrupts are enabled by a corresponding “1” bit in this register.

#### Utility Commands

These commands are used to read or write the 8292 registers that are not directly accessible. All utility commands are written with AO = 1, 04 = 0, 03 = 00. Specify the particular command. For writing into registers the general sequence is:

1. wait for IBF = 0 in Interrupt Status Register
2. write the appropriate command to the 8292, 3. write the desired register value to the 8292 with
4. wait for indication of completion from 8292
   (IBF = 0).

For reading a register the general sequence is:

1. wait for IBF = 0 in Interrupt Status Register
2. write the appropriate command to the 8292
3. wait for a TCI (Task Complete Interrupt)
4. Read the value of the accessed register from the
   8292 with A0 = 0.

**WEVC** — Write to Event Counter
(Command = 0E2H)

The byte written following this command will be loaded into the event counter register and event counter status for byte counting. The internal counter is incremented on a high to low transition of the COUNT (T1) input. In this application example NDAC is connected to count. The counter is an 8 bit register and therefore can count up to 256 bytes (writing 0 to the EC implies a count of 256). If longer blocks are desired, the main CPU must handle the interrupts every 256 counts and carefully observe the timing constraints.

Because the counter has a frequency range from 0 to 133 kHz when using a 6 MHz crystal, this feature may not be usable with all devices on the GPIB. The 8291 can easily transfer data at rates up to 250 kHz and even faster with some tuning of the system. There is also a 500 ns minimum high time requirement for COUNT which can potentially be violated by the 8291 in continuous acceptor handshake mode (i.e., TNDDV1 + TDVND2–C = 350 + 350 = 700 max). When cable delays are taken into consideration, this problem will probably never occur.

When the 8292 has completed the command, IBF will become a “0” and will cause an interrupt if masked on.

**WTOUT** — Write to Time Out Register
(Command = 0E1H)

The byte written following this command will be used to determine the number of increments used for the time out functions. Because the register is 8 bits, the maximum time out is 256 time increments. This
is probably enough for most instruments on the GPIB but is not enough for a manually stepped operation using a GPIB logic analyzer like Ziatech's ZT488. Also, the 488 Standard does not set a lower limit on how long a device may take to do each action. Therefore, any use of a time out must be able to be overridden (this is a good general design rule for service and debugging considerations).

The time out function is implemented in the 8292's firmware and will not be an accurate time. The counter counts backwards to zero from its initial value. The function may be enabled/disabled by a bit in the Error mask register. When the command is complete IBF will be set to a “0” and will cause an interrupt if masked on.

REVC — Read Event Counter Status
(Command = 0E3H)

This command transfers the content of the Event Counter to the DBBOUT register. The firmware then sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value from the 8292 with A0 = 0.

RINM — Read Interrupt Mask Register
(Command = 0E5H)

This command transfers the content of the Interrupt Mask register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

RERM — Read Error Mask Register
(Command = 0EAH)

This command transfers the content of the Error Mask register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

RCST — Read Controller Status Register
(Command = 0E6H)

This command transfers the content of the Controller Status register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

RTOU — Read Time Out Status Register
(Command = 0E9H)

This command transfers the content of the Time Out Status register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

If this register is read while a time-out function is in process, the value will be the time remaining before time-out occurs. If it is read after a time-out, it will be zero. If it is read when no time-out is in process, it will be the last value reached when the previous timing occurred.

RBST — Read Bus Status Register
(Command = 0E7H)

This command causes the firmware to read the GPIB management lines, DAV and the SYC pin and place a copy in DBBOUT. TCI is set to “1” and will cause an interrupt if masked on. The CPU may read the value.

RERF — Read Error Flag Register
(Command = 0E4H)

This command transfers the content of the Error Flag register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

This register is also placed in DBBOUT by an IACK command if ERR remains set. TCI is set to “1” in this case also.

IACK — Interrupt Acknowledge
(Command = A1 A2 A3 A4 1 A5 1 1)

This command is used to acknowledge any combinations of the five SPI interrupts (A1–A5): SYC, ERR, SRQ, EV, and IFCR. Each bit A1–A5 is an individual acknowledgement to the corresponding bit in the Interrupt Status Register. The command clears SPI but it will be set again if all of the pending interrupts were not acknowledged.

If A2 (ERR) is “1”, the Error Flag register is placed in DBBOUT and TCI is set. The CPU may then read the Error Flag without issuing an RERF command.

OPERATION COMMANDS

The following diagram (Fig. 13) is an attempt to show the interrelationships among the various 8292 Operation Commands. It is not meant to replace the complete controller state diagram in the IEEE Standard.

RST — Reset (Command = 0F2H)

This command has the same effect as an external reset applied to the chip’s pin #4. The 8292’s actions are:

1. All outputs go to their electrical high state. This means that SPI, TCI, OBFI, IBFI, CLTH will be TRUE and all other GPIB signals will be FALSE.
2. The 8292’s firmware will cause the above mentioned five signals to go FALSE after approximately 17.5 usec. (at 6 MHz).
3. These registers will be cleared: Interrupt Status, Interrupt Mask, Error Mask, Time Out, Event Counter, Error Flag.
4. If the 8292 is the System Controller (SYC is TRUE), then IFC will be sent TRUE for approximately 100 usec and the Controller function will end up in charge of the bus. If the 8292 is not the
APPLICATIONS

System Controller then it will end up in an Idle state.

5. TCI will not be set.

**RSTI** — Reset Interrupts (Command = 0F3)

This command clears all pending interrupts and error flags. The 8292 will stop waiting for actions to occur (e.g., waiting for ATN to go FALSE in a TCNTR command or waiting for the proper handshake state in a TCSY command). TCI will not be set.

**ABORT** — Abort all operations and Clear Interface (Command = 0F9H)

If the 8292 is not the System Controller, this command acts like a NOP and flags a USER ERROR in the Error Flag Register. No TCI will occur.

If the 8292 is the System Controller then IFC is set TRUE for approximately 100 μsec and the 8292 becomes the Controller-in-Charge and asserts ATN. TCI will be set, only if the 8292 was NOT the CIC.

**STCNI** — Start Counter Interrupts (Command = 0FEH)

Enables the EV Counter Interrupt. TCI will not be set. Note that the counter must be enabled by a GSEC command.

**SPCNI** — Stop Counter Interrupts (Command = 0F0H)

The 8292 will not generate an EV interrupt when the counter reaches 0. Note that the counter will continue counting. TCI will not be set.

**SREM** — Set Interface to Remote Control (Command = 0F8H)

If the 8292 is the System Controller, it will set REN and TCI TRUE. Otherwise it only sets the User Error Flag.

**SLOC** — Set Interface to Local Mode (Command = 0F7H)

If the 8292 is the System Controller, it will set REN FALSE and TCI TRUE. Otherwise, it only sets the User Error Flag.

**EXPP** — Execute Parallel Poll (Command = 0F5H)

If not Controller-in-Charge, the 8292 will treat this as a NOP and does not set TCI. If it is the Controller-in-Charge then it sets IDY (EOI & ATN) TRUE and generates a local DAV pulse (that never reaches the GPIB because of gates in the 8293). If the 8291 is configured as a listener, it will capture the Parallel Poll Response byte in its data register. TCI is not generated, the CPU must detect the BI (Byte In) from the 8291. The 8292 will be ready to accept another command before the BI occurs; therefore the 8291’s BI serves as a task complete indication.

**GTSB** — Go To Standby (Command = 0F6H)

If the 8292 is not the Controller-in-Charge, it will treat this command as a NOP and does not set TCI TRUE. Otherwise, it goes to Controller Standby State (CSBS), sets ATN FALSE and TCI TRUE. This command is used as part of the Send, Receive, Transfer and Serial Poll System commands (see next section) to allow the addressed talker to send data/status.

If the data transfer does not start within the specified Time-Out, the 8292 sets TOUT2 TRUE in the Error Flag Register and sets SPI (if enabled). The controller continues waiting for a new command. The CPU must decide to wait longer or to regain control and take corrective action.

**GSEC** — Go to Standby and Enable Counting (Command = 0F4H)

This command does the same things as GTSB but also initializes the event counter to the value previously stored in the Event Counter Register (default value is 256) and enables the counter. One may wire the count input to NDAC to count bytes. When the counter reaches zero, it sets EV (and SPI if enabled) in Interrupt Status and will set EV every 256 bytes thereafter. Note that there is a potential loss of count information if the CPU does not respond to the EV/SPI before another 256 bytes have been transferred. TCI will be set at the end of the command.

**TCSY** — Take Control Synchronously (Command = 0FDH)

If the 8292 is not in Standby, it treats this command as a NOP and does not set TCI. Otherwise, it waits.
for the proper handshake state and sets ATN TRUE. The 8292 will set TOUT3 if the handshake never assumes the correct state and will remain in this command until the handshake is proper or a RSTI command is issued. If the 8292 successfully takes control, it sets TCI TRUE.

This is the normal way to regain control at the end of a Send, Receive, Transfer or Serial Poll System Command. If TCSY is not successful, then the controller must try TCAS (see warning below).

**TCAS — Take Control Asynchronously** (Command = 0FCH)

If the 8292 is not in Standby, it treats this command as a NOP and does not set TCI. Otherwise, it arbitrarily sets ATN TRUE and TCI TRUE. Note that this action may cause devices on the bus to lose a data byte or cause them to interpret a data byte as a command byte. Both Actions can result in anomalous behavior. TCAS should be used only in emergencies. If TCAS fails, then the System Controller will have to issue an ABORT to clean things up.

**GIDL — Go to Idle** (Command = 0F1H)

If the 8292 is not the Controller in Charge and Active, then it treats this command as a NOP and does not set TCI. Otherwise, it sets ATN FALSE, becomes Not Controller in Charge, and sets TCI TRUE. This command is used as part of the Pass Control System Command.

**TCNTR — Take (Receive) Control** (Command = 0FAH)

If the 8292 is not Idle, then it treats this command as a NOP and does not set TCI. Otherwise, it waits for the current Controller-in-Charge to set ATN FALSE. If this does not occur within the specified Time Out, the 8292 sets TOUT1 in the Error Flag Register and sets SPI (if enabled). It will not proceed until ATN goes false or it receives an RSTI command. Note that the Controller in Charge must previously have sent this controller (via the 8291's command pass through register) a Pass Control message. When ATN goes FALSE, the 8292 sets CIC, ATN and TCI TRUE and becomes Active.

**SOFTWARE DRIVER OUTLINE**

The set of system commands discussed below is shown in Figure 14. These commands are implemented in software routines executed by the main CPU.

The following section assumes that the Controller is the System Controller and will not Pass Control. This is a valid assumption for 99+% of all controllers. It also assumes that no DMA or Interrupts will be used. SYC (System Control Input) should not be changed after Power-on in any system — it adds unnecessary complexity to the CPU's software.

In order to use polling with the 8292 one must enable TCI but not connect the pin to the CPU's interrupt pin. TCI must be readable by some means. In this application example it is connected to bit 1 port 6FH on the ZT7488/18. In addition, the other three 8292 interrupt lines and the 8291 interrupt are also on that port (SPI-Bit 2, IBFI-Bit 4, OBFI-Bit 3, 8291 INT-Bit 0).

These drivers assume that only primary addresses will be used on the GPIB. To use secondary addresses, one must modify the test for valid talk/listen addresses (range macro) to include secondaries.

<table>
<thead>
<tr>
<th>INIT</th>
<th>INITIALIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TALKER/LISTENER</td>
<td></td>
</tr>
<tr>
<td>SEND</td>
<td>SEND DATA</td>
</tr>
<tr>
<td>RECEV</td>
<td>RECEIVE DATA</td>
</tr>
<tr>
<td>XFER</td>
<td>TRANSFER DATA</td>
</tr>
<tr>
<td>CONTROLLER</td>
<td></td>
</tr>
<tr>
<td>TRIG</td>
<td>GROUP EXECUTE TRIGGER</td>
</tr>
<tr>
<td>DCLR</td>
<td>DEVICE CLEAR</td>
</tr>
<tr>
<td>SPOL</td>
<td>SERIAL POLL</td>
</tr>
<tr>
<td>PPEN</td>
<td>PARALLEL POLL ENABLE</td>
</tr>
<tr>
<td>PPUN</td>
<td>PARALLEL POLL DISABLE</td>
</tr>
<tr>
<td>PPD</td>
<td>PARALLEL POLL UNCONFIGURE</td>
</tr>
<tr>
<td>PPOL</td>
<td>PARALLEL POLL</td>
</tr>
<tr>
<td>PCTL</td>
<td>PASS CONTROL</td>
</tr>
<tr>
<td>RTOL</td>
<td>RECEIVE CONTROL</td>
</tr>
<tr>
<td>SRQ</td>
<td>SERVICE REQUESTED</td>
</tr>
</tbody>
</table>

| SYSTEM CONTROLLER | |
| REME | REMOTE ENABLE |
| LOC | LOCAL |
| IFC | ABORT/INTERFACE CLEAR |

**Figure 14. Software Driver Routines**

**INITIALIZATION**

8292 — Comes up in Controller Active State when SYC is TRUE. The only initialization needed is to enable the TCI interrupt mask. This is done by writing 0A0H to Port 68H.

8291 — Disable both the major and minor addresses because the 8291 will never see the 8292's commands/addresses (refer to earlier hardware discussion). This is done by writing 60H and 0E0H to Port 66H.
APPLICATIONS

Set Address Mode to Talk-only by writing 80H to Port 64H.
Set internal counter to 3 MHz to match the clock input coming from the 8085 by writing 23H to Port 65H. High speed mode for the handshakes will not be used here even though the hardware uses three-state drivers.

Set up Int. pins for Port 6FH
;Task complete must be on
;In controller usage, the 8291
;Is set to talk only and/or listen only
;Talk only is our rest state
;3 MHz in this ap note example
;Releases 8291 from init. state

TALKER/LISTENER ROUTINES

Send Data

SEND<listener list pointer> <count> <EOS> <data buffer pointer>

This system command sends data from the CPU to one or more devices. The data is usually a string of ASCII characters, but may be binary or other forms as well. The data is device-specific. My Talk Address (MTA) must be output to satisfy the GPIB requirement of only one talker at a time (any other talker will stop when MTA goes out). The MTA is not needed as far as the 8291 is concerned — it will be put into talk-only mode (ton).

This routine assumes a non-null listener list in that it always sends Universal Unlisten. If it is desired to send data to the listeners previously addressed, one could add a check for a null list and not send UNL. Count must be 255 or less due to an 8 bit register. This routine also always uses an EOS character to terminate the string output; this could easily be eliminated and rely on the count. Items in brackets () are optional and will not be included in the actual code in Appendix A.

SEND:
Output-to-8291 MTA, UNL
Put EOS into 8291
While 20H ≤ listener ≤ 3EH
output-to-8291 listener
Increment listen list pointer
Output-to-8292 GTSB
Enable-8291
Output EOI on EOS sent
If count < > 0 then
While not (end or count = 0)
(could check tout 2 here)
Output-to-8291 data
Increment data buffer pointer
Decrement count
Output-to-8292 TCSY
(If tout3 then take control async)
Enable 8291
No output EOI on EOS sent
Return

;We will talk, nobody listen
;End of string compare character
;GPIB listen addresses are
;"space" thru " >" ASCII
;Address all listeners
;8292 stops asserting ATN, go to standby
;Send EOI along with EOS character
;Wait for EOS or end of count
;Optionally check for stuck bus-tout 2
;Output all data, one byte at a time
;8085 CREG will count for us
;8292 asserts ATN, take control sync.
;If unable to take control sync.
;Restore 8291 to standard condition
Receive Data

RECV <talker> <count> <EOS> <data buffer pointer>

This system command is used to input data from a device. The data is typically a string of ASCII characters.

This routine is the dual of SEND. It assumes a new talker will be specified, a count of less than 257, and an EOS character to terminate the input. EOI received will also terminate the input. Figure 15 shows the flow chart for the RECV ending conditions. My Listen Address (MLA) is sent to facilitate analysis by a GPIB logic analyzer like the Ziatech ZT488. Otherwise, the bus would appear to have no listener even though the 8291 will be listening.

Note that although the count may go to zero before the transmission ends, the talker will probably be left in a strange state and may have to be cleared by the controller. The count ending of RECV is therefore used as an error condition in most situations.
**APPLICATIONS**

**RECV:**
- Put EOS into 8291
- If $40H \leq \text{taker} \leq 5EH$ then
  - Output-to-8291 talker
  - Increment talker pointer
  - Output-to-8291 UNL, MLA
  - Enable-8291
    - Holdoff on end
    - End on EOS received
    - Ion, reset ton
    - Immediate execute pon
  - Output-to-8292 GTSB
  - While not (end or count = 0 (or tout2))
    - Input-from-8291 data
    - Increment data buffer pointer
    - Decrement count
    - (If count = 0 then error)
    - Output-to-8292 TCSY
    - (If Tout3 then take control async.)
  - Enable-8291
    - No holdoff on end
    - No end on EOS received
    - Ion, reset ton
    - Finish handshake
    - Immediate execute pon
  - Return error-indicator

; End of string compare character
; GPIB talk addresses are
; "@" thru "\" ASCII
; Do this for consistency's sake
; Everyone except us stop listening

; Stop when EOS character is
; Detected by 8291
; Listen only (no talk)

; 8292 stops asserting ATN, go to standby
; wait for EOS or EOI or end of count
; optionally check for stuck bus-tout2
; input data, one byte at a time

; Use 8085 C register as counter
; Count should not occur before end
; 8292 asserts ATN take control
; If unable to take control sync.
; Put 8291 back as needed for
; Controller activity and
; Clear holdoff due to end

; Complete holdoff due to end, if any
; Needed to reset Ion

---

**Figure 17.** RECV from "R"; EOS = 0DH

**Figure 18.** XFER from "\" to "1", "2", "\"; EOS = 0DH
APPLICATIONS

Transfer Data

*XFER <Talker > < Listener list > < EOS >*

This system command is used to transfer data from a talker to one or more listeners where the controller does not participate in the transfer of the ASCII data. This is accomplished through the use of the 8291's continuous acceptor handshake mode while in listen-only.

This routine assumes a device list that has the ASCII talker address as the first byte and the string (one or more) of ASCII listener addresses following. The EOS character or an EOI will cause the controller to take control synchronously and thereby terminate the transfer.

*XFER:*

Output-to-8291: Talker, UNL
While 20H ≤ listen ≤ 3EH
   Output-to-8291: Listener
   Increment listen list pointer
Enable-8291
   Ion, no ton
   Continuous AH mode
   End on EOS received
   Immediate execute PON
Put EOS into 8291
Output-to-8292: GTSB

Upon end (or tout2) then
   Take control synchronously
Enable-8291
   Finish handshake
   Not continuous AH mode
   Not END on EOS received
   ton
   Immediate execute pon
Return

;Send talk address and unlisten
;Send listen address
;Controller is pseudo listener
;Handshake but don't capture data
;Capture EOS as well as EOI
;Initialize the 8291
;Set up EOS character
;Go to standby
;8292 waits for EOS or EOI and then
;Regains control
;Go to Ready for Data

CONTROLLER

Group Execute Trigger

*TRIG < Listener list >*

This system command causes a group execute trigger (GET) to be sent to all devices on the listener list. The intended use is to synchronize a number of instruments.

*TRIG:*

Output-to-8291 UNL
While 20H ≤ listen ≤ 3EH
   Output-to-8291 Listener
   Increment listen list pointer
Output-to-8291 GET
Return

;Everybody stop listening
;Check for valid listen address
;Address each listener
;Terminate on any non-valid character
;Issue group execute trigger
Device Clear

DCLR < Listener list >

This system command causes a device clear (SDC) to be sent to all devices on the listener list. Note that this is not intended to clear the GPIB interface of the device, but should clear the device-specific logic.

DCLR:
Output-to-8291 UNL
While 20H ≤ Listener ≤ 3EH
Output-to-8291 listener
Increment listen list pointer
Output-to-8291 SDC
Return

;Everybody stop listening
;Check for valid listen address
;Address each listener
;Terminate on any non-valid character
;Selective device clear

Serial Poll

SPOL < Talker list > < status buffer pointer >

This system command sequentially addresses the designated devices and receives one byte of status from each. The bytes are stored in the buffer in the same order as the devices appear on the talker list. MLA is output for completeness.
**APPLICATIONS**

**SPOL:**
Output-to-8291 UNL, MLA, SPE

While $40H \leq \text{talker} \leq 5EH$
Output-to-8291 talker
Increment talker list pointer
Enable-8291
Ion, reset Ion
Immediate execute pon
Output-to-8292 GTSB
Wait for data in (BI)
Output-to-8292 TCSY
Input-from-8291 data
Increment buffer pointer
Enable 8291
Ion, reset Ion
Immediate execute pon
Output-to-8291 SPD
Return

;Unlisten, we listen, serial poll enable
;Only one byte of serial poll
;Status wanted from each talker
;Check for valid transfer
;Address each device to talk
;One at a time

;Listen only to get status
;This resets Ion
;Go to standby
;Serial poll status byte into 8291
;Take control synchronously
;Actually get data from 8291

;Resets Ion
;Send serial poll disable after all devices polled

---

**Figure 21. SPOL "Q", "R", "K", "A"**

**Parallel Poll Enable**

**PPEN < Listener list> < Configuration Buffer pointer >**

This system command configures one or more devices to respond to Parallel Poll, assuming they implement subset PP1. The configuration information is stored in a buffer with one byte per device in the same order as devices appear on the listener list. The configuration byte has the format $XXXXIP3P2PI$ as defined by the IEEE Std. $P3P2P1$ indicates the bit # to be used for a response and I indicates the assertion value. See Sec. 2.9.3.3 of the Std. for more details.
APPLICATIONS

**PPEN:**
- Output-to-8291 UNL
- While 20H ≤ Listener ≤ 3EH
  - Output-to-8291 listener
  - Output-to-8291 PPC, (PPE or data)
  - Increment listener list pointer
  - Increment buffer pointer
- Return

;Universal unlisten
;Check for valid listener
;Stop old listener, address new
;Send parallel poll info
;Point to next listener
;One configuration byte per listener

---

**Parallel Poll Disable**

**PPDS <listener list>**

This system command disables one or more devices from responding to a Parallel Poll by issuing a Parallel Poll Disable (PPD). It does not deconfigure the devices.

**PPDS:**
- Output-to-8291 UNL
  - While 20H ≤ Listener ≤ 3EH
  - Output-to-8291 listener
  - Increment listener list pointer
  - Output-to-8291 PPC, PPD
- Return

;Universal Unlisten
;Check for valid listener
;Address listener
;Disable PP on all listeners

---

**Figure 23. PPDS “₁”, “₊”, “₋”**

**Figure 24. PPUN**
APPLICATIONS

Parallel Poll Unconfigure

PPUN
This system command deconfigures the Parallel Poll response of all devices by issuing a Parallel Poll Unconfigure message.

PPUN:
Output-to-8291 PPU
Return

;Unconfigure all parallel poll

Conduct a Parallel Poll

PPOL
This system command causes the controller to conduct a Parallel Poll on the GPIB for approximately 12.5 usec (at 6 MHz). Note that a parallel poll does not use the handshake; therefore, to ensure that the device knows whether or not its positive response was observed by the controller, the CPU should explicitly acknowledge each device by a device-dependent data string. Otherwise, the response bit will still be set when the next Parallel Poll occurs. This command returns one byte of status.

PPOL:
Enable-8291
lon
Immediate execute pon
Output-to-8292 EXPP
Upon BI
Input-from-8291 data
Enable-8291
ton
Immediate execute pon
Return Data (status byte)

;Listen only
;This resets ton
;Execute parallel poll
;When byte is input
;Read it
;Talk only
;This resets lon

Pass Control

PCTL <talker>
This system command allows the controller to relinquish active control of the GPIB to another controller. Normally some software protocol should already have informed the controller to expect this, and under what conditions to return control. The 8291 must be set up to become a normal device and the CPU must handle all commands passed through, otherwise control cannot be returned (see Receive Control below). The controller will go idle.

PCTL:
If 40H ≤ talker ≤ 5EH then
if talker < > MTA then
output-to-8291 talked, TCT
Enable-8291
not ton, not lon
Immediate execute pon
My device address, mode I
Undefined command pass through
(Parallel Poll Configuration)
Output-to-8292 GIDL
Return

;Cannot pass control to myself
;Take control message to talker
;Set up 8291 as normal device
;Reset ton and lon
;Put device number in Register 6
;Required to receive control
;Optional use of PP
;Put controller in idle
Receive Control

**RCTL**

This system command is used to get control back from the current controller-in-charge if it has passed control to this inactive controller. Most GPIB systems do not use more than one controller and therefore would not need this routine.

To make passing and receiving control a manageable event, the system designer should specify a protocol whereby the controller-in-charge sends a data message to the soon-to-be-active controller. This message should give the current state of the system, why control is being passed, what to do, and when to pass control back. Most of these issues are beyond the scope of this Ap Note.

---

**RCTL:**

Upon CPT
- If (command=TCT) then
  - If TA then
    - Enable-8291
    - Disable major device number
    - ton
    - Mask off interrupts
    - Immediate execute pon
- ;Wait for command pass through bit in 8291
- ;If command is take control and
- ;We are talker addressed
- ;Controller will use ton and lon
- ;Talk only mode

---
Service Request

**SRQD**

This system command is used to detect the occurrence of a Service Request on the GPIB. One or more devices may assert SRQ simultaneously, and the CPU would normally conduct a Serial Poll after calling this routine to determine which devices are SRQing.
APPLICATIONS

SRQD:
If SRQ then
   Output-to-8292 IACK.SRQ
   Return SRQ
Else return no SRQ

SYSTEM CONTROLLER
Remote Enable

REME
This system command asserts the Remote Enable line (REN) on the GPIB. The devices will not go remote until they are later addressed to listen by some other system command.

REME:
Output-to-8292 SREM
Return

;8292 asserts remote enable line

Local

LOCL
This system command deasserts the REN line on the GPIB. The devices will go local immediately.

LOCL:
Output-to-8292 SLOC
Return

;8292 stops asserting remote enable

Figure 29. LOCL
Figure 30. IFCL
APPLICATIONS

Interface Clear/Abort

IFCL
This system command asserts the GPIB's Interface Clear (IFC) line for at least 100 microseconds. This causes all interface logic in all devices to go to a known state. Note that the device itself may or may not be reset, too. Most instruments do totally reset upon IFC. Some devices may require a DCLR as well as an IFCL to be completely reset. The (system) controller becomes Controller-in-Charge.

IFCL:
Output-to-8292 ABORT
Return
; 8292 asserts Interface Clear
; For 100 microseconds

INTERRUPTS AND DMA CONSIDERATIONS
The previous sections have discussed in detail how to use the 8291, 8292, 8293 chip set as a GPIB controller with the software operating in a polling mode and using programmed transfer of the data. This is the simplest mode of use, but it ties up the microprocessor for the duration of a GPIB transaction. If system design constraints do not allow this, then either Interrupts and/or DMA may be used to free up processor cycles.

The 8291 and 8292 provide sufficient interrupts that one may return to do other work while waiting for such things as 8292 Task Completion, 8291 Next Byte In, 8291 Last Byte Out, 8292 Service Request In, etc. The only difficulty lies in integrating these various interrupt sources and their matching routines into the overall system's interrupt structure. This is highly situation-specific and is beyond the scope of this Ap Note.

The strategy to follow is to replace each of the WAIT routines (see Appendix A) with a return to the main code and provide for the corresponding interrupt to bring the control back to the next section of GPIB code. For example WAITO (Wait for Byte Out of 8291) would be replaced by having the BO interrupt enabled and storing the (return) address of the next instruction in a known place. This co-routine structure will then be activated by a BO interrupt. Fig. 31 shows an example of the flow of control.

Figure 31. GPIB Interrupt & Co-Routine Flow of Control

6-470
APPLICATIONS

DMA is also useful in relieving the processor if the average length of a data buffer is long enough to overcome the extra time used to set up a DMA chip. This decision will also be a function of the data rate of the instrument. The best strategy is to use the DMA to handle only the data buffer transfers on SEND and RECV and to do all the addressing and control just as shown in the driver descriptions.

Another major reason for using a DMA chip is to increase the data rate and therefore increase the overall transaction rate. In this case the limiting factor becomes the time used to do the addressing and control of the GPIB using software like that in Appendix A. The data transmission time becomes insignificant at DMA speeds unless extremely long buffers are used.

Refer to Figure 11 for a typical DMA and interrupt based design using the 8291, 8292, 8293. A system like this can achieve a 250K byte transfer rate while under DMA control.

APPLICATION EXAMPLE

This section will present the code required to operate a typical GPIB instrument set up as shown in Fig. 32. The HP5328A universal counter and the HP3325 function generator are typical of many GPIB devices; however, there are a wide variety of software protocols to be found on the GPIB. The Ziatech ZT488 GPIB analyzer is used to single step the bus to facilitate debugging the system. It also serves as a training/familiarization aid for newcomers to the bus.

This example will set up the function generator to output a specific waveform, frequency and amplitude. It will then tell the counter to measure the frequency and Request Service (SRQ) when complete. The program will then read in the data. The assembled source code will be found at the end of Appendix A.

```
SEND
  LSTN: "2", COUNT: 15, EOS: 0DH, DATA: "FU1FR37KHAM2VO (CR)"
  ;SETS UP FUNCTION GEN. TO 37 KHZ SINE, 2 VOLTS PP
  ;COUNT EQUAL TO # CHAR IN BUFFER
  ;EOS CHARACTER IS (CR) = 0DH = CARRIAGE RETURN
SEND
  LSTN: "1", COUNT: 6, EOS: "T" DATA: "PR4G7T"
  ;SETS UP COUNTER FOR P:INITIALIZE, F4: FREQ CHAN A
  ;G7:0.1 HZ RESOLUTION, T:TRIGGER AND SRQ
  ;COUNT IS EQUAL TO # CHAR
WAIT FOR SRQ
SPOL TALK: "Q", DATA: STATUS 1
  ;CLEAR THE SRO—IN THIS EXAMPLE ONLY FREQ CTR ASSERTS SRQ
RECV TALK: "Q", COUNT: 17, EOS: 0AH,
  ;GETS 17 BYTES OF DATA FROM COUNTER
  ;COUNT IS EXACT BUFFER LENGTH
  ;DATA SHOWN IS TYPICAL HP5328A READING THAT WOULD BE RECEIVED
```

Figure 32. GPIB Example Configuration
APPLICATIONS

CONCLUSION
This Application Note has shown a structured way to view the IEEE 488 bus and has given typical code sequences to make the Intel 8291, 8292, and 8293's behave as a controller of the GPIB. There are other ways to use the chip set, but whatever solution is chosen, it must be integrated into the overall system software.

The ultimate reference for GPIB questions is the IEEE Std 488, -1978 which is available from IEEE, 345 East 47th St., New York, NY, 10017. The ultimate reference for the 8292 is the source listing for it (remember it's a pre-programmed UPI-41A) which is available from INSITE, Intel Corp., 3065 Bowers Ave., Santa Clara, CA 95051.

APPENDIX A

ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0
GPIB CONTROLLER SUBROUTINES

LOC OBJ LINE SOURCE STATEMENT

1 $TITLE ('GPIB CONTROLLER SUBROUTINES')
2
3 ; GPIB CONTROLLER SUBROUTINES
4
5 ;
6 ; for Intel 8291, 8292 on ZT 7488/18
7 ; Bert Forbes, Ziatech Corporation
8 ; 2410 Broad Street
9 ; San Luis Obispo, CA, USA 93401
10 ;
11 ; General Definitions & Equates
12 ; 8291 Control Values
13 ;
14 ; 1000
15 ; ORG 1000H ; For ZT7488/18 w/8085
16 ;
17 PRT91 EQU 60H ; 8291 Base Port #
18 ;
19 ; Reg #0 Data in & Data out
20 DIN EQU PRT91+1 ; 91 Data in reg
21 DOUT EQU PRT91+0 ; 91 Data out reg
22 ;
23 ; Reg # 1 Interrupt 1 Constants
24 INT1 EQU PRT91+1 ; Int Reg 1
25 INTM1 EQU PRT91+1 ; INT Mask Reg. 1
26 BOM EQU 02 ; 91 BO INTRP Mask
27 BIM EQU 01 ; 91 BI INTRP Mask
28 ENDMK EQU 10H ; 91 END INTRP Mask
29 CPT EQU 80H ; 91 command pass thru int bit
30 ;
31 ; Reg #2 Interrupt 2
32 INT2 EQU PRT91+2
33 ;
34 ; Reg #4 Address Mode Constants
35 ADRMD EQU PRT91+4 ; 91 address mode register #
36 TON EQU 00H ; 91 talk only mode & not listen only
37 LON EQU 48H ; 91 listen only & not ton
38 TLO EQU 0C0H ; 91 talk & listen only
39 MODEL EQU 01 ; mode 1 addressing for device
40 ;
41 ; Reg #4 (Read) Address Status Register
42 ADRST EQU PRT91+4 ; reg #4
43 EOIST EQU 20H
44 TA EQU 2
45 LA EQU 1 ; listener active
46 ;
47 ; Reg #5 (Write) Auxiliary Mode Register
48 AUXMD EQU PRT91+5 ; 91 auxiliary mode register #
49 CLKRT EQU 23H ; 91 3 Mhz clock input

6-472 231324-001
APPLICATIONS

0003 50 FNHSK EQU 03 ; finish handshake command
0006 51 SDEIO EQU 06 ; send EOF with next byte
0008 52 AXRA EQU 08H ; aux. reg A pattern
0001 53 HSHSK EQU 1 ; hold off handshake on all bytes
0002 54 HOEND EQU 2 ; hold off handshake on end
0003 55 CAHCY EQU 3 ; continuous AH cycling
0004 56 EDEOS EQU 4 ; end on EOS received
0008 57 EDIS EQU 8 ; output EDI on EOS sent
000F 58 VSCMD EQU 0FH ; valid command pass through
0007 59 NYCMD EQU 07H ; invalid command pass through
00A0 60 AXRB EQU 0AH ; Aux. reg. A pattern
0001 61 CPTEN EQU 01H ; command pass thru enable
42 ;
63 ; Reg #5 (Read)
64 CPTRG EQU PRT91+5
65 ;
66 ; Reg #6 Address 0/1 reg. constants
65 67 ADR#1 EQU PRT91+6
66 68 DTDDL1 EQU 0AH ; Disable major talker & listener
66 69 DTDDL2 EQU 0E0H ; Disable minor talker & listener
70 ;
71 ; Reg #7 EOS Character Register
0067 72 EOSR EQU PRT91+7
73 ;
74 ;
75 ; 8292 CONTROL VALUES
76 ;
77 ;
78 ;
0068 79 PRT92 EQU PRT91+8 ; 8292 Base Port # (CS7)
80 ;
0068 81 INTMR EQU PRT92+0 ; 92 INTRP Mask Reg
82 INTM EQU 0A0H ; TCI
83 ;
0068 84 ERRM EQU PRT92+0 ; 92 Error Mask Reg
0001 85 TOUT1 EQU 01 ; 92 Time Out for Pass Control
0002 86 TOUT2 EQU 02 ; 92 Time Out for Standby
0004 87 TOUT3 EQU 04 ; 92 Time Out for Take Control Sync
0005 88 EVREG EQU PRT92+0 ; 92 Event Counter Pseudo Reg
0068 89 TOREG EQU PRT92+0 ; 92 Event Counter Pseudo Reg
90 ;
0069 91 CMD92 EQU PRT92+1 ; 92 Command Register
92 ;
0069 93 INTST EQU PRT92+1 ; 92 Interrupt Status Reg
0010 94 EVBIT EQU 10H ; Event Counter Bit
0002 95 IBFBT EQU 02 ; Input Buffer Full Bit
0020 96 SROBT EQU 20H ; Seg bit
97 ;
0068 98 ERFLG EQU PRT92+0 ; 92 Error Flag Pseudo Reg
0068 99 CLRST EQU PRT92+0 ; 92 Controller Status Pseudo Reg
0068 9A BUSST EQU PRT92+0 ; 92 GPIB (Bus) Status Pseudo Reg
0068 9B EVCST EQU PRT92+0 ; 92 Event Counter Status Pseudo Reg
0068 9C TOST EQU PRT92+0 ; 92 Time Out Status Pseudo Reg
103 ;
104 ; 8292 OPERATION COMMANDS
105 ;
106 ;
00F0 107 SPCNI EQU 0FH ; Stop Counter Interrupts
00F1 108 GIDL EQU 0FH ; Go to idle
00F2 109 RSET EQU 02H ; Reset
00F3 110 RSTI EQU 0FH ; Reset Interrupts
00F4 111 GSEC EQU 0FH ; Goto standby, enable counting
00F5 112 EXPP EQU 0FH ; Execute parallel poll
00F6 113 GSHA EQU 0FH ; Goto standby
00F7 114 SLOC EQU 07H ; Set local mode
00F8 115 SREM EQU 08H ; Set interface to remote
00F9 116 ABORT EQU 0FH ; Abort all operation, clear interface
00FA 117 TCNTR EQU 0AH ; Take control (Receive control)
00FC 118 TCASY EQU 0FCH ; Take control asynchronously
00FD 119 TCSY EQU 0FCH ; Take control synchronously
00FE 120 STCNI EQU 0FEH ; Start counter interrupts
121 ;
122 ;
APPLICATIONS

| 123 ; | 8292 | UTILITY COMMANDS |
| 124 ; |
| 125 ; |
| 00E1 | WOUT | EQU $E1H ;Write to timeout reg |
| 00E2 | WEVC | EQU $E2H ;Write to event counter |
| 00E3 | REV | EQU $E3H ;Read event counter status |
| 00E4 | RERF | EQU $E4H ;Read error flag reg |
| 00E5 | RINM | EQU $E5H ;Read interrupt mask reg |
| 00E6 | RCLST | EQU $E6H ;Read controller status reg |
| 00E7 | RBST | EQU $E7H ;Read GPIB bus status reg |
| 00E8 | RTOUT | EQU $E8H ;Read timeout status reg |
| 00EA | RERM | EQU $EAH ;Read error mask reg |
| 00B | IACK | EQU $08H ;Interrupt Acknowledge |
| 136 ; |
| 137 ; |
| 138 ; |
| 139 ; |
| 140 ; |
| 141 ; |
| 006F | PRTF | EQU PRTF1+0FH ;ZT7488 port 6F for interrupts |
| 0002 | TCIF | EQU $02H ;Task complete interrupt |
| 0004 | SPIF | EQU $04H ;Special interrupt |
| 0008 | OBFF | EQU $08H ;92 Output (to CPU) Buffer full |
| 0010 | IBFF | EQU $10H ;92 Input (from CPU) Buffer empty |
| 0001 | 80F | EQU $0FH ;91 Int line (BO in this case) |
| 148 ; |
| 149 ; |
| 150 ; |
| 0001 | MDA | EQU 1 ;My device address is 1 |
| 0041 | MTA | EQU MDA+40H ;My talk address is 1 ("A") |
| 0021 | MLA | EQU MDA+20H ;My listen address is 1 ("!") |
| 003F | UNL | EQU $3FH ;Universal unlisten |
| 0000 | GET | EQU $08 ;Group Execute Trigger |
| 0004 | SDC | EQU $04H ;Device Clear |
| 0018 | SPE | EQU $18H ;Serial poll enable |
| 0019 | SPD | EQU $19H ;Serial poll disable |
| 0005 | PPC | EQU $05 ;Parallel poll configure |
| 0070 | PPD | EQU $70H ;Parallel poll disable |
| 0068 | PPE | EQU $60H ;Parallel poll disable |
| 0015 | PPU | EQU $15H ;Parallel poll unconfigured |
| 0009 | TCT | EQU $09 ;Take control (pass control) |
| 164 ; |
| 165 ; |
| 166 ; |
| 147 ; |
| 158 ; |
| 161 | MACRO DEFINITIONS |
| 169 | SETF | MACRO ORA A ;Sets flags on A register |
| 170 | ENDM |
| 171 | |
| 172 ; |
| 173 | WAITO | MACRO ;Wait for last 91 byte to be done |
| 174 | LOCAL | WAITL |
| 175 | WAITL: | IN INTI ;Get INTI status |
| 176 | ANI | BOM ;Check for byte out |
| 177 | JZ | WAITL ;If not, try again |
| 178 | ENDM |
| 179 ; |
| 180 ; |
| 181 | WAITI | MACRO ;Wait for 91 byte to be input |
| 182 | LOCAL | WAITL |
| 183 | WAITL: | IN INTI ;Get INTI status |
| 184 | MOV | B,A ;Save status in B |
| 185 | ANI | 8IM ;Check for byte in |
| 186 | JZ | WAITL ;If not, just try again |
| 187 | ENDM |
| 188 ; |
| 189 | WAITX | MACRO ;Wait for 92's TCI to go false |
| 190 | LOCAL | WAITL |
| 191 | WAITL: | IN PRTF |
| 192 | ANI | TCIF |
| 193 | JNZ | WAITL |
| 194 | ENDM |
| 195 ; |

PORT P BIT ASSIGNMENTS

GPIB MESSAGES (COMMANDS)
APPLICATIONS

196 WAITT MACRO
197 LOCAL WAITL
198 WAITL: IN PRTF ;Get task complete int,etc.
199 ANI TCIF ;Mask it
200 JZ WAITL ;Wait for task to be complete
201 ENDM
202
203 RANGE MACRO LOWER,UPPER,LABEL
204 ;Checks for value in range
205 ;branches to label if not
206 ;in range. Falls through if
207 ;lower <= ( (H)(L) ) <= upper.
208 ;Get next byte.
209 MOV A,M
210 CPI LOWER
211 JM LABEL
212 CPI UPPER+1
213 JP LABEL
214 ENDM
215
216 CLRA MACRO
217 XRA A ;A XOR A =0
218 ENDM
219
220 ;All of the following routines have these common
221 ;assumptions about the state of the 8291 & 8292 upon entry
222 ;to the routine and will exit the routine in an identical state.
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243 ;INPUTS: None
244 ;OUTPUTS: None
245 ;CALLS: None
246 ;DESTROYS: A,F
247
248 3EA0
249 INIT: MVI A,INTM ;Enable TCI
250 OUT INTMR ;Output to 92's intr. mask reg
251 MVI A,DTDL1 ;Disable major talker/listener
252 OUT ADR01
253 MVI A,DTDL2 ;Disable minor talker/listener
254 OUT ADR01
255 MVI A,TON ;Talk only mode
256 OUT ADRMD
257 MVI A,CLKRT ;3 MHZ for delay timer
258 OUT AUXMD
259
260 3E80
261 AF
262 XRA A ;A XOR A =0
263 OUT INT1
264 OUT INT2 ;Disable all 91 mask hits
265 OUT AUXMD ;Immediate execute PON
266
267
268 ;SEND ROUTINE
269

6-475
APPLICATIONS

278 ;
279 ;
280 ;
281 ;
282 ;
283 ;
284 ;
285 ;
286 ;
287 ;
288 ;
289 ;
290 ;
291 ;
292 ;
293 ;
294 ;
295 ;
296 ;
297 ;
298 ;
299 ;
300 ;
301 ;
302 ;
303 ;
304 ;
305 ;
306 ;
307 ;
308 ;
309 ;
310 ;
311 ;
312 ;
313 ;
314 ;
315 ;
316 ;
317 ;
318 ;
319 ;
320 ;
321 ;
322 ;
323 ;
324 ;
325 ;
326 ;
327 ;
328 ;
329 ;
330 ;
331 ;
332 ;
333 ;
334 ;
335 ;
336 ;
337 ;
338 ;
339 ;
340 ;
341 ;
342 ;

282 SEND: MVI A, MTA ; Send MTA to turn off any
283 OUT DOUT ; previous talker
284 WAITO
285 $??001: IN INT1 ; Get Int1 status
286+ ANI BOM ; Check for byte out
287+ JZ ??001 ; If not, try again
288 MVI A, UNL ; Send universal unlisten
289 OUT DOUT ; to stop previous listeners
290 MOV A, B ; Get EOS character
291 OUT EOSR ; Output it to 8291
292 \* While listener...
293 SEND2: RANGE 20H, 3EH, SEND2 ; Check next listen address
294+ ; Checks for value in range
295+ ; branches to label if not
296+ ; in range. Falls through if
297+ ; lower <= (H) (L) <= upper.
298+ ; Get next byte.
300 MOV A, M
301 FE20; CPI 20H
302 FA4710; JM SEND2
303 FE3F; CPI 3EH+1
304 FE24710; JP SEND2
305 $??002: IN INT1 ; Get Int1 status
306+ ANI BOM ; Check for byte out
307+ JZ ??002 ; If not, try again
308 MOV A, M ; Get this listener
309 OUT DOUT ; Output to GPIB
310 INX H ; Increment listen address pointer
311 JMP SEND1 ; Loop till non-valid listen
312 WAITO ; Wait for previous listener sent
313 $??003: IN INT1 ; Get Int1 status
314+ ANI BOM ; Check for byte out
315+ JZ ??003 ; If not, try again
316+ WAITO required for early versions
317 ; Get TCI to go false
318 ; Get TCI on GTSB
319 $??004: IN PRTF ; Get task complete int, etc.
320+ ANI TCEF ; Mask it
321 $??005: IN PRTF ; Get task complete int, etc.
322+ ANI TCEF ; Mask it
323+ JZ ??005 ; Wait for task to be complete
324+ $??006: IN PRTF
325+ ; Get TCI on GTSB
326+ JNZ ??004 ; Wait for TCI to go false
327+ JNZ ??004 ; Get TCI on GTSB
328+ ??005: IN PRTF ; Get task complete int, etc.
329+ ANI TCEF ; Mask it
330+ JZ ??005 ; Wait for task to be complete
331 ;
332 ;
333 ;
334 MOV A, C ; Get count
335 SETF ; Set flags
336+ ORA A
337 JZ SEND5 ; If count=0, send no data
338 SEND3: LDAX D ; Get data byte
339 OUT DOUT ; Output to GPIB
340 CMP B ; Test EOS ... this is faster
341+ MOV A, C
342+ ; Use less code than using

6-476 231324-001
APPLICATIONS

106D CA7F10 343 JZ SEND5 ;If char = EOS, go finish
1070 DB61 344 SEND4: WAITO
1072 E602 345+??0006: IN INT1 ;Get Intl status
1074 CA7B10 346+ ANI BOM ;Check for byte out
1077 13 347+ JZ ??0006 ;If not, try again
1078 0D 348 INX D ;Increment buffer pointer
1079 C2610 349 DCR C ;Decrement count
107C C38810 350 JNZ SEND3 ;If count < > 0, go send
107F L3 351 JMP SEND5 ;Else go finish
1080 0D 352 SEND5: INX D ;for consistency
1084 DB61 353 DCR C ;"
1087 0D 354 WAITO

;This ensures that the standard entry
1081 DA61 355+??0007: IN INT1 ;Get Intl status
1083 E602 356+ ANI BOM ;Check for byte out
1085 CA8110 357+ JZ ??0007 ;If not, try again
1088 3EFD 358 SEND5: MVI A,TCF ;Take control synchronously
109A D369 359 OUT CMD92
109C 3E00 360 MVI A,AXR ;Reset send EOI on EOS
109E D365 361 OUT AUXMD 
109F DB6F 362 WAITX ;Wait for TCI false
10A2 E602 363+??0008: IN PRTF
10A4 CA8110 364+ ANI TCF
10A9 C29010 365+ JNZ ??0008
10A9 0D 366 WAIT ;Wait for TCI
10AD DB6F 367+??0009: IN PRTF ;Get task complete int, etc.
10B0 C602 368+ ANI TCIF ;Mask it
10B5 CA9710 369+ JZ ??0009 ;Wait for task to be complete
10BE C9 370 RET

*********************************************************************

;*************************************************************372

;**********************************************************************

373 ; RECEIVE ROUTINE
375 ;
376 ;
377 ; INPUT: HL talker pointer
378 ; DE data buffer pointer
379 ; C count (max buffer size) 0 implies 255
380 ; B EOS character
381 ; OUTPUT: Fills buffer pointed at by DE
382 ; CALLS: None
383 ; DESTROYS: A, BC, DE, HL, F
384 ;
385 ; RETURNS: A=0 normal termination--EOS detected
386 ; A=40 Error--- count overrun
387 ; A<40 or A>5EH Error---bad talk address
388 ;
389 ;
109F 78 390 RECV: MOV A,B ;Get EOS character
10A0 D367 391 OUT ESR ;Output it to 91
10A5 DA3911 392 RANGE 40H,5EH,RECV6
10A8 E5F5 393+ ;Checks for value in range
10AA F23911 394+ ;Branches to label if not
10AD D360 395+ ;in range. Falls through if
10AF 23 396+ ;lower <= (H)(L) <= upper.
10B0 DB61 397+ ;Get next byte.
10B2 E602 398+ MOV A,M
10B4 CA8110 399+ CPI 40H
10B5 FA3911 400+ JM RECV6
10B8 E5F5 401+ CPI 5EH+1
10BB F23911 402+ JP RECV6
10BC D360 403+ ;Valid if 40H<= talk <=5EH
10BD 23 404 OUT DOUT ;Output talker to GPIB
10BE 24 405 INX H ;Incr pointer for consistency
10BF DB61 406 WAITO

;*************************************************************370

;*************************************************************371


6-477 231324-001
### APPLICATIONS

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10C2 3E21</td>
<td>416</td>
</tr>
<tr>
<td>10C4 D350</td>
<td>417</td>
</tr>
<tr>
<td>10C6 3E86</td>
<td>418</td>
</tr>
<tr>
<td>10C8 D365</td>
<td>419</td>
</tr>
<tr>
<td>420</td>
<td>WAITO</td>
</tr>
<tr>
<td>10CA DB61</td>
<td>421+??012:</td>
</tr>
<tr>
<td>10CC E602</td>
<td>422+</td>
</tr>
<tr>
<td>10CE CACA10</td>
<td>423+</td>
</tr>
<tr>
<td>10D0 3E40</td>
<td>424</td>
</tr>
<tr>
<td>10D2 D364</td>
<td>425</td>
</tr>
<tr>
<td>426</td>
<td>CLRA ; Immediate XEO PON</td>
</tr>
<tr>
<td>10D5 AF</td>
<td>427+</td>
</tr>
<tr>
<td>10D6 D355</td>
<td>428</td>
</tr>
<tr>
<td>10D8 3F6F</td>
<td>429</td>
</tr>
<tr>
<td>10DA D359</td>
<td>430</td>
</tr>
<tr>
<td>431</td>
<td>WAITX ; Wait for TCI=0</td>
</tr>
<tr>
<td>10DC DB6F</td>
<td>432+??013:</td>
</tr>
<tr>
<td>10DE E502</td>
<td>433+</td>
</tr>
<tr>
<td>10E0 C2DC10</td>
<td>434+</td>
</tr>
<tr>
<td>435</td>
<td>WAITT ; Wait for TCI=1</td>
</tr>
<tr>
<td>10E3 DB6F</td>
<td>436+??014:</td>
</tr>
<tr>
<td>10E5 E602</td>
<td>437+</td>
</tr>
<tr>
<td>10EA DB51</td>
<td>439</td>
</tr>
<tr>
<td>10EC 47</td>
<td>440</td>
</tr>
<tr>
<td>10ED E610</td>
<td>441</td>
</tr>
<tr>
<td>10EF C20511</td>
<td>442</td>
</tr>
<tr>
<td>10F2 78</td>
<td>443</td>
</tr>
<tr>
<td>10F3 E601</td>
<td>444</td>
</tr>
<tr>
<td>10F5 CABA10</td>
<td>445</td>
</tr>
<tr>
<td>10F8 DB6A</td>
<td>446</td>
</tr>
<tr>
<td>10FA 12</td>
<td>447</td>
</tr>
<tr>
<td>10FB 13</td>
<td>448</td>
</tr>
<tr>
<td>10FC 0D</td>
<td>449</td>
</tr>
<tr>
<td>10FD C2BA10</td>
<td>450</td>
</tr>
<tr>
<td>110F 3E92</td>
<td>451</td>
</tr>
<tr>
<td>110F C31711</td>
<td>452</td>
</tr>
<tr>
<td>453 ;</td>
<td></td>
</tr>
<tr>
<td>1108 78</td>
<td>454</td>
</tr>
<tr>
<td>1108 E601</td>
<td>455</td>
</tr>
<tr>
<td>1108 C21A10</td>
<td>456</td>
</tr>
<tr>
<td>1108 DB61</td>
<td>457</td>
</tr>
<tr>
<td>110D C30511</td>
<td>458</td>
</tr>
<tr>
<td>1110 DB68</td>
<td>459</td>
</tr>
<tr>
<td>1111 C21A10</td>
<td>460</td>
</tr>
<tr>
<td>1112 12</td>
<td>461</td>
</tr>
<tr>
<td>1114 0D</td>
<td>462</td>
</tr>
<tr>
<td>1115 0600</td>
<td>463</td>
</tr>
<tr>
<td>464 ;</td>
<td></td>
</tr>
<tr>
<td>1117 3E80</td>
<td>465</td>
</tr>
<tr>
<td>1119 D369</td>
<td>466</td>
</tr>
<tr>
<td>467</td>
<td>WAITX ; Wait for TCI=0 (7 tcy)</td>
</tr>
<tr>
<td>111B D68F</td>
<td>468+??015:</td>
</tr>
<tr>
<td>111D E602</td>
<td>469+</td>
</tr>
<tr>
<td>111F C21A11</td>
<td>470+</td>
</tr>
<tr>
<td>471</td>
<td>WAITT ; Wait for TCI=1</td>
</tr>
<tr>
<td>1122 D26F</td>
<td>472+??016:</td>
</tr>
<tr>
<td>1124 E602</td>
<td>473+</td>
</tr>
<tr>
<td>1126 CA2211</td>
<td>474+</td>
</tr>
<tr>
<td>475 ;</td>
<td></td>
</tr>
<tr>
<td>476 ; if timeout 3 is to be checked, the above WAITT should</td>
<td></td>
</tr>
<tr>
<td>477 ; be omitted &amp; the appropriate code to look for TCI or</td>
<td></td>
</tr>
<tr>
<td>478 ; TOUT3 inserted here.</td>
<td></td>
</tr>
<tr>
<td>479 ;</td>
<td></td>
</tr>
<tr>
<td>1129 3E80</td>
<td>480</td>
</tr>
<tr>
<td>112A D355</td>
<td>481</td>
</tr>
<tr>
<td>112D 3E80</td>
<td>482</td>
</tr>
<tr>
<td>112F D364</td>
<td>483</td>
</tr>
<tr>
<td>1131 3E03</td>
<td>484</td>
</tr>
<tr>
<td>1133 D365</td>
<td>485</td>
</tr>
<tr>
<td>1135 AF</td>
<td>487+</td>
</tr>
<tr>
<td>1136 D365</td>
<td>488</td>
</tr>
<tr>
<td>1138 78</td>
<td>489</td>
</tr>
</tbody>
</table>

---

**6-478**

---

231324-001
APPLICATIONS

491 ;
492 ;********************************************************************************************
493 ;
494 ;
495 ;INPUTS: HL device list pointer
496 ;  B EOS character
497 ;
498 ;OUTPUTS: None
499 ;
500 ;DESTROYS: A, HL, F
501 ;RETURNS: A<>0 normal, A < > 0 bad talker
502 ;
503 ;
504 ;NOTE: XFER will not work if the talker
505 ; uses EOI to terminate the transfer.
506 ; Intel will be making hardware
507 ; modifications to the 8291 that will
508 ; correct this problem. Until that time,
509 ; only EOS may be used without possible
510 ; loss of the last data byte transfered.
511 XFER: RANGE 40H,5EH,XFER4 ;Check for valid talker
512+ ;Checks for value in range
513+ ;branches to label if not
514+ ;in range. Falls through if
515+ ;lower <= ( (H)(L) ) <= upper.
516+ ;Get next byte.

113A 7E 517+ MOV A,M
113B FE40 518+ CPI 40H
113D FA811 519+ JM XFER4
1140 FE5F 520+ CPI 5EH+1
1142 F2811 521+ JP XFER4
1145 D350 522 OUT DOUT ;Send it to GPIB
1147 23 523 INX H ;Incr pointer
1148 DB61 524 WAITO
114A E602 525+??0017: IN INT1 ;Get Int1 status
114C CA411 526+ ANI BOM ;Check for byte out
114F 3E3F 527+ JZ ??0017 ;If not, try again
1151 D350 528 OUT DOUT ;Get next byte.

115A 7E 530 XFER1: RANGE 20H,3EH,XFER2 ;Check for valid listener
115B FA61 531+ ;Checks for value in range
115C 3E3F 532+ ;branches to label if not
115D 3EH 533+ ;in range. Falls through if
115E 7E 534+ ;lower <= ( (H)(L) ) <= upper.
115F 540+ ;Get next byte.

1160 E602 541 WAITO
1162 CA511 542+??0018: IN INT1 ;Get Int1 status
1165 7E 543+ CPI 28H
1166 D360 544+ OUT DOUT ;Get listener
1167 23 545 INX H ;Incr pointer
1169 C3511 546 JMP XFER1 ;Loop until non-valid listener

1170 CA611 547 CLRA A
1173 3E87 548 CLRA A
1175 D365 549+ XRA A,AXRA+AH CY+EDEOS ;Invisible handshake
1177 3E40 550 MOV A,LY ;Continuous AH mode
1179 D364 551 OUT ADRMD ;Listen only

1181 3E6 552 MOV A,0 ;Get EOS
1183 D369 553 OUT CMD92 ;Output it to 91

6-479
APPLICATIONS

1185 DB6F
1187 E682
1189 C28511
118C DB6F
118E E682
1190 C28511
1193 DB61
1195 E610
1197 C9311
1199 3E8D
119C D369
119E DB6F
11A9 DB6F
11AC DB6F
11AD E602
11A2 C29E11
11A5 DB6F
11A7 E602
11A9 CAA511
11AC 3E80
11AE D365
11BB 3E80
11BD 3E3F
11BE D360
11BF E602
11CC 3E8D
11CD E602
11CF CAC811
11D2 7E
11D3 D360
11D5 23
11D6 C3C011
11D9 DB61
11DB E602
11DD C9D11
11ED 3E88
11EE D340
11E4 DB61
11E6 E602

564 WAITX
565+??0020 IN PRTF
566+ ANI TCIF
567+ JNZ ??0020
568 WAITT ;Wait for TCS
569+??0021 IN PRTF ;Get task complete int,etc.
570+ ANI TCIF ;Mask it
571+ JZ ??0021 ;Wait for task to be complete
572 IN XFER3 ;Get END status hit
573+ IN ENDMK ;Mask it
574+ JZ XFER3 ;Take control synchronously
575+ IN A,TCSY
576+ OUT CMD92
577+ IN A,TCSY
578+??0022 IN PRTF
579+ ANI TCIF
580+ JNZ ??0022
581+ WAITT ;Wait for TCI
582+??0023 IN PRTF ;Get task complete int,etc.
583+ ANI TCIF ;Mask it
584+ JZ ??0023 ;Wait for task to be complete
585+ OUT AUXMD ;Not cont AH or END on EOS
586+ OUT AUXMD
587+ MV1 A, AXRA ;Finish handshake
588+ OUT AUXMD
589+ MV1 A, TON ;Talk only
590+ OUT ADRMD
591+ CLRA ;Normal return A=0
592+ XRA A ;A XOR A =0
593+ OUT AUXMD ;Immediate XEQ PON
594 XFF34: RET
595+
596+ ;*****************************************************************************
597+
598+
599+ ; TRIGGER ROUTINE
600+
601+
602+;INPUTS: HL listener list pointer
603+;OUTPUTS: None
604+;CALLS: None
605+;DESTROYS: A, HL, F
606+
607+
608+;TRIG: MV1 A, UNL ;
609+;OUT DOUT ;Send universal unlisten
610+;TRIG1: RANGE 28H, 3EH, TRIG2 ;Check for valid listen
611+ ;Checks for value in range
612+ ;branches to label if not
613+ ;in range. Falls through if
614+ ;lower <= (H)(L) <= upper.
615+ ;Get next byte.
616+;TRIG2: WAITO ;Wait for UNL to finish
617+;MOV A, M
618+;CPI 20H
619+;JMP TRIG2
620+;JP TRIG2
621+;WAITO
622+??0024 IN INT1 ;Get Int1 status
623+;ANI BOM ;Check for byte out'
624+;JZ ??0024 ;If not, try again
625+;MOV A, M ;Get listener
626+;OUT DOUT ;Send Listener to GPIB
627+;INX H
628+;JMP TRIG1 ;Loop until non-valid char
629+;TRIG2: WAITO ;Wait for last listen to finish
630+??0025 IN INT1 ;Get Int1 status
631+;ANI BOM ;Check for byte out
632+;JZ ??0025 ;If not, try again
633+;MV1 A, GET ;Send group execute trigger
634+;OUT DOUT ;to all addressed listeners
635+;WAITO
636+??0026 IN INT1 ;Get Int1 status
637+;ANI BOM ;Check for byte out
APPLICATIONS

11E8 CAE411
11E9 C9
11E8+ 638+ JZ ??0026 ;If not, try again
11E9 639 RET
11E8 640 ;
11E9 641 ;*****************************************************************************
11E8 642 ;
11E9 643 ;DEVICE CLEAR ROUTINE
11E8 644 ;
11E9 645 ;
11E8 646 ;
11E9 647 ;INPUTS: HL listener pointer
11E8 648 ;OUTPUT: None
11E9 649 ;CALLS: None
11E8 650 ;DESTROYS: A, HL, F
11E9 651 ;
11EC 3E3F
11EE D360
11EC+ 652 DCLR: MVI A,UNL
11EE+ 653 OUT DOUT
11EC+ 654 DCLR1: RANGE 20H,3EH,DCLR2
11EF 7E
11F1 FE20
11F3 FA0912
11F6 FE3F
11F8 F20912
11F9 668+ MOV A,M
11FD DB61
11FF CAFB11
1202 7E
1203 D360
1205 23
1206 C3F011
1209 DB61
120B DB61
120D CA0912
1210 3E84
1212 D360
1214 DB61
1216 EB02
1218 CA1412
121B C9
121C 3E3F
121E D360
1220 DB61
1222 EB02
1224 CA0912
1227 3E21
1229 D360
122B DB61
122D EB02
122F CA2812
1232 3E18
1234 D360
1236 DB61
121C+ 655+ ;Checks for value in range
11F9+ 656+ ;branches to label if not
11F1+ 657+ ;in range. Falls through if
11F3+ 658+ ;lower <= ( H)(L) <= upper.
11F6+ 659+ ;Get next byte.
11F8+ 660+ MVI OUT RANGE
11FD+ 661+ CPI 20H
11FF+ 662+ JM DCLR2
1202+ 663+ CPI 3EH+1
1204+ 664+ JP DCLR2
1205+ 665+ WAITO
1209+ 666+ ??0027: IN INT1 ;Get IntI status
120B+ 667+ ANI BOM ;Check for byte out
120D+ 668+ JZ ??0027 ;If not, try again
1210+ 669+ MOV A,M
1212+ 670+ OUT DOUT ;Send listener to GPIB
1214+ 671+ INX H
1216+ 672 JMP DCLR1
1218+ 673 DCLR2: WAITO
121B+ 674+ ??0028: IN INT1 ;Get IntI status
121D+ 675+ ANI BOM ;Check for byte out
121F+ 676+ JZ ??0028 ;If not, try again
1221+ 677+ MVI A,SDC ;Send device clear
1223+ 678+ OUT DOUT ;To all addressed listeners
1225+ 679+ WAITO
1227+ 680+ ??0029: IN INT1 ;Get IntI status
1229+ 681+ ANI BOM ;Check for byte out
122B+ 682+ JZ ??0029 ;If not, try again
122D+ 683+ RET
121C+ 684+ ;
121E+ 685+ ;*****************************************************************************
121C+ 686+ ;
121E+ 687+ ; SERIAL POLL ROUTINE
121C+ 688+ ;
121E+ 689+ ;INPUTS: HL talker list pointer
121C+ 690+ ; DE status buffer pointer
121E+ 691+ ;OUTPUTS: Fills buffer pointed to by DE
121C+ 692+ ;CALLS: None
121E+ 693 ;DESTROYS: A, BC, DE, HL, F
121C+ 694+ ;
121E+ 695+ SPOL: MVI A,UNL ;Universal unlisten
121C+ 696+ OUT DOUT
121E+ 697+ WAITO
1220+ 698+ ??0030: IN INT1 ;Get IntI status
1222+ 699+ ANI BOM ;Check for byte out
1224+ 700+ JZ ??0030 ;If not, try again
1226+ 701+ MVI A,MLA ;My listen address
1228+ 702+ OUT DOUT
122A+ 703+ WAITO
122B+ 704+ ??0031: IN INT1 ;Get IntI status
122D+ 705+ ANI BOM ;Check for byte out
122F+ 706+ JZ ??0031 ;If not, try again
1231+ 707+ MVI A,SPE ;Serial poll enable
1233+ 708+ OUT DOUT ;To be formal about it
1235+ 709+ WAITO
1236+ 710+ ??0032: IN INT1 ;Get IntI status
APPLICATIONS

1238 BS02 711+ ANI BOM ;Check for byte out
123A CA3512 713+ MOV A,M ;Get next byte.
123D 7E 719+ MOV A,M ;Get talker
123E FE40 720+ CPI 40H
1240 FA9412 721+ JM SPOL2
1243 FE85F 722+ CPI 5EH+1
1245 F29412 723+ JP SPOL2
1248 7E 724 MOV A,M ;Get talker
1249 D360 725 OUT DOUT ;Send to GPIB
124A 23 726 INX H ;Incr talker list pointer
124C 3E40 727 MVI A,LOV ;Listen only
124E D364 728 OUT ADRMD
1250 DB561 729 WAITO ;Wait for talk address to complete
1252 E502 730+??0033: IN INT1 ;Get Int1 status
1254 CA5012 731+ ANI BOM ;Check for byte out
1257 AF 733 CLR A ;Pattern for immediate XEQ PON
1259 D365 735 OUT AUXMD ;A XOR A =0
125A 3E6F 736 MVI A,GTB ;Goto standby
125C D349 737 OUT CM902
125E 781 738 WAITX ;Wait for TCI false
1260 E502 739+??0034: IN PRTF
1262 C25E12 740+ ANI TCIF
1265 DB56F 741+??0034: IN JNZ ???0034
1267 E602 742 WAITI ;Wait for TCI
1269 CA5512 743+??0035: IN PRTF ;Get task complete int,etc.
126C DB61 744+ ANI TCIF ;Mask it
126A 47 745+??0035: IN INT1 ;Get Int1 status
126F E601 746 WAITI ;Wait for status byte input
1271 CA6C12 747+??0036: IN INT1 ;Get Int1 status
1274 3EFD 750+??0036: IN INT1 ;Get Int1 status
1276 D359 751 MVI A,TCIF
1278 DB5F 752 OUT CM902 ;Take control sync
127A 6002 753 WAITX ;Wait for TCI false
127C 7C7812 754+??0037: IN PRTF
127D 6002 755+??0037: IN PRTF
127F 7C7812 756+ JNZ ???0037
1281 E602 757 WAITI ;Wait for TCI
1283 CA7F12 758+??0038: IN PRTF ;Get task complete int,etc.
1286 D600 759+ ANI TCIF ;Mask it
1289 13 760+??0038: IN INT1 ;Get Int1 status
128A 12 761 IN D ;Get serial poll status byte
128B 12 762 STAX D ;Store it in buffer
128C D344 763 INCX D ;Incr pointer
128D 3E80 764 MVI A,TCSY ;Talk only for controller
128E AF 765 OUT ADRMD
128F D365 766 CLR A
1291 C33D12 767+ XRA A ;A XOR A =0
1292 771 768 OUT AUXMD ;Immediate XEQ PON
1294 3E19 769 JMP SPOL1 ;Go on to next device on list
1296 D360 770 ;Serial poll disable
1298 DB61 771 WAITO ;We know 80 was set (WAITO above)
129A E602 772 MVI A,SPD
129C CA9812 773 OUT DOUT
129F AF 774 WAITO
12A0 D365 775+??0039: IN INT1 ;Get Int1 status
12A2 C9 776+ ANI BOM ;Check for byte out
12A5 CA9812 777+ JZ ???0039
12A8 D360 778 CLR A
12AC 3E19 779+ XRA A ;A XOR A =0
12A0 D365 780 OUT AUXMD ;Immediate XEQ PON to clear LA
12A2 C9 781 RET
12A4 782 ;
12A6 783 ;*******************************************************
APPLICATIONS

785 ; PARALLEL POLL ENABLE ROUTINE
786 ;
787 ; INPUTS: HL listener list pointer
788 ;
789 ; OUTPUTS: None
790 ; CALLS: None
791 ; DESTROYS: A, DE, HL, F
792 ;
12A3 3E3F 794 PPEN: MVI A, UNL ; Universal unlisten
12A5 D360 795 OUT DOUT
12A6 28H, 3EH, PPEN2 ; Check for valid listener
12A7 7E 796 PPEN1: RANGE 28H, 3EH, PPEN2 ; Check for valid listener
12A8 EE20 797 ; Checks for value in range
12A9 FE3F 798 ; Branches to label if not
12AA FADB12 799 ; in range. Falls through if
12AB 8602 800 ; lower <= ( (H) (L) ) <= upper.
12AC DB61 801 ; Get next byte.
12AD 808+ ???014: MOV A, M
12AE FE20 802+ CPI 28H
12AF FADB12 803+ JM PPEN2
12AH FE3F 804+ CPI 3EH+1
12B0 DB61 805+ OUT DOUT
12B1 807 WAITO ; Valid wait 91 data out reg
12B2 808+ ???0040: IN INT1 ; Get Int1 status
12B3 E602 809+ ANI BOM ; Check for byte out
12B4 CSB212 810+ JZ ???0040 ; If not, try again
12B5 7E 811+ MOV A, M ; Get listener
12B6 D350 812+ OUT DOUT
12B7 813 WAITO
12B8 DB61 814+ ???0041: IN INT1 ; Get Int1 status
12B9 E602 815+ ANI BOM ; Check for byte out
12BA CSB212 816+ JZ ???0041 ; If not, try again
12BB 817+ MVI A, PPC ; Parallel poll configure
12BC D350 818+ OUT DOUT
12BD 819 WAITO
12BE DB61 820+ ???0042: IN INT1 ; Get Int1 status
12BF E602 821+ ANI BOM ; Check for byte out
12C0 CSB212 822+ JZ ???0042 ; If not, try again
12C1 823+ LDAX D ; Get matching configuration byte
12C2 F660 824+ ORI PPE ; Merge with parallel poll enable
12C3 D360 825+ OUT DOUT
12C4 826+ INX H ; Incr pointers
12C5 827+ INX D
12C6 C3A712 828+ JMP PPEN1 ; Loop until invalid listener char
12C7 C3A712 829 PPEN2: WAITO
12C8 DB61 830+ ???0043: IN INT1 ; Get Int1 status
12C9 EE02 831+ ANI BOM ; Check for byte out
12CA CSB212 832+ JZ ???0043 ; If not, try again
12CB C9 833+ RET
12CC 834+ ; PARALLEL POLL DISABLE ROUTINE
12CD 836+ ;
12CE 837+ INPUTS: HL listener list pointer
12CF 838+ OUTPUTS: None
12D0 839+ CALLS: None
12D1 840+ DESTROYS: A, HL, F
12D2 841+ ;
12D3 3E3F 842 PPDS: MVI A, UNL ; Universal unlisten
12D5 D350 843 OUT DOUT
12D6 844 PPDS1: RANGE 28H, 3EH, PPDS2 ; Check for valid listener
12D7 EE20 845+ ; Checks for value in range
12D8 FADB12 846+ ; Branches to label if not
12D9 FE3F 847+ ; in range. Falls through if
12DA 853+ ; lower <= ( (H) (L) ) <= upper.
12DB FADB12 848+ ; Get next byte.
12DC 849+ 850+ MOV A, M
12DD E602 851+ CPI 28H
12DE FADB12 852+ JM PPDS2
12DF FE3F 853+ CPI 3EH+1
12E0 FADB12 854+ JP PPDS2
12E1 DB61 855+ WAITO
12E2 856+ ???0044: IN INT1 ; Get Int1 status
12E3 EE02 857+ ANI BOM ; Check for byte out
12E4 CSB212 858+ JZ ???0044 ; If not, try again
12E5 7E 6-483
12E6 231324-001
## APPLICATIONS

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,M</td>
<td>Get listener</td>
</tr>
<tr>
<td>OUT DOUT</td>
<td>Incr pointer</td>
</tr>
<tr>
<td>JMP PPDS1</td>
<td>Loop until invalid listener</td>
</tr>
<tr>
<td>IN INT1</td>
<td>Get Int1 status</td>
</tr>
<tr>
<td>ANI BOM</td>
<td>Check for byte out</td>
</tr>
<tr>
<td>JZ ??045</td>
<td>If not, try again</td>
</tr>
<tr>
<td>MVI A,PPC</td>
<td>Parallel poll configure</td>
</tr>
<tr>
<td>OUT DOUT</td>
<td></td>
</tr>
<tr>
<td>IN INT1</td>
<td>Get Int1 status</td>
</tr>
<tr>
<td>ANI BOM</td>
<td>Check for byte out</td>
</tr>
<tr>
<td>JZ ??046</td>
<td>If not, try again</td>
</tr>
<tr>
<td>MVI A,PPC</td>
<td>Parallel poll disable</td>
</tr>
<tr>
<td>OUT DOUT</td>
<td></td>
</tr>
<tr>
<td>WAITO</td>
<td></td>
</tr>
<tr>
<td>IN INT1</td>
<td>Get Int1 status</td>
</tr>
<tr>
<td>ANI BOM</td>
<td>Check for byte out</td>
</tr>
<tr>
<td>JZ ??047</td>
<td>If not, try again</td>
</tr>
<tr>
<td>RET</td>
<td></td>
</tr>
</tbody>
</table>

**PARALLEL POLL UNCONFIGURE ALL ROUTINE**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPUN: MVI A,PPU</td>
<td>Parallel poll unconfigure</td>
</tr>
<tr>
<td>OUT DOUT</td>
<td></td>
</tr>
<tr>
<td>WAITO</td>
<td></td>
</tr>
<tr>
<td>IN INT1</td>
<td>Get Int1 status</td>
</tr>
<tr>
<td>ANI BOM</td>
<td>Check for byte out</td>
</tr>
<tr>
<td>JZ ??048</td>
<td>If not, try again</td>
</tr>
<tr>
<td>RET</td>
<td></td>
</tr>
</tbody>
</table>

**CONDUCT A PARALLEL POLL**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPOL: MVI A,LOM</td>
<td>Listen only</td>
</tr>
<tr>
<td>OUT ADRMD</td>
<td></td>
</tr>
<tr>
<td>CLRA</td>
<td>Immediate XEQ PON</td>
</tr>
<tr>
<td>XRA A</td>
<td>A XOR A = 0</td>
</tr>
<tr>
<td>OUT AUXMD</td>
<td>Reset TON</td>
</tr>
<tr>
<td>MVI A,EXPP</td>
<td>Execute parallel poll</td>
</tr>
<tr>
<td>OUT CMD92</td>
<td></td>
</tr>
<tr>
<td>WAITI</td>
<td>Wait for completion = BI on 91</td>
</tr>
<tr>
<td>IN INT1</td>
<td>Get INT1 status</td>
</tr>
<tr>
<td>ANI BIM</td>
<td>Check for byte in</td>
</tr>
<tr>
<td>JZ ??049</td>
<td>If not, just try again</td>
</tr>
<tr>
<td>MVI A,TON</td>
<td>Talk only</td>
</tr>
<tr>
<td>OUT ADRMD</td>
<td></td>
</tr>
<tr>
<td>CLRA</td>
<td>Immediate XEQ PON</td>
</tr>
<tr>
<td>XRA A</td>
<td>A XOR A = 0</td>
</tr>
<tr>
<td>OUT AUXMD</td>
<td>Reset LON</td>
</tr>
<tr>
<td>IN DIN</td>
<td>Get PP byte</td>
</tr>
<tr>
<td>RET</td>
<td></td>
</tr>
</tbody>
</table>

**PASS CONTROL ROUTINE**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HL pointer to talker</td>
<td></td>
</tr>
</tbody>
</table>

6-484
APPLICATIONS

933 ;CALLS: None
934 ;DESTROYS: A, HL, P
935 PCTL: RANGE 40H,5EH,PCTL1 ;Is it a valid talker?
936+ ;Checks for value in range
937+ ;branches to label if not
938+ ;in range. Falls through if
939+ ;lower <= ( (H)(L) ) <= upper.
940+ ;Get next byte.

1344 7E 941+ MOV A,M
1345 FE40 942+ CPI 40H
1347 FABA13 943+ JM PCTL1
134A FE5F 944+ CPI 5EH+1
134C F28A13 945+ JP PCTL1
134F FE41 946+ CPI MTA ;Is it my talker address
1351 CA8A13 947 JZ PCTL1 ;Yes, just return
1354 D360 948 OUT DOUT ;Send on GPIB
1356 DB61 949 WAITO

1358 E632 950+ ANI BOM ;Check for byte out
135A CA5613 952+ JZ ??0050 ;If not, try again
135D 3E89 953 MVI A,TCT ;Take control message
135F D364 954 OUT DOUT
1356 DB61 955+ WAITO

1363 E682 956+ ANI BOM ;Check for byte out
1365 CA613 958+ JZ ??0051 ;If not, try again
1368 3E81 959 MVI A,MODE1 ;Not talk only or listen only
136A D364 960 OUT ADRMD ;Enable 91 address mode 1
136C AF 961 CLRA

136D D365 962+ XRA A ;A XOR A =0
136F E681 963 OUT AUXMD ;Immediate XEQ PON
1371 C7213 964 MVI A,MDA ;My device address
1373 3E81 965 OUT ADR01 ;enabled to talk and listen
1375 D365 966 MVI A,AXRB+CPTEN ;Command pass thru enable
1377 3E81 967 OUT AUXMD

968 ;*******optional PP configuration goes here********

1379 D369 969 MVI A,GIDL ;92 go idle command
137D D369 970 OUT CMD92
137F D369 971 WAITX

1382 DB6F 972+??0052: IN PRTF ;Get next byte
137D E682 973+ ANI TCIF
137F C7213 974+ JNZ ??0052
1371 D366 975 WAITT ;Wait for TCI

1385 CA8213 976+??0053: IN PRTF ;Get task complete int, etc.
1384 E682 977+ ANI TCIF ;Mask it
1385 CA8213 978+ JZ ??0053 ;Wait for task to be complete
1389 23 979 INX H
138A C9 980 PCTL1: RET

981 ;
982 ;
983 ;******************************
984 ;
985 ;RECEIVE CONTROL ROUTINE
986 ;
987 ;INPUTS: None
988 ;OUTPUTS: None
989 ;CALLS: None
990 ;DESTROYS: A, P
991 ;RETURNS: 0= invalid (not take control to us or CPT bit not on)
992 ;< > 0 = valid take control-- 92 will now be in control
993 ;NOTE: THIS CODE MUST BE TIGHTLY INTEGRATED INTO ANY USER
994 ;SOFTWARE THAT FUNCTIONS WITH THE 8291 AS A DEVICE.
995 ;NORMALLY SOME ADVANCE WARNING OF IMPENDING PASS
996 ;CONTROL SHOULD BE GIVEN TO US BY THE CONTROLLER
997 ;WITH OTHER USEFUL INFO. THIS PROTOCOL IS SITUATION
998 ;SPECIFIC AND WILL NOT BE COVERED HERE.
999 ;
1000 ;

138B DB61 1001 RCTL: IN INT1 ;Get INT1 reg (i.e. CPT etc.)
138D E680 1002 ANI CPT ;Is command pass thru on?
138F CACT13 1003 JZ RCTL2 ;No, invalid -- go return
1392 DB65 1004 IN CPRDG ;Get command
1394 FE09 1005 CPI TCT ;Is it take control?
APPLICATIONS

1395 C2CA13 1005 JNZ RCTL1 ;No, go return invalid
1399 DB64 1007 IN ADRST ;Get address status
139A EB02 1008 ANI TA ;Is TA on?
139D CACA13 1009 JZ RCTL1 ;No -- go return invalid
13A0 3E60 1010 MVI A,DTDL1 ;Disable talker listener
13A2 D356 1011 OUT ADR01
13A4 3E80 1012 MVI A,TON ;Talk only
13A6 D354 1013 OUT ADRMD
1014 CLRA
13A8 AF 1015+ XRA A ;A XOR A = 0
13A9 D361 1016+ OUT INT1 ;Mask off INT bits
13AB D362 1017+ OUT INT2
13AD D365 1018+ OUT AUXMD
13AF 3E8A 1019 MVI A,TCNTR ;Take (receive) control 92 command
13B1 D369 1020 MVI OUT CMD92
13B3 3E0F 1021 MVI A,VSCMD ;Valid command pattern for 91
13B5 D355 1022+ OUT AUXMD

1023 ;******* optional TOUT1 check could be put here *******
1024 WAITX
1025 DB6F 1025+??0054: IN PRTF
1028 1026+ ANI TCIF
1029+ 1027+ JNZ ??0054 ;Wait for TCI
13A6 DB6F 1029+??0055: IN PRTF ;Get task complete int,etc.
13C8 E602 1030+ ANI TCIF ;Mask it
13C2 C2B13 1031+ JZ ??0055 ;Wait for task to be complete
13C5 3E89 1032 MVI A,TCT ;Valid return pattern
13C7 3CFC13 1033 JMP RCTL2 ;Only one return per routine
13CA 3E8F 1034 RCTL1: MVI A,VSCMD ;Acknowledge CPT
13CC D365 1035 OUT AUXMD
1036+ CLRA ;Error return pattern
13CE AF 1037+ XRA A ;A XOR A = 0
13CF C9 1038 RCTL2: RET

1039 ;
1040+ SRO ROUTINE
1041 ;
1042 ;
1043+
1044+ ; INPUTS: None
1045+ ; OUTPUTS: None
1046+ ; CALLS: None
1047 ; RETURNS: A = 0 no SRQ occurred
1048 ;
1049 ;
1050 ;
13D8 DB69 1051 SRQD: IN INTST ;Get 92's INTRO status
13D2 E620 1052 ANI SRQB1 ;Mask off SRQ
13D4 CAE213 1053 JZ SRQ01 ;Not set--- go return
13D7 F608 1054 ORI IACK ;Set--- must clear it with IACK
13D9 D369 1055 OUT CMD92
13DA DB69 1056+ SRQD1: IN INTST ;Get IBF
13DD E622 1057+ ANI IBFR1 ;Mask it
13DF CAD08 1058+ JZ SRQ02 ;Wait if not set
13E2 C9 1059+ SRQ02: RET
1060 ;
1061+ REMOTE ENABLE ROUTINE
1062 ;
1063+ REMOTE ENABLE ROUTINE
1064 ;
1065+ ; INPUTS: None
1066+ ; OUTPUTS: None
1067+ ; CALLS: NONE
1068+ ; DESTROYS: A, F
1069 ;
13E3 3E88 1070 RMEM: MVI A,SREM
13E5 D369 1071 OUT CMD92 ;92 asserts remote enable
1072+ WAITX ;Wait for TCI = 0
13E7 DB6F 1073+??0056: IN PRTF
13E9 E620 1074+ ANI TCIF
13EB C2B13 1075+ JNZ ??0056
1076+ WAITT ;Wait for TCI
13ED DB6F 1077+??0057: IN PRTF ;Get task complete int,etc.
13F0 E622 1078+ ANI TCIF ;Mask it
13F2 CAE213 1079+ JZ ??0057 ;Wait for task to be complete
APPLICATIONS

13F5 C9

1080  RET
1081 ;
1082 ;******************************************************************************
1083 ;
1084 ;LOCAL ROUTINE
1085 ;
1086 ;
1087 ;INPUTS: None
1088 ;OUTPUTS: None
1089 ;CALLS: None
1090 ;DESTROYS: A, F
1091 ;
13F6 3EF7
1092 LOCL: MVI A, SLOC
13F8 D359
1093 OUT CMD92 ;92 stops asserting remote enable
1094 WAITX ;Wait for TCI = 0
13FA DB6F
1095+??0058: IN PRTF
13FC E502
1096+ ANI TCIF
13FE C2FA13
1097+ JNZ ??0058
1098 WAITT ;Wait for TCI
1401 DB6F
1099+??0059: IN PRTF ;Get task complete int, etc.
1403 E502
1100+ ANI TCIF ;Mask it
1405 CA0114
1101+ JZ ??0059 ;Wait for task to be complete
1408 C9
1102 RET
1103 ;
1104 ;******************************************************************************
1105 ;
1106 ;INTERFACE CLEAR / ABORT ROUTINE
1107 ;
1108 ;
1109 ;INPUTS: None
1110 ;OUTPUTS: None
1111 ;CALLS: None
1112 ;DESTROYS: A, F
1113 ;
1114 ;
1409 3EF9
1115 IFCL: MVI A, ABORT
1410 D359
1116 OUT CMD92 ;Send IFC
1117 WAITX ;Wait for TCI = 0
140D DB6F
1118+??0060: IN PRTF
140F E502
1119+ ANI TCIF
1411 C20D14
1120+ JNZ ??0060
1121 WAITT ;Wait for TCI
1414 DB6F
1122+??0061: IN PRTF ;Get task complete int, etc.
1416 E602
1123+ ANI TCIF ;Mask it
1418 CA1414
1124+ JZ ??0061 ;Wait for task to be complete
1125 ;Delete both WAITX & WAITT if this routine
1126 ;is to be called while the 9292 is
1127 ;Controller-in-Charge. If not C.I.C. then
1128 ;TCI is set, else nothing is set (IFC is sent)
1129 ;and the WAIT'S will hang forever
1130 RET
1132 ;
APPLICATIONS

APPLICATION EXAMPLE CODE FOR 8085

1134 ;
0032 1135 FCDNL EQU '2' ; Func gen device num "2" ASCII, lsn
0031 1136 FCDNL EQU '1' ; Freq ctr device num "1" ASCII, lsn
0051 1137 FCDNT EQU 'Q' ; Freq ctr talk address
000D 1138 CR EQU $DH ; ASCII carriage return
000A 1139 LF EQU $AH ; ASCII line feed
00FF 1140 LEND EQU $FFH ; List end for Talk/Listen lists
0040 1141 SRQM EQU $40H ; Bit indicating device sent SRQ

1142 ;
1143 FCDATA: DB 'FU1PR37THAM2VO',CR ; Data to set up func. gen
1144 LIM1 EQU 15 ; Buffer length
1145 FCDATA: DB 'PF4G7T' ; Data to set up freq ctr
1146 LIM2 EQU 6 ; Buffer length
1147 LL1: DB FCDNL, LEND ; Listen list for freq ctr
1148 LL2: DB FCDNL, LEND ; Listen list for func. gen
1149 TL1: DB FCDNT, LEND ; Talk list for freq ctr
1150 ;
1151 ; SETUP FUNCTION GENERATOR
1152 MVI B, CR ; EOS
1153 MVI C, LIM1 ; Count
1154 LXI D, FCDATA ; Data pointer
1155 LXI H, LL2 ; Listen list pointer
1156 CALL SEND ;
1157 ;
1158 ; SETUP FREQ COUNTER
1159 ;
1160 MVI B, 'T' ; EOS
1161 MVI C, LIM2 ; Count
1162 LXI D, FCDATA ; Data pointer
1163 LXI H, LL1 ; Listen list pointer
1164 CALL SEND
1165 ;
1166 ; WAIT FOR SRQ FROM FREQ CTR
1167 ;
1168 LOOP: CALL SRQD ; Has SRQ occurred ?
1169 JZ LOOP ; No, wait for it
1170 ;
1171 ; SERIAL POLL TO CLEAR SRQ
1172 ;
1173 LXI D, SPBYTE ; Buffer pointer
1174 LXI H, TL1 ; Talk list pointer
1175 CALL SPOL ; Get status byte
1176 DCX D ; Backup buffer pointer to ctr byte
1177 LDAX D ; Ctr should have said yes
1178 ANI SRQM ; Did ctr assert SRQ ?
1179 JZ ERROR ; Ctr should have said yes
1180 ;
1181 ; RECEIVE READING FROM COUNTER
1182 ;
1183 MVI B, 'L' ; EOS
1184 MVI C, LIM3 ; Count
1185 LXI H, TL1 ; Talk list pointer
1186 LXI D, FCDATI ; Data in buffer pointer
1187 CALL RECVC
1188 JMZ ERROR
1189 ;
1190 ;******* rest of user processing goes here *****
1191 ;
1192 ;
1193 ERROR: NOP ; User dependent error handling
1194 ;
1195 ORG 3C00H
1196 SPBYTE: DS 1 ; Location for serial poll byte
0011 1197 LIM3 EQU 17 ; Max freq counter input

6-488 231324-001
APPLICATIONS

3C01  1198 FCDATI  DS  LIM3  fFreq ctr input buffer
1199  END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

ABORT A 0009  ADH01 A 0056  ADDW A 0054  ADDR A 0054  AUXMD A 0054  AXRA A 0088  AXRB A 0088
SIM A 0081  HOF A 0001  BOM A 0092  BUSST A 0088  CANCY A 0089  CLKRT A 0023  CLRA A 0097
CLAST A 0088  CMD92 A 0089  CPT A 0088  CPTEN A 0089  CR A 0089  DCL A 0014
DCLA A 118C  DCLR1 A 118F  DCLR2 A 118F  DIY A 0094  DOT A 0086  DTL1 A 0048  DTLD2 A 0080
EDEOS A 0004  EENDM A 0010  EOIS A 0086  EOIST A 0086  EOSR A 0087  ERFLG A 0058  ERR4 A 0068
ERROR A 1477  EVBH A 0018  EVCST A 0068  EVREG A 0068  EXPP A 00F5  FCDATA A 142A  FCDATI A 3C01
FCMHL A 0031  FCMHL A 0031  FCMHL A 0031  FCMHL A 0031  FCMHL A 0031  FCMHL A 0031  FCMHL A 0031
GSKE A 0084  GTSB A 0075  HSend A 0082  HSend A 0082  IACK A 0088  IFBST A 0092  IFBST A 0092
IFCL A 1489  INIT A 1000  INT1 A 0061  INT2 A 0062  INTM A 0088  INTM1 A 0061  INTM1 A 0061
INITST A 0069  LA A 0081  LEN0 A 007F  LF A 008A  LIM1 A 008F  LIM2 A 0084  LIM3 A 0011
L11 A 1431  LL2 A 1433  LOCL A 13F8  LOOP A 1451  LOOP A 1451  LOOP A 1451  LOOP A 1451
MODEL A 0001  MTA A 0041  IVCM4 A 0047  ORFF A 0088  PCTL A 1344  PCTL1 A 138A  PCC A 0085
PPD A 0078  PPDS A 1280  PPDS1 A 1284  PPDS2 A 128D  PPE A 00F9  PPEX A 12A3  PPEX1 A 12A7
PFEM2 A 1208  PPOL A 1327  PRU A 0015  PPUN A 131A  PRT91 A 006A  PRT92 A 0068  PRF P 00F6
RANGE + 0095  RSET A 0067  RCST A 0065  RCTL A 1388  RCTL2 A 13CA  RCTRL A 13CA  RCTRL A 13CA
RECV1 A 104A  REC2 A 1195  REC3 A 1195  REC4 A 1195  REC5 A 1177  REC6 A 1139  RCLK A 1383
RERF A 0004  REM A 008A  REV C A 00E3  RINM A 0055  SSET A 00F2  STI A 0079  STOUT A 0084
SEND1 A 1046  SEND2 A 1046  SEND3 A 1046  SEND4 A 1046  SEND5 A 1046  SCK A 0071  SDE A 0018
SIPF A 0004  SPOL A 121C  SPOL1 A 123D  SPOL2 A 1294  SPON A 0074  SQRT A 007A  SQRD A 1300
SRQO A 1308  SRQ2 A 1362  SRDM A 0048  STCNI A 00FE  TA A 0092  TCSY A 00FC  TCF A 1002
TOST A 0086  TOUT1 A 0011  TOUT2 A 0092  TOUT3 A 0094  TRIG A 11BC  TRIG1 A 11CB  TRIG2 A 11DB
UXL A 003F  VSCMD A 000F  WAITI A 0002  WAITO A 0001  WAITT A 0004  WAITX A 0003  WEVC A 00E2
WOUT A 0081  XFER1 A 113A  XFER1 A 1153  XFER2 A 115C  XFER3 A 1193  XFER4 A 119A

ASSEMBLY COMPLETE,  NO ERRORS

6-489  231324-001
APPENDIX B

TEST CASES FOR THE SOFTWARE DRIVERS

The following test cases were used to exercise the software routines and to check their action. To provide another device/controller on the GPIB a ZT488 GPIB Analyzer was used. This analyzer acted as a talker, listener or another controller as needed to execute the tests. The sequence of outputs are shown with each test. All numbers are hexadecimal.

SEND TEST CASES

<table>
<thead>
<tr>
<th>B = 44</th>
<th>C = 30</th>
<th>DE = 3E80</th>
<th>HL = 3E70</th>
<th>3E70: 20 30 3E 3F</th>
</tr>
</thead>
</table>

GPIB output:
- 41 ATN 3F ATN 20 ATN 30 ATN 3E ATN 11 44 44 EOI

Ending B = 44
Ending C = 2E
Ending DE = 3E82
Ending HL = 3E73

RECEIVE TEST CASES

<table>
<thead>
<tr>
<th>B = 44</th>
<th>C = 30</th>
<th>DE = 3E80</th>
<th>HL = 3E70</th>
<th>3E70: 5E</th>
</tr>
</thead>
</table>

GPIB output:
- 40 ATN 50 ATN 5E ATN 3F ATN 3F ATN 21 ATN 21 ATN 21 ATN 21 ATN

ZT488 Data
- 1 2 3 4 44,EOI 44 5,EOI

Ending A = 0
Ending B = 0
Ending C = 2B
Ending DE = 3E85
Ending HL = 3E71

SERIAL POLL TEST CASES

| C = 30 | DE = 3E80 | HL = 3E70 | 3E70: 40 50 5E 5F |

GPIB output:
- 3F ATN 21 ATN 18 ATN

6-490
APPLICATIONS

<table>
<thead>
<tr>
<th>GPIB output:</th>
<th>3F ATN</th>
<th>19 ATN</th>
</tr>
</thead>
<tbody>
<tr>
<td>output: 21 ATN</td>
<td>Ending C = 30</td>
<td></td>
</tr>
<tr>
<td>output: 18 ATN</td>
<td>Ending DE = 3E80</td>
<td></td>
</tr>
<tr>
<td>output: 40 ATN</td>
<td>Ending HL = 3E70</td>
<td></td>
</tr>
<tr>
<td>input*: 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>output: 50 ATN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>input*: 41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>output: 5E ATN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>input*: 7F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>output: 19 ATN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*NOTE: leave ZT488 in single step mode even on input
Ending C = 30
Ending DE = 3E83
Ending HL = 3E73
Ending 3E80: 00 41 7F

PASS CONTROL TEST CASES

| HL = 3E70  | 3E70  | 3E70  |
| 3E70: 40  | 41(MTA) 5F |

GPIB output: 40 ATN
09 ATN
ATN
Ending HL = 3E71 3E70 3E70
Ending A = 02 41(MTA) 5F

RECEIVE CONTROL TEST CASES

GPIB input

Run Receive Control
GPIB Input
Ending A =

PARALLEL POLL ENABLE TEST CASES

| DE = 3E80  | 3E80  |
| HL = 3E70  | 3E70  |
| 3E70: 20 30 3E 3F | 3F |
| 3E80: 01 02 03 |

GPIB output: 3F ATN 3F ATN
20 ATN
05 ATN
61 ATN
30 ATN
05 ATN
62 ATN
3E ATN
05 ATN
63 ATN

Ending DE = 3E83 3E80
Ending HL = 3E73 3E70

PARALLEL POLL DISABLE TEST CASES

| HL = 3E70  | 3E70  |
| 3E70: 20 30 3E 3F | 3F |

6-491
231324-001
APPLICATIONS

GPIB output: 3F ATN 3F ATN
20 ATN 05 ATN
30 ATN 70 ATN
3E ATN
05 ATN
70 ATN

Ending HL = 3E73 3E70

PARALLEL POLL UNCONFUGURE TEST CASE

GPIB output: 15 ATN

PARALLEL POLL TEST CASES

Set DIO # 1 2 3 4 5 6 7 8 None
Ending A 1 2 4 8 10 20 40 80 0

SRQ TEST

Set SRQ momentarily Reset SRQ
Ending A = 02 00

TRIGGER TEST

HL = 3E70
DE = 3E80
BC = 4430
3E70: 20 30 3E 3F

GPIB output: 3F ATN
20 ATN
30 ATN
3E ATN
08 ATN

Ending HL = 3E73
DE = 3E80
BC = 4430

DEVICE CLEAR TEST

HL = 3E70
DE = 3E80
BC = 4430
3E70: 20 30 3E 3F

GPIB output: 3F ATN
20 ATN
30 ATN
3E ATN
14 ATN

Ending HL = 3E73
DE = 3E80
RC = 4430

6-492 231324-001
APPLICATIONS

XFER TEST

B = 44
HL = 3E70
3E70: 40 20 30 3E 3F
GPIB output: 40 ATN
3F ATN
20 ATN
30 ATN
3E ATN
GPIB input: 0
1
2
3
44
Ending A = 0
B = 44
HL = 3E74

APPLICATION EXAMPLE
GPIB OUTPUT/INPUT

GPIB output: 41 ATN
3F ATN
32 ATN
46
55
31
46
52
33
37
4B
48
41
4D
32
56
4F
0D EOI
41 ATN
3F ATN
31 ATN
50
46
34
47
37
54 EOI
GPIB input: SRQ
GPIB output: 3F ATN
21 ATN
18 ATN
51 ATN
GPIB input: 40 SRQ
GPIB output: 19 ATN
51 ATN
# APPLICATIONS

3F ATN  
21 ATN

GPIB input:  
20  
2B  
20  
20  
33  
37  
30  
30  
30  
2E  
30  
45  
2B  
30  
0D  
0A

GPIB output: XX ATN

---

## APPENDIX C

### REMOTE MESSAGE CODING

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Message Name</th>
<th>C</th>
<th>T</th>
<th>D</th>
<th>ID</th>
<th>NN</th>
<th>Bus Signal Line(s) and Coding That Asserts the True Value of the Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACG</td>
<td>addressed command group</td>
<td>M</td>
<td>AC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0 X X X X XXX</td>
</tr>
<tr>
<td>ATN</td>
<td>attention</td>
<td>U</td>
<td>UC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X X X X XXX</td>
</tr>
<tr>
<td>DAB</td>
<td>data byte</td>
<td>(Notes 1, 9)</td>
<td>M</td>
<td>DD</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>DAC</td>
<td>data accepted</td>
<td>U</td>
<td>HS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X X X X XXX</td>
</tr>
<tr>
<td>DAV</td>
<td>data valid</td>
<td>U</td>
<td>HS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X X X X XXX</td>
</tr>
<tr>
<td>DCL</td>
<td>device clear</td>
<td>M</td>
<td>UC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>END</td>
<td>end</td>
<td>U</td>
<td>ST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X X X XXX</td>
</tr>
<tr>
<td>EOS</td>
<td>end of string</td>
<td>(Notes 2, 9)</td>
<td>M</td>
<td>DD</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>GET</td>
<td>group execute trigger</td>
<td>M</td>
<td>AC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>GTL</td>
<td>go to local</td>
<td>M</td>
<td>AC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IDY</td>
<td>identify</td>
<td>U</td>
<td>UC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X X X XXX</td>
</tr>
<tr>
<td>IFC</td>
<td>interface clear</td>
<td>U</td>
<td>UC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X X X XXX</td>
</tr>
<tr>
<td>LAG</td>
<td>listen address group</td>
<td>M</td>
<td>AD</td>
<td>Y</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>LLO</td>
<td>local lock out</td>
<td>M</td>
<td>UC</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MLA</td>
<td>my listen address</td>
<td>(Note 3)</td>
<td>M</td>
<td>AD</td>
<td>Y</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MTA</td>
<td>my talk address</td>
<td>(Note 4)</td>
<td>M</td>
<td>AD</td>
<td>Y</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MSA</td>
<td>my secondary address</td>
<td>(Note 5)</td>
<td>M</td>
<td>SE</td>
<td>Y</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(Note 3):  

(Note 4):  

(Note 5):  

---

6-494  
231324-001
## APPLICATIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Message Name</th>
<th>C</th>
<th>D</th>
<th>D</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>null byte</td>
<td>M</td>
<td>D</td>
<td>D</td>
<td>0 0 0 0 0 0 XXX X X X X X</td>
</tr>
<tr>
<td>OSA</td>
<td>other secondary address</td>
<td>M</td>
<td>S</td>
<td>E</td>
<td>(OSA = SCG ∧ MSA)</td>
</tr>
<tr>
<td>OTA</td>
<td>other talk address</td>
<td>M</td>
<td>A</td>
<td>D</td>
<td>(OTA = TAG ∧ MTA)</td>
</tr>
<tr>
<td>PCG</td>
<td>primary command group</td>
<td>M</td>
<td>—</td>
<td>—</td>
<td>(PCG = ACG V UCG V LAG V TAG)</td>
</tr>
<tr>
<td>PPC</td>
<td>parallel poll configure</td>
<td>M</td>
<td>C</td>
<td>Y</td>
<td>0 0 0 1 0 1 XXX 1 X X X X</td>
</tr>
<tr>
<td>PPE</td>
<td>parallel poll enable</td>
<td>M</td>
<td>E</td>
<td>Y</td>
<td>1 1 1 S P P P XXX 1 X X X X</td>
</tr>
<tr>
<td>PPD</td>
<td>parallel poll disable</td>
<td>M</td>
<td>S</td>
<td>E</td>
<td>Y 1 1 1 D D D 4 3 2 1 XXX 1 X X X X</td>
</tr>
<tr>
<td>PPR1</td>
<td>parallel poll response 1</td>
<td>U</td>
<td>T</td>
<td>X</td>
<td>X X X X X X X 1 XXX 1 1 X X X</td>
</tr>
<tr>
<td>PPR2</td>
<td>parallel poll response 2</td>
<td>U</td>
<td>T</td>
<td>X</td>
<td>X X X X X X X X 1 XXX 1 1 X X X</td>
</tr>
<tr>
<td>PPR3</td>
<td>parallel poll response 3</td>
<td>U</td>
<td>T</td>
<td>X</td>
<td>X X X X X X X X 1 XXX 1 1 X X X</td>
</tr>
<tr>
<td>PPR4</td>
<td>parallel poll response 4</td>
<td>U</td>
<td>T</td>
<td>X</td>
<td>X X X X X X X X 1 XXX 1 1 X X X</td>
</tr>
<tr>
<td>PPR5</td>
<td>parallel poll response 5</td>
<td>U</td>
<td>T</td>
<td>X</td>
<td>X X X X X X X X 1 XXX 1 1 X X X</td>
</tr>
<tr>
<td>PPR6</td>
<td>parallel poll response 6</td>
<td>U</td>
<td>T</td>
<td>X</td>
<td>X X X X X X X X 1 XXX 1 1 X X X</td>
</tr>
<tr>
<td>PPR7</td>
<td>parallel poll response 7</td>
<td>U</td>
<td>T</td>
<td>X</td>
<td>X X X X X X X X 1 XXX 1 1 X X X</td>
</tr>
<tr>
<td>PPR8</td>
<td>parallel poll response 8</td>
<td>U</td>
<td>T</td>
<td>X</td>
<td>X X X X X X X X 1 XXX 1 1 X X X</td>
</tr>
<tr>
<td>PPU</td>
<td>parallel poll unconfigure</td>
<td>M</td>
<td>C</td>
<td>Y</td>
<td>0 0 1 0 1 0 1 XXX 1 X X X X</td>
</tr>
<tr>
<td>REN</td>
<td>remote enable</td>
<td>U</td>
<td>U</td>
<td>C</td>
<td>X X X X X X X X 1 XXX 1 X X X</td>
</tr>
<tr>
<td>RDF</td>
<td>ready for data</td>
<td>U</td>
<td>H</td>
<td>S</td>
<td>X X X X X X X X 000 X X X X</td>
</tr>
<tr>
<td>RQS</td>
<td>request service</td>
<td>U</td>
<td>T</td>
<td>X</td>
<td>X X X X X X X 000 X X X X</td>
</tr>
<tr>
<td>SCG</td>
<td>secondary command group</td>
<td>M</td>
<td>E</td>
<td>Y</td>
<td>1 1 1 X X X X XXX 1 X X X X</td>
</tr>
<tr>
<td>SDC</td>
<td>selected device clear</td>
<td>M</td>
<td>A</td>
<td>C</td>
<td>Y 0 0 0 1 0 0 0 XXX 1 X X X X</td>
</tr>
<tr>
<td>SPD</td>
<td>serial poll disable</td>
<td>M</td>
<td>U</td>
<td>C</td>
<td>Y 0 0 1 1 0 0 0 XXX 1 X X X X</td>
</tr>
<tr>
<td>SPE</td>
<td>serial poll enable</td>
<td>M</td>
<td>U</td>
<td>C</td>
<td>Y 0 0 1 1 0 0 0 XXX 1 X X X X</td>
</tr>
<tr>
<td>SRQ</td>
<td>service request</td>
<td>U</td>
<td>T</td>
<td>X</td>
<td>X X X X X X X X 000 X X X X</td>
</tr>
<tr>
<td>STB</td>
<td>status byte</td>
<td>M</td>
<td>S</td>
<td>T</td>
<td>S X S S S S S S S XXX 0 X X X X X</td>
</tr>
<tr>
<td>TCT</td>
<td>take control</td>
<td>M</td>
<td>A</td>
<td>C</td>
<td>Y 0 0 1 1 0 0 1 XXX 1 X X X X</td>
</tr>
<tr>
<td>TAG</td>
<td>talk address group</td>
<td>M</td>
<td>A</td>
<td>D</td>
<td>Y 0 1 X X X X XXX 1 X X X X</td>
</tr>
<tr>
<td>UCG</td>
<td>universal command group</td>
<td>M</td>
<td>U</td>
<td>C</td>
<td>Y 0 0 1 1 X X X XXX 1 X X X X</td>
</tr>
<tr>
<td>UNL</td>
<td>unlisten</td>
<td>M</td>
<td>A</td>
<td>D</td>
<td>Y 0 1 1 1 1 1 XXX 1 X X X X</td>
</tr>
<tr>
<td>UNT</td>
<td>untalk</td>
<td>M</td>
<td>A</td>
<td>D</td>
<td>Y 1 0 1 1 1 1 XXX 1 X X X X</td>
</tr>
</tbody>
</table>

The I/O coding on ATN when sent concurrent with multiline messages has been added to this revision for interpretive convenience.

**NOTES:**

1. D1-D8 specify the device dependent data bits.
2. E1-E8 specify the device dependent code used to indicate the EOS message.
3. L1-L5 specify the device dependent bits of the device's listen address.
4. T1-T5 specify the device dependent bits of the device's talk address.
5. S1-S5 specify the device dependent bits of the device's secondary address.
6. S specifies the sense of the PPR.

<table>
<thead>
<tr>
<th>S</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

P1-P3 specify the PPR message to be sent when a parallel poll is executed.

The following table shows the PPR Message and its corresponding P3, P2, and P1 values.

<table>
<thead>
<tr>
<th>P3</th>
<th>P2</th>
<th>P1</th>
<th>PPR Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>PPR1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>PPR8</td>
</tr>
</tbody>
</table>

(7) D1-D4 specify don't-care bits that shall not be decoded by the receiving device. It is recommended that all zeroes be sent.
(8) S1-S6, S8 specify the device dependent status. (DIO7 is used for the RQS message.)
(9) The source of the message on the ATN line is always the C function, whereas the messages on the DIO and EOI lines are enabled by the T function.
(10) The source of the messages on the ATN and EOI lines is always the C function, whereas the source of the messages on the DIO lines is always the PP function.
(11) This code is provided for system use, see 6.3.
USING THE 8291A GPIB TALKER/LISTENER
INTRODUCTION

This application note explains the Intel® 8291A GPIB (General Purpose Interface Bus) Talker/Listener as a component, and shows its use in GPIB interface design tasks.

The first section of this note presents an overview of IEEE 488 (GPIB). The second section introduces the Intel® GPIB component family. A detailed explanation of the 8291A follows. Finally, some application examples using the component family are presented.

Figure 1. Interface Capabilities and Bus Structure
OVERVIEW OF IEEE 488/GPIB

The GPIB is a parallel interface bus with an asynchronous Interlocking data exchange handshake mechanism. It is designed to provide a common communication interface among devices over a maximum distance of 20 meters at a maximum speed of 1 Mbps. Up to 15 devices may be connected together. The asynchronous interlocking handshake dispenses with a common synchronization clock, and allows intercommunication among devices capable of running at different speeds. During any transaction, the data transfer occurs at the speed of the slowest device involved.

The GPIB finds use in a diversity of applications requiring communication among digital devices over short distances. Common examples are: programmable instrumentation systems, computer to peripherals, etc.

The interface is completely defined in the IEEE Std. 488-1978.

A typical implementation consists of logical devices which talk (talker), listen (listeners), and control GPIB activity (controllers).

Interface Functions

The interface between any device and the bus may have a combination of several different capabilities (called "functions"). Among a total of ten functions defined, the Talker, Listener, Source Handshake, Acceptor Handshake and Controller are the more common examples. The Talker function allows a device to transmit data. The Listener function allows reception. The Source and Acceptor Handshakes, synchronized with the Talker and Listener functions respectively, exchange the handshake signals that coordinate data transfer. The Controller function allows a device to activate the interface functions of the various devices through commands. Other interface functions are: Service request, Remote local, Parallel poll, Device clear and Device trigger. Each interface may not contain all these functions. Further, most of these functions may be implemented to various levels (called "subsets") of capability. Thus, the overall capability of an interface may be tailored to the needs of the communicating device.

Electrical Signal Lines

As shown in Figure 1, the GPIB is composed of eight data lines (D08-D01), five interface management lines (IFC, ATN, SRQ, REN, EOI), and three transfer control lines (DAV, NRFD, NDAC).

The eight data lines are used to transfer data and commands from one device to another with the help of the management and control lines. Each of the five interface management lines has a specific function.

ATN (attention) is used by the Controller to indicate that it (the controller) has access to the GPIB and that its output on the data lines is to be interpreted as a command. ATN is also used by the controller along with EOI to indicate a parallel poll.

SRQ (service request) is used by a device to request service from the controller.

REN (remote enable) is used by the controller to specify the command source of a device. A device can be issued commands either locally through its front panel or by the controller.

EOI (end or identify) may be used by the controller as well as a talker. A controller uses EOI along with ATN to demand a parallel poll. Used by a talker, EOI indicates the last byte of a data block.

IFC (interface clear) forces a complete GPIB interface to the idle state. This could be considered the GPIB's "interface reset." GPIB architecture allows for more than one controller to be connected to the bus simultaneously. Only one of these controllers may be in command at any given time. This device is known as the controller-in-charge. Control can be passed from one controller to another. Only one among all the controllers present on a bus can be the system controller. The system controller is the only device allowed to drive IFC.

Transfer Control Lines

The transfer control lines conduct the asynchronous interlocking three-wire handshake.

DAV (data valid) is driven by a talker and indicates that valid data is on the bus.

NRFD (not ready for data) is driven by the listeners and indicates that not all listeners are ready for more data.

NDAC (not data accepted) is used by the listeners to indicate that not all listeners have read the GPIB data lines yet.

The asynchronous 3-wire handshake flowchart is shown in Figure 2. This is a concept fundamental to the asynchronous nature of the GPIB and is reviewed in the following paragraphs.

Assume that a talker is ready to start a data transfer. At the beginning of the handshake, NRFD is false indicating that the listener(s) is ready for data. NDAC is true indicating that the listener(s) has not accepted the data, since no data has been sent yet. The talker places data on the data lines, waits for the required settling time, and then indicates valid data by driving DAV true. All active listeners drive NRFD true indicating that they are not
Figure 2. Handshake Flowchart

FLOW DIAGRAM OUTLINES SEQUENCE OF EVENTS DURING TRANSFER OF DATA BYTE. MORE THAN ONE LISTENER AT A TIME CAN ACCEPT DATA BECAUSE OF LOGICAL CONNECTION OF NRFD AND NDAC LINES.
ready for more data. They then read the data and drive NDAC false to indicate acceptance. The talker responds by deasserting DAV and readies itself to transfer the next byte. The listeners respond to DAV false by driving NDAC true. The talker can now drive the data lines with a new data byte and wait for NRFD to be false to start the next handshake cycle.

**Bus Commands**

When ATN and DAV are true data patterns which have been placed by the controller on the GPIB, they are interpreted as commands by the other devices on the interface. The GPIB standard contains a repertory of commands such as MTA (My Talk Address), MSA (My Secondary Address), SPE (Serial Poll Enable), etc. All other patterns in conjunction with ATN and DAV are classified as undefined commands and their meaning is user-dependent.

**Addressing Techniques**

To allow the controller to issue commands selectively to specific devices, three types of addressing exist on the GPIB: talk only/listen only (ton/Ion), primary, and secondary.

Ton/Ion is a method where the ability of the GPIB interface to talk or listen is determined by the device and not by the GPIB controller. With this method, fixed roles can be easily designated in simple systems where reassignment is not necessary. This is appropriate and convenient for certain applications. For example, a logic analyzer might be interfaced via the GPIB to a line printer in order to document some type of failure. In this case, the line printer simply listens to the logic analyzer, which is a talker.

The controller addresses devices through three commands, MTA (my talk address), MLA (my listen address), and MSA (my secondary address). The device address is imbedded in the command bit pattern. The device whose address matches the imbedded pattern is enabled. Some devices may have the same logical talk and listen addresses. This is allowable since the talker and listener are separate functions. However, two of the same functions cannot have the same address.

In primary addressing, a device is enabled to talk (listen) by receiving the MTA (MLA) message.

Secondary addressing extends the address field from 5 to 10 bits by allowing an additional byte. This additional byte is passed via the MSA message. Secondary addressing can also be used to logically divide devices into various subgroups. The MSA message applies only to the device(s) whose primary address immediately precede it.

**INTEL’S® GPIB COMPONENTS**

The logic designer implementing a GPIB interface has, in the past, been faced with a difficult and complex discrete logic design. Advances in LSI technology have produced sophisticated microprocessor and peripheral devices which combine to reduce this once complex interface task to a system consisting of a small set of integrated circuits and some software drivers. A microprocessor hardware/software solution and a high-level language source code provide an additional benefit in end-product maintenance. Product changes are a simple matter of revising the product software. Field changes are as easy as exchanging EPROMS.

Intel® has provided an LSI solution to GPIB interfacing with a talker/listener device (8291A), a controller device (8292), and a transceiver (8293). An interface with all capabilities except for the controller function can be built with an 8291A and a pair of 8293’s. The addition of the 8292 produces a complete interface. Since most devices in a GPIB system will not have the controller function capability, this modular approach provides the least cost to the majority of interface designs.

**Overview of the 8291A GPIB Talker/Listener**

The Intel® 8291A GPIB Talker/Listener operates over a clock range of 1 to 8 MHz and is compatible with the MCS-85, iAPX-86, and 8051 families of microprocessors.

A detailed description of the 8291A is given in the data sheet.

The 8291A implements the following functions: Source Handshake (SH), Acceptor Handshake (AH), Talker Extended (TE), Service Request (SRQ), Listener Extended (LE), Remote/Local (RL), Parallel Poll (PP2), Device Clear (DC), and Device Trigger (DT).

Current states of the 8291A can be determined by examining the device’s status read registers. In addition, the 8291A contains 8 write registers. These registers are shown in Figure 3. The three register select pins RS3-RS0 are used to select the desired register.

The data — in register moves data from the GPIB to the microprocessor or to memory when the 8291A is addressed to listen. When the 8291A is addressed to talk, it uses the data -out register to move data onto the GPIB. The serial poll mode and status registers are used to request service and program the serial poll status byte.

A detailed description of each of the registers, along with state diagrams can be found in the 8291A data sheet.
<table>
<thead>
<tr>
<th>READ REGISTERS</th>
<th>REGISTER SELECT CODE</th>
<th>WRITE REGISTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RS2 RS1 RS0</strong></td>
<td><strong>RS2 RS1 RS0</strong></td>
<td><strong>RS2 RS1 RS0</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA IN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPT</th>
<th>APT</th>
<th>GET</th>
<th>END</th>
<th>DEC</th>
<th>ERR</th>
<th>BO</th>
<th>BI</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>CPT</th>
<th>APT</th>
<th>GET</th>
<th>END</th>
<th>DEC</th>
<th>ERR</th>
<th>BO</th>
<th>BI</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERRUPT STATUS 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INT</th>
<th>SPAS</th>
<th>LLO</th>
<th>REM</th>
<th>SPC</th>
<th>LLOC</th>
<th>REMC</th>
<th>ADSC</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>DMA0</th>
<th>DMA1</th>
<th>SPC</th>
<th>LLOC</th>
<th>REMC</th>
<th>ADSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERRUPT STATUS 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S8</th>
<th>SRQS</th>
<th>S6</th>
<th>S5</th>
<th>S4</th>
<th>S3</th>
<th>S2</th>
<th>S1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>S8</th>
<th>RSV</th>
<th>S6</th>
<th>S5</th>
<th>S4</th>
<th>S3</th>
<th>S2</th>
<th>S1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERIAL POLL STATUS 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ton</th>
<th>Lon</th>
<th>EOI</th>
<th>LPAS</th>
<th>TPAS</th>
<th>LA</th>
<th>TA</th>
<th>MJMN</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>TO</th>
<th>LO</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>ADM1</th>
<th>ADM0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS STATUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPT7</th>
<th>CPT6</th>
<th>CPT5</th>
<th>CPT4</th>
<th>CPT3</th>
<th>CPT2</th>
<th>CPT1</th>
<th>CPT0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>CNT2</th>
<th>CNT1</th>
<th>CNT0</th>
<th>COM4</th>
<th>COM3</th>
<th>COM2</th>
<th>COM1</th>
<th>COM0</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND PASS THROUGH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| INT | DTO | DLO | AD5-0 | AD4-0 | AD3-0 | AD2-0 | AD1-0 | 1  | 1  | 0  | ARS | DT  | DL  | AD5 | AD4 | AD3 | AD2 | AD1 |
|-----|-----|-----|-------|-------|-------|-------|-------|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| ADDRESS 0 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

| X  | DT1 | DL1 | AD5-1 | AD4-1 | AD3-1 | AD2-1 | AD1-1 | 1  | 1  | 1  | EC7 | EC6 | EC5 | EC4 | EC3 | EC2 | EC1 | EC0 |
|----|-----|-----|-------|-------|-------|-------|-------|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| ADDRESS 1 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Figure 3. 8291A REGISTERS**
Address Mode

The address mode and status registers are used to program the addressing modes and track addressing states. The auxiliary mode register is used to select a variety of functions. The command pass through register is used for undefined commands and extended addresses. The address 0/1 register is used to program the addresses to which the 8291A will respond. The address 0 and address 1 registers allow reading of these programmed addresses plus trading of the interrupt bit. The EOS register is used to program the end of sequence character.

Detailed descriptions of the addressing modes available with the 8291A are described in the 8291A data sheet. Examples of how to program these modes are shown below.

1. MODE: — Talker has single address of 01H
   — Listener has single address of 02H

<table>
<thead>
<tr>
<th>CPU WRITES TO:</th>
<th>PATTERN</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Mode Register</td>
<td>0000 0001</td>
<td>Select Mode 1 Addressing</td>
</tr>
<tr>
<td>Address 0/1 Register</td>
<td>0010 0001</td>
<td>Major is Talking. Address = 01H</td>
</tr>
<tr>
<td>Address 0/1 Register</td>
<td>1100 0100</td>
<td>Minor is Listener. Address = 02H</td>
</tr>
</tbody>
</table>

2. MODE: — Talker has single address of 01H
   — Listener has single address of 02H

<table>
<thead>
<tr>
<th>CPU WRITES TO:</th>
<th>PATTERN</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Mode Register</td>
<td>0000 0001</td>
<td>Select Mode 1 Addressing</td>
</tr>
<tr>
<td>Address 0/1 Register</td>
<td>0100 0010</td>
<td>Major is Listener. Address = 02H</td>
</tr>
<tr>
<td>Address 0/1 Register</td>
<td>1010 0001</td>
<td>Minor is Talking. Address = 01H</td>
</tr>
</tbody>
</table>

Note that in both of the above examples, the listener will respond to a MLA message with five least significant bits equal to 02H and the talker to a 01H.

3. MODE: — Talker and listener both share a single address of 03H.

<table>
<thead>
<tr>
<th>CPU WRITES TO:</th>
<th>PATTERN</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Mode Register</td>
<td>0000 0001</td>
<td>Selects Mode 1 Addressing</td>
</tr>
<tr>
<td>Address 0/1 Register</td>
<td>0000 0011</td>
<td>Talker and Listener Address = 03</td>
</tr>
<tr>
<td>Address 0/1 Register</td>
<td>1110 0000</td>
<td>Minor Address is disabled</td>
</tr>
</tbody>
</table>

4. MODE: — Talker and listener have a primary address of 04H and a secondary address of 05H

<table>
<thead>
<tr>
<th>CPU WRITES TO:</th>
<th>PATTERN</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Mode Register</td>
<td>0000 0010</td>
<td>Selects Mode 2 Addressing</td>
</tr>
<tr>
<td>Address 0/1 Register</td>
<td>0000 0100</td>
<td>Primary Address = 04H</td>
</tr>
<tr>
<td>Address 0/1 Register</td>
<td>1000 0101</td>
<td>Minor Address is disabled</td>
</tr>
</tbody>
</table>

5. MODE: — Talker has a primary address of 06H. Listener has a primary address of 07H

<table>
<thead>
<tr>
<th>CPU WRITES TO:</th>
<th>PATTERN</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Mode Register</td>
<td>0000 0011</td>
<td>Select Mode 3</td>
</tr>
<tr>
<td>Address 0/1 Register</td>
<td>0010 0110</td>
<td>Talker Address = 06</td>
</tr>
<tr>
<td>Address 0/1 Register</td>
<td>1100 0111</td>
<td>Listener Primary = 07</td>
</tr>
</tbody>
</table>

The CPU will verify the secondary addresses which could be the same or different.
APPLICATION OF THE 8291A

This phase of the application note will examine programming of the 8291A, corresponding bus commands and responses, CPU interruption, etc. for a variety of GPIB activities. This should provide the reader with a clear understanding of the role the 8291A performs in a GPIB system. The talker function, listener function, remote message handling, and remote/local operations including local lockout, are discussed.

Talker Functions

TALK-ONLY (ton). In talk only mode the 8291A will not respond to the MTA message from a controller. Generally, ton is used in an environment which does not have a controller. Ton is also employed in an interface that includes the controller function.

When the 8291A is used with the 8292, the sequence of events for initialization are as follows:

1) The Interrupt/Enable registers are programmed.
2) Ton is selected.
3) Settling time is selected.
4) EOS character is programmed.
5) “Pon” local message is sent.
6) CPU waits for Byte Out (BO) and sends a byte to the data out register.

Addressed Talker (Via MTA Message)

The GPIB controller will direct the 8291A to talk by sending a My Talk Address (MTA) message containing the 8291A's talk address. The sequence of events is as follows:

1) The interrupt enable and serial poll mode registers are programmed.
2) Mode 1 is selected.
3) Settling time is selected.
4) Talker and listener addresses are programmed.
5) Power on (pon) local message is sent.
6) CPU waits for BO and reads the byte from the data-in register.

Remote/Local and Lockout

Remote and local refer to the source of control of a device connected to the GPIB. Remote refers to control from the GPIB controller-in-charge. Local refers to control from the device's own system. Reference should be made to the RL state diagram in the 8291A data sheet.

Upon “pon” the 8291A is in the local state. In this state the REM bit in Interrupt Status 1 Register is reset. When the GPIB controller takes control of the bus it will drive the REN (remote enable) line true. This will cause the REM bit and REMC (remote/local change) bit to be set. The distinction between remote and local modes is necessary in that some types of devices will have local controls which have functions which are also controlled by remote messages.

In the local state the device is allowed to store, but not respond to, remote messages which control functions which are also controlled by local messages. A device

not respond to the My Listen Address (MLA) message from the controller. The sequence of events is as follows:

1) The Interrupt Enable registers are programmed.
2) Lon is selected.
3) EOS characted is programmed.
4) “Pon” local message is sent.
5) CPU waits for BI and reads the byte from the data-in register.

Note that enabling both ton and lon can create an internal loopback as long as another listener exists.

Lister Functions

LISTEN-ONLY (lon). In listen-only mode the 8291A will
which has been addressed to listen will exit the local state and go to the remote state if the REN message is true and the local rtl (return to local) message is false. The state of the "rtl" local message is ignored and the device is "locked" into the local state if the LLO remote message is true. In the Remote state the device is not allowed to respond to local messages which control function that are also controlled by remote messages. A device will exit the remote state and enter the local state when REN goes false. It will also enter the local state if the GTL (go to local) remote message is true and the device has been addressed to listen. It will also enter the local state if the rtl message is true and the LLO message is false or ACDS is inactive.

A device will exit the remote state and enter RWLS (remote with lockout state) if the LLO (local lockout) message is true and ACDS is active. In this mode, those local message which control functions which are also controlled by remote messages are ignored. In other words, the "rtl" message is ignored. A device will exit RWLS and go to the local state if REN goes false. The device will exit RWLS and go to LWLS if the GTL message is true and the device is addressed to listen.

Polling

The IEEE-488 standard specifies two methods for a slave device to let the controller know that it needs service. These two methods are called Serial and Parallel Poll. The controller performs one of these two polling methods after a slave device requests service. As implied in the name, a Serial Poll is when the controller sequentially asks each device if it requested service. In a Parallel Poll the controller asks all of the devices on the GPIB if they requested service, and they reply in parallel.

Serial Poll

When the controller performs a Serial Poll, each slave device sends back to the controller a Serial Poll Status Byte. One of the bits in the Serial Poll Status Byte indicates whether this device requested service or not. The remaining 7 bits are user defined, and they are used to indicate what type of service is required. The IEEE-488 spec only defines the service request bit, however HP has defined a few more bits in the Serial Poll Status Byte. This can be seen in figure 4.

![Figure 4. The Serial Poll Status Byte](image-url)
When a slave device needs service it drives the SRQ line on the GPIB bus (true, low). For the 8291A this is done by setting bit 7 in the Serial Poll Status Byte. The CPU in the controller may be interrupted by SRQ or it may poll a register to determine the state of SRQ. Using the 8292 one could either poll the interrupt status register for the SRQ interrupt status bit, or enable SRQ to interrupt the CPU. After the controller recognizes a service request, it goes into the serial poll routine.

The first thing the controller does in the serial poll routine is assert ATN. When ATN is asserted true the controller takes control of the GPIB, and all slave devices on the bus must listen. All bytes sent over the bus while ATN is true are commands. After the controller takes control, it sends out a Universal Unlisten (UNL), which tells all previously addressed listeners to stop listening. The controller then sends out a byte called SPE (Serial Poll Enable). This command notifies all of the slaves on the bus that the controller has put the GPIB in the Serial Poll Mode State (SPMS). Now the controller addresses the first slave device to TALK and puts itself in the listen mode. When the controller resets ATN the device addressed to talk transmits to the controller its Poll Status Byte. If the device just polled was the one requesting service, the SRQ line on the GPIB goes false, and bit 7 in the serial poll status byte of the 8291A is reset. If more than one device is requesting service, SRQ remains low until all of the devices requesting service have been polled, since SRQ is wire-ored. To continue the Serial Poll, the controller asserts ATN, addresses the next device to talk then reads the Serial Poll Status Byte. When the controller is finished polling it asserts ATN, sends the universal untalk command (UNT), then sends the Serial Poll Disable command (SPD). The flow of the serial poll can be seen from the example in figure 5.

The following section describes the events which happen in a serial poll when 8291A and 8292 are the controller, and another 8291A is the slave device. While going through this section the reader should refer to the register diagrams for the 8291A and 8292.

A. DEVICE A REQUESTS SERVICE (SRQ BECOMES TRUE)
The slave devices rsv bit in the 8291A's serial poll mode register is set.

B. CONTROLLER RECOGNIZES SRQ AND ASSERTS ATN
The 8292's SPI pin 33 interrupts the CPU. The CPU reads the 8292's Interrupt status register and finds the SRQ bit set. The CPU tells the 8292 to 'Take Control Synchronously' by writing a OFDH to the 8292's command register.

C. THE CONTROLLER SENDS OUT THE FOLLOWING COMMANDS: UNIVERSAL UNLISTEN (UNL), SERIAL POLL ENABLE (SPE), MY TALK ADDRESS (MTA).
The MT A is a command which tells one of the devices on the bus to talk.

The CPU in the controller waits for a BO (byte out) interrupt in the 8291A's interrupt status 1 register before it writes to the Data Out register a 3FH (UNL), 18H (SPE), 010XXXXX (MTA). The X represents the programmable address of a device on the GPIB. When the 8291A in the slave device receives its talk address, the ADSC bit in the Interrupt Status register 2 is set, and in the Address Status Register TA and TPAS bits are set.

D. CONTROLLER RECONFIGURES ITSELF TO LISTEN AND RESETS ATN
The CPU in the controller puts the 8291A in the listen only mode by writing a 40H to the Address Mode register of the 8291A, and then a 00H to the Aux Mode register. The second write is an 'Immediate Execute pon' which must be done when switching addressing modes such as talk only to listen only. To reset ATN the CPU tells the 8292 to 'Go To Standby' by writing a 0F6H to the command register. The moment ATN is reset, the 8291A in the slave device sets SPAS in Interrupt Status 2 register, and transmits the serial poll status byte. SRQS in the Serial Poll Status byte of the 8291A slave device is reset, and the SRQ line on the GPIB bus becomes false.

E. THE CONTROLLER READS THE SERIAL POLL STATUS BYTE, SETS ATN, THEN RECONFIGURES ITSELF TO TALK
The CPU in the controller waits for the Byte In (BI) in the 8291A's Interrupt Status 1 register. When this bit is set the CPU reads the Data In register to receive the Serial Poll Status Byte. Since bit 7 is set, this was the device which requested service. The CPU in the controller tells the 8292 to 'Take Control Synchronously' which asserts ATN. The moment ATN is asserted true the 8291A in the slave device resets SPAS, and sets the Serial Poll Com-
plete (SPC) bit in the Interrupt Status 2 register. The controller reconfigures itself to talk by setting the TO bit in the Address Mode register and then writing a OOH to the Aux Mode register.

F. THE CONTROLLER SENDS THE COMMANDS UNIVERSAL UNTALK (UNT), AND SERIAL POLL DISABLE (SPD) THEN RESETS THE SRQ BIT IN THE 8292 INTERRUPT STATUS REGISTER

The CPU in the controller waits for the BO Interrupt status bit to be set in the Interrupt Status 1 register of the 8291A before it writes 5FH (UNT) and 19H (SPD) to the Data Out register. The CPU then writes a 2BH to the 8292’s command register to reset the SRQ status bit in the Interrupt Status register. When the 8291A in the slave device receives the UNT command the ADSC bit in the Interrupt Status 2 register is set, and the TA and TPAS bits in the Address Status register will be reset. At this point the controller can service the slave device's request.

Note that in the software listing of AP-66 (USING THE 8292 GPIB CONTROLLER) there is a bug in the serial poll routines. In the ‘SRQ ROUTINE’ when the CPU finds that the SRQ bit in the interrupt status register is set, it immediately writes the interrupt Acknowledge command to the 8292 to reset this bit. However the SRQ GPIB line will still be driven true until the slave device driving SRQ has been polled. Therefore, the SRQ status bit in the 8292 will become set and latched again, and as a result the SRQ status bit in the 8292 will still be set after the serial poll. The proper time to reset the SRQ bit in the 8292 is after SRQ on the GPIB becomes false.

Parallel Poll

The 8291A supports an additional method for obtaining status from devices known as parallel poll (PPOL). This method limits the controller to a maximum of 8 devices at a time since each device will produce a single bit response on the GPIB data lines. As shown in the state diagrams, there are three basic parallel poll sates: PPIS (parallel poll idle state), PPSS (parallel poll standby state), and PPAS (parallel poll active state).

In PPIS, the device's parallel poll function is in the idle state and will not respond to a parallel poll. PPSS is the standby state, a state in which the device will respond to a parallel poll from the controller. The response is initiated by the controller driving both ATN and EOI true simultaneously.

The 8291A state diagram shows a transition from PPIS to PPSS with the “lpe” message. This is a PP2 implementation for a parallel poll. This “lpe” (local poll enable) local message is achieved by writing 011 USP P3 P1 to the Aux Mode Register with U=0. The S bit is the sense bit. If the “ist” (individual status) local message value matches the sense bit, then the 8291A will give a true response to a parallel poll. Bits P3-P1 identify which data line is used for a response.

For example, assume the programmer decides that the system containing the 8291A shall participate in parallel poll. The programmer, upon system initialization would write to the Aux Mode Register and reset the U bit and set the S bit plus identify a data line (P3-P1 bits). At “pon,” the 8291A would not resond true to a parallel poll unless the parallel poll flag is set (via Aux Mode Register command).

When a status condition in the user system occurs and the programmer decides that this condition warrants a true response, then programmers software should set the parallel poll flag. Since the S bit value matches the “ist” (set) condition a true response will be given to all parallel polls.

An additional method of parallel polling reading exists known as a PPI implementation. In this case the controller sends a PPE (parallel poll enable) message. PPE contains a bit pattern similar to the bit pattern used to program the “lpe” local message. The 8291A will receive this as an undefined command and use it to generate an “lpe” message. Thus the controller is specifying the sense bits and data lines for a response. A PPD (parallel poll disable) message exists which clears the bits SP P3 P1 and sets the U bit. This also will be received by the 8291A and used to generate an “lpe” false local message.

The actual sequence of events is as follows. The controller sends a PPC (parallel poll configure) message. This is an undefined command which is received in the CPT register and the handshake is held off. The local CPU reads this bit pattern, decodes it, and sends a VSCMD message to the Aux Mode Register. The controller then sends a ppe message which is also received as an undefined command in the CPT register. The local CPU reads this, decodes it clears the MSB, and writes this to the Aux Mode Register generating the “lpe” message.

The controller then sends ATN and EOI true and the 8291A drives the appropriate data line if the “ist” (parallel poll flag) is true. The controller will then send a PPD (parallel poll disable) message (again, an undefined command). The CPU reads this from the CPT register and uses it to write a new “lpe” message (this “lpe” message will be false). The controller then sends a PPU (parallel poll unconfigure) message. Since this is also an undefined command, it goes into the CPT register. When the local CPU decodes this, the CPU should clear the “ist” (parallel poll flag).

APPLICATION EXAMPLES

In the course of developing this application note, two complete and identical GPIB systems were built. The
schematics and block diagrams are contained in Appendix I. These systems feature an 8088 CPU, 8237 DMA controller, serial I/O (8251 A and 8253), RAM, EPROM, and a complete GPIB talker/listener controller. Jumper switches were provided to select between a controller function and a talker/listener function. This system design is based on the design of Intel's SDK-86 prototyping kit and thus shares the same I/O and memory addresses. This system uses the same download software to transfer object files from Intel development systems.

Two Software Drivers

Two software drivers were developed to demonstrate a ton/on environment. These two programs (BOARD 1 and BOARD 2) are contained in Appendix 2.

In this example, one of the systems (BOARD 1) initially is programmed in talk-only mode and synchronization is achieved by waiting for the listening board to become active. This is sensed by the lack of a GPIB error since a condition of no active listener produces an ERR status condition. Board I upon detecting the presence of an active listener switches to the talk only mode.

The controller then performs a parallel poll. The parallel poll indicates 1 bit of status of each device in a group of up to 8 devices. Such information could be used by an application program to determine whether optional devices are part of a system configuration. Such optional devices might include mass storage devices, printers, etc. where the application software for the controller might need to format data to match each type of device. Once the PPOL sequence is finished, the HP 9835A offers the user the opportunity to execute user commands from the keyboard. At this time the HP 9835A sits in a loop waiting for an SRQ condition. When the operator hits a key on the keyboard, the HP 9835A processor is interrupted and vectors to a service routine where the key is read and the appropriate routine is executed. The HP 9835A will then return to the loop checking for SRQ true. For this application, the valid keys are G,D,R,H, and X. Pressing the “G” key causes the GET command to be sent across the bus. A message to this effect is printed in the CRT and the HP 9835A returns. The “D” key causes the SDC message to be sent with the 8291 A being the addressed device. Again, an appropriate message is output on the HP 9835A CRT.

The characters are stored in the sequence entered into a buffer whose maximum size is 80 characters. Pressing the “CONTINUE” key terminates storing characters in the array and all characters including the carriage return and line feed are sent. EOI is then sent true with a false byte of OOH. This false byte is due to the 1975 standard which allows asynchronous sending and reception of EOI. (The 8291 A supports the later 1978 standard which eliminates this false byte).

After any key command is serviced control returns to the loop which checks for SRQ active. Should SRQ be active, then the keyboard interrupt is disabled and a message printed to indicate that SRQ has been received true.

The controller then performs a parallel poll.

This is an example of how parallel poll may be used to

8291A with HP 9835A

An example of the 8291 A used in conjunction with a bus controller is also included in this application note. In this example, the 8291 A system used in previous experiments was connected via the GPIB to a Hewlett-Packard 9835 A desktop computer. This computer contains, in addition to a GPIB interface, a black and white CRT, keyboard, tape drive for high quality data cassettes, and a calculator type printer. The software for the HP 9835A is shown in Appendix 3. The user should refer to the operation manuals for the HP 9835A for information on the features and programming methods for the HP 9835A.

In this example, the 8292 was removed from its socket and the OPTA and OPTB pins of the two 8293 transceiver reconfigured to modes 0 and 1. Optionally, the mode pins could have been left wired for modes 2 and 3 and the 8292 left in its socket with its SYC pin wired to ground. This would have produced the same effect.
quickly check which group of devices contains a device sending SRQ. The eight devices in a group would, of course, have software drivers which allow a true response to a PPOL if that device is currently driving SRQ true. This would be a valuable method of isolation of the SRQ source in a system with a large number of devices. In this application program, only the response from the 8291A is of concern and only the 8291A's response is considered. It does, however, demonstrate the technique employed. If a true response from the 8291A is detected, then a message to this effect is printed on the HP 9835A CRT screen. From this process, the controller has identified the device requesting service and will use a serial poll from the DEC and REMC status bits.

The controller software then prints a message to indicate that it is about to perform a serial poll. This serial poll will return to the controller the current status of the 8291A and clear the service request. The status byte received is then printed on the CRT screen of the HP 9835A. One of the 8291A status bits indicates that the 8291A system has a field (on line or less) of data to transfer to the HP 9835A. If this bit is set, then the HP 9835A addresses the 8291A system to talk. The data is sent by the 8291A system to the CRT screen of the HP 9835A. The HP 9835 then enables the keyboard interrupts and goes into its SRQ checking loop.

Appendix 4 contains the software for the 8291A system which is connected to the HP 9835A via the GPIB. This software throws away the first byte of data it receives since this transfer was used by the HP 9835A to test when the 8291A system came on line.

Next, both status registers are read and stored in the two variable STAT 1 and STAT 2. It is necessary to store the status since reading the status registers clears the status bits.

Initially, six status bits are evaluated (END, GET, CPT, DEC, REMC, ADSC). Some of these conditions require that additional status bits be evaluated.

If END is true, then the 8291A system has received a block from the HP 9835A and the contents of a buffer is printed on the CRT screen. Next, the CPT bit is checked. PPC and PPE are the only valid undefined commands in this example.

Next, the GET bit is examined and if true, the CRT screen connected to the serial channel on the 8291A system prints a message to indicate that the trigger command has been received. A similar process occurs with the DEC and REMC status bits.

Address Status Change (ADSC) is checked to see if the 8291A has been addressed or unaddressed by the controller. If ADSC is false, then the software checks the keyboard at the CRT terminal. If ADSC is set, then the TA and LA bits are read and evaluated to determine whether the 8291A has been addressed to talk or listen. The DMA controller is set to start transfers at the start of the character buffer and the type of transfer is determined by whether the 8291A is in TADS or LADS. We only need to set up the DMA controller since the transfers will be transparent to the system processor. The keyboard from the CRT terminal is then checked. If a key has been hit, then this character is stored in the character buffer and the buffer printer set to the next character location. This process repeats until the received character is a line feed. The line feed is echoed to the CRT, the serial poll status byte updated and the SRQ line driven true. This allows the 8291A system to store up to one line of characters before requesting a transfer to the controller. Recall that upon receiving an SRQ, the controller will perform a serial poll and subsequently address the 8291A to talk. The 8291A system then goes back to reading the status register thus repeating the process.

**CONCLUSION**

This application note has shown a basic method to view the IEEE 488 bus, when used in conjunction with Intel's® 8291A.

The main reference for GPIB questions is the IEEE Standard 488 - 1978. Reference 8291A's data sheet for detailed information on it.

Additional Intel® GPIB products include ISBX-488, which is a multimode board consisting of the 8291A, 8292, and 8293.

**REFERENCES**

8291A Data Sheet
8292 Data Sheet
8293 Data Sheet
Application Note #66 “Using the 8292 GPIB Controller”
PLM-86 User Manual
HP 9835A User's Manual
IEEE—488—1978 Standard
APPENDIX 1
SYSTEM BLOCK DIAGRAM WITH 8088
APPENDIX 2
SOFTWARE DRIVERS FOR BLOCK DATA TRANSFER

PL/M-86 COMPILER: BOARD 1
ISIS-II PL/M-86 V11 COMPILATION OF MODULE BOARD 1
OBJECT MODULE PLACED IN: F1 BRD1 OBJ
COMPILED INVOKED BY: PLM86 F1 BRD1. SRC SYMBOLS MEDIUM

/* BOARD 1 TPT PROGRAM */
/* THIS BOARD TALKS TO THE OTHER BOARD BY */
/* TRANSFERRING A BLOCK OF DATA VIA THE 8237 */
/* COUPLED WITH THE 8291A THE 8291A IS PROGRAM- */
/* MED TO SEND END WHEN RECOGNIZING THE LAST */
/* DATA BYTE'S BIT PATTERN WHILE DATA IS BEING */
/* TRANSFERRED, THE PROCESSOR PERFORMS I/O READS */
/* OF THE 8237 COUNT REGISTERS TO SIMULATE BUS */
/* ACTIVITY AND TO DETERMINE WHEN TO TURN THE */
/* LINE AROUND. AFTER THE 8237 HAS REACHED */
/* TERMINAL COUNT, THE 8291A IS PROGRAMMED TO */
/* THE LISTENER STATE AND WAITS FOR THE BLOCK */
/* TO BE TRANSMITTED BACK FROM THE SECOND BOARD. */
/* THIS DATA IS PLACED IN A SECOND BUFFER AND */
/* ITS CONTENTS COMPARED WITH THE ORIGINAL DATA */
/* TO CHECK FOR INTERFACE INTEGRITY */

1 BOARD1.

DO

/* PROCEDURES */

2 1 CO PROCEDURE (XXX);
3 2 DECLARE XXX BYTE,
4 2 SER*STAT LITERALLY 'OFFF2H',
5 2 SER*DATA LITERALLY 'OFFFOH',
6 2 TXRDY LITERALLY '01H',
7 2 DO WHILE (INPUT (SER*STAT) AND TXRDY) <> TXRDY;
8 2 END;
9 2 OUTPUT (SER*DATA) = XXX;
10 2 END CO;

/* SETUP BUFFERS */

8 1 DECLARE BUFF2 (100) BYTE; /* RAM STORAGE AREA */
9 1 DECLARE BUFF1 (100) BYTE DATA

(1, 2, 3, 4, 5, 6, 7, 8, 9, 10H,
31H, 32H, 33H, 34H, 35H, 36H, 37H, 38H, 39H, 40H,
41H, 42H, 43H, 44H, 45H, 46H, 47H, 48H, 49H, 50H,
51H, 52H, 53H, 54H, 55H, 56H, 57H, 58H, 59H, 60H,
61H, 62H, 63H, 64H, 65H, 66H, 67H, 68H, 69H, 70H,
71H, 72H, 73H, 74H, 75H, 76H, 77H, 78H, 79H, 80H,
81H, 82H, 83H, 84H, 85H, 86H, 87H, 88H, 89H, 90H.

6-510
PL/M-86 Compiler Board

91H, 92H, 93H, 94H, 95H, 96H, 97H, 98H, 99H, ODH);

DECLARE BUFF3(17) BYTE DATA
(ODH, OAH, 'COMPARE ERROR', ODH, OAH); /* ROM STORAGE AREA */

/* 6237 PORT ADDRESSES */

DECLARE

CLEAR$FF LITERALLY 'OFFDFDH', /* MASTER CLEAR */
START$O$LO LITERALLY 'OFFD10H',
START$O$HI LITERALLY 'OFFD10H',
O$COUNT$LO LITERALLY 'OFFD10H',
O$COUNT$HI LITERALLY 'OFFD10H',
SET$MODE LITERALLY 'OFFD10H',
CMD$37 LITERALLY 'OFFD10H',
SET$MASK LITERALLY 'OFFD10H'.

/* 6237 COMMAND - DATA BYTES */

DECLARE DMA$ADR$TALK POINTER;
DECLARE DMA$ADR$LSTN POINTER;

DECLARE

RD$TRANSFER LITERALLY '48H',
WR$TRANSFER LITERALLY '44H',
NORM$TIME LITERALLY '20H',
TC$LO1 LITERALLY 'OFFH',
TC$HI1 LITERALLY '00H',
TC$LO2 LITERALLY '99D', /* 100 XFERS */
TC LITERALLY '01H',
I BYTE;

DECLARE

DMA$WRD$TALK (2) WORD AT (@DMA$ADR$TALK),
DMA$WRD$LSTN (2) WORD AT (@DMA$ADR$LSTN);

/* 8291A PORT ADDRESSES */

DECLARE

PORT$OUT LITERALLY 'OFFCOH', /* DATA OUT */
PORT$IN LITERALLY 'OFFCOH'
STATUS$1 LITERALLY 'OFFC1H', /* INTR STAT 1 */
STATUS$2 LITERALLY 'OFFC2H', /* INTR STAT 2 */
ADDR$STATUS LITERALLY 'OFFC4H',
COMMAND$MOD LITERALLY 'OFFC5H', /* CMD PASS THRU */
ADDR$0 LITERALLY 'OFFC6H',
EOS$REG LITERALLY 'OFFC7H', /* EOS REGISTER */
/* 8291A COMMAND - DATA BYTES */

PL/M-86 COMPILER  BOARD1

DECLARE

END$EOI LITERALLY 'BBH',
DONE LITERALLY '10H',
PON LITERALLY '00H',
RESET LITERALLY '02H',
CLEAR LITERALLY '00H',
DMA$REG$L LITERALLY '10H',
DMA$REG$T LITERALLY '20H',
MODI$LO LITERALLY '08H',
MODI$LO LITERALLY '40H',
EOS LITERALLY '0DH',
PRESCALER LITERALLY '23H',
HIGH$SPEED LITERALLY '0A4H',
OKAY LITERALLY '0FFFFH',
XYZ BYTE.
MATCH WORD,
B0 LITERALLY '02H',
BI LITERALLY '01H',
EPR LITERALLY '04H',

/* CODE BEGINS */

START91.

OUTPUT (STATUS$2) =CLEAR, /* SHUT-OFF DMA REG BITS TO */
/* PREVENT EXTRA DMA REGS */
/* FROM 8291A */

/* MANIPULATE DMA ADDRESS VARIABLES */

DMA$ADR$Y TALK =(@BUFF1);
DMA$ADR$Y LSTN =(@BUFF2);
DMA$WRD$TALK(1)=SHL (DMA$WRD$TALK(1), 4);
DMA$WRD$TALK(0)=DMA$WRD$TALK(0) + DMA$WRD$TALK(1);
DMA$WRD$LSTN(1)=SHL (DMA$WRD$LSTN(1), 4);
DMA$WRD$LSTN(0)=DMA$WRD$LSTN(0) + DMA$WRD$LSTN(1);

【INIT37】
/* INIT 8237 FOR TALKER FUNCTIONS */

OUTPUT (CLEAR$FF) =CLEAR; /* TOGGLE MASTER CLEAR */
OUTPUT (CMD$37) =NORM$TIME;
OUTPUT (SET$MODE) =RD$TRANSFER;
OUTPUT (SET$MASK) =CLEAR;
OUTPUT (START$0$LO) =DMA$WRD$TALK(0);
DMA$WRD$TALK(0) =SHR (DMA$WRD$TALK(0), 8);
OUTPUT (START$0$HI) =DMA$WRD$TALK(0);
OUTPUT (0$COUNT$LO) =TC$LO2;
OUTPUT (0$COUNT$HI) =TC$HI2;
/* INIT 8291A FOR TALKER FUNCTIONS */

PL/M-86 COMPILER  BOARD1
34 1  OUTPUT (EOS*REG) = EOS;
35 1  OUTPUT (COMMAND*MOD) = END*EOI; /* EOI ON EOS SENT */
36 1  OUTPUT (ADDR*STATUS) = MOD1*TO, /* TALK ONLY */
37 1  OUTPUT (COMMAND*MOD) = PRESCALER,
38 1  OUTPUT (COMMAND*MOD) = HIGH*SPEED,
39 1  OUTPUT (COMMAND*MOD) = PON;

40 1  DO WHILE ((INPUT (STATUS$1) AND BO) = 0.,
41 2   END; /* WAIT FOR BO INTR */
42 1  OUTPUT (PORT$OUT) = OAAH;

43 1  DO WHILE ((INPUT (STATUS$1) AND ERR) = ERR;
44 2   DO WHILE ((INPUT (STATUS$1) AND BO) = 0;
45 3   END. /* WAIT FOR BO INTR */
46 2   OUTPUT (PORT$OUT) = OAAH;
47 2   END.

48 1  OUTPUT (STATUS$2) = DMA*REG$T, /* ENABLE DMA REGS */
49 1  DO WHILE ((INPUT (CMD$37) AND TC) = TC;
50 2   /* WAIT FOR TC = G */
51 1  END,

51 1  INIT37L,

52 1  OUTPUT (STATUS$2) = CLEAR, /* DISABLE DMA REGS */

53 2  /* INIT 8237 FOR LISTENER FUNCTIONS */
54 1  OUTPUT (CLEAR$FF) = CLEAR, /* TOGGLE MASTER RESET */
55 1  OUTPUT (CMD$37) = NOP*TIME,
56 1  OUTPUT (SET*MODE) = WR*TRANSFER,
57 1  OUTPUT (SET*MASK) = CLEAR;
58 1  OUTPUT (START$O*LO) = DMA*WRD$LSTN (O),
59 1  OUTPUT (START$O*HI) = DMA*WRD$LSTN (O),
60 1  OUTPUT (O*COUNT$LO) = TC$LO1,
61 1  OUTPUT (O*COUNT$HI) = TC$HI1,

61 1  /* INIT 8291A FOR LISTENER FUNCTIONS */
62 1  OUTPUT (COMMAND$MOD) = RESET,
63 1  OUTPUT (ADDR$STATUS) = MOD1*LO; /* LISTEN ONLY */
64 1  OUTPUT (COMMAND$MOD) = PON;

64 1  DO WHILE ((INPUT (STATUS$1) AND BI) = 0;
65 2  END. /* WAIT FOR BI INTR */
66 1  XYZ = INPUT (PORT$IN),

67 1  OUTPUT (STATUS$2) = DMA*REQ$L, /* ENABLE DMA REGS */
68 1  DO WHILE ((INPUT (STATUS$1) AND DNE) = DNE,
69 2   /* WAIT FOR EOI RECEIVED */
PL/M-86 COMPILER BOARD 1

70 1 CMPBLKS

    /* COMPARE THE TWO BUFFERS CONTENTS */
    MATCH=CMFB (@BUFF1, @BUFF2, 100);

71 1 IF MATCH = OKAY THEN GOTO START91;

    /* SEND ERROR MESSAGE IN BUFFER 3 */

73 1 DO I=0 TO 16,

74 2 CALL CO (BUFF 3 (I));

75 2 END;

76 1 GOTO START91;

77 1 END;

MODULE INFORMATION

| CODE AREA SIZE |   =01DBH | 475D |
| CONSTANT AREA SIZE | =0075H | 117D |
| VARIABLE AREA SIZE | =0070H | 112D |
| MAXIMUM STACK SIZE | =0006H | 6D |

243 LINES READ
0 PROGRAM ERROR (S)

END OF PL/M-86 COMPILATION
PL/M-86 COMPILER BOARD2

ISIS-II PL/M-86 V1.1 COMPILATION OF MODULE BOARD2
OBJECT MODULE PLACED IN FI BRD2, OBJ
COMPILER INVOKED BY. PLM86 FI. BRD2, SRC

/* BOARD 2 TPT PROGRAM */
/* */
/* THIS BOARD LISTENS TO THE OTHER BOARD (1) */
/* AND DMA'S DATA INTO A BUFFER, WHILE WAITING */
/* FOR THE END INTERRUPT BIT TO BECOME ACTIVE */
/* UPON END ACTIVE, THE DATA IN THE BUFFER IS */
/* SENT BACK TO THE FIRST BOARD VIA THE GPIF */
/* WHEN THE BLOCK IS FINISHED THE 8291A IS */
/* PROGRAMMED BACK INTO THE LISTENER MODE */

1
BOARD2,
DO;

/* 8237 PORT ADDRESSES */

DECLARE

CLEAR$FF LITERALLY 'OFFDHH', /*MASTER CLEAR */
START$O$Lo LITERALLY 'OFFDOH',
START$O$Hi LITERALLY 'OFFDOH',
O$COUNT$Lo LITERALLY 'OFFD1H',
O$COUNT$Hi LITERALLY 'OFFD1H',
SET$MODE LITERALLY 'OFFDBH',
CMD$37 LITERALLY 'OFFDBH',
SET$MASK LITERALLY 'OFFDFH',

/* 8237 COMMAND - DATA BYTES */

DECLARE

RD$TRANSFER LITERALLY '48H',
WR$TRANSFER LITERALLY '44H',
ADDR$1A LITERALLY '00H',
ADDR$1B LITERALLY '01H',
NORM$TIME LITERALLY '20H',
TC$L01 LITERALLY 'OFFH',
TC$H11 LITERALLY '00H',
TC$L02 LITERALLY '99D',
TC$H12 LITERALLY '00H',
TC LITERALLY '01H',

/* 8291A PORT ADDRESSES */

DECLARE

PORT$OUT LITERALLY 'OFFCOH', /* DATA IN */
PORT$IN LITERALLY 'OFFCOH',
STATUS$1 LITERALLY 'OFFC1H', /* INTR STAT 1 */
STATUS$2 LITERALLY 'OFFC2H', /* INTR STAT 2 */
ADDR$STATUS LITERALLY 'OFFC4H', /* ADDR STAT */
COMMAND$MODE LITERALLY 'OFFC5H', /* CMD PASS THRU */
PL/M-86 COMPILER ROAND2

ADDR$0 LITERALLY 'OFFC6H',
EOS$REG LITERALLY 'OFFC7H', /* EOS REGISTER */

/* 8291A COMMAND - DATA BYTES */

DECLARE

END$EO! LITERALLY '78H',
DNE LITERALLY '10H',
PON LITERALLY 'OOH',
RESET LITERALLY '02H',
CLEAR LITERALLY '0OH',
DMA$REG$0 LITERALLY '1OH',
DMA$REG$1 LITERALLY '20H',
MOD1$TO LITERALLY '80H',
MOD1$LO LITERALLY '40',
EOS LITERALLY 'ODH',
PRESCALER LITERALLY '23H',
HIGH$SPEED LITERALLY 'A4H',
XYZ BYTE,
BO LITERALLY '02H',
BI LITERALLY '01H',
ERR LITERALLY '04H',

6 1 START91,

OUTPUT (STATUS$2) =CLEAR; /* END INITILIZATION STATE */

/* INIT 8237 FOR LISTENER FUNCTION */

7 1 INIT37L;

OUTPUT (CLEAR$FF) =CLEAR; /* TOGGLE MASTER RESET */
OUTPUT (CMD$37) =NORM$TIME,
OUTPUT (SET$MODE) =WR$TRANSFER, /* BLOCK XFER MODE */
OUTPUT (SET$MASK) =CLEAR,
OUTPUT (START$O$LO) =ADDR$1A,
OUTPUT (START$O$HI) =ADDR$1B,
OUTPUT (O$COUNT$LO) =TC$LO1,
OUTPUT (O$COUNT$HI) =TC$HI1;

/* INIT 8291A FOR LISTENER FUNCTIONS */

15 1 OUTPUT (COMMAND$MOD) =RESET;
16 1 OUTPUT (ADDR$STATUS) =MOD1$LO;
17 1 OUTPUT (COMMAND$MOD) =PON;
18 1 DO WHILE (INPUT (STATUS$1) AND BI) =0;
19 2 END; /* WAIT FOR BI INTR */
20 1 XYZ= INPUT (PORT$IN);
21 1 OUTPUT (STATUS$2) =DMA$REG$0;

/* WAIT UNTIL EOI RCVD AND END INTR-BIT SET */

22 1 DO WHILE (INPUT (STATUS$1) AND DNE ) <> DNE;
PL/M-86 COMPILER BOARD2

END;

INIT37;
/* INIT 8237 FOR TALKER FUNCTION */

OUTPUT (STATUS$2) =CLEAR; /* CLEAR 8291A DRQ */
OUTPUT (CLEAR$FF) =CLEAR;
OUTPUT (CMD$37) =NORM$TIME;
OUTPUT (SET$MODE) =RD$TRANSFER, /* BLOCK XFER MODE */
OUTPUT (SET$MASK) =CLEAR;
OUTPUT (START$O$LO) =ADDR$1A;
OUTPUT (START$O$HI) =ADDR$1B;
OUTPUT (O$COUNT$LO) =TC$LO2;
OUTPUT (O$COUNT$HI) =TC$HI2;
/* INIT 8291A FOR TALKER FUNCTION */

OUTPUT (EOS$REG) =EOS;
OUTPUT (COMMAND$MOD) =END$EOI; /* EOI ON EOS SENT */
OUTPUT (ADDR$STATUS) =MOD1$TO; /* TALK ONLY */
OUTPUT (COMMAND$MOD) =PRESCALER;
OUTPUT (COMMAND$MOD) =HIGH$SPEED;
OUTPUT (COMMAND$MOD) =PON;

DO WHILE (INPUT (STATUS$1) AND BO) =0;
END; /* WAIT FOR BO INTR */

OUTPUT (PORT$OUT) =OAAH;

DO WHILE (INPUT (STATUS$1) AND ERR) =ERR;
DO WHILE (INPUT (STATUS$1) AND BO) =0;
END; /* WAIT FOR BO INTR */

OUTPUT (PORT$OUT) =OAAH;
END.

OUTPUT (STATUS$2) =DMA$REG$T;
/* WAIT FOR TC=0 */

DO WHILE (INPUT (CMD$37) AND TC) <> TC;
END;

GOTO START91;

END;

MODULE INFORMATION

CODE AREA SIZE =0122H 290D
CONSTANT AREA SIZE =0000H 0D
VARIABLE AREA SIZE =0001H 1D
MAXIMUM STACK SIZE =0000H 0D
152 LINES READ
0 PROGRAM ERROR (S)
APPENDIX 3
SOFTWARE FOR HP 9835A

10 REM SEND IN
TERFACE CLEAR
20 ABORTIO 7
30 REM FORCE E
RRORS UNTIL LIST
ENERS ACTIVE
40 Frerr: OUT
PUT 704 USING ";K";"B"
50 Chkstat: STAT
ATUS 7;Stat1,Stat2,Stat3,Stat4
60 Err=Stat2 A
ND 1
70 IF Err=1 TH
EN GOTO Frerr
80 PRINT CHR$(12),"LISTENERS A
RE ON LINE"
90 REM CONFIGU
RE PPOLL
100 PPOLL CONFIGU
TURE 704;"000001
00"
110

! response on
bit 4
120 PRINT CHR$(12),"PARALLEL PO
LL CONFIGURED"
130 REM ENABLE
KEYBOARD INTERRU
PT
140 PRINT "COMM
AND = ? (HIT
'H' FOR LIST)"
150 Keyen: ON K
BD GOSUB 610
160 STATUS 7;St
at1,Stat2,Stat3,
Stat4
170 Sra=BINAND(Stat1,128)
180 IF Sra=0 TH
EN GOTO Keyen
190 OFF KBD
200 PRINT CHR$(12),"SRQ RECEIVE
D"
210 PRINT "SEND
ING PARALLEL POL
L RESPONSE MESSA
GE"
220 REM EXECUTI
NG PARALLEL POL
230 Ppollbyte=P
OLL(?)
240 PRINT "PAR
ALLEL POLL BYTE =
";Ppollbyte
250 PRINT "----
----------
----
260 Ppollbyte=BINAND(Ppollbyte,8)
270 IF Ppollbyt
e=0 THEN GOTO PS
291
280 PRINT "SR
OTH FROM 8291"
281 PRINT "COMM
AND = ? (HIT
'H' FOR LIST)"
290 GOTO Keyen
300 P8291: PRIN
T "SR0 IS FROM N
CC 8291 ... THE
ENTERPRISE"
310 PRINT "PERF
ORIGIN SERIAL POLL TO GET STATUS

320 STATUS 704;
Stat
330 PRINT CHR$(12), "Status = ";
Stat
340 Dxfer=BINAN D(Stat,1)
520 IF Dxfer>0 THEN GOTO Rcvr
530 GOTO Keyen
531 Rcvr: REM R READY TO RCV CHAR S FROM GPIB
540 DIM G$[80]
550 ENTER 704 U SING "%,T";G$
560 PRINT CHR$(12),G$
570 PRINT "COMM AND = ? (HIT 'H' FOR LIST)"
580 GOTO Keyen
590 REM INTERRUPT SERVICE ROUTINE
600 REM GET KEY BOARD DATA
610 Whatkey: DIM K$[80]
620 : K$=KBD$
630 IF K$="G" GOTO Get
640 IF K$="D" GOTO Dec
650 IF K$="R" GOTO Rem
660 IF K$="H" GOTO Help
670 IF K$="X" GOTO Xmit
680 Get: TRIGGE
R 704
690 PRINT CHR$(12), "GROUP EXECUTE TRIGGER SENT"
700 PRINT " "
710 PRINT "COMM AND = ? (HIT 'H' FOR LIST)"
720 RETURN
730 Dec: RESET 704
740 PRINT CHR$(12), "SELECTIVE DEVICE CLEAR SENT"
750 PRINT: " "
760 PRINT "COMM AND = ? (HIT 'H' FOR LIST)"
770 RETURN
780 Rem: LOCAL 704
790 PRINT CHR$(12), "REMOTE MESSAGE SENT"
800 PRINT " "
810 PRINT "COMM AND = ? (HIT 'H' FOR LIST)"
820 RETURN
830 Help: PRINT CHR$(12)
840 PRINT " @ OPERATOR ALLOW ABLE COMMANDS @ @ "
850 PRINT " hit key result"
860 PRINT " Send GET m essage"
870 PRINT " Send DEC m essage"
880 PRINT " R
   Send REM a
OC message"
890 PRINT " X
   Xmits keyb
doard input to 81
91"
900 PRINT " H
   Prints thia
stable"
910 PRINT " .
920 PRINT " ...
so ahead; TRY IT
!"
930 RETURN

940 Xmit: DIM A
 $[80]
950 PRINT CHR$(
12), "Enter data
to send and hit
CONTINUE"
960 INPUT A$
970 OUTPUT 704;
A$
971 EOI 7;0
980 PRINT " COMM
AND = ? (HIT
' H' FOR LIST)"
990 RETURN
1000 END
APPENDIX 4
SOFTWARE FOR HP 8088/HP 9835A VIA GPIB

PL/M-86 COMPILER
HP/IB

ISIS-II PL/M-86 V1.1 COMPI LATION OF MODULE HP/IB
OBJECT MODULE PLACED IN: F1: HP/I B. OBJ
COMPILER INVOKED BY: PLMB6 : F1: HP/I B. SRC LARGE

1
HP/I B:
/*

PARAMETER DECLARATIONS
*/
DO:

2
DECLARE

ADDR_HI LITERALLY '01H',
ADDR_LO LITERALLY '00H',
ADSC LITERALLY '01H',
BI LITERALLY '01H',
BO LITERALLY '02H',
CHAR$COUNT BYTE,
CHAR BYTE,
CHARS(80) BYTE,
CLEAR LITERALLY '00H',
CPT LITERALLY '00H',
CRLF LITERALLY '0AH',
DEC LITERALLY '08H',
DMA$ADR$LSTN POINTER,
DMA$ADR$TALK POINTER,
DMA$RD$LSTN(2) WORD AT (@DMA$ADR$LSTN),
DMA$RD$TALK(2) WORD AT (@DMA$ADR$TALK),
DMA$REG$L LITERALLY '10H',
DMA$REG$T LITERALLY '20H',
DNE LITERALLY '10H',
END$EOI LITERALLY '88H',
EGS LITERALLY '0DH',
ERR LITERALLY '04H',
GET LITERALLY '20H',
I BYTE,
LISTEN LITERALLY '04H',
MLA LITERALLY '04H',
MODE$1 LITERALLY '01H',
NO$DMA LITERALLY '00H',
NO$RSV LITERALLY '00H',
NORM$TIME LITERALLY '20H',
PDN LITERALLY '00H',
PPC LITERALLY '05H',
PPE$MASK LITERALLY '60H',
PPOLL$CNFG$FLAG LITERALLY '01H',
PPOLL$EN$BYTE BYTE,
PRI$BUF(80) BYTE AT (@CHARS),
RD$XFER LITERALLY '48H',
RESET LITERALLY '02H',
REMC LITERALLY '02H',
RSV LITERALLY '40H',
RXRDY LITERALLY '02H',
PL/M-86 Compiler

PORT DECLARATIONS

DECLARE ADDR#O LITERALLY 'OFFC6H',
ADDR#STATUS LITERALLY 'OFFC4H',
CLEAR#OFF LITERALLY 'OFFDDH',
CMD#37 LITERALLY 'OFFD8H',
COMMAND#MOD LITERALLY 'OFFC5H',
COUNT#HI LITERALLY 'OFFD1H',
COUNT#LO LITERALLY 'OFFD1H',
CPT#REG LITERALLY 'OFFC5H',
EOS#REG LITERALLY 'OFFC7H',
PORT#IN LITERALLY 'OFFCOH',
PORT#OUT LITERALLY 'OFFCOH',
SER#DATA LITERALLY 'OFFFOH',
SER#STAT LITERALLY 'OFFF2H',
SET#MASK LITERALLY 'OFFDFH',
SET#MODE LITERALLY 'OFFD8H',
SPOLL#STAT LITERALLY 'OFFC3H',
START#HI LITERALLY 'OFFD0H',
START#LO LITERALLY 'OFFD0H',
STATUS#1 LITERALLY 'OFFC1H',
STATUS#2 LITERALLY 'OFFC2H';

/* crt messages list */

DECLARE GET#MSG(11) BYTE DATA (ODH, OAH, 'TRIGGER', OAH, ODH);
DECLARE DEC#MSG(16) BYTE DATA (ODH, OAH, 'DEVICE CLEAR', OAH, ODH);
DECLARE REMC#MSG(10) BYTE DATA (ODH, OAH, 'REMOTE', OAH, ODH);
DECLARE CPT#MSG(22) BYTE DATA (ODH, OAH, 'UNDEF CMD RECEIVED', OAH, ODH);
DECLARE HUH#MSG(11) BYTE DATA (ODH, OAH, 'HUH ???', OAH, ODH);

REGER:  PROCEDURE;
PL/M-86 COMPILER

10 2  OUTPUT (SPOLL*STAT)=TRG;
11 2  DO WHILE (INPUT (SPOLL*STAT) AND SRGS)=SRGS;
12 3  END;
13 2  OUTPUT (SPOLL*STAT)=NRDSV;
14 2  END REGSER;
15 1  CO: PROCEDURE(XXX);
16 2  DECLARE
XXX BYTE;
17 2  DO WHILE (INPUT (SER*STAT) AND TXRDY)<<TXRDY;
18 3  END;
19 2  OUTPUT (SER*DATA)=XXX;
20 2  END CO;
21 1  HUH:  PROCEDURE;
22 2  DO I=0 TO 10;
23 3  CALL CO (HUH*MSG(I));
24 3  END;
25 2  END HUH;
26 1  CI:  PROCEDURE;
27 2  IF (INPUT (SER*STAT) AND RXRDY)=RXRDY THEN
28 2  DO:
29 3  I=0;
30 3  CHAR*COUNT=0;
31 3  STORE*CHAR:
32 3  CHAR=(INPUT (SER*DATA) AND 7FH);
33 3  CHAR*COUNT=CHAR*COUNT+1;
34 3  CALL CO (CHAR);
35 3  CHAR(I)=CHAR;
36 3  I=I+1;
37 3  IF CHAR <> CRLF THEN
38 4  DO;
39 5  DO WHILE (INPUT (SER*STAT) AND RXRDY)<>RXRDY;
40 4  END;
41 4  GOTO STORE*CHAR;
42 3  END;
43 3  CALL REGSER;
44 2  END CI;
45 1  TALK*EXEC:  PROCEDURE;
46 2  OUTPUT (STATUS*2)=CLEAR;
47 2  /*
manipulate address bits for DMA controller
*/
48 2  DMA*ADR*TALK=(@CHARS);
49 2  DMA*WRD*TALK(1)=SHL(DMA*WRD*TALK(1),4);
50 2  DMA*WRD*TALK(0)=DMA*WRD*TALK(0)+DMA*WRD*TALK(1);
51 2  OUTPUT (CLEAR*OFF)=CLEAR;
PL/M-86 COMPILER

51 2 OUTPUT (CMD37)=NORM*TIME;
52 2 OUTPUT (SET*MODE)=RD*XFER;
53 2 OUTPUT (SET*MASK)=CLEAR;
54 2 OUTPUT (START*LO)=DMA*WRD*TALK(O);
55 2 DMA*WRD*TALK(0)=SHR(DMA*WRD*TALK(0),B);
56 2 OUTPUT (START*HI)=DMA*WRD*TALK(0);
57 2 OUTPUT (COUNT*LO)=CHAR*COUNT;
58 2 OUTPUT (COUNT*HI)=O;
59 2 OUTPUT (EOS*REG)=EOS;
60 2 OUTPUT (COMMAND*MOD)=END*EOI;
61 2 DO WHILE (INPUT (STATUS*1) AND BO)=O;
62 3 END;
63 2 OUTPUT (PORT*OUT)=OAAH;
64 2 DO WHILE (INPUT (STATUS*1) AND ERR)=ERR;
65 3 DO WHILE (INPUT (STATUS*1) AND BO)=O;
66 4 END;
67 3 OUTPUT (PORT*OUT)=OAAH;
68 3 END;
69 2 OUTPUT (STATUS*2)=DMA*REG*T;
70 2 END TALK*EXEC;

71 1 LISTEN*EXEC: PROCEDURE;
72 2 OUTPUT (STATUS*2)=CLEAR;
73 2 OUTPUT (CLEAR*FF)=CLEAR;
74 2 OUTPUT (CMD*37)=NORM*TIME;
75 2 OUTPUT (SET*MODE)=WR*XFER;
76 2 OUTPUT (SET*MASK)=CLEAR;
77 2 DMA*ADR*LSTN=(@CHARS);
78 2 DMA*WRD*LSTN(1)=SHL(DMA*WRD*LSTN(1),4);
79 2 DMA*WRD*LSTN(0)=DMA*WRD*LSTN(0)+DMA*WRD*LSTN(1);
80 2 OUTPUT (START*LO)=DMA*WRD*LSTN(0);
81 2 DMA*WRD*LSTN(0)=SHR(DMA*WRD*LSTN(0),B);
82 2 OUTPUT (START*HI)=DMA*WRD*LSTN(0);
83 2 OUTPUT (COUNT*LO)=TC*LO;
84 2 OUTPUT (COUNT*HI)=TC*HI;
85 2 OUTPUT (STATUS*2)=DMA*REG*L;
86 2 END LISTEN*EXEC;
87 1 PRINTER: PROCEDURE;
88 2 I=0;
89 2 DO WHILE PRI*BUF(I) <>CRLF;
90 3 CALL CO (PRI*BUF(I));
91 3 I=I+1;
92 3 END;
93 2 CALL CO (PRI*BUF(I));
94 2 END PRINTER;
95 1 ADSC*EXEC: PROCEDURE;
96 2 TA*OR*LA=INPUT (ADDR*STATUS);
97 2 IF (TA*OR*LA AND TALK)=TALK THEN
98 2 CALL TALK*EXEC;
99 2 IF (TA*OR*LA AND LISTEN)=LISTEN THEN
100 2 CALL LISTEN*EXEC;
101 2 END ADSC*EXEC;
102 1 GET*EXEC: PROCEDURE;
103 2 DO I=0 TO 10;
104 3 CALL CO (GET*MSG(I));
105 3 END;
106 2 END GET*EXEC;
107 1 DEC*EXEC: PROCEDURE;
108 2 DO I=0 TO 15;
109 3 CALL CO (DEC*MSG(I));
110 3 END;
111 2 END DEC*EXEC;
112 1 REMC*EXEC: PROCEDURE;
113 2 DO I=0 TO 9;
114 3 CALL CO (REMC*MSG(I));
115 3 END;
116 2 END REMC*EXEC;
117 1 PPOLL*CON: PROCEDURE;
118 2 OUTPUT (COMMAND*MOD)=PPOLL*CNFG*FLAG;
119 2 END PPOLL*CON;
120 1 PPOLL*EN: PROCEDURE;
121 2 PPOLL*EN*BYTE=(UDC AND 6FH);
122 2 OUTPUT (COMMAND*MOD)=PPOLL*EN*BYTE;
123 2 END PPOLL*EN;
124 1 CPT*EXEC: PROCEDURE;
125 2 DO I=0 TO 21;
126 3 CALL CO (CPT*MSG(I));
127 3 END;
128 2 UDC=INPUT (CPT*REG);
129 2 UDC=(UDC AND 7FH);
130 2 IF (UDC AND PPC)=PPC THEN
131 2 CALL PPOLL*CON;
132 2 IF (UDC AND PPE*MASK)=PPE*MASK THEN
133 2 CALL PPOLL*EN;

6-525
230832-001
PL/M-86 COMPILER

END CPT$EXEC;

/*
BEGIN CODE
*/

INIT:

OUTPUT (CLEAR$FF) =CLEAR;
OUTPUT (COMMAND$MOD) =RESET;
OUTPUT (ADDR$STATUS) =MODE$1;
OUTPUT (ADDR$0) =MLA;
OUTPUT (STATUS$2) =NO$DMA;
OUTPUT (COMMAND$MOD) =PON;

LISTENERS:

/\* response to listeners check */

DO WHILE (INPUT (STATUS$1) AND BI)=0;
END;

XYZ=INPUT (PORT$IN);
XYZ=INPUT (STATUS$2);

READSTAT:
/\* read status registers and interpret command */

STAT1=INPUT (STATUS$1);
STAT2=INPUT (STATUS$2);

IF (STAT1 AND DNE)=DNE THEN
CALL PRINTER;
IF (STAT1 AND CPT)=CPT THEN
DO;
CALL CPT$EXEC;
STAT2=(STAT2 AND OFEH);
END;
IF (STAT1 AND GET)=GET THEN
DO;
CALL GET$EXEC;
STAT2=(STAT2 AND OFEH);
END;
IF (STAT1 AND DEC)=DEC THEN
DO;
CALL DEC$EXEC;
STAT2=(STAT2 AND OFEH);
END;
IF (STAT2 AND REMC)=REMC THEN
DO;
CALL REMC$EXEC;
STAT2=(STAT2 AND OFEH);
END;
IF (STAT2 AND ADSC)=ADSC THEN
PL/M-B6 COMPILER

170 1  DO;
171 2  CALL ADSC*EXEC;
172 2  STAT2=(STAT2 AND OFEH);
173 2  END;
174 1  CALL CI;
175 1  GOTO CMD;
176 1  END;

MODULE INFORMATION:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Area Size</td>
<td>0475H</td>
</tr>
<tr>
<td>Constant Area Size</td>
<td>0000H</td>
</tr>
<tr>
<td>Variable Area Size</td>
<td>0061H</td>
</tr>
<tr>
<td>Maximum Stack Size</td>
<td>000AH</td>
</tr>
<tr>
<td>Lines Read</td>
<td>349</td>
</tr>
<tr>
<td>Program Error(S)</td>
<td>0</td>
</tr>
</tbody>
</table>

END OF PL/M-B6 COMPILATION
LSI Transceiver Chips Complete GPIB Interface

Reprinted with permission from COMPUTER DESIGN, March 1982.

June 1982

6-528
LSI TRANSCEIVER CHIPS COMPLETE GPIB INTERFACE

A GPIB interface meeting IEEE 488 standards can be built with only three or four chips!

by Pradip Madan and Jim Frederick

The decision to support the IEEE 488 standard with integrated circuits was based on the potential popularity of the interface standard and its applications potential. Although a serial interface supports many system throughput requirements, a parallel interface over short distances can provide much higher data transfer rates, yet remain economical despite the extra interconnection copper required.

The IEEE 488 standard is for a parallel interface designed to operate over a limited distance. Its general purpose nature makes the general purpose interface bus (GPIB) attractive for a variety of systems, and also allows manufacturers to design their equipment interfaces to a common standard. As a result, users can mix equipment from different manufacturers without having to adapt the interfaces for compatibility. To date, the GPIB has been incorporated in computer peripherals, such as printers, but the most applications have been in programmable instrumentation systems. Other GPIB applications include camera control in computer controlled studios, electronic surveillance, peripheral control, modular add-ons to photocopiers, and so forth.

Pradip Madan is the product manager for microprocessor peripheral components at Intel Corp., 2625 Walsh Ave, Santa Clara, CA 95051, where he has been employed for 2 years. He has a BSEE, an MS in computer science, and an MBA in finance.

Integration benefit

Shortly after the IEEE committee had put the final touches on its standard specifications, engineers began building GPIB interface sub-systems. Because the standard had just been defined, there were no large scale integration (LSI) chips available. Therefore, the first GPIB implementations were board level designs replete with small scale integration (SSI) and medium scale integration (MSI) logic chips. A typical effort included four or five rows of ten chips each.

With the advent of integrated circuit GPIB chips, chip counts dropped dramatically, reliability improved, and space requirements shrank. Consequently, the price range of systems for which GPIB had become practical began to decrease. A fully functional GPIB subsystem can now be constructed with less than one-tenth the number of chips formerly required. In fact, the complete

Jim Frederick is a microcomputer design engineer at Intel Corp. Since joining the company in 1974, he has been involved in several different projects. Mr. Frederick has studied at the College of San Mateo, and the University of Santa Clara.
Choosing appropriate active devices

All of the 8293's functional elements required only four different types of active field effect transistors (FETs). Low threshold enhancement type devices show good high output voltage characteristics, and were used as output pullup devices in push-pull 3-state drivers. Enhancement type FETs were also used for fast switching and low leakage: depletion type devices were used for resistive pullup in buffering. Depletion type FETs also played an important role in meeting the hysteresis specifications of the IEEE 488 standard. Finally, higher threshold depletion type devices were used to prevent the bus lines from being disturbed on power-up and power-down.

Choosing of lengths, smaller size, and lower parasitic capacitance before, HMOS that transceivers. Nine open collector or 3-state line drivers, transceiver, and/or talker/listener/controller data transceiver, and/or talker/listener/controller data transceiver, and/or talker/listener/controller data transceiver, and/or talker/listener/controller data transceiver, and/or talker/listener/controller data transceiver. Thus, a 2-pin select scheme allows a user to select the desired operating mode.

A conventional MOS transistor capable of supplying 48 mA at 0.5 V would have been physically too large. HMOS technology, however, permits such a device to be fabricated in an area of less than 150 mil² (97 mm²). Furthermore, the low speed/power product of HMOS allowed a multi-stage design so that, like transistor-transistor logic (TTL) circuitry, natural hysteresis could be built into the receivers.
**Special layout techniques**

The transceiver was implemented using new layout techniques aimed at reducing the series resistance in the polysilicon gate structures of the large transistors, and routing ac signal paths over metal interconnects in order to reduce capacitance and series resistance. Chip size, 188 x 156 mils (5 x 4 mm), includes a 7-mil (0.2-mm) ground line and two ground pads in order to handle the 432-mA current generated when all drivers are on. Power consumption is 300 mW, typically, with driver or receiver speeds of 20 ns under light loads and speeds of 85 ns under the maximum load of 4500 pF.

**Signaling a new trend?**

Until the advent of the 8293, complex MOS chips relied on bipolar drivers to handle the heavy bus loading found in complex systems. The 8293 could point the way to future microprocessors and controllers that include their own MOS drivers. Such a scheme would significantly reduce the time lost by going through external buffers. It would also provide all the other benefits of system integration.

The 8293 is essentially a non inverting buffer chip capable of driving high currents. The 8291A talker/listener chip and 8292 GPIB controller chip are designed to interface with the 8080, 8085, iAPX 86, iAPX 88, and 8048/8051 microprocessors and single-chip microcomputers. However, the 8291A and 8292 cannot electrically drive a standard IEEE 488 bus by themselves. Thus, the 8293 was designed to interface between the GPIB and a single 8291A or a combination of the 8291A and 8292. (See Fig 2.)

The chip is divided into nine distinct transceivers. Each one's characteristics, such as 3-state or open-collector outputs, and transmit or receive modes of operation, are determined by internal logic control. (See Fig 3.) Thus, in mode 0 talker/listener control configuration the attention (ATN) transceiver is forced into an input-only mode with respect to the bus's ATN line. The end or identify (EOI) transceiver, on the other hand, is either a transmitter or receiver depending on the state of the transmit/receive (T/R2) line. Its interface to the GPIB is 3-state because of the fixed 5 V logic on the EOI transceiver's output control. In mode 1, the talker/listener data configuration, the 8293 is a true transceiver with its operations mode controlled by the state of the T/R line and its output characteristics (3-state or open-collector) determined by the states of the ATN and EOI lines. (See Fig 3.)

**Fig 1** 3-state driver schematic. Nine such open collector drivers are used in the interface.

**Fig 2** 8293 is designed for use in talker/listener implementation (a), or for talker/listener/controller interface (b).
The talker/listener/controller control configuration, mode 2, is a full transceiver mode but the operation mode of the transceivers is determined by more complex combinational logic. (See Fig 4.) The fourth mode (mode 3), which is the talker/listener/controller data configuration, is again a true transceiver whose mode of operation is controlled by the state of the T/R line. In this mode, some additional interval combinational logic is enabled to permit the 8293 to support the 8292 in taking bus control synchronously.

**...complete talker/listener/controller mode logic resides in four LSI chips.**

The 8293's overall mode (mode 0, 1, 2, or 3) is determined by the state on the option pins 26 and 27. For example, if both pins are tied low (0 V), the chip is in mode 0. If both are high (5 V) it is in mode 3. The particular state of these pins will determine the characteristics of the other 26 pins. (See the Table, "8293 Mode Selection Pin Mapping.")

**Talker/listener only**

If the IEEE 488 is to be implemented in a system that is able to talk and listen (eg, a digital multimeter), only talk (eg, a counter), or only listen (eg, a signal generator),
MODE 2

MODE 3

Fig 4 Mode 2 is control configuration. Operating nodes of individual transceivers are controlled by external signals and internal combinational logic. Chip in mode 3 acts like true transceiver, as in mode 1, except some extra functions have been included in order to support controller function. In (a), talker/listener/controller configuration is for control, and in (b), for data.

then the entire interface can be built with a single 8291A and a pair of 8293s. (See Fig 5.) In this configuration, one 8293 handles the eight data lines DIOI to DIO8 and the other handles the data-byte transfer handshake lines and general interface management lines. Both transceivers are connected to the 8291A's ATN, and E01, and T/R1 lines.

Talker/listener/controller

For an IEEE 488 controller (like the HP 85 or Tektronix 4051), the system must be able to take control of the bus or delegate it to another controller. Such an interface scheme can be implemented using an 8291A, an 8292, and a pair of 8293s. (See Fig 6.) The arrangement is similar to that of a talker/listener interface; one 8293 handles the DIO1 through DIO8 bus data lines and the other handles the data byte transfer handshake and general interface management lines. The difference is that pins 26 and 27 have been selected for modes 2 and 3 and several additional control functions have been added. The attention in (ATNI) lines and attention out (ATNO) lines permit the 8292 to monitor the GPIB's ATN line and take control of the bus. In conjunction with the ATN line, the E012 line is used by the 8292 to initiate a polling sequence.

The chip is divided into nine distinct transceivers and each one's characteristics are determined by internal logic.

Lastly, the system controller line (SYC) enables the control function. If it is low, the 8292 is prevented from acting as a controller. If it is switched high, the 8292 can act as a controller. In essence, the SYC controls the direction of the interface clear (IFC) and remote enable (REN) signals.
Fig 5  Talker/listener only implementation can be built using just three chips—single 8291A and a pair of 8293. First (upper) transceiver chip is used for bidirectional data flow on D101 to D108 data lines. Lower 8293 handles some of data byte transfer control lines and general interface management lines.

8293 MODE SELECTION PIN MAPPING

![Diagram of 8293 Mode Selection Pin Mapping]

*These pins are the IEEE 488 bus noninverting driver/receivers. They include all the bus terminations required by the standard, and connect directly to the GPIB connector.
**Summary**

Before the advent of integrated solutions for IEEE 488 implementation, it usually took forty to fifty SSI and MSI chips to build this interface. A large portion of those were eliminated by controllers and interface chips like the 8291A and 8292. Now, with the last part of the interface available in LSI, a fully functional interface can be built using only four LSI chips. The cost of the original design was typically $400 to $500. A set of the three chips, the 8291A, and two 8293s (for a talker/listener function) allows a 15-fold reduction in cost. The power dissipation of a 40-chip interface was in the vicinity of 10 W. The power dissipation of the 4-chip approach is a mere 1.5 W. The size of the PC board is considerably smaller, too, and that lowers the manufacturing costs and improves reliability.

---

**Fig 6** Fully functional talker/listener/controller interface can be built with only four LSI chips; the 8291A, 8292, and a pair of 8293s. Like simpler talker/listener only case, one 8293 handles data transceiver functions while other handles data byte transfer control and general interface management. There are additional control lines enabled which support the controller (8292) activity.
LSI Chips Ease Standard 488
Bus Interfacing

Ronald M. Williams
Intel Corporation, Santa Clara, California
LSI CHIPS EASE STANDARD 488
BUS INTERFACING

Time and cost disadvantages of interfacing to the IEEE Std 488 bus are overcome with a dedicated LSI chip set that incorporates most of its functional and electrical specifications.

Ronald M. Williams  Intel Corporation, Santa Clara, California

Historically, interface techniques proliferated as designers evolved customized links among instruments, controllers, and processors for realtime test measurements or data communications, resulting in excessive and expensive codes, formats, signal levels, and timing factors. Obviously, interface standardization was mandatory to save design costs for engineers, development costs for manufacturers, and system integration costs for users. Thus, IEEE Standard 488-1978 (a revision of ANSI/IEEE Std 488-1975) offers a universal instrumentation system approach to automatic operating measurement configurations that provides compatibility, versatility, and flexibility. This system approach establishes a suitable standard bus for interfacing programmable devices from different manufacturers. Outstanding advantages of the standard bus include byte serial, bit parallel digital data handling, synchronized communication among devices at varying data rates, and hardware interchangeability and interconnection in daisy-chained fashion. However, some restrictive disadvantages that have hindered implementation are highly complex logic protocol, time consuming design analysis, and lack of low cost components to perform the intricate logic control functions. To overcome these drawbacks, a large scale integrated (LSI) chip set has been designed with built-in IEEE Std 488 logic controls. Thus,
interfacing has been significantly simplified for properly connecting processor buses and programming system protocols.

**Interface Overview**

The IEEE Standard 488-1978 bus interface includes electrical, mechanical, and functional specifications* for interconnecting both programmable and nonprogrammable electronic measuring apparatus with other apparatus and accessories necessary to assemble instrumentation systems. The functional specifications occupy about 80% of the document and involve a proportional amount of system design time to implement.

*This article deals with the functional aspects (interface signals that exist on the physical bus) of IEEE Std 488-1978, and is not intended as a complete dissertation on the major elements of the standard. For detailed definitions of the mechanical (physical cable connections), electrical (timing, voltages, and currents), and operational (application software routines) technicalities, interested readers should consult the IEEE Standard Digital Interface for Programmable Instrumentation, IEEE Std 488-1978, Institute of Electrical and Electronics Engineers, Inc, New York, NY 10017, Nov 30, 1978—Ed.

Bus functions encompass 16 active signal lines, 10 interface functions, the protocol by which interface functions send and receive messages, and logical and timing relationships between signal states.

Functional requirements of the standard can be incorporated in either hardware, software, or a combination of both. Some designers have chosen the hardware approach to incorporate all the interface functions, using about 200 medium scale integrated (MSI) and small-scale integrated (SSI) packages. This technique costs about $1000 for a complete interface board. As a result, many cost sensitive implementations of the bus interface use only a subset of its functions custom tailored to the requirements of the devices involved, thereby reducing package count and expense by curtailing the interchangeability advantages.

Other designers have selected the software approach to implement the bus interface. One disadvantage of this approach is that programming is an expensive and extended project; another is that a subroutine has to be executed with each transferred byte. This overhead not only burdens the microprocessor within a device, but also reduces the overall speed of the bus. This approach costs about $200 for the interfacing functions.

---

**Fig 1** IEEE Std 488 active signal lines for multiple devices. Peripheral devices of different characteristics can be easily connected to standard bus interface. Controller (or processor), such as a mini-computer, enables and disables talkers and listeners and manages overall bus activity. Bubble memory functions as both talker and listener. As listener: printer receives characters to be printed. As talker, counter transmits measurements to both controller and listeners.
Combinational hardware/software approaches, although faster than direct software implementations, still require enormous design time and cost about $1000 for a typical interface board.

With a recent alternative approach, however, the bus interface is easier and less expensive to incorporate in instrument designs. LSI circuit chips now include as built-in capabilities most of the functional and some of the electrical portions of the Standard’s specifications, significantly reducing design time and costing about $50 for bus interfacing. Additionally, Intel’s 8291/8292 General Purpose Interface Bus (GPIB) peripheral chip set also incorporates capabilities for bus monitoring, data rate manipulation, and addressing to further simplify bus interface designs.

**Bus Signal Definitions**

The IEEE Std 488 signals are defined as negative true, where the high state (0 = false, ≥2.0 V) and the low state (1 = true, ≤0.8 V) are based on standard transistor-transistor logic (TTL) levels. Of the 16 active signal lines, 8 are data lines, 5 are interface management lines, and 3 are handshake lines (Fig 1). Data input/output lines (DIO1-DIO8) carry ASCII-coded information, as well as device addresses, universal commands, or program instructions. Interface management lines help to supervise the data lines. The primary management line—Attention (ATN)—determines how data lines are processed. When ATN is true, data lines are interpreted as addresses or universal commands by all bus connected devices. When ATN is false, only those devices addressed can use the data lines; in this case, data transmitted are typically device-dependent. With another management line, Interface Clear (IFC), the bus controller returns the system to a known quiescent state. The Service Request (SRQ) line can be used by any device on the interface bus when it has data to send (talker) or needs to receive data (listener). The Remote Enable (REN) line determines whether the system is under front panel or program control. The End Or Identify (EOI) line can be used as a delimiter by a talker (sending) device to indicate an end of message, or by the controller as a polling line.

Handshake lines control the timing relationship of the interface bus (Fig 2). The Data Valid (DAV) line

---

**Fig 2** Three-wire handshaking between single talker and several listeners. Before transfer begins, listener indicates it is ready by asserting Ready For Data (RFD) message to true. Talker then drives all eight data input/output lines. Following settling time specified by standard, talker asserts Data Valid (DAV) message to true. While data are being read, RFD message is asserted to false since device is unable to receive additional data. As each listener completes its read, it indicates acceptance by asserting Data Accepted (DAC) message to true; DAC is not sensed true by talker until all listeners have completed read. After each device indicates acceptance, it indicates readiness for data by asserting RFD to true. New cycle begins when all devices have asserted RFD to true.

---
Fig 3 GPIB talker/listener chip. 8291 chip connects 8-bit microprocessor to noninverting bus transceivers, which, in turn, connect to IEEE Std 488 bus. Microprocessor manipulates data bytes after receipt or before transmission, and monitors talker/listener status. Single chip handles all IEEE Std 488 interface functions, except controller functions.
Fig 4 Bus interface functions. Messages received from interface bus can cause state transitions, just as state transitions can cause messages to be sent on bus (1 and 2). Device dependent data are transferred automatically to microprocessor, without affecting state transitions (3). State changes in one function can cause state changes in another function, resulting in message to be sent (4). Microprocessor can also send local messages to interface functions (5) or remote messages to interface (6).
is used by a talker device to indicate that data are ready to transmit. The Not Ready For Data (NRFD) and Not Data Accepted (NDAC) lines are used by a listener to indicate readiness to receive data and receipt of data, respectively. As a result, a talker knows when all listeners on the bus have received an 8-bit byte of information. Thus, the transmission rate of the bus is only as fast as the slowest listener.

Messages conveyed by all 16 lines are true or false, depending on the states of 10 interface functions. The standard defines each of these interface functions with state diagrams. A function's state can be changed by a controller, another device on the bus, or a state change in another function within a device. Of the 10 interface functions, four provide basic communication capabilities: Source Handshake (SH), Talker (T), Acceptor Handshake (AH), and Listener (L). These functions affect the three handshake lines (DAV, NRFD, and NDAC), eight data lines (DI01-DI08), and EOI management line. The Device Clear (DC) and Device Trigger (DT) interface functions are used to initialize and to trigger a device, respectively. The Parallel Poll (PP) function acts with the EOI line to send a single bit of status information. The Service Request (SRQ) function controls the SRQ management line. The Remote Local (RL) interface uses the REN management line in conjunction with front panel control. The Controller (C) function, which is active in only one device on the bus at a time, determines which device talks or listens.

To date, these 10 interface functions and their intricate interrelationship and timing factors have required difficult and time consuming efforts when designing the interface bus into a digital system.

**Talker/Listener Chip Capabilities**

The 8291 GPIB talker/listener chip, a 40-pin LSI device (Fig 3), performs the inversion necessary to connect an 8-bit microprocessor bus to the negative true IEEE Std 488 bus. In addition, this chip implements most of the Standard's required functions. The microprocessor sets the talker/listener chip to an initial state, manipulates bytes before or after transmission, performs interrupt service routines, causes state changes, monitors other state changes, and enables and disables chip capabilities.

Without microprocessor involvement, the talker/listener chip implements all interface functions, except controller performance, such as handling data transfers, handshake protocols, listener/talker address procedures, device clearing and triggering, service requests, and parallel and serial polling schemes (Fig 4).

Within the chip architecture are eight read (output) and eight write (input) registers. One input register holds the data that are to be moved from the bus to the microprocessor when a device is listening. An output register holds the data byte that is to be transferred to the bus when a device is ready to talk. The other seven write and seven read registers control various chip functions.

Interrupt status registers 1 and 2 store 12 different interrupt flags. For example, one bit in the Interrupt Status 2 register reflects changes in a device's addressed state. The microprocessor can poll both registers to determine which flag caused the interrupt, and can then branch to the appropriate service routine. Two corresponding interrupt mask registers allow designers to mask any interrupt. A serial poll status register holds device status information, and a serial poll mode register is available so that the microprocessor can verify this status. An address mode register contains a device's addressing mode, as determined by the microprocessor. An address status register monitors the address status (ie, active talker or active listener) of a device.

Two address registers store the assigned device addresses. An End-Of-Sequence (EOS) register contains a designer specified end of string code for delimiting data block transfers by flagging the last byte with EOI. A command pass-through register feeds non-GPIB commands to the microprocessor. An auxiliary, mode register holds local messages to control reset, power on, etc.

Among the chip's capabilities are a programmable data transfer rate from 62k to 525k bytes/s, three addressing modes, and an EOS message recognition. With a programmable data transfer rate, the designer controls the handshake rate of the interface to match the data transfer rate to the devices on the bus.

The three addressing modes permit flexibility in designating talkers/listeners. The dual primary address mode, for example, allows both a talker and a listener address to be assigned to a device. With the primary/secondary address mode, multiple devices of the same type can have the same primary address, but a different secondary address. In the third addressing mode, devices can have both dual primary and dual secondary addresses.

Data block transfers are made easier with the EOS register. This register holds the character that signals an end-of-block transfer. When a data byte loaded into the data-out register matches the byte in the EOS register, the talker/listener chip asserts the EOI line, signaling an end of transfer.

**Controller Chip Capabilities**

The 8292 controller chip (Fig 5) implements the controller function of the Standard. In conjunction with the 8291, the controller forms a complete standard interface, including the capability of handling the transfer control protocol. This ability gives the designer an option to accommodate multiple controllers on a single bus.

Additionally, the 8292 performs all the tasks necessary in a complete controller design. It responds to
Fig 5  GPIB controller chip. 8292 chip works in conjunction with 8291 to perform GPIB controller interface functions. It implements local control commands from microprocessor according to IEEE Std 488 protocol. Additionally, it processes such inputs from bus as SRQ and EOI. Furthermore, it can send the full repertoire of GPIB control messages, including REN, IFC, ATN, and EOI.
Fig 6 System configuration using chip set. In conjunction with 8291, 8292 performs complete controller function. Together with shared bus transceivers, chip set forms a complete IEEE Std 488 interface. In addition, DMA interface may be implemented through 8291 with 8237 DMA controller.
service requests (SRQs), configures other devices on the bus for remote control by sending Remote Enable (REN), and sends Interface Clear (IFC), allowing for control seizure to reinitialize the bus. More importantly, the controller chip can take control of the bus synchronously with the handshake, preventing the destruction of any data transmission in progress.

Internally, the controller chip has 10 dedicated registers for programming and for monitoring status. Through the use of the Interrupt Status and Interrupt Mask registers, the designer can configure the controller to interrupt the microprocessor on selected events. An Event Counter and a corresponding status register are available to monitor and control either byte counts or block counts. A Time-Out register may be set by the designer to program a time-out error function; a corresponding status register contains the current value in the time-out counter. In conjunction with these registers, error control can be programmed with the Error Flags and Error Mask registers. Finally, Controller and GPIB Status registers are available. Each of these registers is read or programmed through a dedicated command buffer.

**Chip Set Application**

The talker/listener and controller chips connect to the standard interface bus through noninverting bus transceivers (Fig 6). These transceivers provide the 48-mA bus drive capability needed to meet the electrical portion of the IEEE Std 488 specification—not directly possible with existing metal oxide semiconductor (MOS) parts. The talker/listener chip can interface directly to microprocessor memory through a direct memory access (DMA) controller, such as an 8237.

The microprocessor drives the talker/listener with a short stored program (see Table), containing initialization conditions, such as data transfer rate, address mode, and other designer requirements. Microprocessor data handling is limited to taking bytes off the bus after they arrive or putting bytes on the bus. Interrupt service routines are necessary for each unmasked interrupt. Although 12 interrupts are available, not all have to be used. All other standard bus functions are handled by the 8291.

To send a byte of data, the microprocessor writes the byte into the talker/listener data-out register. The chip then transmits the data byte over the bus lines in conjunction with the handshake lines. Next, the NRFD line is checked to see if it is ready for data. If a ready for data message is detected, the talker/listener sends a DATA signal until it receives a data accepted message from the interface's NDAC line. The 8291 also generates a Byte Out (BO) interrupt, setting the BO flag in the interrupt status register. When its interrupt pin is activated, the microprocessor reads the interrupt status register and responds to the interrupt with an appropriate service routine.

The 8292 handles all hardware aspects of the controller function: SRQ input, ATN, IFC, EOI, and REN outputs. Meanwhile, the designer-defined aspects of a given GPIB system are handled by processor software. For example, the processor is responsible for knowing which device on the bus corresponds to which device address. The processor then uses the 8291 to transmit coded Controller commands as the 8292 asserts ATN.

**Summary**

Bus interface designs that previously required 150 or 200 MSI/SSI chips may now be implemented with a GPIB peripheral chip set. For designers, this hardware set means less design time and cost, resulting in increased reliability and versatility in IEEE Std 488 bus interfaces custom programmed for dedicated applications.

**Bibliography**


Ronald M. Williams is a product manager for peripheral controllers in Insignia Microcomputer Components Div. of Insignia Corporation, Prof. Williams has been involved in introductory computer courses. He received his B.S. degree in mathematics from Rensselaer Polytechnic Institute of Chicago.
DATA ENCRYPTION TUTORIAL

The proliferation of electronic data processing (EDP) applications that involve the storage and the distribution of potentially sensitive information have demonstrated the need for mechanisms to insure data privacy and security. As society becomes increasingly dependent on computers and data communications networks, this need becomes even more acute.

Cryptography

The most efficient technique of providing data security is cryptography: the transformation of data via a secret code into a form which is useless to anyone but authorized recipients.

A cryptographic algorithm can be presented as a sequence of mathematical transformations. Each transformation has its unique inverse operation that changes the encrypted data back into the original plain text. In conventional cryptosystems, a set of specific parameters called a key is supplied along with the plain text/cipher text as an input to the enciphering/deciphering algorithm. The key is specified by the user. The transformation of the plain text and the cipher text depends on the key as well as the enciphering and deciphering algorithms. In fact the algorithms themselves can be made public, because the security of the system depends entirely on the secrecy of the key.

The initial interest in encryption for commercial applications came from financial institutions, most notably banks that are heavily involved in Electronic Fund Transfer (EFT). The American banking system alone, moves more than $400 billion between computers every day. The rapid rise of personal computers, workstations and the use of electronic mail and information retrieval services have spread the need for insuring data privacy and security to many other applications.

The DES

In response to the growing commercial need, the National Bureau of Standards has adopted in 1977 a standard algorithm know as the Data Encryption Standard (DES). The DES, originally developed by IBM, is designed for use with sensitive but unclassified information. The National Bureau of Standards requires that the DES be implemented in system hardware. The standardization insures that certified hardware from different suppliers are compatible.

The DES specifies a method for encrypting 64 bit blocks of clear data into corresponding 64 bit blocks of cipher text using a 56 bit key user specified. The 56 bit key (64 bit with parity) gives the user a total of 256 (seventy quadrillion) possible keys. Because the DES algorithm key is so long, a state of the art computer would take years to explore all possible permutations required to break the code. The most critical factor in protecting the data is guaranteeing the secrecy of the key.

Intel Data Encryption Product Line

Intel offers two peripherals supporting the DES algorithm: the 8294A Data Encryption Unit (DEU) and the 82538 Data Ciphering Processor (DCP).

The 8294A - a preprogrammed 8042- can encrypt and decrypt data at a rate up to 400 Byte/Sec. The 8294A is very well suited for data file protection, off line data encryption prior to transmission, and phone line applications.

The 82538 is a much faster device: 1.5 Mbyte/Sec: This encryption rate is needed in satellite communications systems, data storage onto hard disks, high performance data communications networks like Ethernet. This rate is high enough to accommodate the fly encryption in most of the communications systems and eliminate the need for buffers and interfacing circuitry. High encryption and decryption speed is not the only feature of this device. The 82538 supports bi-directional, half-duplex operations at its top speed. It contains three separate write only registers for encryption, decryption and master keys improving system's security and throughput. The DCP can also be configured in any of the three encryption/decryption modes recommended by the NBS (ECB, CBC or CFB).

The Intel Data Encryption product line solves the need for a broad range of applications. Security features can now be economically designed in data entry terminal as well as in satellite communications systems.
Video Display

Peripherals Section
8275H
PROGRAMMABLE CRT CONTROLLER

- Programmable Screen and Character Format
- 6 Independent Visual Field Attributes
- 11 Visual Character Attributes (Graphic Capability)
- Cursor Control (4 Types)
- Light Pen Detection and Registers
- MCS-51®, MCS-85®, iAPX 86, and iAPX 88 Compatible
- Dual Row Buffers
- Programmable DMA Burst Mode
- Single +5V Supply
- High Performance HMOS-II

The Intel® 8275H Programmable CRT Controller is a single chip device to interface CRT raster scan displays with Intel® microcomputer systems. It is manufactured on intel's advanced HMOS-II process. Its primary function is to refresh the display by buffering the information from main memory and keeping track of the display position of the screen. The flexibility designed in the 8275H will allow simple interface to almost any raster scan CRT display with a minimum of external hardware and software overhead.

Figure 1. Block Diagram
Figure 2. Pin Configuration
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC 3</td>
<td>1</td>
<td>O</td>
<td>Line Count: Output from the line counter which is used to address the character generator for the line positions on the screen.</td>
</tr>
<tr>
<td>LC 2</td>
<td>2</td>
<td>O</td>
<td>DMA Request: Output signal to the 8257 DMA controller requesting a DMA cycle.</td>
</tr>
<tr>
<td>LC 1</td>
<td>3</td>
<td>O</td>
<td>DMA Acknowledge: Input signal from the 8257H DMA controller acknowledging that the requested DMA cycle has been granted.</td>
</tr>
<tr>
<td>LC 0</td>
<td>4</td>
<td>O</td>
<td>Horizontal Retrace: Output signal which is active during the programmed horizontal retrace interval. During this period the VSP output is high and the LTEN output is low.</td>
</tr>
<tr>
<td>DACK</td>
<td>6</td>
<td>I</td>
<td>Vertical Retrace: Output signal which is active during the programmed vertical retrace interval. During this period the VSP output is high and the LTEN output is low.</td>
</tr>
<tr>
<td>HRTC</td>
<td>7</td>
<td>O</td>
<td>Light Pen: Input signal from the CRT system signifying that a light pen signal has been detected.</td>
</tr>
<tr>
<td>VRTC</td>
<td>8</td>
<td>O</td>
<td>Read Input: A control signal to read registers.</td>
</tr>
<tr>
<td>RD</td>
<td>9</td>
<td>I</td>
<td>Write Input: A control signal to write commands into the control registers or write data into the row buffers during a DMA cycle.</td>
</tr>
<tr>
<td>LPEN</td>
<td>11</td>
<td>I</td>
<td>General Purpose Attribute Codes: Outputs which are enabled by the general purpose field attribute codes.</td>
</tr>
<tr>
<td>DB 0</td>
<td>12</td>
<td>I/O</td>
<td>Bi-Directional Three-State Data Bus Lines: The outputs are enabled during a read of the C or P ports.</td>
</tr>
<tr>
<td>DB 1</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB 2</td>
<td>14</td>
<td></td>
<td>GPA 34</td>
</tr>
<tr>
<td>DB 3</td>
<td>15</td>
<td></td>
<td>GPA 33</td>
</tr>
<tr>
<td>DB 4</td>
<td>16</td>
<td></td>
<td>Character Clock (from dot/timing logic).</td>
</tr>
<tr>
<td>DB 5</td>
<td>17</td>
<td></td>
<td>Character Codes: Output from the row buffers used for character selection in the character generator.</td>
</tr>
<tr>
<td>DB 6</td>
<td>18</td>
<td></td>
<td>Chip Select: The read and write are enabled by CS.</td>
</tr>
<tr>
<td>DB 7</td>
<td>19</td>
<td></td>
<td>Port Address: A high input on A0 selects the &quot;C&quot; port or command registers and a low input selects the &quot;P&quot; port or parameter registers.</td>
</tr>
</tbody>
</table>

### Table 1. Pin Descriptions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>40</td>
<td></td>
<td>+5V Power Supply.</td>
</tr>
<tr>
<td>LA 0</td>
<td>39</td>
<td>O</td>
<td>Line Attribute Codes: These attribute codes have to be decoded externally by the dot/timing logic to generate the horizontal and vertical line combinations for the graphic displays specified by the character attribute codes.</td>
</tr>
<tr>
<td>LA 1</td>
<td>38</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>LTEN</td>
<td>37</td>
<td>O</td>
<td>Light Enable: Output signal used to enable the video signal to the CRT. This output is active at the programmed underline cursor position, and at positions specified by attribute codes.</td>
</tr>
<tr>
<td>RVV</td>
<td>36</td>
<td>O</td>
<td>Reverse Video: Output signal used to indicate the CRT circuitry to reverse the video signal. This output is active at the cursor position if a reverse video block is programmed or at the positions specified by the field attribute codes.</td>
</tr>
<tr>
<td>VSP</td>
<td>35</td>
<td>O</td>
<td>Video Suppression: Output signal used to blank the video signal to the CRT. This output is active:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>—during the horizontal and vertical retrace intervals.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>—at the top and bottom lines of rows if underline is programmed to be number 8 or greater.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>—when an end of row or end of screen code is detected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>—when a DMA underrun occurs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>—at regular intervals (1/16 frame frequency for cursor, 1/32 frame frequency for character and field attributes)—to create blinking displays as specified by cursor, character attribute, or field attribute programming.</td>
</tr>
<tr>
<td>GPA 1</td>
<td>34</td>
<td>O</td>
<td>General Purpose Attribute Codes: Outputs which are enabled by the general purpose field attribute codes.</td>
</tr>
<tr>
<td>GPA 0</td>
<td>33</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>HLGT</td>
<td>32</td>
<td>O</td>
<td>Highlight: Output signal used to intensify the display at particular positions on the screen as specified by the character attribute codes or field attribute codes.</td>
</tr>
<tr>
<td>IRQ</td>
<td>31</td>
<td>O</td>
<td>Interrupt Request.</td>
</tr>
<tr>
<td>CCLK</td>
<td>30</td>
<td>I</td>
<td>Character Clock (from dot/timing logic).</td>
</tr>
<tr>
<td>CC 0</td>
<td>29</td>
<td>O</td>
<td>Character Codes: Output from the row buffers used for character selection in the character generator.</td>
</tr>
<tr>
<td>CC 1</td>
<td>28</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>CC 2</td>
<td>27</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>CC 3</td>
<td>26</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>CC 4</td>
<td>25</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>CC 5</td>
<td>24</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>CC 6</td>
<td>23</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td>22</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>A 0</td>
<td>21</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7-2
FUNCTIONAL DESCRIPTION

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8275H to the system Data Bus. This functional block accepts inputs from the System Control Bus and generates control signals for overall device operation. It contains the Command, Parameter, and Status Registers that store the various control formats for the device functional definition.

<table>
<thead>
<tr>
<th>A0</th>
<th>OPERATION</th>
<th>REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Read</td>
<td>PREG</td>
</tr>
<tr>
<td>0</td>
<td>Write</td>
<td>PREG</td>
</tr>
<tr>
<td>1</td>
<td>Read</td>
<td>SREG</td>
</tr>
<tr>
<td>1</td>
<td>Write</td>
<td>CREG</td>
</tr>
</tbody>
</table>

RD (Read)
A "low" on this input informs the 8275H that the CPU is reading data or status information from the 8275H.

WR (Write)
A "low" on this input informs the 8275H that the CPU is writing data or control words to the 8275H.

CS (Chip Select)
A "low" on this input selects the 8275H. No reading or writing will occur unless the device is selected. When CS is high, the Data Bus is in the float state and RD and WR will have no effect on the chip.

DRQ (DMA Request)
A "high" on this output informs the DMA Controller that the 8275H desires a DMA transfer.

DACK (DMA Acknowledge)
A "low" on this input informs the 8275H that a DMA cycle is in progress.

IRQ (Interrupt Request)
A "high" on this output informs the CPU that the 8275H desires interrupt service.

<table>
<thead>
<tr>
<th>A0</th>
<th>RD</th>
<th>WR</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>
FUNCTIONAL DESCRIPTION

Character Counter
The Character Counter is a programmable counter that is used to determine the number of characters to be displayed per row and the length of the horizontal retrace interval. It is driven by the CCLK (Character Clock) input, which should be a derivative of the external dot clock.

Line Counter
The Line Counter is a programmable counter that is used to determine the number of horizontal lines (Sweeps) per character row. Its outputs are used to address the external character generator ROM.

Row Counter
The Row Counter is a programmable counter that is used to determine the number of character rows to be displayed per frame and length of the vertical retrace interval.

Light Pen Registers
The Light Pen Registers are two registers that store the contents of the character counter and the row counter whenever there is a rising edge on the LPEN (Light Pen) input.

Note: Software correction is required.

Raster Timing and Video Controls
The Raster Timing circuitry controls the timing of the HRTC (Horizontal Retrace) and VRTC (Vertical Retrace) outputs. The Video Control circuitry controls the generation of LA0-1 (Line Attribute), HGLT (Highlight), RVV (Reverse Video), LTEN (Light Enable), VSP (Video Suppress), and GPA0-1 (General Purpose Attribute) outputs.

Row Buffers
The Row Buffers are two 80 character buffers. They are filled from the microcomputer system memory with the character codes to be displayed. While one row buffer is displaying a row of characters, the other is being filled with the next row of characters.

FIFOs
There are two 16 character FIFOs in the 8275H. They are used to provide extra row buffer length in the Transparent Attribute Mode (see Detailed Operation section).

Buffer Input/Output Controllers
The Buffer Input/Output Controllers decode the characters being placed in the row buffers. If the character is a character attribute, field attribute or special code, these controllers control the appropriate action. (Examples: An “End of Screen—Stop DMA” special code will cause the Buffer Input Controller to stop further DMA requests. A “Highlight” field attribute will cause the Buffer Output Controller to activate the HGLT output.)
SYSTEM OPERATION

The 8275H is programmable to a large number of different display formats. It provides raster timing, display row buffering, visual attribute decoding, cursor timing, and light pen detection.

It is designed to interface with the 8257 DMA Controller and standard character generator ROMs for dot matrix decoding. Dot level timing must be provided by external circuitry.

Figure 4. 8275H Systems Block Diagram Showing Systems Operation
General Systems Operational Description

The 8275H provides a “window” into the microcomputer system memory.

Display characters are retrieved from memory and displayed on a row by row basis. The 8275H has two row buffers. While one row buffer is being used for display, the other is being filled with the next row of characters to be displayed. The number of display characters per row and the number of character rows per frame are software programmable, providing easy interface to most CRT displays. (See Programming Section.)

The 8275H requests DMA to fill the row buffer that is not being used for display. DMA burst length and spacing is programmable. (See Programming Section.)

The 8275H displays character rows one line at a time.

The number of lines per character row, the underline position, and blanking of top and bottom lines are programmable. (See Programming Section.)

The 8275H provides special Control Codes which can be used to minimize DMA or software overhead. It also provides Visual Attribute Codes to cause special action or symbols on the screen without the use of the character generator (See Visual Attributes Section).

The 8275H also controls raster timing. This is done by generating Horizontal Retrace (HRTC) and Vertical Retrace (VRTC) signals. The timing of these signals is programmable.

The 8275H can generate a cursor. Cursor location and format are programmable. (See Programming Section.)

The 8275H has a light pen input and registers. The light pen input is used to load the registers. Light pen registers can be read on command. (See Programming Section.)

Figure 5. Display of a Character Row
Display Row Buffering

Before the start of a frame, the 8275H requests DMA and one row buffer is filled with characters.

After all the lines of the character row are scanned, the roles of the two row buffers are reversed and the same procedure is followed for the next row.

Figure 6. First Row Buffer Filled

When the first horizontal sweep is started, character codes are output to the character generator from the row buffer just filled. Simultaneously, DMA begins filling the other row buffer with the next row of characters.

Figure 7. Second Buffer Filled, First Row Displayed

This is repeated until all of the character rows are displayed.

Figure 8. First Buffer Filled with Third Row, Second Row Displayed
Display Format

Screen Format

The 8275H can be programmed to generate from 1 to 80 characters per row, and from 1 to 64 rows per frame.

The 8275H can also be programmed to blank alternate rows. In this mode the first row is displayed, the second blanked, the third displayed, etc. DMA is not requested for the blanked rows.

Row Format

The 8275H is designed to hold the line count stable while outputting the appropriate character codes during each horizontal sweep. The line count is incremented during horizontal retrace and the whole row of character codes are output again during the next sweep. This is continued until the whole character row is displayed.

The number of lines (horizontal sweeps) per character row is programmable from 1 to 16.

The output of the line counter can be programmed to be in one of two modes.

In mode 0, the output of the line counter is the same as the line number.

In mode 1, the line counter is offset by one from the line number.

Note: In mode 1, while the first line (line number 0) is being displayed, the last count is output by the line counter (see examples).

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Line Counter Mode 0</th>
<th>Line Counter Mode 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>1111</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0001</td>
</tr>
<tr>
<td>3</td>
<td>0110</td>
<td>0010</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0110</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0111</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>1000</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1001</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>10</td>
<td>1100</td>
<td>1011</td>
</tr>
<tr>
<td>11</td>
<td>1101</td>
<td>1100</td>
</tr>
<tr>
<td>12</td>
<td>1110</td>
<td>1101</td>
</tr>
<tr>
<td>13</td>
<td>1111</td>
<td>1110</td>
</tr>
</tbody>
</table>

Figure 11. Example of a 16-Line Format

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Line Counter Mode 0</th>
<th>Line Counter Mode 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>1001</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0001</td>
</tr>
<tr>
<td>3</td>
<td>0110</td>
<td>0010</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0111</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0100</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>0110</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>0111</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1000</td>
</tr>
</tbody>
</table>

Figure 12. Example of a 10-Line Format

Mode 0 is useful for character generators that leave address zero blank and start at address 1. Mode 1 is useful for character generators which start at address zero.
Underline placement is also programmable (from line number 0 to 15). This is independent of the line counter mode.

If the line number of the underline is greater than 7 (line number MSB = 1), then the top and bottom lines will be blanked.

If the line number of the underline is less than or equal to 7 (line number MSB = 0), then the top and bottom lines will not be blanked.

Dot width and character width are dependent upon the external timing and control circuitry.

Dot level timing circuitry should be designed to accept the parallel output of the character generator and shift it out serially at the rate required by the CRT display.

Dot width is a function of dot clock frequency.

Character width is a function of the character generator width.

Horizontal character spacing is a function of the shift register length.

Note: Video control and timing signals must be synchronized with the video signal due to the character generator access delay.
**Raster Timing**

The character counter is driven by the character clock input (CCLK). It counts out the characters being displayed (programmable from 1 to 80). It then causes the line counter to increment, and it starts counting out the horizontal retrace interval (programmable from 2 to 32). This is constantly repeated.

![Figure 16. Line Timing](image)

The line counter is driven by the character counter. It is used to generate the line address outputs (LC_0-3_) for the character generator. After it counts all of the lines in a character row (programmable from 1 to 16), it increments the row counter, and starts over again. (See Character Format Section for detailed description of Line Counter functions.)

The row counter is an internal counter driven by the line counter. It controls the functions of the row buffers and counts the number of character rows displayed.

![Figure 17. Row Timing](image)

After the row counter counts all of the rows in a frame (programmable from 1 to 64), it starts counting out the vertical retrace interval (programmable from 1 to 4).

![Figure 18. Frame Timing](image)

The Video Suppression Output (VSP) is active during horizontal and vertical retrace intervals.

Dot level timing circuitry must synchronize these outputs with the video signal to the CRT Display.
DMA Timing

The 8275H can be programmed to request burst DMA transfers of 1 to 8 characters. The interval between bursts is also programmable (from 0 to 55 character clock periods ±1). This allows the user to tailor his DMA overhead to fit his system needs.

The first DMA request of the frame occurs one row time before the end of vertical retrace. DMA requests continue as programmed, until the row buffer is filled. If the row buffer is filled in the middle of a burst, the 8275H terminates the burst and resets the burst counter. No more DMA requests will occur until the beginning of the next row. At that time, DMA requests are activated as programmed until the other buffer is filled.

The first DMA request for a row will start at the first character clock of the preceding row. If the burst mode is used, the first DMA request may occur a number of character clocks later. This number is equal to the programmed burst space.

If, for any reason, there is a DMA underrun, a flag in the status word will be set.

Interrupt Timing

The 8275H can be programmed to generate an interrupt request at the end of each frame. This can be used to reinitialize the DMA controller. If the 8275H interrupt enable flag is set, an interrupt request will occur at the beginning of the last display row.

The DMA controller is typically initialized for the next frame at the end of the current frame.
VISUAL ATTRIBUTES AND SPECIAL CODES

The characters processed by the 8275H are 8-bit quantities. The character code outputs provide the character generator with 7 bits of address. The Most Significant Bit is the extra bit and it is used to determine if it is a normal display character (MSB = 0), or if it is a Visual Attribute or Special Code (MSB = 1).

There are two types of Visual Attribute Codes. They are Character Attributes and Field Attributes.

Character Attribute Codes

Character attribute codes are codes that can be used to generate graphics symbols without the use of a character generator. This is accomplished by selectively activating the Line Attribute outputs (LA0–1), the Video Suppression output (VSP), and the Light Enable output. The dot level timing circuitry can use these signals to generate the proper symbols.

Character attributes can be programmed to blink or be highlighted individually. Blinking is accomplished with the Video Suppression output (VSP). Blink frequency is equal to the screen refresh frequency divided by 32. Highlighting is accomplished by activating the Highlight output (HGLT).

Character Attributes

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
</tbody>
</table>

HIGHLIGHT BLINK CHARACTER ATTRIBUTE CODE

Figure 22. Typical Character Attribute Logic
# Table 2. Character Attributes

Character attributes were designed to produce the following graphics:

<table>
<thead>
<tr>
<th>CHARACTER ATTRIBUTE CODE “CCCC”</th>
<th>OUTPUTS</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LA1</td>
<td>LA0</td>
<td>VSP</td>
</tr>
<tr>
<td>0000 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0001 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0010 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0011 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0100 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0101 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0110 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0111 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1000 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1001 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1010 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1011 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1100 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1111 Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

*Character Attribute Code 1011 is not recommended for normal operation. Since none of the attribute outputs are active, the character Generator will not be disabled, and an indeterminate character will be generated.

Character Attribute Codes 1101, 1110, and 1111 are illegal.

Blinking is active when B = 1.

Highlight is active when H = 1.
Special Codes
Four special codes are available to help reduce memory, software, or DMA overhead.

Special Control Character

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>00 S S</td>
<td>SPECIAL CONTROL Code</td>
</tr>
</tbody>
</table>

The End of Row Code (00) activates VSP and holds it to the end of the line.
The End of Row-Stop DMA Code (01) causes the DMA Control Logic to stop DMA for the rest of the row when it is written into the Row Buffer. It affects the display in the same way as the End of Row Code (00).
The End of Screen Code (10) activates VSP and holds it to the end of the frame.
The End of Screen-Stop DMA Code (11) causes the DMA Control Logic to stop DMA for the rest of the frame when it is written into the Row Buffer. It affects the display in the same way as the End of Screen Code (10).

If the Stop DMA feature is not used, all characters after an End of Row character are ignored, except for the End of Screen character, which operates normally. All characters after an End of Screen character are ignored.

Note: If a Stop DMA character is not the last character in a burst or row, DMA is not stopped until after the next character is read. In this situation, a dummy character must be placed in memory after the Stop DMA character.

Field Attributes
The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the code up to, and including, the character which precedes the next field attribute code, or up to the end of the frame. The field attributes are reset during the vertical retrace interval.

There are six field attributes:

1. **Blink** — Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.

2. **Highlight** — Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).

3. **Reverse Video** — Characters following the code are caused to appear with reverse video by activating the Reverse Video output (RVV).

4. **Underline** — Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).

5,6. **General Purpose** — There are two additional 8275 outputs which act as general purpose, independently programmable field attributes. GPA0-1 are active high outputs.

Field Attribute Code

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>O</td>
<td>HIGHLIGHT</td>
</tr>
<tr>
<td>0</td>
<td>R</td>
<td>BLINK</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>GENERAL PURPOSE</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>REVERSE VIDEO</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>UNDERLINE</td>
</tr>
</tbody>
</table>

H = 1 FOR HIGHLIGHTING  
B = 1 FOR BLINKING      
R = 1 FOR REVERSE VIDEO  
U = 1 FOR UNDERLINE  
GG = GPA1, GPA0

*More than one attribute can be enabled at the same time. If the blinking and reverse video attributes are enabled simultaneously, only the reversed characters will blink.
The 8275H can be programmed to provide visible or invisible field attribute characters.

If the 8275H is programmed in the visible field attribute mode, all field attributes will occupy a position on the screen. They will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character.

If the 8275H is programmed in the invisible field attribute mode, the FIFOs are activated.

Each row buffer has a corresponding FIFO. These FIFOs are 16 characters by 7 bits in size.

When a field attribute is placed in the row buffer during DMA, the buffer input controller recognizes it and places the next character in the proper FIFO.

When a field attribute is placed in the Buffer Output Controller during display, it causes the controller to immediately put a character from the FIFO on the Character Code outputs (CC0-6). The chosen Visual Attributes are also activated.

Since the FIFO is 16 characters long, no more than 16 field attribute characters may be used per line in this mode. If more are used, a bit in the status word is set and the first characters in the FIFO are written over and lost.

Note: Since the FIFO is 7 bits wide, the MSB of any characters put in it are stripped off. Therefore, a Visual Attribute or Special Code must not immediately follow a field attribute code. If this situation does occur, the Visual Attribute or Special Code will be treated as a normal display character.

Field and Character Attribute Interaction

Character Attribute Symbols are affected by the Reverse Video (RVV) and General Purpose (GPA0-1) field attributes. They are not affected by Underline, Blink or Highlight field attributes; however, these characteristics can be programmed individually for Character Attribute Symbols.
Cursor Timing

The cursor location is determined by a cursor row register and a character position register which are loaded by command to the controller. The cursor can be programmed to appear on the display as:

1. a blinking underline
2. a blinking reverse video block
3. a non-blanking underline
4. a non-blanking reverse video block

The cursor blinking frequency is equal to the screen refresh frequency divided by 16.

If a non-blanking reverse video cursor appears in a non-blanking reverse video field, the cursor will appear as a normal video block.

If a non-blanking underline cursor appears in a non-blanking underline field, the cursor will not be visible.

Light Pen Detection

A light pen consists of a micro switch and a tiny light sensor. When the light pen is pressed against the CRT screen, the micro switch enables the light sensor. When the raster sweep reaches the light sensor, it triggers the light pen output.

If the output of the light pen is presented to the 8275H LPEN input, the row and character position coordinates are stored in a pair of registers. These registers can be read on command. A bit in the status word is set, indicating that the light pen signal was detected. The LPEN input must be a 0 to 1 transition for proper operation.

Note: Due to internal and external delays, the character position coordinate will be off by at least three character positions. This has to be corrected in software.

Device Programming

The 8275H has two programming registers, the Command Register (CREG) and the Parameter Register (PREG). It also has a Status Register (SREG). The Command Register can only be written into and the Status Registers can only be read from. They are addressed as follows:

<table>
<thead>
<tr>
<th>Addr</th>
<th>OPERATION</th>
<th>REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Read</td>
<td>PREG</td>
</tr>
<tr>
<td>0</td>
<td>Write</td>
<td>PREG</td>
</tr>
<tr>
<td>1</td>
<td>Read</td>
<td>SREG</td>
</tr>
<tr>
<td>1</td>
<td>Write</td>
<td>CREG</td>
</tr>
</tbody>
</table>

The 8275H expects to receive a command and a sequence of 0 to 4 parameters, depending on the command. If the proper number of parameter bytes are not received before another command is given, a status flag is set, indicating an improper command.

INSTRUCTION SET

The 8275H instruction set consists of 8 commands.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>NO. OF PARAMETER BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>4</td>
</tr>
<tr>
<td>Start Display</td>
<td>0</td>
</tr>
<tr>
<td>Stop Display</td>
<td>0</td>
</tr>
<tr>
<td>Read Light Pen</td>
<td>2</td>
</tr>
<tr>
<td>Load Cursor</td>
<td>2</td>
</tr>
<tr>
<td>Enable Interrupt</td>
<td>0</td>
</tr>
<tr>
<td>Disable Interrupt</td>
<td>0</td>
</tr>
<tr>
<td>Preset Counters</td>
<td>0</td>
</tr>
</tbody>
</table>

In addition, the status of the 8275H (SREG) can be read by the CPU at any time.
1. Reset Command:

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>A0</th>
<th>DESCRIPTION</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write 1</td>
<td>Reset Command</td>
<td>0 0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>Write 0</td>
<td>Screen Comp Byte 1</td>
<td>S H H H H H H H</td>
<td>H H H H H H H H H</td>
</tr>
<tr>
<td></td>
<td>Write 0</td>
<td>Screen Comp Byte 2</td>
<td>V V R R R R R R</td>
<td>R R R R R R R R R</td>
</tr>
<tr>
<td></td>
<td>Write 0</td>
<td>Screen Comp Byte 3</td>
<td>U U U J L L L L L</td>
<td>L L L L L L L L L</td>
</tr>
<tr>
<td></td>
<td>Write 0</td>
<td>Screen Comp Byte 4</td>
<td>M F C C Z Z Z Z Z</td>
<td>Z Z Z Z Z Z Z Z Z</td>
</tr>
</tbody>
</table>

Action — After the reset command is written, DMA requests stop, 8275 interrupts are disabled, and the VSP output is used to blank the screen. HRTC and VRTC continue to run. HRTC and VRTC timing are random on power-up.

As parameters are written, the screen composition is defined.

Parameter — S  Spaced Rows

<table>
<thead>
<tr>
<th>S</th>
<th>FUNCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal Rows</td>
</tr>
<tr>
<td>1</td>
<td>Spaced Rows</td>
</tr>
</tbody>
</table>

Parameter — HHHHHHH  Horizontal Characters/Row

<table>
<thead>
<tr>
<th>H H H H H H</th>
<th>NO. OF CHARACTERS PER ROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 0 0 1</td>
<td>2</td>
</tr>
<tr>
<td>0 0 0 0 0 0</td>
<td>3</td>
</tr>
<tr>
<td>1 0 0 1 1 1</td>
<td>80</td>
</tr>
<tr>
<td>1 0 1 0 0 0</td>
<td>Undefined</td>
</tr>
<tr>
<td>1 1 1 1 1 1</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

Parameter — VV  Vertical Retrace Row Count

<table>
<thead>
<tr>
<th>V V</th>
<th>NO. OF ROW COUNTS PER VRTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>2</td>
</tr>
<tr>
<td>1 0</td>
<td>3</td>
</tr>
<tr>
<td>1 1</td>
<td>4</td>
</tr>
</tbody>
</table>

Parameter — RRRRRR  Vertical Rows/Frame

<table>
<thead>
<tr>
<th>R R R R R R</th>
<th>NO. OF ROWS/FRAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 0 0 1</td>
<td>2</td>
</tr>
<tr>
<td>0 0 0 0 1 0</td>
<td>3</td>
</tr>
<tr>
<td>1 1 1 1 1 1</td>
<td>64</td>
</tr>
</tbody>
</table>

Parameter — UUUU  Underline Placement

<table>
<thead>
<tr>
<th>U U U</th>
<th>LINE NUMBER OF UNDERLINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>2</td>
</tr>
<tr>
<td>0 1 0</td>
<td>3</td>
</tr>
<tr>
<td>1 1 1</td>
<td>16</td>
</tr>
</tbody>
</table>

Parameter — LLLL  Number of Lines per Character Row

<table>
<thead>
<tr>
<th>L L L L</th>
<th>NO. OF LINES/ROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>2</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>3</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>16</td>
</tr>
</tbody>
</table>

Parameter — M  Line Counter Mode

<table>
<thead>
<tr>
<th>M</th>
<th>LINE COUNTER MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Mode 0 (Non-Offset)</td>
</tr>
<tr>
<td>1</td>
<td>Mode 1 (Offset by 1 Count)</td>
</tr>
</tbody>
</table>

Parameter — F  Field Attribute Mode

<table>
<thead>
<tr>
<th>F</th>
<th>FIELD ATTRIBUTE MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Transparent</td>
</tr>
<tr>
<td>1</td>
<td>Non-Transparent</td>
</tr>
</tbody>
</table>

Parameter — CC  Cursor Format

<table>
<thead>
<tr>
<th>C C</th>
<th>CURSOR FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Blinking reverse video block</td>
</tr>
<tr>
<td>0 1</td>
<td>Blinking underline</td>
</tr>
<tr>
<td>1 0</td>
<td>Nonblinking reverse video block</td>
</tr>
<tr>
<td>1 1</td>
<td>Nonblinking underline</td>
</tr>
</tbody>
</table>

Parameter — ZZZZ  Horizontal Retrace Count

<table>
<thead>
<tr>
<th>Z Z Z Z</th>
<th>NO. OF CHARACTER COUNTS PER HRTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>2</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>4</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>6</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>32</td>
</tr>
</tbody>
</table>

Note: uuuu MSB determines blanking of top and bottom lines (1 = blanked, 0 = not blanked)
2. Start Display Command:

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>A₀</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write 1</td>
<td>Start Display</td>
<td>0 0 1 5 S S B B</td>
</tr>
</tbody>
</table>

No parameters

5. Load Cursor Position:

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>A₀</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write 1</td>
<td>Load Cursor</td>
<td>1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Parameters</td>
<td>Write 0</td>
<td>Char Number</td>
<td>(Char Position in Row)</td>
</tr>
<tr>
<td></td>
<td>Write 0</td>
<td>Row Number</td>
<td>(Row Number)</td>
</tr>
</tbody>
</table>

**Action** — The 8275H is conditioned to place the next two parameter bytes into the cursor position registers. Status flags not affected.

6. Enable Interrupt Command:

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>A₀</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write 1</td>
<td>Enable Interrupt</td>
<td>1 0 1 0 0 0 0 0</td>
</tr>
</tbody>
</table>

No parameters

**Action** — The interrupt enable status flag is set and interrupts are enabled.

7. Disable Interrupt Command:

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>A₀</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write 1</td>
<td>Disable Interrupt</td>
<td>1 1 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

No parameters

**Action** — Interrupts are disabled and the interrupt enable status flag is reset.

8. Preset Counters Command:

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>A₀</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write 1</td>
<td>Preset Counters</td>
<td>1 1 1 0 0 0 0 0</td>
</tr>
</tbody>
</table>

No parameters

**Action** — The internal timing counters are preset, corresponding to a screen display position at the top left corner. Two character clocks are required for this operation. The counters will remain in this state until any other command is given.

This command is useful for system debug and synchronization of clustered CRT displays on a single CPU. After this command, two additional clock cycles are required before the first character of the first row is put out.

**Note:** Software correction of light pen position is required.
Status Flags

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>A0</th>
<th>DESCRIPTION</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Read</td>
<td>Status Word</td>
<td>I E</td>
<td>I R</td>
</tr>
</tbody>
</table>

IE — (Interrupt Enable) Set or reset by command. It enables vertical retrace interrupt. It is automatically set by a “Start Display” command and reset with the “Reset” command.

IR — (Interrupt Request) This flag is set at the beginning of display of the last row of the frame if the interrupt enable flag is set. It is reset after a status read operation.

LP — This flag is set when the light pen input (LPEN) is activated and the light pen registers have been loaded. This flag is automatically reset after a status read.

IC — (Improper Command) This flag is set when a command parameter string is too long or too short. The flag is automatically reset after a status read.

VE — (Video Enable) This flag indicates that video operation of the CRT is enabled. This flag is set on a “Start Display” command, and reset on a “Stop Display” or “Reset” command.

DU — (DMA Underrun) This flag is set whenever a data underrun occurs during DMA transfers. Upon detection of DU, the DMA operation is stopped and the screen is blanked until after the vertical retrace interval. This flag is reset after a status read.

FO — (FIFO Overrun) This flag is set whenever the FIFO is overrun. It is reset on a status read.
ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias .................................. 0°C to 70°C
Storage Temperature ................................................ -65°C to +150°C
Voltage On Any Pin With Respect to Ground ............... -0.5V to +7V
Power Dissipation .................................................. 1 Watt

*NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. CHARACTERISTICS \((T_A = 0°C \text{ to } 70°C, \ V_{CC} = 5V \pm 5\%\)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V_{IL})</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
</tr>
<tr>
<td>(V_{IH})</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>(V_{CC} + 0.5V)</td>
<td>V</td>
</tr>
<tr>
<td>(V_{OL})</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td>V</td>
<td>(I_{OL} = 2.2\ mA)</td>
</tr>
<tr>
<td>(V_{OH})</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td>V</td>
<td>(I_{OH} = -400\ \mu A)</td>
</tr>
<tr>
<td>(I_{IL})</td>
<td>Input Load Current</td>
<td>(\pm 10)</td>
<td>(\mu A)</td>
<td>(V_{IN} = V_{CC} \text{ to } 0V)</td>
</tr>
<tr>
<td>(I_{OFL})</td>
<td>Output Float Leakage</td>
<td>(\pm 10)</td>
<td>(\mu A)</td>
<td>(V_{OUT} = V_{CC} \text{ to } 0.45V)</td>
</tr>
<tr>
<td>(I_{CC})</td>
<td>(V_{CC}) Supply Current</td>
<td>160</td>
<td>mA</td>
<td></td>
</tr>
</tbody>
</table>

CAPACITANCE \((T_A = 25°C, \ V_{CC} = \text{GND} = 0V)\)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C_{IN})</td>
<td>Input Capacitance</td>
<td>10</td>
<td>pF</td>
<td></td>
</tr>
<tr>
<td>(C_{I/O})</td>
<td>I/O Capacitance</td>
<td>20</td>
<td>pF</td>
<td></td>
</tr>
</tbody>
</table>

A.C. CHARACTERISTICS \((T_A = 0°C \text{ to } 70°C, \ V_{CC} = 5.0V \pm 5\%, \ \text{GND} = 0V)\)

Bus Parameters

READ CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_{AR})</td>
<td>Address Stable Before READ</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{RA})</td>
<td>Address Hold Time for READ</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{RR})</td>
<td>READ Pulse Width</td>
<td>250</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{RD})</td>
<td>Data Delay from READ</td>
<td>200</td>
<td>ns</td>
<td>(C_L = 150\ pF)</td>
</tr>
<tr>
<td>(t_{DF})</td>
<td>READ to Data Floating</td>
<td>100</td>
<td>ns</td>
<td>(C_L = 150\ pF)</td>
</tr>
</tbody>
</table>

WRITE CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_{AW})</td>
<td>Address Stable Before WRITE</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{WA})</td>
<td>Address Hold Time for WRITE</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{WW})</td>
<td>WRITE Pulse Width</td>
<td>250</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{DW})</td>
<td>Data Setup Time for WRITE</td>
<td>150</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(t_{WD})</td>
<td>Data Hold Time for WRITE</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

7-20

210464-002
A.C. CHARACTERISTICS (Continued)

CLOCK TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCLK</td>
<td>Clock Period</td>
<td>480</td>
<td>320</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tKH</td>
<td>Clock High</td>
<td>240</td>
<td>120</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tKL</td>
<td>Clock Low</td>
<td>160</td>
<td>120</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tKR</td>
<td>Clock Rise</td>
<td>5</td>
<td>30</td>
<td>5</td>
<td>30</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tKF</td>
<td>Clock Fall</td>
<td>5</td>
<td>30</td>
<td>5</td>
<td>30</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

OTHER TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCC</td>
<td>Character Code Output Delay</td>
<td>150</td>
<td>150</td>
<td></td>
<td></td>
<td>ns</td>
<td>C_L = 50 pF</td>
</tr>
<tr>
<td>tHR</td>
<td>Horizontal Retrace,Output Delay</td>
<td>200</td>
<td>150</td>
<td></td>
<td></td>
<td>ns</td>
<td>C_L = 50 pF</td>
</tr>
<tr>
<td>tLC</td>
<td>Line Count Output Delay</td>
<td>400</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tAT</td>
<td>Control/Attribute Output Delay</td>
<td>275</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
<td>C_L = 50 pF</td>
</tr>
<tr>
<td>tVR</td>
<td>Vertical Retrace Output Delay</td>
<td>275</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
<td>C_L = 50 pF</td>
</tr>
<tr>
<td>tRI</td>
<td>IRQ� from RD^1</td>
<td>250</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
<td>C_L = 50 pF</td>
</tr>
<tr>
<td>tWQ</td>
<td>DRO^ from WR^1</td>
<td>250</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
<td>C_L = 50 pF</td>
</tr>
<tr>
<td>tRQ</td>
<td>DRO^ from WR^1</td>
<td>200</td>
<td>200</td>
<td></td>
<td></td>
<td>ns</td>
<td>C_L = 50 pF</td>
</tr>
<tr>
<td>tLR</td>
<td>DACK^ to WR^1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tRL</td>
<td>WR^ to DACK^</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tPR</td>
<td>LPEN Rise</td>
<td>50</td>
<td>50</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tPH</td>
<td>LPEN Hold</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tDI</td>
<td>DACK Inactive Period</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING LOAD CIRCUIT

C_L INCLUDES JIG CAPACITANCE

---

A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING LOAD CIRCUIT

---

7-21

210464-002
WAVEFORMS

TYPICAL DOT LEVEL TIMING

EXT DOT CLK

CCLK*

CCQ-6

FIRST CHARACTER CODE
SECOND CHARACTER CODE

ROM ACCESS

FIRST CHARACTER
SECOND CHARACTER

ATTRIBUTES & CONTROLS

ATTRIBUTES & CONTROLS FOR FIRST CHAR

SHIFT REGISTER SETUP

VIDEO (FROM SHIFT REGISTER)

FIRST CHARACTER
SECOND CHARACTER

ATTRIBUTES & CONTROLS

ATTRIBUTES & CONTROLS FOR 2ND CHAR

*CCLK IS A MULTIPLE OF THE DOT CLOCK AND AN INPUT TO THE 8275

LINE TIMING

CCLK

CCQ-6

FIRST DISPLAY CHARACTER
SECOND DISPLAY CHARACTER
LAST DISPLAY CHARACTER
FIRST RETRACE CHARACTER
LAST RETRACE CHARACTER

PROGRAMMABLE FROM 1 TO 80 CHARACTERS
PROGRAMMABLE FROM 1 TO 32 CCLKS

HLRT

PRESENT LINE COUNT

NEXT LINE COUNT

VIDEO CONTROLS AND ATTRIBUTES

*LAQ-1, VSP, LTEN, HGLT, RVV, GPAQ-1

*LC

7-22
WAVEFORMS (Continued)

ROW TIMING

FRAME TIMING

RUPT TIMING

INTERNAL ROW COUNTER

FIRST LINE COUNT
SECOND LINE COUNT
LAST LINE COUNT

PROGRAMMABLE FROM 1 TO 16 LINES

INTERNAL ROW COUNTER

FIRST DISPLAY ROW
SECOND DISPLAY ROW
LAST DISPLAY ROW

PROGRAMMABLE FROM 1 TO 64 ROWS

INTERNAL ROW COUNTER

FIRST RETRACE ROW
LAST RETRACE ROW

PROGRAMMABLE FROM 1 TO 4 ROWS

A0

CS

RD

I/O
WAVEFORMS (Continued)

DMA TIMING

WRITE TIMING

READ TIMING

CLOCK TIMING
The Intel 8276H Small System CRT Controller is a single chip device intended to interface CRT raster scan displays with Intel microcomputers in minimum device-count systems. Its primary function is to refresh the display by buffering character information from main memory and keeping track of the display position of the screen. The flexibility designed into the 8276H will allow simple interface to almost any raster scan CRT display. It can be used with the 8051 Single Chip Microcomputer for a minimum IC count design. It is manufactured on Intel's advanced HMOS-II process.
### Table 1. Pin Descriptions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC3</td>
<td>1</td>
<td>O</td>
<td>Line count. Output from the line counter which is used to address the character generator for the line positions on the screen</td>
</tr>
<tr>
<td>LC2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LC1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LC0</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRDY</td>
<td>5</td>
<td>O</td>
<td>Buffer ready. Output signal indicating that a Row Buffer is ready for loading of character data</td>
</tr>
<tr>
<td>BS</td>
<td>6</td>
<td>I</td>
<td>Buffer select. Input signal enabling WR for character data into the Row Buffers</td>
</tr>
<tr>
<td>HRTC</td>
<td>7</td>
<td>O</td>
<td>Horizontal retrace. Output signal which is active during the programmed horizontal retrace interval. During this period the VSP output is high and the LTEN output is low</td>
</tr>
<tr>
<td>VRTC</td>
<td>8</td>
<td>O</td>
<td>Vertical retrace. Output signal which is active during the programmed vertical retrace interval. During this period the VSP output is high and the LTEN output is low</td>
</tr>
<tr>
<td>RD</td>
<td>9</td>
<td>I</td>
<td>Read input. A control signal to read registers</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>I</td>
<td>Write input. A control signal to write commands into the control registers or write data into the row buffers</td>
</tr>
<tr>
<td>NC</td>
<td>11</td>
<td></td>
<td>No connection.</td>
</tr>
<tr>
<td>DB0</td>
<td>12</td>
<td>I/O</td>
<td>Bidirectional data bus. Three-state lines. The outputs are enabled during a read of the C or P ports</td>
</tr>
<tr>
<td>DB1</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB3</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB4</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB5</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB6</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB7</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ground</td>
<td>20</td>
<td></td>
<td>Ground.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>40</td>
<td></td>
<td>±5V power supply.</td>
</tr>
<tr>
<td>NC</td>
<td>39</td>
<td></td>
<td>No connection.</td>
</tr>
<tr>
<td>NC</td>
<td>38</td>
<td></td>
<td>No connection.</td>
</tr>
<tr>
<td>LTEN</td>
<td>37</td>
<td>O</td>
<td>Light enable. Output signal used to enable the video signal to the CRT. This output is active at the programmed underline cursor position, and at positions specified by attribute codes</td>
</tr>
<tr>
<td>RVV</td>
<td>36</td>
<td>O</td>
<td>Reverse video. Output signal used to activate the CRT circuitry to reverse the video signal. This output is active at the cursor position if a reverse video block cursor is programmed or at the positions specified by the field attribute codes</td>
</tr>
<tr>
<td>VSP</td>
<td>35</td>
<td>O</td>
<td>Video suppression. Output signal used to blank the video signal to the CRT. This output is active:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>— during the horizontal and vertical retrace intervals</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>— at the top and bottom lines of rows if underline is programmed to be number 8 or greater</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>— when an end of row or end of screen code is detected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>— when a Row Buffer underrun occurs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>— at regular intervals (1/16 frame frequency for cursor, 1/32 frame frequency for attributes)—to create blinking displays as specified by cursor or field attribute programming</td>
</tr>
<tr>
<td>GPA1</td>
<td>34</td>
<td>O</td>
<td>General purpose attribute codes. Outputs which are enabled by the general purpose field attribute codes</td>
</tr>
<tr>
<td>GPA0</td>
<td>33</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>HLGT</td>
<td>32</td>
<td>O</td>
<td>Highlight. Output signal used to intensify the display at particular positions on the screen as specified by the field attribute codes</td>
</tr>
<tr>
<td>INT</td>
<td>31</td>
<td>O</td>
<td>Interrupt output.</td>
</tr>
<tr>
<td>CCLK</td>
<td>30</td>
<td>I</td>
<td>Character clock (from dot/timing logic)</td>
</tr>
<tr>
<td>CC6</td>
<td>29</td>
<td>O</td>
<td>Character codes. Output from the row buffers used for character selection in the character generator</td>
</tr>
<tr>
<td>CC5</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC4</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC3</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC2</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC1</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC0</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td>22</td>
<td>I</td>
<td>Chip select. Enables RD of status or WR of command or parameters</td>
</tr>
<tr>
<td>C/ P</td>
<td>21</td>
<td>I</td>
<td>Port address. A high input on this pin selects the &quot;C&quot; port or command registers and a low input selects the &quot;P&quot; port or parameter registers</td>
</tr>
</tbody>
</table>

ORDER NUMBER. 210668-002
FUNCTIONAL DESCRIPTION

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8276H to the system Data Bus.

This functional block accepts inputs from the System Control Bus and generates control signals for overall device operation. It contains the Command, Parameter, and Status Registers that store the various control formats for the device functional definition.

<table>
<thead>
<tr>
<th>C/P</th>
<th>OPERATION</th>
<th>REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Read</td>
<td>RESERVED</td>
</tr>
<tr>
<td>0</td>
<td>Write</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>1</td>
<td>Read</td>
<td>STATUS</td>
</tr>
<tr>
<td>1</td>
<td>Write</td>
<td>COMMAND</td>
</tr>
</tbody>
</table>

RD (READ)
A "low" on this input informs the 8276H that the CPU is reading status information from the 8276H.

WR (WRITE)
A "low" on this input informs the 8276H that the CPU is writing data or control words to the 8276H.

CS (CHIP SELECT)
A "low" on this input selects the 8276H for RD or WR of Commands, Status, and Parameters.

BRDY (BUFFER READY)
A "high" on this output indicates that the 8276H is ready to receive character data.

BS (BUFFER SELECT)
A "low" on this input enables WR of character data to the 8276H row buffers.

INT (INTERRUPT)
A "high" on this output informs the CPU that the 8276H needs interrupt service.

Character Counter

The Character Counter is a programmable counter that is used to determine the number of characters to be displayed per row and the length of the horizontal retrace interval. It is driven by the CCLK (Character Clock) input, which should be derived from the external dot clock.

Line Counter

The Line Counter is a programmable counter that is used to determine the number of horizontal lines (Raster Scans) per character row. Its outputs are used to address the external character generator.

Row Counter

The Row Counter is a programmable counter that is used to determine the number of character rows to be displayed per frame and length of the vertical retrace interval.

Raster Timing and Video Controls

The Raster Timing circuitry controls the timing of the HRTC (Horizontal Retrace) and VRTC (Vertical Retrace) outputs. The Video Control circuitry controls the generation of HGLT (Highlight), RVV (Reverse Video), LTEN (Light Enable), VSP (Video Suppress), and GPA0-1 (General Purpose Attribute) outputs.

Row Buffers

The Row Buffers are two 80-character buffers. They are filled from the microcomputer system memory with the character codes to be displayed. While one row buffer is displaying a row of characters, the other is being filled with the next row of characters.

Buffer Input/Output Controllers

The Buffer Input/Output Controllers decode the characters being placed in the row buffers. If the character is a field attribute or special code, they control the appropriate action. (Example: A "Highlight" field attribute will cause the Buffer Output Controller to activate the HGLT output.)
SYSTEM OPERATION

The 8276H is programmable to a large number of different display formats. It provides raster timing, display row buffering, visual attribute decoding and cursor timing.

It is designed to interface with standard character generators for dot matrix decoding. Dot level timing must be provided by external circuitry.

General Systems Operational Description

Display characters are retrieved from memory and displayed on a row-by-row basis. The 8276H has two row buffers. While one row buffer is being used for display, the other is being filled with the next row of characters to be displayed. The number of display characters per row and the number of character rows per frame are software programmable, providing easy interface to most CRT displays. (See Programming Section.)

The 8276H uses BRDY to request character data to fill the row buffer that is not being used for display.

The 8276H displays character rows one scan line at a time. The number of scan lines per character row, the underline position, and blanking of top and bottom lines are programmable. (See Programming Section.)

The 8276H provides special Control Codes which can be used to minimize overhead. It also provides Visual Attribute Codes to cause special action on the screen without the use of the character generator. (See Visual Attributes Section.)

The 8276H also controls raster timing. This is done by generating Horizontal Retrace (HRTC) and Vertical Retrace (VRTC) signals. The timing of these signals is also programmable.

The 8276H can generate a cursor. Cursor location and format are programmable. (See Programming Section.)

Figure 3. CRT System Block Diagram
Display Row Buffering

Before the start of a frame, the 8276H uses BRDY and BS to fill one row buffer with characters.

When the first horizontal sweep is started, character codes are output to the character generator from the row buffer just filled. Simultaneously, the other row buffer is filled with the next row of characters.

After all the lines of the character row are scanned, the buffers are swapped and the same procedure is followed for the next row.

This process is repeated until all of the character rows are displayed.

Row Buffering allows the CPU access to the display memory at all times except during Buffer Loading (about 25%). This compares favorably to alternative approaches which restrict CPU access to the display memory to occur only during horizontal and vertical retrace intervals (80% of the bus time is used to refresh the display.)
Display Format

SCREEN FORMAT
The 8276H can be programmed to generate from 1 to 80 characters per row, and from 1 to 64 rows per frame.

ROW FORMAT
The 8276H is designed to hold the line count stable while outputting the appropriate character codes during each horizontal sweep. The line count is incremented during horizontal retrace and the whole row of character codes are output again during the next sweep. This is continued until the entire character row is displayed.

The number of lines (horizontal sweeps) per character row is programmable from 1 to 16.

The output of the line counter can be programmed to be in one of two modes.

In mode 0, the output of the line counter is the same as the line number.
In mode 1, the line counter is offset by one from the line number.

**Note:** In mode 1, while the first line (line number 0) is being displayed, the last count is output by the line counter (see examples).

![Figure 10. Example of a 16-Line Format](image)

If the line number of the underline is greater than 7 (line number MSB = 1), then the top and bottom lines will be blanked.

![Figure 11. Example of a 10-Line Format](image)

![Table: Example of a 16-Line Format](image)

If the line number of the underline is less than or equal to 7 (line number MSB = 0), then the top and bottom lines will *not* be blanked.

![Table: Example of a 10-Line Format](image)

![Figure 12. Underline in Line Number 10](image)

![Figure 13. Underline in Line Number 7](image)

Mode 0 is useful for character generators that leave address zero blank and start at address 1. Mode 1 is useful for character generators which start at address zero.

Underline placement is also programmable (from line number 0 to 15). This is independent of the line counter mode.

If the line number of the underline is greater than the maximum number of lines, the underline will not appear.

Blanking is accomplished by the VSP (Video Suppression) signal. Underline is accomplished by the LTEN (Light Enable) signal.

7-31 ORDER NUMBER 210668-002
**DOT FORMAT**

Dot width and character width are dependent upon the external timing and control circuitry.

Dot level timing circuitry should be designed to accept the parallel output of the character generator and shift it out serially at the rate required by the CRT display.

The line counter is driven by the character counter. It is used to generate the line address outputs (LC0–3) for the character generator. After it counts all of the lines in a character row (programmable from 1 to 16), it increments the row counter, and starts over again. (See Character Format Section for detailed description of Line Counter functions.)

The row counter is an internal counter driven by the line counter. It controls the functions of the row buffers and counts the number of character rows displayed.

---

**Raster Timing**

The character counter is driven by the character clock input (CCLK). It counts out the characters being displayed (programmable from 1 to 80). It then causes the line counter to increment, and it starts counting out the horizontal retrace interval (programmable from 2 to 32). This process is constantly repeated.

After the row counter counts all of the rows in a frame (programmable from 1 to 64), it starts counting out the vertical retrace interval (programmable from 1 to 4).

---

**Video Suppression Output (VSP)** is active during horizontal and vertical retrace intervals.

Dot level timing circuitry must synchronize these outputs with the video signal to the CRT Display.
Interrupt Timing

The 8276H can be programmed to generate an interrupt request at the end of each frame. If the 8276H interrupt enable flag is set, an interrupt request will occur at the beginning of the last display row.

![Diagram of Interrupt Timing](image)

**Figure 18. Beginning of Interrupt**

INT will go inactive after the status register is read.

![Diagram of End of Interrupt](image)

**Figure 19. End of Interrupt**

A reset command will also cause INT to go inactive, but this is not recommended during normal service.

**Note:** Upon power-up, the 8276H Interrupt Enable Flag may be set. As a result, the user’s cold start routine should write a reset command to the 8276H before system interrupts are enabled.

Special Codes

Four special codes are available to help reduce bus usage.

### SPECIAL CONTROL CHARACTER

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>00 SS</td>
<td>SPECIAL CONTROL CODE</td>
</tr>
<tr>
<td>0 0</td>
<td>End of Row</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>End of Row-Stop Buffer Loading</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>End of Screen</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>End of Screen-Stop Buffer Loading</td>
<td></td>
</tr>
</tbody>
</table>

The End of Row Code (00) activates VSP and holds it to the end of the line.

The End of Row-Stop Buffer Loading (BRDY) Code (01) causes the Buffer Loading Control Logic to stop buffer loading for the rest of the row upon being written into the Row Buffer. It affects the display in the same way as the End of Row Code (00).

The End of Screen Code (10) activates VSP and holds it to the end of the frame.

The End of Screen-Stop Buffer Loading (BRDY) Code (11) causes the Row Buffer Control Logic to stop buffer loading for the rest of the frame upon being written. It affects the display in the same way as the End of Screen Code (10).

If the Stop Buffer Loading feature is not used, all characters after an End of Row character are ignored, except for the End of Screen character, which operates normally. All characters after an End of Screen character are ignored.

**Note:** If a Stop Buffer Loading is not the last character in a row, Buffer Loading is not stopped until after the next character is read. In this situation, a dummy character must be placed in memory after the Stop Buffer Loading character.

VISUAL ATTRIBUTES AND SPECIAL CODES

The characters processed by the 8276H are 8-bit quantities. The character code outputs provide the character generator with 7 bits of address. The Most Significant Bit is the extra bit and it is used to determine if it is a normal display character (MSB = 0), or if it is a Field Attribute or Special Code (MSB = 1).

Field Attributes

The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the code up to, and including, the character which precedes the next field attribute code, or up to the end of the frame. The field attributes are reset during the vertical retrace interval.
The 8276H can be programmed to provide visible field attribute characters; all field attribute codes will occupy a position on the screen. These codes will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character.

There are six field attributes:

1. **Blink** — Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.

2. **Highlight** — Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).

3. **Reverse Video** — Characters following the code are caused to appear with reverse video by activating the Reverse Video output (RVV).

4. **Underline** — Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).

5. **General Purpose** — There are two additional 8276H outputs which act as general purpose, independently programmable field attributes. GPA0-1 are active high outputs.

### Field Attribute Codes

<table>
<thead>
<tr>
<th>Field Attribute</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blink</td>
<td>1</td>
</tr>
<tr>
<td>Highlight</td>
<td>2</td>
</tr>
<tr>
<td>Reverse Video</td>
<td>3</td>
</tr>
<tr>
<td>Underline</td>
<td>4</td>
</tr>
<tr>
<td>General Purpose</td>
<td>GPA0 GPA1</td>
</tr>
</tbody>
</table>

**Figure 20. Example of a Visible Field Attribute (Underline Attribute)**

**Field Attribute Code**

- **MSB**
  - 1: Blink
  - 0: Highlight
  - General Purpose
  - Reverse Video
  - Underline

- **LSB**
  - H: Highlight
  - B: Blink
  - G: General Purpose
  - R: Reverse Video
  - U: Underline

**H = 1 for Highlighting**

**B = 1 for Blinking**

**R = 1 for Reverse Video**

**U = 1 for Underline**

**GG = GPA1, GPA0**

**Note:** More than one attribute can be enabled at the same time. If the blinking and reverse video attributes are enabled simultaneously, only the reversed characters will blink.

### Cursor Timing

The cursor location is determined by a cursor row register and a character position register which are loaded by command to the controller. The cursor can be programmed to appear on the display as:

1. a blinking underline
2. a blinking reverse video block
3. a non-blinking underline
4. a non-blinking reverse video block

The cursor blinking frequency is equal to the screen refresh frequency divided by 16.

If a non-blinking reverse video cursor appears in a non-blinking reverse video field, the cursor will appear as a normal video block.

If a non-blinking underline cursor appears in a non-blinking underline field, the cursor will not be visible.

### Device Programming

The 8276H has two programming registers, the Command Register and the Parameter Register. It also has a Status Register. The Command Register can only be written into and the Status Register can only be read from. They are addressed as follows:

<table>
<thead>
<tr>
<th>C/P</th>
<th>OPERATION</th>
<th>REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Read</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>Write</td>
<td>Parameter</td>
</tr>
<tr>
<td>1</td>
<td>Read</td>
<td>Status</td>
</tr>
<tr>
<td>1</td>
<td>Write</td>
<td>Command</td>
</tr>
</tbody>
</table>

The 8276H expects to receive a command and a sequence of 0 to 4 parameters, depending on the command. If the proper number of parameter bytes are not received before another command is given, a status flag is set, indicating an improper command.
Instruction Set

The 8276H instruction set consists of 7 commands.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>NO. OF PARAMETER BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>4</td>
</tr>
<tr>
<td>Start Display</td>
<td>0</td>
</tr>
<tr>
<td>Stop Display</td>
<td>0</td>
</tr>
<tr>
<td>Load Cursor</td>
<td>2</td>
</tr>
<tr>
<td>Enable Interrupt</td>
<td>0</td>
</tr>
<tr>
<td>Disable Interrupt</td>
<td>0</td>
</tr>
<tr>
<td>Preset Counters</td>
<td>0</td>
</tr>
</tbody>
</table>

In addition, the status of the 8276H can be read by the CPU at any time.

1. RESET COMMAND

<table>
<thead>
<tr>
<th>OPERACTION</th>
<th>CP</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write 1</td>
<td>Reset Command</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write 0</td>
<td>Screen Comp Byte 1</td>
<td>S H H H H H H H H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write 0</td>
<td>Screen Comp Byte 2</td>
<td>V V R R R R R R R</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write 0</td>
<td>Screen Comp Byte 3</td>
<td>U U U U L L L L</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write 0</td>
<td>Screen Comp Byte 4</td>
<td>M I C C Z Z Z Z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Action—After the reset command is written, BRDY goes inactive, 8276H interrupts are disabled, and the VSP output is used to blank the screen. HRTC and VRTC continue to run. HRTC and VRTC timing are random on power-up.

As parameters are written, the screen composition is defined.

Parameter—VV Vertical Retrace Row Count

<table>
<thead>
<tr>
<th>V V</th>
<th>NO. OF ROW COUNTS PER VRTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>2</td>
</tr>
<tr>
<td>1 0</td>
<td>3</td>
</tr>
<tr>
<td>1 1</td>
<td>4</td>
</tr>
</tbody>
</table>

Parameter—RRRRRR Vertical Rows/Frame

<table>
<thead>
<tr>
<th>R R R R R R</th>
<th>NO. OF ROWS/FRAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 0 0 1</td>
<td>2</td>
</tr>
<tr>
<td>0 0 0 0 1 0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 1 1</td>
<td>64</td>
</tr>
</tbody>
</table>

Parameter—UUUU Underline Placement

<table>
<thead>
<tr>
<th>U U U U</th>
<th>LINE NUMBER OF UNDERLINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>2</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>16</td>
</tr>
</tbody>
</table>

Parameter—LLLL Number of Lines per Character Row

<table>
<thead>
<tr>
<th>L L L L</th>
<th>NO. OF LINES/ROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>2</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>16</td>
</tr>
</tbody>
</table>

Parameter—MM Line Counter Mode

<table>
<thead>
<tr>
<th>M</th>
<th>LINE COUNTER MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Mode 0 (Non-Offset)</td>
</tr>
<tr>
<td>1</td>
<td>Mode 1 (Offset by 1 Count)</td>
</tr>
</tbody>
</table>

Parameter—CC Cursor Format

<table>
<thead>
<tr>
<th>C C</th>
<th>CURSOR FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Blinking reverse video block</td>
</tr>
<tr>
<td>0 1</td>
<td>Blinking underline</td>
</tr>
<tr>
<td>1 0</td>
<td>Non-blinking reverse video block</td>
</tr>
<tr>
<td>1 1</td>
<td>Non-blinking underline</td>
</tr>
</tbody>
</table>
Parameter—ZZZZ Horizontal Retrace Count

<table>
<thead>
<tr>
<th>Z Z Z Z</th>
<th>NO. OF CHARACTER COUNTS PER HRTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>2</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>4</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>32</td>
</tr>
</tbody>
</table>

Note: uuuu MSB determines blanking of top and bottom lines (1 = blanked, 0 = not blanked).

2. START DISPLAY COMMAND

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>C/P</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write</td>
<td>1 Start Display</td>
<td>0 0 1 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Action—8276H interrupts are enabled, BRDY goes active, video is enabled, Interrupt Enable and Video Enable status flags are set.

3. STOP DISPLAY COMMAND

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>C/P</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write</td>
<td>1 Stop Display</td>
<td>0 1 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Action—Disables video, interrupts remain enabled, HRTC and VRTC continue to run, Video Enable status flag is reset, and the “Start Display” command must be given to reenable the display.

4. LOAD CURSOR POSITION

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>C/P</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write</td>
<td>1 Load Cursor</td>
<td>1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Parameters</td>
<td>Write</td>
<td>0 Char Number</td>
<td>(Char Position in Row)</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>0 Row Number</td>
<td>(Row Number)</td>
</tr>
</tbody>
</table>

Action—The 8276H is conditioned to place the next two parameter bytes into the cursor position registers. Status flag not affected.

5. ENABLE INTERRUPT COMMAND

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>C/P</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write</td>
<td>1 Enable Interrupt</td>
<td>1 0 1 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Action—The interrupt enable flag is set and interrupts are enabled.

6. DISABLE INTERRUPT COMMAND

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>C/P</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write</td>
<td>1 Disable Interrupt</td>
<td>1 1 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Action—Interrupts are disabled and the interrupt enable status flag is reset.

7. PRESET COUNTERS COMMAND

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>C/P</th>
<th>DESCRIPTION</th>
<th>DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Write</td>
<td>1 Preset Counters</td>
<td>1 1 1 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Action—The internal timing counters are preset, corresponding to a screen display position at the top left corner. Two character clocks are required for this operation. The counters will remain in this state until any other command is given.

This command is useful for system debug and synchronization of clustered CRT displays on a single CPU. After this command, two additional clock cycles are required before the first character of the first row is put out.

Status Flags

IE — (Interrupt Enable) Set or reset by command. It enables vertical retrace interrupt. It is automatically set by a “Start Display” command and reset with the “Reset” command.

IR — (Interrupt Request) This flag is set at the beginning of display of the last row of the frame if the interrupt enable flag is set. It is reset after a status read operation.

IC — (Improper Command) This flag is set when a command parameter string is too long or too short. The flag is automatically reset after a status read operation.

VE — (Video Enable) This flag indicates that video operation of the CRT is enabled. This flag is set on a “Start Display” command, and reset on a “Stop Display” or “Reset” command.

BU — (Buffer Underrun) This flag is set whenever a Row Buffer is not filled with character data in time for a buffer swap required by the display. Upon activation of this bit, buffer loading ceases, and the screen is blanked until after the vertical retrace interval.
ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias .......... 0°C to 70°C
Storage Temperature .............. −65°C to +150°C
Voltage On Any Pin
With Respect to Ground .......... −0.5V to +7V
Power Dissipation ................... 1 Watt

*NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_CC = 5V ±5%)

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>MIN.</th>
<th>MAX.</th>
<th>UNITS</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>−0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>V_CC + 0.5V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td></td>
<td>V</td>
<td>I_{OL} = 2.2 mA</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td></td>
<td>V</td>
<td>I_{OH} = −400 μA</td>
</tr>
<tr>
<td>IIL</td>
<td>Input Load Current</td>
<td>±10</td>
<td></td>
<td>μA</td>
<td>V_{IN} = V_CC to 0V</td>
</tr>
<tr>
<td>IOFL</td>
<td>Output Float Leakage</td>
<td>±10</td>
<td></td>
<td>μA</td>
<td>V_{OUT} = V_CC to 0.45V</td>
</tr>
<tr>
<td>ICC</td>
<td>V_CC Supply Current</td>
<td>160</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
</tbody>
</table>

CAPACITANCE (T_A = 25°C; V_CC = GND = 0V)

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>MIN.</th>
<th>MAX.</th>
<th>UNITS</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_{IN}</td>
<td>Input Capacitance</td>
<td>10</td>
<td></td>
<td>pF</td>
<td>f_C = 1 MHz</td>
</tr>
<tr>
<td>C_{I/O}</td>
<td>I/O Capacitance</td>
<td>20</td>
<td></td>
<td>pF</td>
<td>Unmeasured pins returned to V_SS.</td>
</tr>
</tbody>
</table>

A.C. TESTING LOAD CIRCUIT

![A.C. Testing Load Circuit Diagram]

C_i INCLUDES JIG CAPACITANCE
### A.C. CHARACTERISTICS

\( T_A = 0^\circ \text{C} \) to \( 70^\circ \text{C}; \ V_{CC} = 5.0 \text{V} \pm 5%; \ GND = 0 \text{V} \)

#### Bus Parameters

### READ CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{AR} )</td>
<td>Address Stable Before READ</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{RA} )</td>
<td>Address Hold Time for READ</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{RR} )</td>
<td>READ Pulse Width</td>
<td>250</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{RD} )</td>
<td>Data Delay from READ</td>
<td>200</td>
<td></td>
<td>ns</td>
<td>( C_L = 150 \text{pF} )</td>
</tr>
<tr>
<td>( t_{DF} )</td>
<td>READ to Data Floating</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### WRITE CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{AW} )</td>
<td>Address Stable Before WRITE</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{WA} )</td>
<td>Address Hold Time for WRITE</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{WW} )</td>
<td>WRITE Pulse Width</td>
<td>250</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{DW} )</td>
<td>Data Setup Time for WRITE</td>
<td>150</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{WD} )</td>
<td>Data Hold Time for WRITE</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### CLOCK TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>( 8276H )</th>
<th>( 8276-2 )</th>
<th>Min.</th>
<th>Max.</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{CLK} )</td>
<td>Clock Period</td>
<td>480</td>
<td>320</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{KH} )</td>
<td>Clock High</td>
<td>240</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{KL} )</td>
<td>Clock Low</td>
<td>160</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{KR} )</td>
<td>Clock Rise</td>
<td>5</td>
<td>30</td>
<td>5</td>
<td>30</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{KF} )</td>
<td>Clock Fall</td>
<td>5</td>
<td>30</td>
<td>5</td>
<td>30</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### OTHER TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>( 8276H )</th>
<th>( 8276-2 )</th>
<th>Min.</th>
<th>Max.</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{CC} )</td>
<td>Character Code Output Delay</td>
<td>150</td>
<td>150</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
<td>( C_L = 50 \text{pF} )</td>
<td></td>
</tr>
<tr>
<td>( t_{HR} )</td>
<td>Horizontal Retrace Output Delay</td>
<td>200</td>
<td>150</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
<td>( C_L = 50 \text{pF} )</td>
<td></td>
</tr>
<tr>
<td>( t_{LC} )</td>
<td>Line Count Output Delay</td>
<td>400</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
<td>( C_L = 50 \text{pF} )</td>
<td></td>
</tr>
<tr>
<td>( t_{AT} )</td>
<td>Control/Attribute Output Delay</td>
<td>275</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
<td>( C_L = 50 \text{pF} )</td>
<td></td>
</tr>
<tr>
<td>( t_{VR} )</td>
<td>Vertical Retrace Output Delay</td>
<td>275</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
<td>( C_L = 50 \text{pF} )</td>
<td></td>
</tr>
<tr>
<td>( t_{RI} )</td>
<td>( INT ) from ( RD )†</td>
<td>250</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
<td>( C_L = 50 \text{pF} )</td>
<td></td>
</tr>
<tr>
<td>( t_{WO} )</td>
<td>( BRDY ) from ( WR )†</td>
<td>250</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
<td>( C_L = 50 \text{pF} )</td>
<td></td>
</tr>
<tr>
<td>( t_{RQ} )</td>
<td>( BRDY ) from ( WR )†</td>
<td>200</td>
<td>200</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
<td>( C_L = 50 \text{pF} )</td>
<td></td>
</tr>
<tr>
<td>( t_{LR} )</td>
<td>( BSL ) to ( WR )†</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{RL} )</td>
<td>( WR ) to ( BSL )†</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Typical Dot Level Timing

**EXT DOT CLK**

**CC0-C6**

**CHARACTER GENERATOR OUTPUT**

**ATTRIBUTES & CONTROLS**

- **FIRST CHARACTER CODE**
- **SECOND CHARACTER CODE**

- **ROM ACCESS**

**ATTributes & ContRolS**

- **FIRST CHARACTER**
- **SECOND CHARACTER**

**VIDEO (FROM SHIFT REGISTER)**

**ATTRIBUTES & CONTROLS (FROM SYNCHRONIZER)**

*CC0-8 IS A MULTIPLE OF THE DOT CLOCK AND AN INPUT TO THE 8276*

---

**Line Timing**

**CCLK**

**CC0-C6**

**HRTC**

**LC0-3**

**VIDEO CONTROLS AND ATTRIBUTES**

**VSP, LTEL, HOL, RVV, GPAD**

*8276H ORDER NUMBER: 210668-002*
Timing for Buffer Loading

Write Timing

Read Timing

Clock Timing

Input and Output Waveforms for A.C. Tests

FOR A.C. TESTING, INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC 1 AND 0.45V FOR A LOGIC 0. TIMING MEASUREMENTS FOR INPUT AND OUTPUT SIGNALS ARE MADE AT 2.0V FOR A LOGIC 1 AND 0.8V FOR A LOGIC 0.
A Low Cost CRT Terminal Using The 8275

John Katausky
Peripherals Applications
1. INTRODUCTION

The purpose of this application note is to provide the reader with the design concepts and factual tools needed to integrate Intel peripherals and microprocessors into a low cost raster scan CRT terminal. A previously published application note, AP-32, presented one possible solution to the CRT design question. This application note expands upon the theme established in AP-32 and demonstrates how to design a functional CRT terminal while keeping the parts count to a minimum.

For convenience, this application note is divided into seven general sections:

1. Introduction
2. CRT Basics
3. 8275 Description
4. Design Background
5. Circuit Description
6. Software Description
7. Appendix

There is no question that microprocessors and LSI peripherals have had a significant role in the evolution of CRT terminals. Microprocessors have allowed design engineers to incorporate an abundance of sophisticated features into terminals that were previously mere slaves to a larger processor. To complement microprocessors, LSI peripherals have reduced component count in many support areas. A typical LSI peripheral easily replaces between 30 and 70 SSI and MSI packages, and offers features and flexibility that are usually not available in most hardware designs. In addition to replacing a whole circuit board of random logic, LSI circuits also reduce the cost and increase the reliability of design. Fewer interconnects increases mechanical reliability and fewer parts decreases the power consumption and hence, the overall reliability of the design. The reduction of components also yields a circuit that is easier to debug during the actual manufacturing phase of a product.

Until the era of advanced LSI circuitry, a typical CRT terminal consisted of 80 to 200 or more SSI and MSI packages. The first microprocessors and peripherals dropped this component count to between 30 and 50 packages. This application note describes a CRT terminal that uses 20 packages.

2. CRT BASICS

The raster scan display gets its name from the fact that the image displayed on the CRT is built up by generating a series of lines (raster) across the face of the CRT. Usually, the beam starts in the upper left hand corner of the display and simultaneously moves left to right and top to bottom to put a series of zig-zag lines on the screen (Fig. 2.1). Two simultaneously operating independent circuits control the vertical and horizontal movement of the beam.

As the electron beam moves across the face of the CRT, a third circuit controls the current flowing in the beam. By varying the current in the electron beam the image on the CRT can be made to be as bright or as dark as the user desires. This allows any desired pattern to be displayed.

When the beam reaches the end of a line, it is brought back to the beginning of the next line at a rate that is much faster than was used to generate the line. This action is referred to as "retrace". During the retrace period the electron beam is usually shut off so that it doesn’t appear on the screen.

As the electron beam is moving across the screen horizontally, it is also moving downward. Because of this, each successive line starts slightly below the previous line. When the beam finally reaches the bottom right hand corner of the screen, it retraces vertically back to the top left hand corner. The time it takes for the beam to move from the top of the screen to the bottom and back again to the top is usually referred to as a "frame". In the United States, commercial television broadcast use 15,750 Hz as the horizontal sweep frequency (63.5 microseconds per horizontal line) and 60 Hz as the vertical sweep frequency or "frame" (16.67 milliseconds per vertical frame).

Although, the 60 Hz vertical frame and the 15,750 Hz horizontal line are the standards used by commercial broadcasts, they are by no means the only frequency at which CRT's can operate. In fact, many CRT displays use a horizontal scan that is around 18 KHz to 22 KHz and some even exceed 30 KHz. As the
horizontal frequency increases, the number of horizontal lines per frame increases. Hence, the resolution on the vertical axis increases. This increased resolution is needed on high density graphic displays and on special text editing terminals that display many lines of text on the CRT.

Although many CRT's operate at non-standard horizontal frequencies, very few operate at vertical frequencies other than 60 Hz. If a vertical frequency other than 60 Hz is chosen, any external or internal magnetic or electrical variations at 60 Hz will modulate the electron beam and the image on the screen will be unstable. Since, in the United States, the power line frequency happens to be 60 Hz, there is a good chance for 60 Hz interference to exist. Transformers can cause 60 Hz magnetic fields and power supply ripple can cause 60 Hz electrical variations. To overcome this, special shielding and power supply regulation must be employed. In this design, we will assume a standard frame rate of 60 Hz and a standard line rate of 15,750 Hz.

By dividing the 63.5 microsecond horizontal line rate into the 16.67 millisecond vertical rate, it is found that there are 262.5 horizontal lines per vertical frame. At first, the half line may seem a bit odd, but actually it allows the resolution on the CRT to be effectively doubled. This is done by inserting a second set of horizontal lines between the first set (interlacing). In an interlaced system the line sets are not generated simultaneously. In a 60 Hz system, first all of the even-numbered lines are scanned: 0, 2, 4, ... 524. Then all the odd-numbered lines: 1, 3, 5, ... 525. Each set of lines usually contains different data (Fig. 2.2).

Although interlacing provides greater resolution, it also has some distinct disadvantages. First of all, the circuitry needed to generate the extra half horizontal line per frame is quite complex when compared to a noninterlaced design, which requires an integer number of horizontal lines per frame. Next, the overall vertical refresh rate is half that of a noninterlaced display. As a result, flicker may result when the CRT uses high speed phosphors. To keep things as simple as possible, this design uses the noninterlaced approach.

The first thing any CRT controller must do is generate pulses that define the horizontal line timing and the vertical frame timing. This is usually done by dividing a crystal reference source by some appropriate numbers. On most raster scan CRT's the horizontal frequency is very forgiving and can vary by around 500 Hz or so and produce no ill effects. This means that the CRT itself can track a horizontal frequency between 15250 Hz and 16250 Hz, or in other words, there can be 256 to 270 horizontal lines per vertical frame. But, as mentioned earlier, the vertical frequency should be 60 Hz to insure stability.

The characters that are viewed on the screen are formed by a series of dots that are shifted out of the controller while the electron beam moves across the face of the CRT. The circuits that create this timing are referred to as the dot clock and character clock. The character clock is equal to the dot clock divided by the number of dots used to form a character along the horizontal axis and the dot clock is calculated by the following equation:

\[
\text{DOT CLOCK (Hz)} = \frac{(N + R) \times D \times L \times F}{D}
\]

where N is the number of displayed characters per row,
R is the number of retrace character time increments,
D is the number of dots per character,
L is the number of horizontal lines per frame and
F is the frame rate in Hz.

In this design \(N = 80\), \(R = 20\), \(D = 7\), \(L = 270\), and \(F = 60\) Hz. If the numbers are plugged in, the dot clock is found to be 11.34 MHz.

The retrace number, \(R\), may vary from system to system because it is used to establish the margins on the left and right hand sides of the CRT. In this particular design \(R = 20\) was empirically found it be optimum. The number of dots per character may vary depending on the character generator used and the number of dot clocks the designer wants to place between characters. This design uses a 5 X 7 dot matrix and allows 2 dot clock periods between characters (see Fig. 2.3); since 5 + 2 equals 7, we find that \(D = 7\).
APPLICATIONS

The number of lines per frame can be determined by the following equation:

$$L = (H \times Z) + V$$

where, $H$ is the number of horizontal lines per character,

$Z$ is the number of character lines per frame and

$V$ is the number of horizontal lines during vertical retrace. In this design, a 5 X 7 dot matrix is to be placed on a 7 X 10 field, so $H = 10$. Also, 25 lines are to be displayed, so $Z = 25$. As mentioned before, $V = 20$. When the numbers are plugged into the equation, $L$ is found to be equal to 270 lines per frame.

The designer should be cautioned that these numbers are interrelated and that to guarantee proper operation on a standard raster scan CRT, $L$ should be between 256 and 270. If $L$ does not lie within these bounds the horizontal circuits of the CRT may not be able to lock onto the driving signal and the image will roll horizontally. The chosen $L$ of 270 yields a horizontal frequency of 16,200 KHz on a 60 Hz frame and this number is within the 500 Hz tolerance mentioned earlier.

The $V$ number is chosen to match the CRT in much the same manner as the $R$ number mentioned earlier. When the electron beam reaches the bottom right corner of the screen it must retrace vertically to the top left corner. This retrace action requires time, usually between 900-1200 microseconds. To allow for this, enough horizontal sync times must be inserted during vertical retrace. Twenty horizontal sync times at 61.5 microseconds yield a total of 1234.5 microseconds, which is enough time to allow the beam to return to the top of the screen.

The choices of $H$ and $Z$ largely relate to system design preference. As $H$ increases, the character size along the vertical axis increases. $Z$ is simply the number of lines of characters that are displayed and this, of course, is entirely a system design option.

![Figure 2-3. 5 X 7 Dot Matrix](image)

![Figure 3-1. 8275 Block Diagram/Pin Configuration](image)
3. 8275 DESCRIPTION

A block diagram and pin configuration of the 8275 are shown in Fig. 3.1. The following is a description of the general capabilities of the 8275.

3.1 CRT DISPLAY REFRESHING

The 8275, having been programmed by the designer to a specific screen format, generates a series of DMA request signals, resulting in the transfer of a row of characters from display memory to the 8275's row buffers. The 8275 presents the character codes to an external character generator ROM by using outputs CCO-CC6. External dot timing logic is then used to transfer the parallel output data from the character generator ROM serially to the video input of the CRT. The character rows are displayed on the CRT one line at a time. Line count outputs LC0-LC3 are applied to the character generator ROM to perform the line selection function. The display process is illustrated in Figure 3.2. The entire process is repeated for each display row. At the beginning of the last displayed row, the 8275 issues an interrupt by setting the IRQ output line. The 8275 interrupt output will normally be connected to the interrupt input of the system central processor.

The interrupt causes the CPU to execute an interrupt service subroutine. The service subroutine typically re-initializes DMA controller parameters for the next display refresh cycle, polls the system keyboard controller, and/or executes other appropriate functions. A block diagram of a CRT system implemented with the 8275 CRT Controller is provided in Figure 3.3. Proper CRT refreshing requires that certain 8275 parameters be programmed prior to the beginning of display operation. The 8275 has two types of programming registers, the Command Registers (CREG) and the Parameter Registers (PREG). It also has a Status Register (SREG). The Command Registers may only be written to and the Status Registers may only be read. The 8275 expects to receive a command followed by a sequence of from 0 to 4 parameters, depending on the command. The 8275 instruction set consist of the eight commands shown in Figure 3.4.

To establish the format of the display, the 8275 provides a number of user programmable display format parameters. Display formats having from 1 to 80 characters per row, 1 to 64 rows per screen, and 1 to 16 horizontal lines per row are available.

In addition to transferring characters from memory
to the CRT screen, the 8275 features cursor position control. The cursor position may be programmed, via X and Y cursor position registers, to any character position on the display. The user may select from four cursor formats. Blinking or non-blinking underline and reverse video block cursors are available.

3.2 CRT TIMING

The 8275 provides two timing outputs, HRTC and VRTC, which are utilized in synchronizing CRT horizontal and vertical oscillators to the 8275 refresh cycle. In addition, whenever HRTC or VRTC is active, a third timing output, VSP (Video Suppress) is true, providing a blinking signal to the dot timing logic. The dot timing logic will normally inhibit the video output to the CRT during the time when video suppress signal is true. An additional timing output, LTEN (Light Enable) is used to provide the ability to force the video output high regardless of the state of VSP. This feature is used by the 8275 to place a cursor on the screen and to control attribute functions. Attributes will be considered in the next section.

The HLGT (Highlight) output allows an attribute function to increase the CRT beam intensity to a level greater than normal. The fifth timing signal, RVV (Reverse Video) will, when enabled, cause the system video output to be inverted.

Figure 3-4. 8275's Instruction Set
Character attributes were designed to produce the following graphics:

<table>
<thead>
<tr>
<th>CHARACTER ATTRIBUTE CODE &quot;CCCC&quot;</th>
<th>OUTPUTS</th>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LA1</td>
<td>LA0</td>
<td>VSP</td>
</tr>
<tr>
<td>Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Underline</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Below Underline</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Underline</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Below Underline</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Above Underline</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Underline</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Below Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Above Underline</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Underline</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Below Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Underline</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Below Underline</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Above Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Below Underline</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Above Underline</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Underline</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Below Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Above Underline</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Underline</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Below Underline</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Above Underline</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Underline</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Below Underline</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Above Underline</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Underline</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Below Underline</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Character Attribute Code 1011 is not recommended for normal operation. Since none of the attribute outputs are active, the character Generator will not be disabled, and an indeterminate character will be generated.

Character Attribute Codes 1101, 1110, and 1111 are illegal.

Blinking is active when B = 1.

Highlight is active when H = 1.

Figure 3-5. Character Attributes
3.3 SPECIAL FUNCTIONS

VISUAL ATTRIBUTES—Visual attributes are special codes which, when retrieved from display memory by the 8275, affect the visual characteristics of a character position or field of characters. Two types of visual attributes exist, character attributes and field attributes.

Character Attribute Codes: Character attribute codes can be used to generate graphics symbols without the use of a character generator. This is accomplished by selectively activating the Line Attribute outputs (LAO-LAI), the Video Suppression output (VSP), and the Light Enable output (LTEN). The dot timing logic uses these signals to generate the proper symbols. Character attributes can be programmed to blink or be highlighted individually. Blinking is accomplished with the Video Suppression output (VSP). Blink frequency is equal to the screen refresh frequency divided by 32. Highlighting is accomplished by activating the Highlight output (HGLT). Character attributes were designed to produce the graphic symbols shown in Figure 3.5.

Field Attribute Codes: The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the field attribute code up to, and including, the character which precedes the next field attribute code, or up to the end of the frame.

There are six field attributes:

1. **Blink** — Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.

2. **Highlight** — Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).

3. **Reverse Video** — Characters following the code are caused to appear in reverse video format by activating the Reverse Video output (RVV).

4. **Underline** — Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).

5. **General Purpose** — There are two additional 8275 outputs which act as general purpose, independently programmable field attributes. These attributes may be used to select colors or perform other desired control functions.

The 8275 can be programmed to provide visible or invisible field attribute characters as shown in Figure 3.6. If the 8275 is programmed in the visible field attribute mode, all field attributes will occupy a position on the screen. They will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character. If the 8275 is programmed in the invisible field attribute mode, the 8275 row buffer FIFOs are activated. The FIFOs effectively lengthen the row buffers by 16 characters, making room for up to 16 field attribute characters per display row. The FIFOs are 126 characters by 7 bits in size. When a field attribute is placed in the row buffer during DMA, the buffer input controller recognizes it and places the next character in the proper FIFO. When a field attribute is placed in the buffer output controller during display, it causes the controller to immediately put a character from the FIFO on the Character Code outputs (CCO-6). The chosen attributes are also activated.
LIGHT PEN DETECTION — A light pen consists fundamentally of a switch and light sensor. When the light pen is pressed against the CRT screen, the switch enables the light sensor. When the raster sweep coincides with the light sensor position on the display, the light pen output is input and the row and character position coordinates are stored in two 8275 internal registers. These registers can be read by the microprocessor.

SPECIAL CODES — Four special codes may be used to help reduce memory, software, or DMA overhead. These codes are placed in character positions in display memory.

1. End Of Row Code - Activates VSP. VSP remains active until the end of the line is reached. While VSP is active, the screen is blanked.

2. End Of Row-Stop DMA Code - Causes the DMA Control Logic to stop DMA for the rest of the row when it is written into the row buffer. It affects the display in the same way as the End of Row Code.

3. End Of Screen Code - Activates VSP. VSP remains active until the end of the frame is reached.

4. End Of Screen-Stop DMA Code - Causes the DMA Control Logic to stop DMA for the rest of the frame when it is written into the row buffer. It affects the display in the same way as the End of Screen Code.

PROGRAMMABLE DMA BURST CONTROL — The 8275 can be programmed to request single-byte DMA transfers of DMA burst transfers of 2, 4, or 8 characters per burst. The interval between bursts is also programmable. This allows the user to tailor the DMA overhead to fit the system needs.

4. DESIGN BACKGROUND

4.1 DESIGN PHILOSOPHY

Since the cost of any CRT system is somewhat proportional to parts count, arriving at a minimum part count solution without sacrificing performance has been the motivating force throughout this design effort. To successfully design a CRT terminal and keep the parts count to a minimum, a few things became immediately apparent.

1. An 8085 should be used.
2. Address and data buffering should be eliminated.
3. Multi-port memory should be eliminated.
4. DMA should be eliminated.

Decision 1 is obvious, the 8085's on-board clock generator, bus controller and vectored interrupts greatly reduce the overall part count considerably. Decision 2 is fairly obvious; if a circuit can be designed so that loading on the data and address lines is kept to a minimum, both the data and address buffers can be eliminated. This easily saves three to eight packages and reduces the power consumption of the design. Both decisions 3 and 4 require a basic understanding of current CRT design concepts.

In any CRT design, extreme time conflicts are created because all essential elements require access to the bus. The CPU needs to access the memory to control the system and to handle the incoming characters, but, at the same time, the CRT controller needs to access the memory to keep the raster scan display refreshed. To resolve this conflict two common techniques are employed, page buffering and line buffering.

In the page buffering approach the entire screen memory is isolated from the rest of the system. This isolation is usually accomplished with three-state buffers or two line to one line multiplexers. Of course, whenever a character needs to be manipulated the CPU must gain access to the buffered memory and, again, possible contention between the CPU and the CRT controller results. This contention is usually resolved in one of two ways, (1) the CPU is always given priority, or; (2) the CPU is allowed to access the buffered memory only during horizontal and vertical retrace times.

Approach 1 is the easiest to implement from a hardware point of view, but if the CPU always has priority the display may temporarily blink or "flicker" while the CPU accesses the display memory. This, of course, occurs because when the CPU accesses the display memory the CRT controller is not able to retrieve a character, so the display must be blanked during this time. Aesthetically, this "flickering" is not desirable, so approach 2 is often used.

The second approach eliminates the display flickering encountered in the previously mentioned technique, but additional hardware is required. Usually the vertical and horizontal blank signals are gated with the buffered memory select lines and this line is used to control the CPU's ready line. So, if the CPU wants to use the buffered memory, its ready line is asserted until horizontal or vertical retrace times. This, of course, will impact the CPU's overall throughput.

Both page buffered approaches require a significant amount of additional hardware and for the most part are not well suited for a minimum parts count type of terminal. This guides us to the line buffered approach. This approach eliminates the separate buffered memory for the display, but, at the same time, introduces a few new problems that must be solved.
In the line buffered approach both the CPU and the CRT controller share the same memory. Every time the CRT controller needs a new character or line of data, normal processing activity is halted and the CRT controller accesses memory and displays the data. Just how the CRT controller needs to acquire the display data greatly affects the performance of the overall system. Whether the CRT controller needs to gain access to the main memory to acquire a single character or a complete line of data depends on the presence or absence of a separate line or row buffer.

If no row buffer is present the CRT controller must go to the main memory to fetch every character. This of course, is not a very efficient approach because the processor will be forced to relinquish the bus 70% to 80% of the time. So much processor inactivity greatly affects the overall system performance. In fact terminals that use this approach are typically limited to around 1200 to 2400 baud on their serial communication channels. This low baud rate is in general not acceptable, hence this approach was not chosen.

If a separate row buffer is employed the CRT controller only has to access the memory once for each displayed character per line. This forces the processor to relinquish the bus only about 20% to 35% of the time and a full 4800 to 9600 baud can be achieved. Figure 4.1 illustrates these different techniques.

The 8275 CRT controller is ideal for implementing the row buffer approach because the row buffer is contained on the device itself. In fact, the 8275 contains two 80-byte row buffers. The absence of two row buffers allow one buffer to be filled while the other buffer is displaying the data. This dual row buffer approach enhances CPU performance even further.

4.2 USING THE 8275 WITHOUT DMA

Until now the process of filling the row buffer has only been alluded to. In reality, a DMA technique is usually used. This approach was demonstrated in AP-32 where an 8257 DMA controller was mated to an 8275 CRT controller. In order to minimize component count, this design eliminates the DMA controller and its associated circuitry while replacing them with a special interrupt-driven transfer.

The only real concern with using the 8275 in an interrupt-driven transfer mode is speed. Eighty characters must be loaded into the 8275 every 617 micro seconds and the processor must also have time to perform all the other tasks that are required. To minimize the overhead associated with loading the characters into the 8275 a special technique was employed. This technique involves setting a special
transfer bit and executing a string of POP instructions. The string of POP instructions is used to rapidly move the data from the memory into the 8275. Figure 4.2 shows the basic software structure.

In this design the 8085's SOD line was used as the special transfer bit. In order to perform the transfer properly this special bit must do two things: (1) turn processor reads into DACK plus WR for the 8275 and (2) mask processor fetch cycles from the 8275, so that a fetch cycle does not write into the 8275. Conventional logic could have been used to implement this special function, but in this design a small bipolar programmable read only memory was used. Figure 4.3 shows a basic version of the hardware.

At first, it may seem strange that we are supplying a DACK when no DMA controller exist in the system. But the reader should be aware that all Intel peripheral devices that have DMA lines actually use DACK as a chip select for the data. So, when you want to write a command or read status you assert CS and WR or RD, but when you want to read or write data you assert DACK and RD or WR. The peripheral device doesn't "know" if a DMA controller is in the circuit or not. In passing, it should be mentioned that DACK and CS should not be asserted on the same device at the same time, since this combination yields an undefined result.

This POP technique actually compares quite favorably in terms of time to the DMA technique. One POP instruction transfers two bytes of data to the 8275 and takes 10 CPU clock cycles to execute, for a net transfer rate of one byte every five clock cycles. The DMA controller takes four clock cycles to transfer one byte but, some time is lost in synchronization. So the difference between the two techniques is one clock cycle per byte maximum. If we compare the overall speed of the 8085 to the speed of the 8080 used in AP-32, we find that at 3 MHz we can transfer one byte every 1.67 microseconds using the 8085 and POP technique vs. 2 microseconds per byte for the 2 MHz 8080 using DMA.

5. CIRCUIT DESCRIPTION

5.1 SCOPE OF THE PROJECT

A fully functional, microprocessor-based CRT terminal was designed and constructed using the 8275 CRT controller and the 8085 as the controlling element. The terminal had many of the functions found in existing commercial low-cost terminals and more sophisticated features could easily be added with a modest amount of additional software. In order to minimize component count LSI devices were used whenever possible and software was used to replace hardware.

5.2 SYSTEM TARGET SPECIFICATIONS

The design specifications for the CRT terminal were as follows:

Display Format
- 80 characters per display row
- 25 display rows

Character Format
- 5 X 7 dot matrix character contained within a 7 X 10 matrix
- First and seventh columns blanked
- Ninth line cursor position
- Blinking underline cursor

Special Characters Recognized
- Control characters
- Line feed
- 'Carriage Return
- Backspace
- Form feed

Escape Sequences Recognized
- ESC, A, Cursor up
- ESC, B, Cursor down
- ESC, C, Cursor right
- ESC, D, Cursor left
- ESC, E, Clear screen
- ESC, H, Home cursor
- ESC, J, Erase to the end of the screen
- ESC, K, Erase the current line

Characters Displayed
- 96 ASCII alphanumeric characters
- Special control characters
Characters Transmitted
- 96 ASCII alphanumeric characters
- ASCII control characters

Program Memory
- 2K bytes of 2716 EPROM

Display/Buffer/Stack Memory
- 2K bytes 2114 static memory (4 packages)

Data Rate
- 9600 BAUD using 3MHz 8085

CRT Monitor
- Ball Bros TV-12, 12MHz B.W.

Keyboard
- Any standard un-encoded ASCII keyboard

Screen Refresh Rate
- 60 Hz

5.3 HARDWARE DISCRIPTION
A block diagram of the CRT terminal is shown in Figure 5.1. The diagram shows only the essential system features. A detailed schematic of the CRT is contained in the Appendix. The terminal was constructed on a simple 6" by 6" wire wrap board. Because of the minimum bus loading no buffering of any kind was needed (see Figure 5.2).

The “heart” of the CRT terminal is the 8085 microprocessor. The 8085 initializes all devices in the system, loads the CRT controller, scans the keyboard, assembles the characters to be transmitted, decodes the incoming characters and determines where the character is to be placed on the screen. Clearly, the processor is quite busy.

A standard list of LSI peripheral devices surround the 8085. The 8251A is used as the serial communication link, the 8255A-5 is used to scan the keyboard and read the system variables through a set of

Figure 5-1. CRT Terminal Block Diagram

Worst case bus loading:

<table>
<thead>
<tr>
<th>Description</th>
<th>8275</th>
<th>8255A-5</th>
<th>8253-5</th>
<th>8253-5</th>
<th>8251A</th>
<th>2x 2114</th>
<th>2716</th>
<th>8212</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address:</td>
<td>20pf</td>
<td>20pf</td>
<td>20pf</td>
<td>20pf</td>
<td>20pf</td>
<td>10pf</td>
<td>12pf</td>
<td>12pf</td>
</tr>
<tr>
<td>Data Bus:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>工商联:</td>
<td>20pf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Only A8 - A15 are important since A0 - A7 are latched by the 8212

Address Bus: 4x 2114 20pf
2116 6pf

This loading assures that all components will be compatible with a 3MHz 8085 and that no wait states will be required

Figure 5-2. Bus Loading
switches, and the 8253 is used as a baud rate generator and as a “horizontal pulse extender” for the 8275.

The 8275 is used as the CRT controller in the system, and a 2716 is used as the character generator. To handle the high speed portion of the terminal the 8275 is surrounded by a small handful of TTL. The program memory is contained in one 2716 EPROM and the data and screen memory use four 2114-type RAMs.

All devices in this system are memory mapped. A bipolar PROM is used to decode all of the addresses for the RAM, ROM, 8275, and 8253. As mentioned earlier, the bipolar prom also turns READs into DACK's and WR's for the 8275. The 8255 and 8253 are decoded by a simple address line chip select method. The total package count for the system is 20, not including the serial line drivers.

If this same terminal were designed using the MCS-85 family of integrated circuits, additional part savings could have been realized. The four 2114's could have been replaced by two 8185's and the 8255 and the 2716 program PROM could have been replaced by one 8755. Additionally, since both the 8185 and the 2716 have address latches no 8212 would be needed, so the total parts count could be reduced by three or four packages.

5.4 SYSTEM OPERATION

The 8085 CPU initializes each peripheral to the appropriate mode of operation following system reset. After initialization, the 8085 continually polls the 8251A to see if a character has been sent to the terminal. When a character has been received, the 8085 decodes the character and takes appropriate action. While the 8085 is executing the above “foreground” programs, it is being interrupted once every 617 microseconds by the 8275. This “background” program is used to load the row buffers on the 8275. The 8085 is also interrupted once every frame time, or 16.67 ms, to read the keyboard and the status of the 8275.

As discussed earlier, a special POP technique was used to rapidly move the contents of the display RAM into the 8275’s row buffers. The characters are then synchronously transferred to the character code outputs CC0-CC6, connected to the character generator address lines A3-A9 (Figure 5.3). Line count outputs LC0-LC2 from the 8275 are applied to the character generator address lines A0-A2. The 8275 displays character rows one line at a time. The line count outputs are used to determine which line of the character selected by A3-A8 will be displayed. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line of the character row is selected. This process continues until the last line of the row is transferred to the dot timing logic.

The dot timing logic latches the output of the character generator ROM into a parallel in, serial out synchronous shift register. This shift register is clocked at the dot clock rate (11.34 MHz) and its output constitutes the video input to the CRT.

5.5 SYSTEM TIMING

Before any specific timing can be calculated it is necessary to determine what constraints the chosen CRT places on the overall timing. The requirements for the Ball Bros. TV-12 monitor are shown in Table 5.1. The data from Table 5.1, the 8275 specifications, and the system target specifications are all that is needed to calculate the system's timing.

![Figure 5-3 Character Generator/Dot Timing Logic Block Diagram](image)

Table 5-1

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical Blanking Time (VRTC)</td>
<td>900 μsec nominal</td>
</tr>
<tr>
<td>Vertical Drive Pulswidth</td>
<td>300 μsec ≤ PW ≤ 1.4 ms</td>
</tr>
<tr>
<td>Horizontal Blanking Time (HRTC)</td>
<td>11 μsec nominal</td>
</tr>
<tr>
<td>Horizontal Drive Pulswidth</td>
<td>25 μsec ≤ PW ≤ 30 μsec</td>
</tr>
<tr>
<td>Horizontal Repetition Rate</td>
<td>15,750 ±500 pps</td>
</tr>
</tbody>
</table>

![Figure 5-4. Row Format](image)
First, let's select and "match" a few numbers. From our target specifications, we see that each character is displayed on a $7 \times 10$ field, and is formed by a $5 \times 7$ dot matrix (Figure 5.4). The 8275 allows the vertical retrace time to be only an integer multiple of the horizontal character line. This means that the total number of horizontal lines in a frame equals 10 times the number of character lines plus the vertical retrace time, which is programmed to be either 1, 2, 3, or 4 character lines. Twenty-five display lines
APPLICATIONS

require 250 horizontal lines. So, if we wish to have a horizontal frequency in the neighborhood of 15,750 Hz we must choose either one or two character lines for vertical retrace. To allow for a little more margin at the top and bottom of the screen, two character lines were chosen for vertical retrace. This choice yields a net 250 + 20 = 270 horizontal lines per frame. So, assuming a 60 Hz frame:

60 Hz * 270 = 16,200 Hz (horizontal frequency)

This value falls within our target specification of 15,750 Hz with a 500 Hz variation and also assures timing compatibility with the Ball monitor since, 20 horizontal sync times yield a vertical retrace time of:

61.7 microseconds X 20 horizontal sync times = 1,234.5 milliseconds

This number meets the nominal VRTC and vertical drive pulse width time for the Ball monitor. A horizontal frequency of 16,200 Hz implies a 1/16,200 = 61.73 microseconds period.

It is now known that the terminal is using 250 horizontal lines to display data and 20 horizontal lines to allow for vertical retrace and that the horizontal frequency is 16,200 Hz. The next thing that needs to be determined is how much time must be allowed for horizontal retrace. Unfortunately, this number depends almost entirely on the monitor used. Usually, this number lies somewhere between 15 and 30 percent of the total horizontal line time, which in this case is 1/16,200 Hz or 61.73 microseconds. Since in most designs a fixed number of characters can be displayed on a horizontal line, it is often useful to express retrace as a given number of character times. In this design, 80 characters can be displayed on a horizontal line and it was empirically found that allowing 20 horizontal character times for retrace gave the best results. So, in reality, there are 100 character times in every given horizontal line, 80 are used to display characters and 20 are used to allow for retrace. It should be noted that if too many character times are used for retrace, less time will be left to display the characters and the display will not “fill out” the screen. Conversely, if not enough character times are allowed for retrace, the display may “run off” the screen.

One hundred character times per complete horizontal line means that each character requires

61.73 microseconds / 100 character times = 617.3 nanoseconds.

If we multiply the 20 horizontal retrace times by the

---

Figure 5-6. CRT System Timing
617.3 nanoseconds needed for each character, we find

\[ 617.3 \text{ nanoseconds} \times 20 \text{ retrace times} = 12.345 \text{ microseconds} \]

This value falls short of the 25 to 30 microseconds required by the horizontal drive of the Ball monitor. To correct for this, an 8253 was programmed in the one-shot mode and was used to extend the horizontal drive pulsewidth.

Now that the 617.3 nanosecond character clock period is known, the dot clock is easy to calculate. Since each character is formed by placing 7 dots along the horizontal:

\[ \text{DOT CLOCK PERIOD} = \frac{617.3 \text{ ns}}{(\text{CHARACTER CLK PERIOD})/7 \text{ DOTS}} \]

\[ \text{DOT CLOCK PERIOD} = 88.183 \text{ nanoseconds} \]

\[ \text{DOT CLOCK FREQUENCY} = \frac{1}{\text{PERIOD}} = 11.34 \text{ MHz} \]

Figures 5.5 and 5.6 illustrate the basic dot timing and the CRT system timing, respectively.

6. SYSTEM SOFTWARE

6.1 SOFTWARE OVERVIEW.

As mentioned earlier the software is structured on a “foreground-background” basis. Two interrupt-driven routines, FRAME and POPDAT (Fig. 6.1) request service every 16.67 milliseconds and 617 microseconds respectively, frame is used to check the baud rate switches, update the system pointers and decode and assemble the keyboard characters. POPDAT is used to move data from the memory into the 8275's row buffer rapidly.

The foreground routine first examines the line-local switch to see whether to accept data from the USART or the keyboard. If the terminal is in the local mode, action will be taken on any data that is entered through the keyboard and the USART will be ignored on both output and input. If the terminal is in the line mode data entered through the keyboard will be transmitted by the USART and action will be taken on any data read out of the USART.

When data has been entered in the terminal the software first determines if the character received was an escape, line feed, form feed, carriage return, back space, or simply a printable character. If an escape was received the terminal assumes the next received character will be a recognizable escape sequence character. If it isn't no operation is performed.

After the character is decoded, the processor jumps to the routine to perform the required task. Figure 6.2 is a flow chart of the basic software operations; the program is listed in Appendix 6.8.

---

**Figure 6.1. Frame and Popdat Interrupt Routines**
6.2 SYSTEM MEMORY ORGANIZATION

The display memory organization is shown in Figure 6.3. The display begins at location 0800H in memory and ends at location 0FCFH. The 48 bytes of RAM from location 0FD0H to 0FFFH are used as system stack and temporary system storage. 2K bytes of PROM located at 0000H through 07FFH contain the systems program.

6.3 MEMORY POINTERS AND SCROLLING

To calculate the location of a character on the screen, three variables must be defined. Two of these variables are the X and Y position of the cursor (CURSX, CURSY). In addition, the memory address defining the top line of the display must be known, since scrolling on the 8275 is accomplished simply by changing the pointer that loads the 8275’s row buffers from memory. So, if it is desired to scroll the display up or down all that must be changed is one 16-bit memory pointer. This pointer is entered into the system by the variable TOPAD (TOP Address) and always defines the top line of the display. Figure 6.4 details screen operation during scrolling.

Subroutines CALCU (Calculate) and ADX (ADd X axis) use these three variables to calculate an absolute memory address. The subroutine CALCU is used whenever a location in the screen memory must be altered.

6.4 SOFTWARE TIMING

One important question that must be asked about the terminal software is, “How fast does it run”. This is important because if the terminal is running at 9600 baud, it must be able to handle each received character in 1.04 milliseconds. Figure 6.5 is a flowchart of the subroutine execution times. It should be pointed out that all of the times listed are “worst case” execution times. This means that all routines assume they must do the maximum amount of data manipulation. For instance, the PUT routine assumes that the character is being placed in the last column and that a line feed must follow the placing of the character on the screen.

How fast do the routines need to execute in order to assure operation at 9600 baud? Since POPDAT interrupts occur every 617 microseconds, it is possible to receive two complete interrupt requests in every character time (1042 microseconds) at 9600 baud.
### APPLICATIONS

<table>
<thead>
<tr>
<th>ROW 1</th>
<th>0800H 0801H</th>
<th>.084FH</th>
<th>ROW 2</th>
<th>0850H 0851H</th>
<th>.089FH</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW 2</td>
<td>0850H 0851H</td>
<td>.089FH</td>
<td>ROW 3</td>
<td>08A0H 08A1H</td>
<td>.08EFH</td>
</tr>
<tr>
<td>ROW 3</td>
<td>08A0H 08A1H</td>
<td>.08EFH</td>
<td>ROW 4</td>
<td>08F0H 08F1H</td>
<td>.093FH</td>
</tr>
<tr>
<td>ROW 4</td>
<td>08F0H 08F1H</td>
<td>.093FH</td>
<td>ROW 5</td>
<td>0940H 0941H</td>
<td>.098FH</td>
</tr>
<tr>
<td>ROW 5</td>
<td>0940H 0941H</td>
<td>.098FH</td>
<td>ROW 6</td>
<td>0990H 0991H</td>
<td>.090FH</td>
</tr>
<tr>
<td>ROW 6</td>
<td>0990H 0991H</td>
<td>.090FH</td>
<td>ROW 7</td>
<td>09E0H 09E1H</td>
<td>.0A2FH</td>
</tr>
<tr>
<td>ROW 7</td>
<td>09E0H 09E1H</td>
<td>.0A2FH</td>
<td>ROW 8</td>
<td>0A30H 0A31H</td>
<td>.0A7FH</td>
</tr>
<tr>
<td>ROW 8</td>
<td>0A30H 0A31H</td>
<td>.0A7FH</td>
<td>ROW 9</td>
<td>0A80H 0A81H</td>
<td>.0ACFH</td>
</tr>
<tr>
<td>ROW 9</td>
<td>0A80H 0A81H</td>
<td>.0ACFH</td>
<td>ROW 10</td>
<td>0AD0H 0AD1H</td>
<td>.0B1FH</td>
</tr>
<tr>
<td>ROW 10</td>
<td>0AD0H 0AD1H</td>
<td>.0B1FH</td>
<td>ROW 11</td>
<td>0B20H 0B21H</td>
<td>.0B6FH</td>
</tr>
<tr>
<td>ROW 11</td>
<td>0B20H 0B21H</td>
<td>.0B6FH</td>
<td>ROW 12</td>
<td>0B70H 0B71H</td>
<td>.0BBFH</td>
</tr>
<tr>
<td>ROW 12</td>
<td>0B70H 0B71H</td>
<td>.0BBFH</td>
<td>ROW 13</td>
<td>0B60H 0B61H</td>
<td>.0C0FH</td>
</tr>
<tr>
<td>ROW 13</td>
<td>0B60H 0B61H</td>
<td>.0C0FH</td>
<td>ROW 14</td>
<td>0C60H 0C61H</td>
<td>.0C5FH</td>
</tr>
<tr>
<td>ROW 14</td>
<td>0C60H 0C61H</td>
<td>.0C5FH</td>
<td>ROW 15</td>
<td>0C60H 0C61H</td>
<td>.0C5FH</td>
</tr>
<tr>
<td>ROW 15</td>
<td>0C60H 0C61H</td>
<td>.0C5FH</td>
<td>ROW 16</td>
<td>0C0H 0C1H</td>
<td>.0CFFH</td>
</tr>
<tr>
<td>ROW 16</td>
<td>0C0H 0C1H</td>
<td>.0CFFH</td>
<td>ROW 17</td>
<td>0D00H 0D01H</td>
<td>.0D4FH</td>
</tr>
<tr>
<td>ROW 17</td>
<td>0D00H 0D01H</td>
<td>.0D4FH</td>
<td>ROW 18</td>
<td>0D50H 0D51H</td>
<td>.0DFFH</td>
</tr>
<tr>
<td>ROW 18</td>
<td>0D50H 0D51H</td>
<td>.0DFFH</td>
<td>ROW 19</td>
<td>0D50H 0D51H</td>
<td>.0DFFH</td>
</tr>
<tr>
<td>ROW 19</td>
<td>0D50H 0D51H</td>
<td>.0DFFH</td>
<td>ROW 20</td>
<td>0D00H 0D01H</td>
<td>.0DFFH</td>
</tr>
<tr>
<td>ROW 20</td>
<td>0D00H 0D01H</td>
<td>.0DFFH</td>
<td>ROW 21</td>
<td>0E40H 0E41H</td>
<td>.0E8FH</td>
</tr>
<tr>
<td>ROW 21</td>
<td>0E40H 0E41H</td>
<td>.0E8FH</td>
<td>ROW 22</td>
<td>0E90H 0E91H</td>
<td>.0EDFH</td>
</tr>
<tr>
<td>ROW 22</td>
<td>0E90H 0E91H</td>
<td>.0EDFH</td>
<td>ROW 23</td>
<td>0F0H 0F01H</td>
<td>.0FFFH</td>
</tr>
<tr>
<td>ROW 23</td>
<td>0F0H 0F01H</td>
<td>.0FFFH</td>
<td>ROW 24</td>
<td>0F30H 0F31H</td>
<td>.0FFFH</td>
</tr>
<tr>
<td>ROW 24</td>
<td>0F30H 0F31H</td>
<td>.0FFFH</td>
<td>ROW 25</td>
<td>0F80H 0F81H</td>
<td>.0FCFH</td>
</tr>
<tr>
<td>ROW 25</td>
<td>0F80H 0F81H</td>
<td>.0FCFH</td>
<td>ROW 1</td>
<td>0800H 0801H</td>
<td>.084FH</td>
</tr>
<tr>
<td>ROW 1</td>
<td>0800H 0801H</td>
<td>.084FH</td>
<td>ROW 2</td>
<td>0850H 0851H</td>
<td>.089FH</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>After Initialization</th>
<th>After 1 Scroll</th>
<th>After 2 Scrolls</th>
<th>After 3 Scrolls</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW 3</td>
<td>08A0H 08A1H</td>
<td>.08EFH</td>
<td>ROW 4</td>
</tr>
<tr>
<td>ROW 4</td>
<td>08F0H 08F1H</td>
<td>.093FH</td>
<td>ROW 5</td>
</tr>
<tr>
<td>ROW 5</td>
<td>0940H 0941H</td>
<td>.098FH</td>
<td>ROW 6</td>
</tr>
<tr>
<td>ROW 6</td>
<td>0990H 0991H</td>
<td>.090FH</td>
<td>ROW 7</td>
</tr>
<tr>
<td>ROW 7</td>
<td>09E0H 09E1H</td>
<td>.0A2FH</td>
<td>ROW 8</td>
</tr>
<tr>
<td>ROW 8</td>
<td>0A30H 0A31H</td>
<td>.0A7FH</td>
<td>ROW 9</td>
</tr>
<tr>
<td>ROW 9</td>
<td>0A80H 0A81H</td>
<td>.0ACFH</td>
<td>ROW 10</td>
</tr>
<tr>
<td>ROW 10</td>
<td>0AD0H 0AD1H</td>
<td>.0B1FH</td>
<td>ROW 11</td>
</tr>
<tr>
<td>ROW 11</td>
<td>0B20H 0B21H</td>
<td>.0B6FH</td>
<td>ROW 12</td>
</tr>
<tr>
<td>ROW 12</td>
<td>0B70H 0B71H</td>
<td>.0BBFH</td>
<td>ROW 13</td>
</tr>
<tr>
<td>ROW 13</td>
<td>0B60H 0B61H</td>
<td>.0C0FH</td>
<td>ROW 14</td>
</tr>
<tr>
<td>ROW 14</td>
<td>0C60H 0C61H</td>
<td>.0C5FH</td>
<td>ROW 15</td>
</tr>
<tr>
<td>ROW 15</td>
<td>0C60H 0C61H</td>
<td>.0C5FH</td>
<td>ROW 16</td>
</tr>
<tr>
<td>ROW 16</td>
<td>0C60H 0C61H</td>
<td>.0C5FH</td>
<td>ROW 17</td>
</tr>
<tr>
<td>ROW 17</td>
<td>0D00H 0D01H</td>
<td>.0D4FH</td>
<td>ROW 18</td>
</tr>
<tr>
<td>ROW 18</td>
<td>0D50H 0D51H</td>
<td>.0DFFH</td>
<td>ROW 19</td>
</tr>
<tr>
<td>ROW 19</td>
<td>0D50H 0D51H</td>
<td>.0DFFH</td>
<td>ROW 20</td>
</tr>
<tr>
<td>ROW 20</td>
<td>0D00H 0D01H</td>
<td>.0DFFH</td>
<td>ROW 21</td>
</tr>
<tr>
<td>ROW 21</td>
<td>0E40H 0E41H</td>
<td>.0E8FH</td>
<td>ROW 22</td>
</tr>
<tr>
<td>ROW 22</td>
<td>0E90H 0E91H</td>
<td>.0EDFH</td>
<td>ROW 23</td>
</tr>
<tr>
<td>ROW 23</td>
<td>0F0H 0F01H</td>
<td>.0FFFH</td>
<td>ROW 24</td>
</tr>
<tr>
<td>ROW 24</td>
<td>0F30H 0F31H</td>
<td>.0FFFH</td>
<td>ROW 25</td>
</tr>
<tr>
<td>ROW 25</td>
<td>0F80H 0F81H</td>
<td>.0FCFH</td>
<td>ROW 1</td>
</tr>
<tr>
<td>ROW 1</td>
<td>0800H 0801H</td>
<td>.084FH</td>
<td>ROW 2</td>
</tr>
<tr>
<td>ROW 2</td>
<td>0850H 0851H</td>
<td>.089FH</td>
<td>ROW 3</td>
</tr>
</tbody>
</table>

---

**Figure 6-4. Screen Memory During Scrolling**
baud. Each POPDAT interrupt executes in 211 microseconds maximum. This means that each routine must execute in:

\[
1042 - 2 \times 211 = 620 \text{ microseconds}
\]

By adding up the times for any loop, it is clear that all routines meet this speed requirement, with the exception of ESC J. This means that if the terminal is operating at 9600 baud, at least one character time must be inserted after an ESC J sequence.

---

**Figure 6-5. Timing Flowchart**
Appendix 7.1

CRT TERMINAL SCHEMATICS
Appendix 7.2
KEYBOARD INTERFACE

The keyboard used in this design was a simple unencoded ASCII keyboard. In order to keep the cost to a minimum a simple scan matrix technique was implemented by using two ports of an 8255 parallel I/O device.

When the system is initialized the contents of the eight keyboard RAM locations are set to zero. Once every frame, which is 16.67 milliseconds the contents of the keyboard ram is read and then rewritten with the contents of the current switch matrix. If a non-zero value of one of the keyboard RAM locations is found to be the same as the corresponding current switch matrix, a valid key push is registered and action is taken. By operating the keyboard scan in this manner an automatic debounce time of 16.67 milliseconds is provided.

Figure 7.2A shows the actual physical layout of the keyboard and Figure 7.2B shows how the individual keys were encoded. On Figure 7.2B the scan lines are the numbers on the bottom of each key position and the return lines are the numbers at the top of each key position. The shift, control, and caps lock key were brought in through separate lines of port C of the 8255. Figure 7.3 shows the basic keyboard matrix.

In order to guarantee that two scan lines could not be shorted together if two or more keys are pushed simultaneously, isolation diodes could be added as shown in Figure 7.4.

---

**Figure 7-2A. Keyboard Layout**

**Figure 7-2B. Keyboard Encoding**

---

TOP NUMBER = RETURN LINE
BOTTOM NUMBER = SCAN LINE
### Appendix 7.3
#### ESCAPE/CONTROL/DISPLAY CHARACTER SUMMARY

<table>
<thead>
<tr>
<th>BIT</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>@</td>
<td>P</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>SOH</td>
<td>DC1</td>
<td>DC2</td>
<td>R</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>010</td>
<td>STX</td>
<td>DC3</td>
<td>T</td>
<td>3</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>011</td>
<td>ETX</td>
<td>DC4</td>
<td>D</td>
<td>4</td>
<td>T</td>
<td>D</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>010</td>
<td>EOT</td>
<td>U</td>
<td>E</td>
<td>5</td>
<td>F</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>ACK</td>
<td>V</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>BEL</td>
<td>^</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>G</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>100</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>H</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>101</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>I</td>
<td>Y</td>
</tr>
<tr>
<td>101</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>J</td>
<td>J</td>
<td>J</td>
<td>J</td>
<td>J</td>
</tr>
<tr>
<td>101</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>,</td>
<td>K</td>
<td>K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>SO</td>
<td>RS</td>
<td>^</td>
<td>&gt;</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>?</td>
<td>O</td>
<td>-</td>
<td>O</td>
<td>-</td>
</tr>
</tbody>
</table>

**NOTE**

- Shaded blocks: Functions terminal will react to. Others can be generated but are ignored upon receipt.
As stated earlier, all of the logic necessary to convert the 8275 into a non-DMA type of device was performed by a single small bipolar prom. Besides turning certain processor READS into DACKS and WRITES for the 8275, this 32 by 8 prom decoded addresses for the system ram, rom, as well as for the 8255 parallel I/O port.

Any bipolar prom that has a by eight configuration could function in this application. This particular device was chosen simply because it is the only “by eight” prom available in a 16 pin package. The connection of the prom is shown in detail in Figure 7.5 and its truth table is shown in Figure 7.6. Note that when a fetch cycle (M I) is not being performed, the state of the SOD line is the only thing that determines if memory reads will be written into the 8275’s row buffers. This is done by pulling both DACK and WRITE low on the 8275.

Also note that all of the outputs of the bipolar prom MUST BE PULLED HIGH by a resistor. This prevents any unwanted assertions when the prom is disabled.
As previously mentioned, the character generator used in this terminal is a 2716 or 2758 EPROM. A 1K by 8 device is sufficient since a 128 character 5 by 7 dot matrix only requires 8K of memory. Any “standard” or custom character generator could have been used.

The three low-order line count outputs (LC0-LC2) from the 8275 are connected to the three low-order address lines of the character generator and the seven character generator output lines (CC0-CC6) are connected to A3-A9 of the character generator. The output from the character generator is loaded into a shift register and the serial output from the shift register is the video output of the terminal.

Now, let’s assume that the letter “E” is to be displayed. The ASCII code for “E” is 45H. So, 45H is presented to the address lines A2-A9 of the character generator. The scan lines will now count each line from zero to seven to “form” the character as shown in Fig. 7.7. This same procedure is used to form all 128 possible characters.

It should be obvious that “custom” character fonts could be made just by changing the bit patterns in the character generator PROM. For reference, Appendix 7.6 contains a HEX dump of the character generator used in this terminal.

45H = 01000101
Address to Prom = 01000101 SL2 SL1 SL0 = 228H - 22FH

Depending on state of Scan lines.

Character generator output

<table>
<thead>
<tr>
<th>Rom Address</th>
<th>Rom Hex Output</th>
<th>Bit Output*</th>
</tr>
</thead>
<tbody>
<tr>
<td>228H</td>
<td>3E</td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>229H</td>
<td>02</td>
<td>.</td>
</tr>
<tr>
<td>22AH</td>
<td>02</td>
<td>.</td>
</tr>
<tr>
<td>22BH</td>
<td>0E</td>
<td>.</td>
</tr>
<tr>
<td>22CH</td>
<td>02</td>
<td>.</td>
</tr>
<tr>
<td>22DH</td>
<td>02</td>
<td>.</td>
</tr>
<tr>
<td>22EH</td>
<td>3E</td>
<td>.</td>
</tr>
<tr>
<td>22FH</td>
<td>00</td>
<td>.</td>
</tr>
</tbody>
</table>

Bits 0, 6 and 7 are not used.

* note bit output is backward from convention.
Appendix 7.7
COMPOSITE VIDEO

In this design, it was assumed that the monitor required a separate horizontal drive, vertical drive, and video input. However, many monitors require a composite video signal. The schematic shown in Figure 7.8 illustrates how to generate a composite video signal from the output of the 8275.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the vertical and horizontal timing is used to "center" the display. IC3 is used to mix the vertical and horizontal retrace and Q1 along with the R1, R2, and R3 mix the video and the retrace signal and provide the proper DC levels.

Figure 7-8. Composite Video

Appendix 7.8
SOFTWARE LISTINGS

ISIS-II 8080/8085 MACRO ASSEMBLER, X108

LOC OBJ SEQ SOURCE STATEMENT

;NO DMA 8275 SOFTWARE ALL I/O IS MEMORY MAPPED
;SYSTEM ROM 0000H TO 07FFH
;SYSTEM RAM 0000H TO 0FPPH
;8275 WRITE 1000H TO 13FFH
;8275 READ 1400H TO 17FFH
;8255 READ/WRITE 1800H TO 1FPP
;8255 ENABLED BY A14
;8251 ENABLED BY A15

1800 10 PORTA EQU 1800H
1801 11 PORTB EQU 1801H
1802 12 PORTC EQU 1802H
1803 13 CMD55 EQU 1803H
A000 14 USDF EQU 0000H
A001 15 CNTB EQU 0001H
A002 16 CNT1 EQU 0002H
A003 17 CNT2 EQU 0003H
A004 18 CNTM EQU 0004H
A005 19 CTRM EQU 0005H
A006 20 CTRS EQU 1401H
A007 21 CTRT EQU 0006H
A008 22 INT75 EQU 1401H
A009 23 TPDIS EQU 0008H
A00A 24 BADIS EQU 0009H
A00B 25 LAST EQU 000AH
A00C 26 CURB EQU 18H
A00D 27 LENGTH EQU 0050H
A00E 28 STPTR EQU 005AH
20 29 ;START PROGRAM
30 30 ;ALL VARIABLES ARE INITIALIZED BEFORE ANYTHING ELSE

0000 F3 32 DI
0001 31E00F 34 LXI SP,STPTR
0002 21E00F 35 LXI H,TPDIS
0003 22E00F 36 SHLD TOPAD
000A 22E00F 37 SHLD CURAD
000D 3000 38 MVI A,00H
000F 32E10F 39 STA CURSY
0010 32E20F 3A STA CURSX
0011 32E30F 3B STA KBCHR
0015 32E50F 41 STA USCHR
0018 32E70F 42 STA USCHR
001B 32E80F 43 STA KEYDN

7-70

207780-001
## APPLICATIONS

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>001E</td>
<td>32E0F</td>
<td>STA KEVOK; ZERO KEVOK</td>
</tr>
<tr>
<td>0021</td>
<td>32E0F</td>
<td>STA ESCP; ZERO ESCAPE</td>
</tr>
<tr>
<td>0024</td>
<td>C39800</td>
<td>JMP LPAD; JUMP AND SET EVERYTHING UP</td>
</tr>
<tr>
<td>002C</td>
<td>C36701</td>
<td>ONG 002CH</td>
</tr>
<tr>
<td>0030</td>
<td>E1</td>
<td>JMP FRAME</td>
</tr>
<tr>
<td>0034</td>
<td>F5</td>
<td>POPDAT: PUSH P5W; SAVE A AND FLAGS</td>
</tr>
<tr>
<td>0035</td>
<td>E5</td>
<td>PUSH H; SAVE H AND L</td>
</tr>
<tr>
<td>0036</td>
<td>D5</td>
<td>PUSH D; SAVE D AND E</td>
</tr>
<tr>
<td>0037</td>
<td>210000</td>
<td>LXI H, 0000H; ZERO H AND L</td>
</tr>
<tr>
<td>003A</td>
<td>39</td>
<td>DAD SP; PUT STACK POINTER IN H AND L</td>
</tr>
<tr>
<td>003B</td>
<td>BB</td>
<td>XCHG; PUT STACK IN D AND E</td>
</tr>
<tr>
<td>003B</td>
<td>2A0B0F</td>
<td>LATD; CURAD; GET POINTER</td>
</tr>
<tr>
<td>003C</td>
<td>FA</td>
<td>SPHI; PUT CURRENT LINE INTO SP</td>
</tr>
<tr>
<td>0040</td>
<td>EC0</td>
<td>WIFI A, C0H; SET MASK FOR SIM</td>
</tr>
<tr>
<td>0042</td>
<td>3B</td>
<td>SIM</td>
</tr>
<tr>
<td>0043</td>
<td>E1</td>
<td>REPT (LENTH/2)</td>
</tr>
<tr>
<td>0044</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0045</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0046</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0047</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0048</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0049</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>004A</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>004B</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>004C</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>004D</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>004E</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>004F</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0050</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0051</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0052</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0053</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0054</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0055</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0056</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0057</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0058</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0059</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>005A</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>005B</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>005C</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>005D</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>005E</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>005F</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0060</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0061</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0062</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0063</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0064</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0065</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0066</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0067</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0068</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>0069</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>006A</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>006B</td>
<td>E1</td>
<td>POP H</td>
</tr>
<tr>
<td>006C</td>
<td>30</td>
<td>SIM; SET UP A</td>
</tr>
<tr>
<td>006D</td>
<td>210000</td>
<td>LXI H, 0000H; ZERO H</td>
</tr>
<tr>
<td>006E</td>
<td>39</td>
<td>DAD SP; ADD STACK</td>
</tr>
<tr>
<td>0070</td>
<td>E2</td>
<td>XCHG; PUT STACK IN H AND L</td>
</tr>
<tr>
<td>0071</td>
<td>BB</td>
<td>SHIL; RESTORE STACK</td>
</tr>
<tr>
<td>0072</td>
<td>F9</td>
<td>LHI; LAST; PUT BOTTOM DISPLAY IN H AND L</td>
</tr>
<tr>
<td>0073</td>
<td>21000F</td>
<td>LXI H, LAST; PUT BOTTOM DISPLAY IN H AND L</td>
</tr>
<tr>
<td>0074</td>
<td>E3</td>
<td>XCHG; SWAP REGISTERS</td>
</tr>
<tr>
<td>0077</td>
<td>7A</td>
<td>MOV A, D; PUT HIGH ORDER IN A</td>
</tr>
<tr>
<td>0078</td>
<td>BC</td>
<td>CMP H; SEE IF SAME AS H</td>
</tr>
<tr>
<td>0079</td>
<td>28400</td>
<td>JNZ KPTK; IF NOT LEAVE</td>
</tr>
<tr>
<td>007A</td>
<td>78</td>
<td>MOV A, E; PUT LOW ORDER IN A</td>
</tr>
<tr>
<td>007B</td>
<td>BD</td>
<td>CMP L; SEE IF SAME AS L</td>
</tr>
<tr>
<td>007C</td>
<td>28400</td>
<td>JNZ KPTK; IF NOT LEAVE</td>
</tr>
<tr>
<td>0081</td>
<td>210000</td>
<td>LXI H, TPDIS; LOAD H AND L WITH TOP OF SCREEN MEMORY</td>
</tr>
<tr>
<td>0084</td>
<td>22E0F</td>
<td>SHLD CURAD; PUT BACK CURRENT ADDRESS</td>
</tr>
<tr>
<td>0087</td>
<td>361B</td>
<td>MVI A, 1BH; SET MASK</td>
</tr>
<tr>
<td>0089</td>
<td>30</td>
<td>SIM; OUTPUT MASK</td>
</tr>
</tbody>
</table>

7-71

20778-001
APPLICATIONS

008A D1 132 POP D ;GET D AND E
008B E1 133 POP H ;GET H AND L
008C F1 134 POP PSW ;GET A AND FLAGS
008D F9 135 EI ;TURN ON INTERRUPTS
008E C9 136 RET ;GO BACK

008F 3E18 137 THIS IS THE EXIT ROUTINE FOR THE FRAME INTERRUPT

0091 30 138
0092 C1 139
0093 D1 140
0094 E1 141
0095 F1 142
0096 FB 143
0097 C9 144

0098 32F0F 145 Bypass: MVI A,18H ;SET MASK
0099 32F0F 146 SIM ;OUTPUT THE MASK
009A C1 147 POP B ;GET B AND C
009B D1 148 POP D ;GED D AND E.
009C E1 149 POP H ;GET H AND L
009D F1 150 POP PSW ;GET A AND FLAGS
009E FB 151 EI ;ENABLE INTERRUPT
009F 3E18 152 RET ;GO BACK

00A0 7C 153
00A1 210008 154
00A2 3620 155
00A3 23 156
00A4 7C 157
00A5 88 158
00A6 C2A700 159
00A7 7D 160
00A8 09 161
00A9 C2A700 162

00A0 3E18 163 STAX ;PU'E IT
00A1 3E18 164 STAX
00A2 3E18 165 STAX
00A3 3E18 166 STAX
00A4 3E18 167 STAX
00A5 3E18 168 STAX
00A6 3E18 169 STAX
00A7 3E18 170 STAX
00A8 3E18 171 STAX
00A9 3E18 172 STAX
00AA 3E18 173 STAX
00AB 3E18 174 STAX
00AC 3E18 175 STAX
00AD 3E18 176 STAX
00AE 3E18 177 STAX
00AF 3E18 178 STAX
00B0 3E18 179 STAX
00B1 3E18 180 STAX
00B2 3E18 181 STAX
00B3 3E18 182 STAX
00B4 3E18 183 STAX
00B5 3E18 184 STAX

00B6 3F900 185 ;8255 INITIALIZATION

00B7 3D32 186
00B8 32E50 187
00B9 33232 188
00BA 32E50 189
00BB 32E50 190
00BC 32D00 191
00BD 32D00 192
00BE 32E50 193
00BF 32E50 194
00C0 32E50 195

00C1 32B18 196
00C2 32B18 197
00C3 32B18 198
00C4 32E50 199
00C5 32E50 200
00C6 32E50 201
00C7 32E50 202
00C8 32E50 203
00C9 32E50 204
00CC 32E50 205
00CD 32E50 206
00CE 32E50 207
00CF 32E50 208
00D0 32E50 209

00D1 32D00 210
00D2 32D00 211
00D3 32D00 212
00D4 32D00 213
00D5 32D00 214
00D6 32D00 215
00D7 32D00 216
00D8 C9 217

00D9 32E50 218
00DA 32E50 219

00DB 32E50 220
00DC 32E50 221
00DD 32E50 222
00DE 32E50 223
00DF 32E50 224
00E0 32E50 225
00E1 32E50 226
00E2 32E50 227
00E3 32E50 228
00E4 32E50 229
00E5 32E50 230
00E6 32E50 231
00E7 32E50 232
00E8 32E50 233
00E9 32E50 234
00EA 32E50 235
00EB 32E50 236
00EC 32E50 237
00ED 32E50 238
00EE 32E50 239
00EF 32E50 240
00F0 32E50 241
00F1 32E50 242
00F2 32E50 243
00F3 32E50 244
00F4 32E50 245
00F5 32E50 246
00F6 32E50 247
00F7 32E50 248
00F8 C9 249
00F9 32E50 250
00FA 32E50 251
00FB 32E50 252
00FC 32E50 253
00FD 32E50 254
00FE 32E50 255
00FF 32E50 256

7-72
207780-001
APPLICATIONS

219 ;8275 INITIALIZATION

009 210108
008C 3600
006F 26
00FF 364F
0101 3658
0103 3660
0105 365D
0107 23
0108 DB9803
010B 3624
010D 3623

221 IN75: LXI H,CRTS

222 MVI M,00H ;RESET AND STOP DISPLAY

223 DL H

224 MVI M,4FH ;SCREEN PARAMETER BYTE 1

225 MVI M,58H ;SCREEN PARAMETER BYTE 2

226 MVI M,8DH ;SCREEN PARAMETER BYTE 3

227 MVI M,9DDH ;SCREEN PARAMETER BYTE 4

228 INX H ;HL=1001H

229 CALL LDCUR ;LOAD THE CURSOR

230 MVI M,02H ;RESET COUNTER

231 MVI M,23H ;START DISPLAY

232 ;THIS ROUTINE READS BOTH THE KEYBOARD AND THE USART

233 ;AND TAKES PROPER ACTION DEPENDING ON HOW THE LINE-LOCAL

234 ;SWITCH IS SET

235

236

010F 3E18
237 SETUP: MVI A,18H ;SET MASK

238 SIM ;LOAD MASK

239 EI ;ENABLE INTERRUPTS

240

241 ;READ THE USART

242

0113 20
243 RXRDY: AIM ;GET LINE LOCAL

0114 E680
244 ANI 80H ;IS IT ON OR OFF?

0116 C22101
245 DJNZ KEYPIN ;LEAVE IF IT IS ON

0119 3A01A0
246 LDA USTF ;READ 8251 FLAGS

011C E682
247 ANI 02H ;LOOK AT RXRDY

011E C25C01
248 JNZ OK7 ;IF HAVE CHARACTER GO TO WORK

0121 3A030F
249 KEYINP: MVI A,KEYDN ;GET KEYBOARD CHARACTER

0124 E680
250 ANI 80H ;IS IT THERE

0126 C23101
251 JNZ KEYS ;IF KEY IS PRESSED LEAVE

0129 3E2D0F
252 MVI A,0AH ;ZERO A

012B 3E2D0F
253 STA KEYOK ;CLEAR KEYOK

012C C31301
254 JMP RXRDY ;LOOP AGAIN

0131 3E2D0F
255 KEYS: LDA KEYOK ;WAS KEY DOWN

0134 4F
256 MOV C,A ;SAVE A IN C

0135 3E2B0F
257 LDA KBCHR ;GET KEYBOARD CHARACTER

0136 1F 89
258 CMP C ;IS IT THE SAME AS KEYOK

0139 C21301
259 JZ RXRDY ;IF SAME LOOP AGAIN

013C 3E2D0F
260 STA KEYOK ;IF NOT SAVE IT

013E 3E2D0F
261 STA USCHR ;SAVE IT

0142 20
262 RIM ;GET LINE LOCAL

0143 E680
263 ANI 80H ;WHICH WAY

0145 CA4B01
264 JZ TRANS ;LEAVE IF LINES

0148 C34E02
265 JMP CBRC2 ;TIME TO DO SOME WORK

014B 3A01A0
266 TRANS: LDA USTF ;GET USART FLAGS

014E E681
267 ANI 01H ;READY TO TRANSMIT?

0150 CA4B01
268 JZ TRANS ;LOOP IF NOT READY

0153 3A070F
269 LDA USCHR ;GET CHARACTER

0156 3E2B0A
270 STA USTD ;PUT IN USART

0159 C3EF01
271 JMP SETUP ;LEAVE

015C 3A0A0A
272 OK7: LDA USTD ;READ USART

015F B77F
273 ANI 07H ;STRIP MSB

0161 3E270F
274 STA USCHR ;PUT IT IN MEMORY

0164 C34E02
275 JMP CBRC2 ;LEAVE

276 ;THIS ROUTINE CHECKS THE BAUD RATE SWITCHES, RESETS THE

277 ;SCREEN POINTERS AND READS AND LOOKS UP THE KEYBOARD.

278 ;SCREEN POINTERS AND READS AND LOOKS UP THE KEYBOARD.

279 ;

0167 F5
280 FRAME: PUSH PSW ;SAVE A AND FLAGS

0168 E5
281 PUSH H ;SAVE H AND L

0169 D5
282 PUSH D ;SAVE D AND E

016A C5
283 PUSH B ;SAVE B AND C

016B 3A1114
284 LDA INT75 ;READ 8275 TO CLEAR INTERRUPT

285 I ;SET UP THE POINTERS

287

016E 2AE39F
288 LHLH TOPAD ;LOAD TOP IN H AND L

0171 22608F
289 SHLD CURAD ;STORE TOP IN CURRENT ADDRESS

290 ;SET UP BAUD RATE

291

0174 3A9218
292 LDA PORTC ;READ BAUD RATE SWITCHES

0177 E68F
293 ANI 08H ;STRIP OFF 4 MSB'S

0179 47
294 MOV B,A ;SAVE IN B

017D 3AC30F
295 LDA BAUD ;GET BAUD RATE

017A 88
296 CMP 3 ;SEE IF SAME AS A

017E 4CD6C0
297 CNZ STRBAUD ;IF NOT SAME DO SOMETHING

298 ;READ KEYBOARD

299

300

0181 3AE08F
301 LDA KEYDOWN ;SEE IF A KEY IS DOWN

0184 3E40
302 ANI 4H ;GET THE FLAGS

0186 C22901
303 JNZ KEYDOWN ;IF KEY IS DOWN JUMP AROUND

0189 CDBF01
305 CALL RXRB ;GO READ THE KEYBOARD

018C C38F00
306 JMP BYPASS ;LEAVE
APPLICABLE

ROUTINE ASSUMES THAT THE CHARACTER IS LOWER CASE ASCII AND SUBTRACTS 20H, WHICH CONVERTS THE CHARACTER TO

UPPER CASE ASCII

; THE ROUTINES SHOWN AND CNTDOWN SET BIT 6 AND 7 RESPECTIVELY

; IN THE ACC.

; THE ROUTINE CHECKS FOR ESCAPE CHARACTERS, LF, CR, FF, AND BACK SPACE

; THE CHARACTERS FOLLOWING AN ESCAPE, THE COMMANDS ARE

; COMPATIBLE WITH INTEL'S CREDIT TEXT EDITOR

; THIS ROUTINE MOVES THE CURSOR DOWN ONE CHARACTER LINE

; THIS ROUTINE MOVES THE CURSOR DOWN ONE CHARACTER LINE
APPLICATIONS

029F CDE403 483 ; THIS ROUTINE CLEARS THE SCREEN.
029F CDE403 484 CLEAR: CALL CISCR ; GO CLEAR THE SCREEN
029F CDE403 485 JMP SETUP ; GO BACK
029F CDE403 486 ; THIS ROUTINE CLEARS ALL LINES BELOW THE LOCATION
029F CDE403 487 OF THE CURSOR.
029F CDE403 488
029F CDE403 489 CURST: CALL CALCUL ; CALCULATE ADDRESS
029F CDE403 490 CALL ADX ; ADD X POSITION
029F CDE403 491 LXI B, 4F20H ; PUT SPACE AND LAST X IN B AND C
029F CDE403 492 LDA CURSX ; GET X CURSOR
029F CDE403 493 CMP B ; SEE IF AT END OF LINE
029F CDE403 494 JMP OVR1 ; LEAVE IF X IS AT END OF LINE
029F CDE403 495 JZ OVR1 ; LEAVE IF X IS AT END OF LINE
029F CDE403 496 JNZ OVR1 ; LEAVE IF X IS AT END OF LINE
029F CDE403 497 LLP: INR A ; MOVE A OVER ONE X POSITION
029F CDE403 498 INX H ; INCREMENT MEMORY POINTER
029F CDE403 499 MOV A, C ; PUT A SPACE IN MEMORY
029F CDE403 500 CMP B ; SEE IF A = 4FH
029F CDE403 501 JMP SImJP ; IF NOT LOOP AGAIN
029F CDE403 502 OVR1: LXI B, LAST ; PUT LAST LINE IN 3C
029F CDE403 503 INX H ; POINT HL TO LAST LINE
029F CDE403 504 MOV A, B ; GET B
029F CDE403 505 CMP H ; SAME AS H?
029F CDE403 506 JNZ CONCL ; LEAVE IF NOT
029F CDE403 507 MOV A, C ; GET C
029F CDE403 508 CMP L ; SAME AS L?
029F CDE403 509 JNZ CONCL ; LEAVE IF NOT
029F CDE403 510 LXI H, TEDIS ; GET TOP OF DISPLAY
029F CDE403 511 CONCL: LXI HCYSX ; GET Y CURSOR
029F CDE403 512 CPI CURBOT ; IS IT ON THE BOTTOM
029F CDE403 513 JZ SETUP ; LEAVE IF IT IS
029F CDE403 514 MOV A, B ; MOVE IT DOWN ONE LINE
029F CDE403 515 MOV B, A ; MOVE CURSOR IN B FOR LATER
029F CDE403 516 LXI D, LENGTH ; PUT LENGTH OF ONE LINE IN D
029F CDE403 517 CLOOP: MVI M, 0FH ; PUT FOR MEMORY
029F CDE403 518 MOV A, B ; GET CURSOR Y
029F CDE403 519 CPI CURBOT ; ARE WE ON THE BOTTOM
029F CDE403 520 JZ SETUP ; LEAVE IF WE ARE
029F CDE403 521 MOV A, C ; MORE CURSOR DOWN ONE
029F CDE403 522 DAD D ; GET NEXT LINE
029F CDE403 523 MOV B, A ; SAVE A
029F CDE403 524 CPI 0FH ; PUT H IN A
029F CDE403 525 CPI OPH ; COMPAIR TO HIGH LAST
029F CDE403 526 JNZ CLOOP ; LEAVE IF IT IS NOT
029F CDE403 527 MOV A, L ; PUT L IN A
029F CDE403 528 MOV A, H ; MOVE CURSOR UP
029F CDE403 529 CPI OPH ; COMPAIR TO LOW LAST
029F CDE403 530 JNZ CLOOP ; LEAVE IF IT IS NOT
029F CDE403 531 JMP CLOOP ; LOOP AGAIN
029F CDE403 532
029F CDE403 533 CURLIN: CALL CALCUL ; CALCULATE ADDRESS
029F CDE403 534 SHLD LOCBO ; STORE H AND L TO CLEAR LINE
029F CDE403 535 CALL CLINE ; CLEAR THE LINE
029F CDE403 536 JMP SETUP ; GO BACK
029F CDE403 537
029F CDE403 538 ; THIS ROUTINE MOVES THE CURSOR UP ONE LINE.
029F CDE403 539 3AE10F 540 UPCUR: LDA CURSY ; GET Y CURSOR
029F CDE403 541 CPI 0FH ; IS IT ZERO
029F CDE403 542 JZ SETUP ; IF IT IS LEAVE
029F CDE403 543 JNZ SETUP ; IF IT IS LEAVE
029F CDE403 544 DCR A ; MOVE CURSOR UP
029F CDE403 545 STA CURSY ; SAVE NEW CURSOR
029F CDE403 546 CALL LDCUR ; LOAD THE CURSOR
029F CDE403 547 JMP SETUP ; LEAVE
029F CDE403 548
029F CDE403 549 ; THIS ROUTINE MOVES THE CURSOR ONE LOCATION TO THE RIGHT
029F CDE403 550 3AE20F 551 3AE20F 552 RIGHT: LDA CURSX ; GET X CURSOR
029F CDE403 553 CPI 4FH ; IS IT ALL THE WAY OVER?
029F CDE403 554 JNZ NTOVER ; IF NOT JUMP AROUND
029F CDE403 555 LDA CURSX ; GET Y CURSOR
029F CDE403 556 CPI CURBOT ; SEE IF ON BOTTOM
029F CDE403 557 JZ GD18 ; IF WE ARE JUMP
029F CDE403 558 INC A ; INCREMENT Y CURSOR
029F CDE403 559 STA CURSY ; SAVE IT
029F CDE403 560 STA CURSX ; SAVE IT
029F CDE403 561 GD18: MVI A, 08H ; ZERO A
029F CDE403 562 MOV A, 08H ; ZERO X CURSOR
029F CDE403 563 LDCUR ; LOAD THE CURSOR
029F CDE403 564 JMP SETUP ; LEAVE
029F CDE403 565 INC X CURSOR ; INCREMENT X CURSOR
029F CDE403 566 STA CURSX ; SAVE IT
029F CDE403 567 STA CURSY ; SAVE IT
029F CDE403 568 CALL LDCUR ; LOAD THE CURSOR
029F CDE403 569 JMP SETUP ; LEAVE
029F CDE403 570
029F CDE403 571 ; THIS ROUTINE MOVES THE CURSOR LEFT ONE CHARACTER POSITION

7-76
APPLICATIONS

036E 3AE20F 576  LEFT:  LOA  CURSX  ;Get X Cursor
036F 3AE20F 577  CPI  00H  ;Is it all the way over
0370 CED03 578  JNZ  NOVER  ;If not jump around
0373 3AE10F 579  LOA  CURSY  ;Get Cursor Y
0376 FED00 580  CPI  00H  ;Is it zero?
0379 FE90 581  DJNZ  A  ;If it is jump
037B 3AE01F 582  STA  CURSY  ;Move Cursor Y up
037E 3210F 583  STA  CURSX  ;Save it
0380 C6B83 584  CALL  LDCUR  ;Load the Cursor
0383 3C0F1F 585  STA  CURSX  ;Save it
0384 32E20F 586  STA  CURSX  ;Save Cursor X
0387 CDB83 587  CALL  LDCUR  ;Load the Cursor
038A C30F1F 588  JMP  SETUP  ;Leave
038C 3D0 589 ;This routine homes the Cursor.
038E 3E00 590  HOME:  MVX  A,00H  ;Zero A
0390 3AE20F 591  STA  CURSX  ;Zero X Cursor
0393 32E10F 592  STA  CURSX  ;Zero Y Cursor
0396 3EB83 593  CALL  LDCUR  ;Load the Cursor
0399 C30F1F 594  JMP  SETUP  ;Leave
039B 32D0 595 ;This routine sets the escape bit
039D 3E08 596  ESKAP:  MVX  A,00H  ;Load a with escape bit
03A0 32E00F 597  STA  ESCP  ;Set escape location
03A3 C30F1F 598  JMP  SETUP  ;Go back and read uart
03A6 3E00 599 ;This routine does a cr
03A8 3E08 600  CCR:  MVX  A,00H  ;Zero A
03AB 32E20F 601  STA  CURSX  ;Zero X Cursor
03AD 32E10F 602  STA  CURSX  ;Zero Y Cursor
03AF 3EB83 603  CALL  LDCUR  ;Load Cursor into 8275
03B2 C30F1F 604  JMP  SETUP  ;Poll UART again
03B5 3E0B 605 ;This routine loads the cursor
03B7 3E08 606  LDCUR:  MVX  A,00H  ;Put 80H into A
03BA 32D100 607  STA  CRTS  ;Load Cursor into 8275
03BD 3AE20F 608  LDA  CURSX  ;Get Cursor X
03C0 32D001 609  STA  CRTM  ;Put it in 8275
03C3 3AE10F 60A  STA  CURSX  ;Zero Cursor Y
03C6 32D001 60B  STA  CRTM  ;Put it in 8275
03C9 C9 60C  RET  ;Zero Cursor
03C9 39 60D ;This routine does a form feed
03CA CDE403 60E  PFMD:  CALL  CLSCR  ;Call clear Screen
03CD 21D000 60F  LAXI  H,TDIS  ;Put top Display in HL
03DF 23E50F 610  SHLD  LDC05  ;Put it in LDC05
03DE CD1004 611  CALL  CLLINE  ;Clear Top Line
03DF 3E08 612  MVX  A,00H  ;Zero A
03E0 32E20F 613  STA  CURSX  ;Zero X Cursor
03E3 32E10F 614  STA  CURSX  ;Zero Y Cursor
03E6 3EB83 615  CALL  LDCUR  ;Load the Cursor
03E9 C30F1F 616  JMP  SETUP  ;Back to UART
03EE 3E0B 617 ;This routine clears the screen by writing end of row
03F0 3E08 618  LDCUR:  MVX  A,00H  ;Put 80H into A
03F3 43E10F 619  MVI  B,CURBOT  ;Load B with MAX Y
03F6 0E18 61A  MVI  A,0FH  ;Max Y plus one
03F9 21D000 61B  LAXI  H,TDIS  ;Load h and L with top of ram
03FEB 11D000 61C  LAXI  D,LNGTH  ;Move 5AH = 80D into D and E
03FF 77 61D  MOVE  A,A  ;Move or into memory
0400 19 61E  CHG  D  ;Change pointer by 80D
0401 05 61F  DCR  B  ;Count the loops
0402 C2E003 620  JC  LOADX  ;Continue if not zero
0405 C9 621  REI  ;Go back
0407 3AF 622 ;This routine does a line feed
040F C6C003 623  LFND:  CALL  LFND1  ;Call routine
040F 3C0F1F 624  JMP  SETUP  ;Poll flags
0412 51 625 ;Line feed
0415 46 626 ;This routine does a line feed
0418 C6C003 627 ;Call LFND1
041F 3C0F1F 628 ;Call routine
0422 51 629 ;Poll Flags
0425 52 62A ;Line Feed
042E 3AE10F 62B  LFND1:  LDA  CURSX  ;Get Y Location of Cursor
0431 FC018 62C  CPI  CURBOT  ;See if at bottom of screen
0434 C5304 62D  JZ  ONBOT  ;If we are, leave
0437 3C 62E  INR  A  ;Increment Y
043A 3E10F 62F  STA  CURSX  ;Save new Cursor

7-77
APPLICATIONS

0408 CD504 658 CALL CALCU ;CALCULATE ADDRESS
0409 22E50F 659 SHLD LOCS0 ;SAVE TO CLEAR LINE
040C CD1504 660 CALL CCLINE ;CLEAR THE LINE
0411 CD8353 661 CALL LCUR ;LOAD THE CURSOR
0414 C9 662 RET ;LEAVE

;THIS ROUTINE CLEARS THE LINE WHOSE FIRST ADDRESS
;IS IN LOCS0. PUSH INSTRUCTIONS ARE USED TO RAPIDLY
CLEAR THE LINE

0415 E3 663 CALL CLINE ;NO INTERRUPTS HERE
0416 2AE50F 664 SHLD LOCS0 ;GET LOC0
0419 115000 665 LXI D,LENGTH ;GET OFFSET
041C 19 666 DAD D ;ADD OFFSET
041E 08 667 XCHG ;PUT START IN DE
0421 39 668 LXI H,LOC80 ;ZERO HL
0422 74 669 DAD ;GET WC80
0423 F9 670 LXI B,H ;GET OFFSET
0425 00 671 DAD ;ADD OFFSET
0427 E5 672 XCHG ;PUT START IN DE
0429 EB 673 LXI H,LOC80 ;LOAD SP IN HL
042B F9 674 LXI D,LENGTH ;GET STACK
042D EB 675 XCHG ;PUT STACK IN DE
042F EB 676 LXI H,LOC80 ;PUT SPACES IN HL
0431 E5 677 LXI H,LOC80 ;NON 00 PUSH INSTRUCTIONS
0433 E5 678 TO CLEAR THE LINE
0435 E5 679 PUSH (LENGTH/2)
0437 E5 680 REPT
0439 E5 681 PUSH H
043B E5 682 ENDM
043D E5 683

0427 E5 684 PUSH H
0429 E5 685 PUSH H
042B E5 686 PUSH H
042D E5 687 PUSH H
042F E5 688 PUSH H
0431 E5 689 PUSH H
0433 E5 690 PUSH H
0435 E5 691 PUSH H
0437 E5 692 PUSH H
0439 E5 693 PUSH H
043B E5 694 PUSH H
043D E5 695 PUSH H
043F E5 696 PUSH H
0441 E5 697 PUSH H
0443 E5 698 PUSH H
0445 E5 699 PUSH H
0447 E5 700 PUSH H
0449 E5 701 PUSH H
044B E5 702 PUSH H
044D E5 703 PUSH H
044F E5 704 PUSH H
0451 E5 705 PUSH H
0453 E5 706 PUSH H
0455 E5 707 PUSH H
0457 E5 708 PUSH H
0459 E5 709 PUSH H
045B E5 710 PUSH H
045D E5 711 PUSH H
045F E5 712 PUSH H
0451 E5 713 PUSH H
0453 E5 714 PUSH H
0455 E5 715 PUSH H
0457 E5 716 PUSH H
0459 E5 717 PUSH H
045B E5 718 PUSH H
045D E5 719 PUSH H
045F E5 720 PUSH H
0451 E5 721 PUSH H
0453 E5 722 PUSH H
0455 E5 723 PUSH H
0457 E5 724 XCHG ;PUT STACK IN HL
0459 E5 725 SHL ;PUT SPACE IN SP
045B E5 726 EI ;ENABLE INTERRUPTS
045D E5 727 RET ;GO BACK
0451 E5 728

;IF CURSOR IS ON THE BOTTOM OF THE SCREEN THIS ROUTINE
;IS USED TO IMPLEMENT THE LINE FEED

0453 2AE30F 729 ONBOT: SHLD TOPAD ;GET TOP ADRESS
0455 22E50F 730 SHLD LOCS0 ;SAVE IT IN LOCS0
0459 115000 731 LXI D,LENGTH ;LINE LENGTH
045C 19 732 DAD D ;ADD HL + DE
045E 01D80F 733 LXI B, LAST ;GET BOTTOM LINE
0460 7C 734 MOV A,H ;GET H
0462 B8 735 CMP B ;SAME AS B
0464 C6 736 MOV A,H ;GET H
0466 B9 737 CMP C ;SAME AS C
0468 C26D04 738 JNZ ARND ;IF NOT SAME
046A 7D 739 MOV A,L ;GET L
046C B9 740 CMP C ;SAME AS C
046E C26D04 741 JNZ ARND ;IF NOT SAME
0470 210000 742 LXI H,TDIS ;LOAD HL WITH TOP OF DISPLAY
0472 22E50F 743 SHLD TOPAD ;SAVE NEW TOP ADDRESS

7-78
APPLICATIONS

| 0478 CD1504 | 745 | CALL | CLLINE  ; CLEAR LINE |
| 0473 CD8803 | 746 | CALL | LOCUR  ; LOAD THE CURSOR |
| 0476 C9    | 747 | RET  |
| 0478 CD1504 | 748 | ; THIS ROUTINE PUTS A CHARACTER ON THE SCREEN AND |
| 0478 CD1504 | 749 | INCREMENTS THE X CURSOR POSITION. A LINE FEED IS |
| 0478 CD1504 | 750 | INSERTED IF THE INCREMENTED CURSOR EQUALS 80D |
| 0477 CD1504 | 751 | CALL | CAU  ; CALCULATE SCREEN POSITION |
| 0477 CD1504 | 752 | MOV  A,M  ; GET FIRST CHARACTER |
| 0477 CD1504 | 753 | CPI  080H  ; IS IT A CLEAR LINE |
| 0477 CD1504 | 754 | SHLD  LOC80  ; SAVE LINE TO CLEAR |
| 0477 CD1504 | 755 | CLI  ; CLEAR LINE |
| 0478 CD1504 | 756 | LHLD  LOC80  ; GET LINE |
| 0478 CD1504 | 757 | CALL | ADX  ; ADD CURSOR X |
| 0478 CD1504 | 758 | LDA  CUR8R  ; ADD CHARACTER |
| 0478 CD1504 | 759 | MOV  M,A  ; PUT IT ON SCREEN |
| 0478 CD1504 | 760 | ADA  ; GET CURSOR X |
| 0478 CD1504 | 761 | RLC  ; INCREMENT CURSOR X |
| 0478 CD1504 | 762 | JMP  OK1  ; HAS IT GONE TOO FAR? |
| 0478 CD1504 | 763 | JMP  OK1:  ; IF NOT GOOD |
| 0478 CD1504 | 764 | JMP  CUR8R  ; LOAD THE CURSOR |
| 0478 CD1504 | 765 | JMP  ; LEAVE |

04A5 21D504
CALCUL:  LXI  H,LINNTAB  ; GET LINE TABLE INTO H AND L
04A8 3EA10F
LDA  CUR8R  ; GET CURSOR INTO A
04A8 07
RCL  B  ; SET UP FOR LOOKUP TABLE
04A8 9309
MOV  B,08H  ; ZERO B
04A8 4F
MOV  C,A  ; PUT CURSOR INTO A
04A9 09
DAD  B  ; ADD LINE TABLE TO Y CURSOR
04A9 7E
MOV  B,09H  ; ZERO B
04A9 4E
MOV  C,A  ; PUT CURSOR INTO A
04A9 23
INC  H  ; CHANGE MEMORY POINTER
04A9 7E
MOV  A,M  ; PUT HIGH LINE TABLE INTO C
04A9 47
MOV  B,0A  ; HIGH IN D
04B0 2100F8
LXI  H,0FP08H  ; TWS COMPLEMENT SCREEN LOCATION
04B0 89
DAD  B  ; SUBTRACT OFFSET
04B0 83
XCHG  ; SAVE HIGH IN C
04BB 2A030F
LXI  H,0FP30H  ; TWS COMPLEMENT SCREEN LOCATION
04B1 19
DAD  D  ; SEE IF WE ARE OFF THE SCREEN
04B2 9C804
JC  FIX  ; IF WE ARE FIX IT
04B3 88
XCHG  ; GET DISPLACED ADDRESS BACK
04B4 C7
RET  ; GO BACK
04B5 2130F8
FIX:  LXI  H,0F830H  ; SCREEN BOUNDARY
04B5 19
DAD  D  ; ADJUST SCREEN
04B6 81
RET  ; GO BACK
04B6 83
; THIS ROUTINE ADDS THE X CURSOR LOCATION TO THE ADDRESS
04B7 89
DAD  D  ; THAT IS IN THE H AND L REGISTERS AND RETURNS THE RESULT
04B8 85
; IN H AND L
04B9 86
; THIS TABLE CONTAINS THE OFFSET ADDRESSES FOR EACH
04B9 81
OF THE 25 DISPLAYED LINES.
0000 810
LINTAB:  LNNUM SET 0
0017 817
REPT  (CURBOT+1)
0018 818
DW  TPDIS+(LENGTH*LNNUM)
0019 819
LNNUM SET (LNNUM+1)
0020 828
ENDM
00D5 0008 821
DW  TPDIS+(LENGTH*LNNUM)
00D6 0012 822
LNNUM SET (LNNUM+1)
00D7 5008 823
LNNUM SET (LNNUM+1)
00D8 0024 824
LNNUM SET (LNNUM+1)
00D9 0035 825
LNNUM SET (LNNUM+1)
00DA 0048 826
LNNUM SET (LNNUM+1)
00DB 0059 827
LNNUM SET (LNNUM+1)
00DC 0070 828
LNNUM SET (LNNUM+1)
00DE 0081 829
LNNUM SET (LNNUM+1)
00DF 0093 830
LNNUM SET (LNNUM+1)
APPLICATIONS

<table>
<thead>
<tr>
<th>Address</th>
<th>ASCII</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0006 832</td>
<td>011111</td>
<td>LINNUM SET (LINNUM+1)</td>
</tr>
<tr>
<td>0010 833</td>
<td>011111</td>
<td>DW TPDIS+(LENGTH*LINNUM)</td>
</tr>
<tr>
<td>0012 834</td>
<td>011111</td>
<td>LINNUM SET (LINNUM+1)</td>
</tr>
<tr>
<td>0014 835</td>
<td>011111</td>
<td>DW TPDIS+(LENGTH*LINNUM)</td>
</tr>
<tr>
<td>0016 836</td>
<td>011111</td>
<td>LINNUM SET (LINNUM+1)</td>
</tr>
<tr>
<td>0018 837</td>
<td>01111</td>
<td>DW TPDIS+(LENGTH*LINNUM)</td>
</tr>
<tr>
<td>001A 838</td>
<td>01111</td>
<td>LINNUM SET (LINNUM+1)</td>
</tr>
<tr>
<td>001C 839</td>
<td>01111</td>
<td>DW TPDIS+(LENGTH*LINNUM)</td>
</tr>
<tr>
<td>001E 83A</td>
<td>01111</td>
<td>LINNUM SET (LINNUM+1)</td>
</tr>
<tr>
<td>0020 83B</td>
<td>01111</td>
<td>DW TPDIS+(LENGTH*LINNUM)</td>
</tr>
<tr>
<td>0022 83C</td>
<td>01111</td>
<td>LINNUM SET (LINNUM+1)</td>
</tr>
<tr>
<td>0024 83D</td>
<td>01111</td>
<td>DW TPDIS+(LENGTH*LINNUM)</td>
</tr>
<tr>
<td>0026 83E</td>
<td>01111</td>
<td>LINNUM SET (LINNUM+1)</td>
</tr>
<tr>
<td>0028 83F</td>
<td>01111</td>
<td>DW TPDIS+(LENGTH*LINNUM)</td>
</tr>
</tbody>
</table>

; KEYBOARD LOOKUP TABLE
; THIS TABLE CONTAINS ALL THE ASCII CHARACTERS
; THAT ARE TRANSMITTED BY THE TERMINAL
; THE CHARACTERS ARE ORGANIZED SO THAT BITS 0, 1 AND 2
; ARE THE SCAN LINES, BITS 3, 4 AND 5 ARE THE RETURN LINES
; BIT 6 IS SHIFT AND BIT 7 IS CONTROL

<table>
<thead>
<tr>
<th>Address</th>
<th>ASCII</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0507 38</td>
<td>888</td>
<td>DB 30H,39H</td>
</tr>
<tr>
<td>0509 39</td>
<td>889</td>
<td>DB 30H,20H</td>
</tr>
<tr>
<td>050A 3A</td>
<td>890</td>
<td>DB 30H,5CH</td>
</tr>
<tr>
<td>050B 3B</td>
<td>891</td>
<td>DB 30H,08H</td>
</tr>
<tr>
<td>050C 3C</td>
<td>892</td>
<td>DB 30H,00H</td>
</tr>
<tr>
<td>050D 3D</td>
<td>893</td>
<td>DB 75H,69H</td>
</tr>
<tr>
<td>050E 3E</td>
<td>894</td>
<td>DB 6FH,78H</td>
</tr>
<tr>
<td>050F 3F</td>
<td>895</td>
<td>DB 58H,5CH</td>
</tr>
<tr>
<td>0510 40</td>
<td>896</td>
<td>DB 0AH,7FH</td>
</tr>
<tr>
<td>0511 41</td>
<td>897</td>
<td>DB 6AH,68H</td>
</tr>
<tr>
<td>0512 42</td>
<td>898</td>
<td>DB 6CH,38H</td>
</tr>
<tr>
<td>0513 43</td>
<td>899</td>
<td>DB 27H,08H</td>
</tr>
<tr>
<td>0514 44</td>
<td>900</td>
<td>DB 0OH,37H</td>
</tr>
<tr>
<td>0515 45</td>
<td>901</td>
<td>DB 60H,2CH</td>
</tr>
<tr>
<td>0516 46</td>
<td>902</td>
<td>DB 2EH,2FH</td>
</tr>
<tr>
<td>0517 47</td>
<td>903</td>
<td>DB 00H,08H</td>
</tr>
<tr>
<td>0518 48</td>
<td>904</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0519 49</td>
<td>905</td>
<td>DB 00H,61H</td>
</tr>
<tr>
<td>051A 4A</td>
<td>906</td>
<td>DB 7AH,78H</td>
</tr>
<tr>
<td>051B 4B</td>
<td>907</td>
<td>DB 63H,76H</td>
</tr>
<tr>
<td>051C 4C</td>
<td>908</td>
<td>DB 52H,6EH</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>VALUE</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>0528-08</td>
<td>899</td>
<td>DB 79H,09H</td>
</tr>
<tr>
<td>0529-08</td>
<td>900</td>
<td>DB 09H,2AH</td>
</tr>
<tr>
<td>0531-08</td>
<td>901</td>
<td>DB 6AH,55H</td>
</tr>
<tr>
<td>0532-08</td>
<td>902</td>
<td>DB 57H,68H</td>
</tr>
<tr>
<td>0533-08</td>
<td>903</td>
<td>DB 08H,71H</td>
</tr>
<tr>
<td>0534-08</td>
<td>904</td>
<td>DB 77H,73H</td>
</tr>
<tr>
<td>0535-08</td>
<td>905</td>
<td>DB 65H,72H</td>
</tr>
<tr>
<td>0536-08</td>
<td>906</td>
<td>DB 74H,09H</td>
</tr>
<tr>
<td>0537-08</td>
<td>907</td>
<td>DB 18H,31H</td>
</tr>
<tr>
<td>0538-08</td>
<td>908</td>
<td>DB 32H,33H</td>
</tr>
<tr>
<td>0539-08</td>
<td>909</td>
<td>DB 34H,35H</td>
</tr>
<tr>
<td>053A-08</td>
<td>910</td>
<td>DB 36H,00H</td>
</tr>
<tr>
<td>053B-08</td>
<td>911</td>
<td>DB 2AH,28H</td>
</tr>
<tr>
<td>053C-08</td>
<td>912</td>
<td>DB 29H,5FH</td>
</tr>
<tr>
<td>053D-08</td>
<td>913</td>
<td>DB 2AH,00H</td>
</tr>
<tr>
<td>053E-08</td>
<td>914</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>053F-08</td>
<td>915</td>
<td>DB 00H,41H</td>
</tr>
<tr>
<td>0540-08</td>
<td>916</td>
<td>DB 0AH,7FH</td>
</tr>
<tr>
<td>0541-08</td>
<td>917</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0542-08</td>
<td>918</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0543-08</td>
<td>919</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0544-08</td>
<td>920</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0545-08</td>
<td>921</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0546-08</td>
<td>922</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0547-08</td>
<td>923</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0548-08</td>
<td>924</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0549-08</td>
<td>925</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>054A-08</td>
<td>926</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>054B-08</td>
<td>927</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>054C-08</td>
<td>928</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>054D-08</td>
<td>929</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>054E-08</td>
<td>930</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>054F-08</td>
<td>931</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0550-08</td>
<td>932</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0551-08</td>
<td>933</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0552-08</td>
<td>934</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0553-08</td>
<td>935</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0554-08</td>
<td>936</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0555-08</td>
<td>937</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0556-08</td>
<td>938</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0557-08</td>
<td>939</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0558-08</td>
<td>940</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>0559-08</td>
<td>941</td>
<td>DB 00H,00H</td>
</tr>
<tr>
<td>055A-08</td>
<td>942</td>
<td>DB 00H,00H</td>
</tr>
</tbody>
</table>

7-81
APPLICATIONS

0586 00
943
944
945 ; THIS IS WHERE THE CONTROL CHARACTERS ARE LOOKED UP
946
......
0587 00
0588 00
0589 00
058A 00
058B 00
058C 00
058D 00
058E 00
058F 15
0590 09
0591 0F
0592 10
0593 0B
0594 0C
0595 0A
0596 7F
0597 0A
0598 0B
0599 0C
059A 00
059B 00
059C 00
059D 00
059E 00
059F 0D
05A0 00
05A1 00
05A2 00
05A3 00
05A4 00
05A5 00
05A6 00
05A7 1A
05A8 10
05A9 03
......
05B3 07
05B4 00
05B5 00
05B6 11
05B7 17
05B8 13
05B9 06
05BA 12
05BB 14
05BC 00
05BD 1B
05BE 1D
05BF 1C
05C0 1C
05C1 14
05C2 1F
05C3 00
05C4 00
05C5 04
05C6 05
05C7 69
05C8 03
05C9 0A
05CA 02
05CB 01
05CC 00
05CD 0A
05CE 00
05CF 00
05D0 00
05D1 28
05D2 00
05D3 14
05D4 00
05D5 0A
05D6 00

977 ; LOOK UP TABLE FOR 8253 BAUD RATE GENERATOR
978
979
980 BDLK:
981
......
982
983
984
985
986

7-82
APPLICATIONS

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

ASSEMBLY COMPLETE, NO ERRORS
Low-Cost CRT Control Does More with Less
Fewer parts make a microprocessor-based CRT controller cost-effective, and interrupt-driven software cuts overhead on the system's CPU.

Low-cost CRT control does more with less

The multitude of components and the CPU overhead long associated with cathode-ray-tube controllers are rapidly becoming conspicuous by their absence. In particular, an intelligent terminal based on Intel's iAPX 88/10 (8088) microprocessor and 8276 small-system CRT controller eliminates all but 22 of the nearly 40 chips required by other CRT controllers (even those with microprocessors and integrated peripherals). It also cuts overhead on the processor to less than 25%, so that the 88/10 is free to implement many intelligent terminal functions as local data processing.

The iAPX 88/10 implementation supplies characters directly to the 8276 by means of interrupt-driven software, eliminating the need for a direct-memory-access (DMA) controller. The design interfaces directly with standard CRT monitors, contact-closure keyboards, and RS-232C serial-communication links (asynchronous or synchronous), to provide a complete stand-alone operator interface.

Although the primary design goal—implementing a low-cost CRT terminal—has excluded some useful CRT features, these are easily made available through additional external hardware. For example, composite video is added with two TTL packages, a transistor, and some resistors and capacitors. Another simple option involves the two general-purpose attribute outputs on the 8276 and lets users select any one of four colors on a color monitor.

Basic system configuration and architecture

Central to the 22-chip CRT controller design is an iAPX 88/10 8-bit microprocessor operating at 5 MHz and supported by two 8185 1-kbit × 8 static RAMs and a 2716 control software PROM (Fig. 1). An 8251A programmable communication interface provides synchronous or asynchronous serial communica-

Thomas Rossi, Applications Mgr Peripheral Components
Intel Corp.
3065 Bowers Ave., Santa Clara, CA 95051

1. Intelligent terminals, built with Intel's iAPX 88/10 (8088) microprocessor and new 8276 small-system CRT controller, take this basic configuration to reduce parts count and minimize overhead on the system CPU.
Low-cost CRT

third port senses option-switch settings and the vertical-retrace signal from the 8276 (for CRT synchronization upon reset).

The CRT dot and character timing is generated by an 8284A clock generator. Another 8253 timer provides the appropriate horizontal-retrace timing for the CRT monitor. In its programmable one-shot mode, this timer generates a 32-μs horizontal-retrace pulse for the CRT monitor (Ball Brothers TV-12). A simple user-initiated change in the software will modify this delay time to suit different CRT monitors. The third and last timer in the 8253 is available for any user-defined need.

A 2716 EPROM on the controller board serves as a user-programmable character generator. A shift register transforms the data from the character EPROM into a serial-bit stream to illuminate dots on the CRT screen. The 2716 character generator helps to create special symbols and characters for word processing, industrial-control applications, or foreign-language displays.

The controller hardware is divided into processor and support, serial and parallel I/O, and CRT-control sections. The processor and support section consists of an iAPX 88/10 microprocessor, which is supported by two 8185 1-kbit × 8 static-RAM devices, and another 2716 EPROM (containing 2 kbytes of control firmware). The iAPX 88/10 uses a 15-MHz crystal (with an 8284A) to operate at a 5-MHz clock rate. The 8185 memories attach directly to the iAPX 88/10 multiplexed bus. An 8282 latches eight address lines (A0–A7) from the multiplexed bus for 2716-program memory access (Fig. 2).

The serial and parallel I/O section of the terminal includes the 8255A programmable peripheral interface, and the CRT section contains the 8276 CRT controller and support circuits. All of the controller's I/O operations are memory mapped (see table).

How the controller board communicates

The CRT-controller board communicates to computer systems and other CRT units through a serial interface. Both RS-232C and TTL-compatible interfaces are available at the PC connector. The unit's standard software supports eight data-transmission rates: 9600, 4800, 2400, 1200, 600, 300, 150, and 110 baud. These rates are switch-selectable on the board. Since the baud-rate clock is generated by an 8253, baud rates may be easily modified in software.

Keyboard scanning is supported through the A and B ports of a 8255A programmable peripheral interface. Therefore, low-cost unencoded keyboards can be used. The eight scan lines (port B) and eight return lines (port A) support a 64-contact closure-key matrix. The three switches attached to port C permit baud-rate selection. Four general-purpose

Memory map of controller I/O operations

<table>
<thead>
<tr>
<th>Address range</th>
<th>Selected device</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000 - 00029</td>
<td>RAM</td>
<td>Interrupt vector</td>
</tr>
<tr>
<td>00030 - 0007F</td>
<td>RAM</td>
<td>Stack, local variables</td>
</tr>
<tr>
<td>00100 - 01001</td>
<td>8276 CS</td>
<td>Display buffer</td>
</tr>
<tr>
<td>01000 - 01900</td>
<td>8276 BS</td>
<td>8276 command/status</td>
</tr>
<tr>
<td>12000 - 12001</td>
<td>8251A</td>
<td>8276 row buffers</td>
</tr>
<tr>
<td>14000 - 14003</td>
<td>8253</td>
<td>Serial channel</td>
</tr>
<tr>
<td>16000 - 18000</td>
<td>8255A</td>
<td>Baud-rate timer</td>
</tr>
<tr>
<td>FF800 - FFFFF</td>
<td>2716</td>
<td>Keyboard, switches</td>
</tr>
</tbody>
</table>

2. The processor and support section of the intelligent terminal's hardware contains two 8185 RAMs attached directly to the iAPX 88/10 multiplexed bus. An 8282 latches eight address lines (A0–A7) from the multiplexed bus for 2716-program memory access.

3. Here are the major functional blocks of the 8276 programmable CRT controller. This device permits software specification of most CRT-screen format characteristics (cursor position, characters/row, rows/frame).
inputs on port C permit the software to sense depression of the caps-lock key, the control key, and the shift key, as well as the position of the line/local switch. The last input on port C senses the status of the vertical retrace (VRTC) output of the 8276, so that the controller can synchronize with the CRT display on power up or after a hardware reset.

All keyboard I/O connects to the terminal board by means of a 40-pin header on its edge. All seven option-switch inputs are also brought to the connector, so that option switches may be installed on the keyboard if desired.

**Software specifies the screen format**

The CRT display is controlled by the 8276 programmable CRT controller (Fig. 3). With this device, most CRT screen-format characteristics—such as the cursor position, the number of characters per row, and the number of rows per frame—can be specified through software. The 8276 handles all display timing including retrace time delays.

In the current design, 2000 characters are displayed on the CRT screen (25 rows of 80 characters). Each character is formed as a 5 x 7-dot matrix within a larger 7 x 10 matrix (Fig. 4). Other screen formats (e.g., 16 rows of 64 characters) can be easily implemented with a few software changes and no hardware changes.

The 8276 contains two 80-character row buffers (see “Row Buffers Reduce System Overhead”). While one buffer displays the current character line on the screen, the 8276 fills the other row buffer from memory. This data transfer begins when the 8276 issues a data request (by means of the BRDY pin), causing an interrupt to the CPU. In response to this interrupt, the CPU activates the RAM's CS and RD inputs, while simultaneously activating the 8276 BS and WR inputs (Fig. 5). Through this technique, a single bus cycle suffices to transfer each byte from the RAM into the CRT row buffer. After the row buffer is filled, the CPU exits the interrupt-service routine.

But the 8276 can do more than simply paint characters on a CRT screen. Its end-of-row-stop buffer-loading code allows the control software to blank individual display lines. Also, the end-of-the-screen-stop buffer-loading code initiates an erase to the end of the screen.

The 8276 supports software selection of visible-field “attributes” that can blink, underline, or highlight (intensify) characters on the screen and can reverse the video-character fields (black letters on a white background). Two general-purpose attribute outputs are provided to control the user-defined display capabilities.

**Hardware provides three support functions**

The 8276 is supported by three hardware functions: a dot/character-clock oscillator, an EPROM character generator, and a character-shift register (Fig. 6). The dot/character-clock oscillator consists of an 8284A operating at 11.34 MHz and providing an 88.2-ns dot clock. A 74LS163 divides this clock by 7 to generate a 1.62-MHz (617-ns) character clock.

---

**Diagram:**

4. The dot-matrix character font used in the low-cost CRT controller creates a 5 x 7 character in a 7 x 10 matrix (example shown is an upper-case A). Top and bottom lines are blanked for character separation, and the remaining line is reserved for cursor/underline display.

5. Row-buffer loading for the 8276 begins when a single 8088 string instruction moves data bytes from the 8185 RAM to the 8276 row buffer. The 8088 CPU “thinks” it is loading the AX register.
The 8276 is programmed to display one raster line every 61.7 μs—a complete character line every 617 μs (ten raster lines). The 8276 is also programmed to refresh the screen every 16.7 ms (60 Hz).

Each character row consists of ten raster lines. Seven lines display the 5 × 7-character matrix, two lines are blanked for row spacing, and one line displays the cursor and underline.

The 8276 uses the line count (LC0-LC3) outputs to indicate the current raster line during the display of each character. These outputs, combined with the character-code outputs (CC0-CC6), are sent to the 2716, which generates the dot pattern for display. This dot pattern is loaded into the shift register and is serially clocked for display by the 11.34-MHz dot clock.

During the vertical-retrace interval, the row buffer for the first line of the next frame is loaded by the iAPX 88/10. When the frame starts, the 8276 outputs the first character on its CC0-CC6 pins; the LC outputs are all zero. Exactly 617 ns later, the next character code is emitted by the 8276. This process continues every 617 ns until all 80 characters have been output. Then the 8276 generates a horizontal-retrace pulse, which is converted to the appropriate pulse width for the CRT monitor by the 8253.

At the end of the first raster line, the 8276 increments the LC outputs. The next nine raster lines are similar to the first—the 8276 outputs the same 80 character codes on the CC0-CC6 pins for each of the raster lines, and the LC outputs are incremented after each raster line.

While the ten raster lines are being displayed, the 8276 is also filling the next row buffer. After the tenth raster line is completed, the 8276 resets the LC count and outputs character codes for the second row on the CC0-CC6 pins. As this row is displayed, the first row buffer is filled with information for the third row. The 8276 alternates row buffers until all 25 rows are displayed. At this time, the vertical-retrace signal is activated, and the scanning process is repeated for the next frame.

During display, the 8276 automatically activates the video-suppress pin (VSP) and/or light-enable outputs (LTEN), as appropriate, to control retrace blanking, generate the cursor, or underline characters.

Software is split between two priorities

The software for the CRT controller is divided into high and low-priority sections. The high-priority "foreground" software is activated each time the 8276 requests (through the iAPX 88/10 NMI interrupt) that an 80-character row buffer be filled. The 8276 row buffer is filled by performing 80 sequential memory reads. As each read is performed, the

![Diagram of CRT control logic](image)

6. CRT control logic supports the 8276. Three hardware functions are involved: a dot/character clock oscillator, an EPROM character generator, and a character-shift register.
If no row buffer is present, the CRT controller must go to main memory to fetch every character during every dot scan line. Thus, the central processing unit is forced to relinquish the system bus 90 to 95% of the time. That CPU inactivity (overhead) greatly degrades total system performance and efficiency. CRT terminals using this approach are typically limited to between 1200 and 2400 baud on their serial-communications channels.

However, with the 8276's row-buffered architecture, the CRT controller need only access the main memory once for each displayed character row. This approach reduces system bus overhead for CRT refreshing to 25% (maximum). The CPU is then free to perform other local-processing functions, for instance, processing data at 9600 baud on a serial-communications channel.

**Row buffers reduce system overhead**

<table>
<thead>
<tr>
<th>PUSHF</th>
<th>save registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH SI</td>
<td>used by</td>
</tr>
<tr>
<td>PUSH CX</td>
<td>subroutine</td>
</tr>
<tr>
<td>MOV SI,CURAD</td>
<td>point to current line</td>
</tr>
<tr>
<td>ADD SI,OFFSET</td>
<td></td>
</tr>
<tr>
<td>CLD</td>
<td>auto increment</td>
</tr>
<tr>
<td>MOV CX,40</td>
<td></td>
</tr>
<tr>
<td>REP LODS</td>
<td>move 40 words</td>
</tr>
<tr>
<td>CMP SI,LAST</td>
<td>check for end of screen</td>
</tr>
<tr>
<td>JNZ KTPK</td>
<td>jump if not at end</td>
</tr>
<tr>
<td>MOV SI,TOPDIS</td>
<td>end-set to top</td>
</tr>
<tr>
<td>KTPK</td>
<td></td>
</tr>
<tr>
<td>MOv CURAD,SI</td>
<td></td>
</tr>
<tr>
<td>POP CX</td>
<td>restore</td>
</tr>
<tr>
<td>POP SI</td>
<td></td>
</tr>
<tr>
<td>POPF</td>
<td></td>
</tr>
</tbody>
</table>

7. A screen-refresh routine illustrates how the iAPX 88/10 load-string (LODS) instruction fills an 8276 row buffer. The 15 lines take 167 $\mu$s and are run every ten CRT lines (every 617 $\mu$s).

**Cumbersome scrolling technique avoided**

A refresh-buffer memory stores all 2000 characters that can be displayed on the CRT screen. The foreground software transfers one row (of 80 characters) at a time to the 8276. Two pointers are employed during normal operation. Under the control of foreground processing software, the current-row pointer contains the address of the next row to be displayed. This pointer must always be correct, so that a row can be transferred to the 8276 when requested. The buffer pointer contains the address of the next CRT buffer location to be written into (from either the keyboard or the serial port). Controlled by the background software, the buffer pointer indicates the cursor's actual location.

The simplest refresh-buffer organization associates the first memory address with the upper left position on the CRT screen. All other characters are stored sequentially (Fig. 9). But this method makes CRT screen scrolling difficult. Scrolling requires that each display line be moved up one row. The top line of the CRT is lost, the bottom line is blanked, and the cursor is placed at the beginning of the bottom line.

With this fixed sequential organization, all characters in the refresh buffer must be moved forward...
Low-cost CRT

by 80 characters (memory locations) to scroll the screen. (Each line moves up one row on the CRT and the last 80 characters in the buffer are blanked.) Moving 1920 characters each time the screen scrolls a single line is very slow and cumbersome.

The low-cost CRT controller avoids this problem with a slight modification of the fixed-sequential scrolling technique. Here, sequential memory orientation is retained while the need to move characters in memory is eliminated. This approach requires an additional display-start pointer that points to the memory location of the first character to be displayed.

At system initialization, the display-start pointer is set to 30H, the buffer-start address. During each vertical-retrace interval, the current-row pointer is initialized from the display-start pointer. Scrolling is performed by merely changing the display-start pointer.

For a single row scroll, the display-start pointer moves ahead 80 characters to location 80H, and the first 80 characters in the buffer are blanked. During the next vertical retrace, the foreground software sets the current-row pointer to the display-start location (80H), and begins transferring characters to the 8276 from this address.

The character in memory-location 80H (previously the first character in the second row) now occupies the first display position on the CRT screen (first character of the first row). When the foreground software reaches the end of the display buffer, the next row is read from the beginning of the buffer (location 30H). Thus, the first 80 characters in the buffer appear on the last display row (Fig. 10).

Each subsequent scroll moves the display start pointer forward by 80 characters. Buffer operations automatically “roll over” to the physical beginning of the buffer after passing the last buffer location.

Since the row-by-row character display is controlled by iAPX 88/10 software, other display techniques may be used. In particular, a linked list structure is extremely adaptable to word-processing and text-editing functions. This method allows each row within a file to be changed independently of other rows.

Because the rows are linked or “chained together” by pointers, rows may be easily inserted or deleted by simply changing pointers. To display a CRT frame, the processor simply follows the pointer chain from one row to the next.

9. This memory/screen-character relationship exists when all characters are stored sequentially, making scrolling difficult.

10. If sequential memory orientation is retained but characters do not have to be moved in memory, scrolling can be much more efficient. Here, scrolling is accomplished simply by changing the display-start pointer. The memory/screen-character relationship is shown after a scroll of one line from the positions illustrated in Fig. 9.

How useful?

<table>
<thead>
<tr>
<th>Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate design application</td>
</tr>
<tr>
<td>Within the next year</td>
</tr>
<tr>
<td>Not applicable</td>
</tr>
</tbody>
</table>
82720 GRAPHICS DISPLAY CONTROLLER

- Displays Low-to-High Resolution Images
- Draws Characters, Points, Lines, Arcs, and Rectangles
- Supports Monochrome, Gray Scale, or Color Displays
- Zooms, Pans and Windows Through a 4 Mpixel Display Memory
- Extremely Flexible Programmable Screen Display, Blanking, and Sync Formats
- Compatible with Intel’s Microprocessor Families
- High-Level Commands Off Load Host Processor from Bit Map Loading and Screen Refresh Tasks
- Supports Graphics, Character, and Mixed Display Modes

FUNCTIONAL DESCRIPTION

Introduction

The 82720 Graphics Display Controller (GDC) is an intelligent microprocessor peripheral designed to drive high-performance raster-scan computer graphics and character CRT displays. Positioned between the video display memory and Intel microprocessor bus, the GDC performs the tasks needed to generate the raster display and manage the display memory. Processor software overhead is minimized by the GDC's sophisticated instruction set, graphics figure drawing, and DMA transfer capabilities. The display memory directly supported by the GDC can be configured in any number of formats and sizes up to 256K 16-bit words. The display can be zoomed and partitioned screen areas can be independently scrolled and panned. With its light pen input and multiple controller capability, the GDC is ideal for most computer graphics applications. Systems implemented with the GDC can be designed to be compatible with standards such as VDI, NAPLPS, GKS, Core, or custom implementations.

Figure 1. Block Diagram

Figure 2. Pin Configuration
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Type</th>
<th>Name and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2XWCLK</td>
<td>1</td>
<td>I</td>
<td>Clock Input</td>
</tr>
<tr>
<td>DBIN</td>
<td>2</td>
<td>O</td>
<td>Display Bus Input: Read strobe output used to read display memory data into the GDC.</td>
</tr>
<tr>
<td>HSYNC</td>
<td>3</td>
<td>O</td>
<td>Horizontal Sync: Output used to initiate the horizontal retrace of the CRT display.</td>
</tr>
<tr>
<td>V/EXT</td>
<td>4</td>
<td>I/O</td>
<td>Vertical Sync: Output used to initiate the vertical retrace of the CRT display.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>In slave mode, this pin is an input used to synchronize the GDC with the master raster timing device.</td>
</tr>
<tr>
<td>BLANK</td>
<td>5</td>
<td>O</td>
<td>Blank: Output used to suppress the video signal.</td>
</tr>
<tr>
<td>RAS (ALE)</td>
<td>6</td>
<td>O</td>
<td>Row Address Strobe (Address Latch Enable): Output used to start the control timing chain when used with dynamic RAMs. When used with static RAMs, this signal is used to demultiplex the display address/data bus.</td>
</tr>
<tr>
<td>DRQ</td>
<td>7</td>
<td>O</td>
<td>DMA Request: Output used to request a DMA transfer from a DMA controller (8237) or I/O processor (8089).</td>
</tr>
<tr>
<td>DACK</td>
<td>8</td>
<td>I</td>
<td>DMA Acknowledge: Input used to acknowledge a DMA transfer from a DMA controller or I/O processor.</td>
</tr>
<tr>
<td>RD</td>
<td>9</td>
<td>I</td>
<td>Read: Input used to strobe GDC Data into the microprocessor.</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>I</td>
<td>Write: Input used to strobe microprocessor data into the GDC.</td>
</tr>
<tr>
<td>A0</td>
<td>11</td>
<td>I</td>
<td>Register Address: Input used to select between commands and data read or written.</td>
</tr>
<tr>
<td>DB0</td>
<td>12</td>
<td>I/O</td>
<td>Bidirectional Microprocessor Data Bus Line: Input enabled by WR. Output enabled by RD.</td>
</tr>
<tr>
<td>DB1</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB3</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB4</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB5</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB6</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB7</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>20</td>
<td></td>
<td>Ground.</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td></td>
<td>+5V Power Supply</td>
</tr>
<tr>
<td>A17</td>
<td>39</td>
<td>O</td>
<td>Graphics Mode: Display Address Bit 17 Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Character Mode: Cursor and Line Counter Bit 4 Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mixed Mode: Cursor and Image Mode Flag</td>
</tr>
<tr>
<td>A16</td>
<td>38</td>
<td>O</td>
<td>Graphics Mode: Display Address Bit 16 Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Character Mode: Line Counter Bit 3 Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mixed Mode: Attribute Blink and Line Counter Reset</td>
</tr>
<tr>
<td>AD15</td>
<td>37</td>
<td>I/O</td>
<td>Graphics Mode: Display Address/Data Bits 13–15</td>
</tr>
<tr>
<td>AD14</td>
<td>36</td>
<td></td>
<td>Character Mode: Line Counter Bits 0–2 Output</td>
</tr>
<tr>
<td>AD13</td>
<td>35</td>
<td></td>
<td>Mixed Mode: Display Address/Data Bits 13–15</td>
</tr>
<tr>
<td>AD12</td>
<td>34</td>
<td>I/O</td>
<td>Display Address/Data Bits 0–12</td>
</tr>
<tr>
<td>AD11</td>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD10</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD9</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD8</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD7</td>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD6</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD5</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD4</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD3</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD2</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD1</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD0</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPEN</td>
<td>21</td>
<td>I</td>
<td>Light Pen Detect Input</td>
</tr>
</tbody>
</table>

7-92
FUNCTIONAL DESCRIPTION (Continued)

Microprocessor Bus Interface
Control of the GOC by the system microprocessor is achieved through an 8-bit bidirectional interface. The status register is readable at any time. Access to the FIFO buffer is coordinated through flags in the status register.

Command Processor
The contents of the FIFO are interpreted by the command processor. The command bytes are decoded, and the succeeding parameters are distributed to their proper destinations within the GOC. The bus interface has priority over the command processor when both access the FIFO simultaneously.

DMA Control
The DMA Control circuitry in the GOC coordinates data transfers when using an external DMA controller. The DMA Request and Acknowledge handshake lines interface with an 8257 or 8237 DMA controller or 8089 I/O processor, so that display data can be moved between the microprocessor memory and the display memory.

Parameter RAM
The 16-byte RAM stores parameters that are used repetitively during the display and drawing processes. In character mode, the RAM holds the partitioned display area parameters. In graphics mode, the RAM also holds the drawing pattern and graphics character.

Video Sync Generator
Based on the clock input, the sync logic generates the raster timing signals for almost any interlaced, non-interlaced, or "repeat field" interlaced video format. The generator is programmed during the idle period following a reset. In video sync slave mode, it coordinates timing between the GOC and another video source.

Memory Timing Generator
The memory timing circuitry provides two memory cycle types: a two-clock period refresh cycle and the read-modify-write (RMW) cycle which takes four clock periods. The memory control signals needed to drive the display memory devices are easily generated from the GOC's RAS(ALE) and DBIN outputs.

Zoom and Pan Controller
Based on the programmable zoom display factor and the display area parameters in the parameter RAM, the zoom and pan controller determines when to advance to the next memory address for display refresh and when to go on to the next display area. A horizontal zoom is produced by slowing down the display refresh rate while maintaining the video sync rates. Vertical zoom is accomplished by repeatedly accessing each line a number of times equal to the horizontal repeat. Once the line count for a display area is exhausted, the controller accesses the starting address and line count of the next display area from the parameter RAM. The system microprocessor, by modifying a display area starting address, allows panning in any direction, independent of the other display areas.

Drawing Processor
The drawing processor contains the logic necessary to calculate the addresses and positions of the pixels of the various graphics figures. Given a starting point and the appropriate drawing parameters, the drawing processor needs no further assistance to complete the figure drawing.

Display Memory Controller
The display memory controller's tasks are numerous. Its primary purpose is to multiplex the address and data information in and out of the display memory. It also contains the 16-bit logic units used to modify the display memory contents during RMW cycles, the character mode line counter, and the refresh counter for dynamic RAMs. The memory controller apportions the video field time between the various types of cycles.

Light Pen Debouncer
Only if two rising edges on the light pen input occur at the same point during successive video fields are the pulses accepted as a valid light pen detection. A status bit indicates to the system microprocessor that the light pen register contains a valid address.

System Operation
The GOC is designed to work with Intel microprocessors to implement high-performance computer graphics systems. System efficiency is maximized through partitioning and a pipelined architecture. At the lowest level, the GOC generates the basic video
raster timing, including sync and blanking signals. Partitioned areas on the screen and zooming are also accomplished at this level. At the next level, video display memory is modified during the figure drawing operations and data moves. Third, display memory address are calculated pixel by pixel as drawing progresses. Outside the GDC at the next level, preliminary calculations are done to prepare drawing parameters. At the fifth level, the picture must be represented as a list of graphics figures drawable by the GDC. Finally, this representation must be manipulated, stored and communicated. The GDC takes care of the high-speed and repetitive tasks required to implement graphics systems.

GENERAL OVERVIEW

In order to minimize system bus loading, the 82720 uses a private video memory for storage of the video image. Up to 512K bytes of video memory can be directly supported. For example, this is sufficient capacity to store a 2048 x 2048 pixel x 1 bit image. Images can be generated on the screen by:

- Drawing Commands
- Program-Controlled Transfers
- DMA Transfers from System Memory

The 82720 can be configured to support a wide variety of graphics applications. It can support:

- High Dot Rates
- Color Planes
- Horizontal Split Screen
- Character Split Screen
- Multiplexed Graphic and Character Display

GRAPHIC DISPLAY CONFIGURATIONS

The 82720 provides the flexibility to handle a wide variety of graphic applications. This flexibility results from having its own private video memory for storage of the graphics image. The organization of this memory determines the performance, the number of bits/pixel and the size of the display. Several different video memory organizations are examined in the following paragraphs.

In the simplest 82720 system, the memory can store up to a 2048 x 2048 x 1 bit image. It can display a 1024 x 1024 x 1 bit section of the image at a maximum dot rate of 44 MHz, or 88 MHz in wide mode. In this configuration, only 1 bit/pixel is used.

By partitioning the memory into multiple banks, color, gray scale and higher bandwidth displays can be supported. By adding various amounts of external logic, many cost/performance tradeoffs for both display and drawing are realizable.

The video memory can be partitioned into 4 banks, each 1024 x 1024 bits. By selecting all 4 memory banks during display, 4 bits/pixel can be provided by a single 82720. Each bank of video memory contributes 1 bit to each pixel. This configuration can support color monitors, again with a maximum dot shift rate of 44 or 88 MHz.

Higher performance may be achieved by using multiple 82720s. Multiple 82720s can be used to support multiple display windows, increased drawing speed, or increased bits per pixel. For display windows, each 82720 controls one window of the display. For increased drawing speed, multiple 82720s are operated in parallel. For increased bits/pixel, each 82720 contributes a portion of the number of bits necessary for a pixel.

CHARACTER DISPLAY CONFIGURATION

Although the 82720 is intended primarily for raster-scan graphics, it can be used as a character display controller. The 82720 can support up to 8K by 13 bits of private video memory in this configuration (1 character = 13 bits). This is sufficient memory to store 4 screens of data containing 25 rows by 80 characters. The 82720 can display up to 256 characters per row. Smooth vertical scrolling of each of 4 independent display partitions is also supported.

MIXED DISPLAY CONFIGURATION

The GDC can support a mixed display system for both graphic and character information. This capability allows the display screen to be partitioned between graphic and character data. It is possible to switch between one graphic display window and one character display window with raster line resolution. A maximum of 256K bytes of video memory is supported in this mode: half is for graphic data, half is for character data. In graphic mode, a one megapixel image can be stored and displayed. In character mode, 64K, 16-bit characters can be stored.

DETAILED OPERATIONAL DESCRIPTION

The GDC can be used in one of three basic modes — Graphics Mode, Character Mode and Mixed Mode. This section of the data sheet describes the following for each mode:

1. Memory organization
2. Display timing
3. Special Display functions
4. Drawing and writing

7-94
Graphics Mode Memory Organization

The Display Memory is organized into 16-bit words (32-bit words in wide mode). Since the display memory can be larger than the CRT display itself, two width parameters must be specified: display memory width and display width. The Display width (in words) is selected by a parameter of the Reset command. The Display memory width (in words) is selected by a parameter of the Pitch command. The height of the Display memory can be larger than the display itself. The height of the Display is selected by a parameter of the Reset command. The GDC can directly address up to 4Mbits (0.5Mbytes) of display RAM in graphics mode.

Graphics Mode Display Timing

All raster blanking and display timings of the GDC are a function of the input clock frequency. Sixteen or 32 bits of data are read from the RAM and loaded into a shift register in each two clock period display cycle. The Address and Data busses of the GDC are multiplexed. In the first part of the cycle, the address of the word to be read is latched into an external demultiplexer. In the second part of the cycle the data is read from the RAM and loaded into the shift register. Since all 16 (32) bits of data are to be displayed, the dot clock is \( 8 \times (16 \times) \) the GDC clock or \( 16 \times (32 \times) \) the Read cycle rate.

Parameters of the Reset or Sync command determine the horizontal and vertical front porch, sync pulse, and back porch timings. Horizontal parameters are specified as multiples of the display cycle time, and vertical parameters as a multiple of the line time.

Another Reset command parameter selects interlaced or non-interlaced mode. A bit in the parameter RAM can define Wide Display Mode. In this mode, while data is being sent to the screen, the display address counter is incremented by two rather than one. This allows the display memory to be configured to deliver 32 bits from each display read cycle.

The V Sync command specifies whether the V Sync Pin is an input or an output. If the V Sync Pin is an output, the GDC generates the raster timing for the display and other CRT controllers can be synchronized to it. If the V Sync pin is an input, the GDC can be synchronized to any external vertical Sync signal.

Graphics Mode Special Display Functions:

**WINDOWING**

The GDC’s Graphics Mode Display can be divided into two windows on the screen, upper and lower. The windows are defined by parameters written into the GDC’s parameter RAM. Each window is specified by a starting address and a window length in lines. If the second window is not used, the first window parameters should be specified to be the same as the active display length.

**ZOOMING**

A parameter of the GDC’s zoom command allows zooming by effectively increasing the size of the dots on the screen. This is accomplished vertically by repeating the same display line. The number of times it is repeated is determined by the display zoom factor parameter. Horizontally, zoom is accomplished by extending each display word cycle and displaying fewer words per line, according to the zoom factor. It is the responsibility of the microprocessor controlling the GDC to provide the shift register clock circuitry with the zoom factor required to slow down the shift registers to the appropriate speed. The frequency of the 2XWCLK should not be changed. The zoom factor must be set to a known state upon initialization.

**PANNING**

Panning is accomplished by changing the starting address of the display window. In this way, panning is possible in any direction, vertically on a line by line basis and horizontally on a word by word basis.

Graphics Mode Drawing and Writing

The GDC can draw solid or patterned lines, arcs, circles, rectangles, slanted rectangles, characters, slanted characters, filled rectangles. Direct access to the bit map is also provided via the DMA Commands and the Read or Write data commands.

**MEMORY MODIFICATION**

All drawing and writing functions take place at the location in the display RAM specified by the cursor. The cursor is not displayed in Graphics Mode. The cursor location is modified by the execution of drawing, reading or writing commands. The cursor will move to the bit following the last bit accessed.
Each bit is drawn by executing a Read-Modify-Write cycle on the display RAM. These R/M/W cycles normally require four 2XWCLK cycles to execute. If the display zoom factor is greater than two, each R/M/W cycle will be extended to the width of a display cycle. Write Data (WDAT), Read Data (RDAT), DMA write (DMAW) and DMA read (DMAR) commands can be used to examine or modify one to 16 bits in each word during each R/M/W cycle. All other graphics drawing commands modify one bit per R/M/W cycle.

An internal 16-bit Mask register determines which bit(s) in the accessed word are to be modified. A one in the Mask register allows the corresponding bit in the display RAM to be modified by the R/M/W cycle. A zero in the Mask register prevents the corresponding bit in the display RAM.

The display RAM bits can be modified in one of four ways. They can be set to 1, reset to 0, complemented or replaced by a pattern.

When replace by a pattern mode is selected, lines, arcs and rectangles will be drawn using the 16-bit pattern in parameter RAM bytes 8 and 9.

In set, reset, or complement mode, parameter RAM bytes 8 and 9 act as another level of masking for line arc and rectangle drawing. As each 16-bit segment of the line or arc is drawn, it is checked against the pattern in the parameter RAM. If the pattern RAM bit is a one, the display RAM bit will be set, reset, or complemented per the proper modes. If the pattern RAM bit is a zero, the display RAM bit won't be modified.

When replace by pattern mode is selected, the graphics character and fill commands will cause the 8 x 8 pattern in parameter RAM bytes 8 to 15 to be written directly into the display RAM in the appropriate locations.

In set, reset, or complement mode, the 8 x 8 pattern in parameter RAM bytes 8 to 15 act as a mask pattern for graphics character or fill commands. If the appropriate parameter RAM bit is set, the display RAM bit will be modified. If the parameter RAM bit is zero, the display RAM bit will not be modified. These modes are selected by issuing a WDAT command without parameters before issuing graphics commands. The pattern in the parameter RAM has no effect on WDAT, RDAT, DMAP, or DMAR operations.

### Drawing and Drawing Commands

After the modification mode has been set and the parameter RAM has been loaded, the final drawing parameters are loaded via the figure specify (FIGS) command. The first parameter specifies the direction in which drawing will occur and the figure type to be drawn. This parameter is followed by one to five more parameters depending on the type of character to be drawn.

The direction parameter specifies one of eight octants in which the drawing or reading will occur. The effect of drawing direction on the various figure types is shown in Figure 9.

RDAT, WDAT, DMAR, and DMA Operations move through the Display memory as shown in the “DMA” Column.

The other parameters required to set up figure reading or drawing are shown in Figure 3.

<table>
<thead>
<tr>
<th>DRAWING TYPE</th>
<th>DC</th>
<th>D</th>
<th>D2</th>
<th>D1</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIAL VALUE*</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>LINE</td>
<td></td>
<td></td>
<td>2(AD) -</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2(AD) -</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2(AD)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARC**</td>
<td>rain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECTANGLE</td>
<td>3</td>
<td>A - 1</td>
<td>B - 1</td>
<td>A - 1</td>
<td></td>
</tr>
<tr>
<td>AREA FILL</td>
<td>B - 1</td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRAPHIC CHARACTER***</td>
<td>B - 1</td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE DATA</td>
<td>W - 1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DMAR</td>
<td>D - 1</td>
<td>C - 1</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DMAR</td>
<td>D - 1</td>
<td>C - 2</td>
<td>(C - 2)/2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>READ DATA</td>
<td>W</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

---

*INITIAL VALUES FOR THE VARIOUS PARAMETERS ARE LOADED WHEN THE FIGS COMMAND BYTE IS PROCESSED.

**CIRCLES ARE DRAWN WITH 8 ARCS, EACH OF WHICH SPAN 45°, SO THAT SIN θ = 1/2 AND SIN 4θ = 0.

***GRAPHIC CHARACTERS ARE A SPECIAL CASE OF BIT-MAP AREA FILLING IN WHICH θ AND A ≥ 8. IF A = 8 THERE IS NO NEED TO LOAD D AND D2.

WHERE:

- 1 = ALL ONES VALUE.
- 0 = NO PARAMETER BYTES SENT TO GDC FOR THIS PARAMETER.
- D = THE LARGER OF DX OR DY.
- D = THE SMALLER OF DX OR DY.
- r = RADIUS OF CURVATURE, IN PIXELS.
- θ = ANGLE FROM MAJOR AXIS TO END OF THE ARC, ≤ 45°.
- θ = ANGLE FROM MAJOR AXIS TO END OF THE ARC, ≤ 45°.
- θ = ROUND UP TO THE NEXT HIGHER INTEGER.
- θ = ROUND DOWN TO THE NEXT LOWER INTEGER.
- A = NUMBER OF PIXELS IN THE INITIALLY SPECIFIED DIRECTION.
- B = NUMBER OF PIXELS IN THE DIRECTION AT RIGHT ANGLES TO THE INITIALLY SPECIFIED DIRECTION.
- W = NUMBER OF WORDS TO BE ACCESSED.
- C = NUMBER OF BYTES TO BE TRANSFERRED IN THE INITIALLY SPECIFIED DIRECTION. (TWO BYTES PER WORD IF WORD TRANSFER MODE IS SELECTED.)
- D = NUMBER OF WORDS TO BE ACCESSED IN THE DIRECTION AT RIGHT ANGLES TO THE INITIALLY SPECIFIED DIRECTION.
- DC = DRAWING COUNT PARAMETER WHICH IS ONE LESS THAN THE NUMBER OF RMW CYCLES TO BE EXECUTED.
- DM = DOTS MASKED FROM DRAWING DURING ARC DRAWING.
- T = NEEDED ONLY FOR WORD READS.

Figure 3. Drawing Parameter Details
After the parameters have been set, line, arc, circle, rectangle or slanted rectangle drawing operations are initiated by the Figure Draw (FIGD) command. Character, slanted character, area fill and slanted area fill drawing operations are initiated by the Graphics Character Draw (GCHR) command. DMA transfers are initiated by the DMA Read or Write (DMAR or DMAW) commands. Data Read Operations are initiated by the Read Data (RDAT) Command. Data Write Operations are initiated by writing a parameter after the WDAT command.

The area fill operation steps and repeats the 8 x 8 graphics character pattern draw operation to fill a rectangular area. If the size of the rectangle is not an integral number of 8 x 8 pixels, the GDC will automatically truncate the pattern at the edges furthest from the starting point.

The Graphics Character Drawing capability can be modified by the Graphics Character Write Zoom Factor (GCHR) parameter of the zoom command. The zoom write factor may be set from 1 to 16 (by using from 0 to 15 in the parameter). Each dot will be repeated in memory horizontally and vertically (adjusted for drawing direction) the number of times specified by the zoom factor.

The WDAT command can be used to rapidly fill large areas in memory with the same value. The mask is set to all 1's, and the least significant bit of the WDAT parameter replaces all bits of each word written.

Character Mode Memory Organization

In character mode, the Display memory is organized into up to 8K characters of up to 13 bits each. Wide mode is also available for characters of up to 26 bits.

The display memory can be larger than the display itself. The display width (in characters) is a parameter of the reset command. The display memory width (in characters) is a parameter of the Pitch Command. The height of the display (in lines) is a parameter of the Reset Command. The display memory height is determined by dividing the number of display memory words by the pitch.

In character mode, the display works almost exactly as it does in graphics mode. The differences lie in the fact that data read from the display RAM is used to drive a character generator as well as attribute logic if desired. In Character mode, address bits 13-16 become line counter outputs used to select the proper line of the character generator, and the address 17 output becomes the cursor and line counter MSB output.

Character Mode Display Timing

In character mode, the display timing works as it does in graphics mode. In addition, the Address 17 output becomes cursor output. The characteristics of the cursor are defined by parameters of the cursor and Character Characteristics (CCHAR) command. One bit allows the cursor output to be enabled or disabled. The height of the cursor is programmable by selecting the top and bottom line between which the cursor will appear. The blink rate is also programmable. The parameter selects the number of frame times that the cursor will be inactive and active, resulting in a 50% duty cycle cursor blinking at 2 x the period specified by the parameter.

The cursor output pin also provides the line counter bit 4 signal, which is valid 10 clocks after the trailing edge of HSYNC.

Character Mode Special Display Functions

WINDOWING

The GDC's Character Mode display can be partitioned into one to four windows on the screen. The windows are defined by parameters written into the GDC's Parameter RAM. Each window is specified by a starting address and a window length in lines.

If windowing is not required, the first window length should be specified to be the same as the active display length.

ZOOMING AND PANNING

In character mode, zooming and pan handling commands function the same way as in Graphics Mode.

Character Mode Drawing and Writing

The GDC can read or write characters of up to 13 bits into or out of the Display RAM.

All reading and writing functions take place at the display RAM location specified by the cursor. The cursor location can be read by issuing the CURD command. The cursor can be moved anywhere within the display memory by the CURS command. The cursor location is also modified by the execution of character read or write commands.

Each character is written or read via a Read/Modify/Write cycle. The mask register contents determine which bit(s) in the character are modified. The mask register can be used to change character codes without modifying attribute bits or vice-versa. The Replace with pattern, Set, Reset and Complement
modes work exactly as they do in graphics mode, with the exception that the parameter RAM Pattern is not used. The pattern used is a parameter of the WDAT command.

The Figure Specify (FIGS) command must be set to Character Display mode, as well as specify the direction the cursor will be moved by read or write data commands.

In character mode, the FIGD and GCHRD commands are not used.

Mixed Mode Memory Organization

In mixed mode, the display memory is organized into two banks of up to 64K words of 16 bits each (32 bits in wide mode).

The display height and width are programmable by the same Reset or Sync command parameters as in the graphics and character modes. The display memory width (in words) is a parameter of the Pitch Command and the height of the display memory is determined by dividing the number of display memory words by the pitch.

An image mode signal is used to switch the external circuitry between graphics and character modes in two display windows.

In a graphics window, the GDC works as it does in pure graphics mode, but on a smaller total memory space (64K words vs 512K words).

In a character window, the GDC works as it does in pure character mode, but the line counter must be implemented externally. The counter is clocked by the horizontal sync pulse and reset by a signal supplied by the GDC.

In mixed mode, the GDC provides both a cursor and an attribute blink timing signal.

Mixed Mode Display Timing

In mixed mode, each word in a graphic area is accessed twice in succession. The AW parameter of the Reset or Sync command should be set to twice its normal value, and the video shift register load signal must be suppressed during the extra access cycle.

In addition, A16 becomes a Multiplexed Attribute and Clear Line Counter signal and A17 becomes a multiplexed cursor and image mode signal. A16 provides an active high line counter reset signal which is valid 10 clocks after the trailing edge of HSYNC. During the active display line time, A16 provides blink timing for external attribute circuitry. This signal blinks at 1/2 the blink rate of the cursor with a 75% on, 25% off duty cycle. A17 provides a signal which selects between graphics or character display, which is also valid 10 clocks after the trailing edge of HSYNC. During the active display time, A17 provides the cursor signal. The cursor timing and characteristics are defined in exactly the same way as in pure character mode.

Mixed Mode Special Display Functions

WINDOWING

The GDC supports two display windows in mixed mode. They can independently be programmed into either graphics or character mode determined by the state of two bits in the parameter RAM. The window location in display memory and size are also determined by parameters in the parameter RAM.

ZOOMING AND PANNING

In mixed mode, zooming and panning commands function the same as in graphics and character mode.

Mixed Mode Drawing and Writing

In mixed mode, the GDC can write or draw in exactly the same ways as in both graphics and character modes. In addition, the FIGS command has a parameter GD (Graphics Drawing Flag) which sets the image mode signal to select the proper RAM bank.

DEVICE PROGRAMMING

The GDC occupies two addresses on the system microprocessor bus through which the GDC's status register and FIFO are accessed. Commands and parameters are written into the GDC FIFO and are differentiated by address bit A0. The status register or the FIFO can be read as selected by the address line.
Commands to the GDC take the form of a command byte followed by a series of parameter bytes as needed for specifying the details of the command. The command processor decodes the commands, unpacks the parameters, loads them into the appropriate registers within the GDC and initiates the required operations.

The commands available in the GDC can be organized into five categories as described in Figure 5.

**VIDEO CONTROL COMMANDS**
1. **RESET**: Resets the GDC to its idle state.
2. **SYNC**: Specifies the video display format.
3. **VSYNC**: Selects master or slave video synchronization mode.
4. **CCHAR**: Specifies the cursor and character row heights.

**DISPLAY CONTROL COMMANDS**
1. **START**: Ends idle mode and unblanks the display.
2. **CTRL**: Controls the blanking and unblanking of the display.
3. **ZOOM**: Specifies zoom factors for the display and graphics characters writing.
4. **CURS**: Sets the position of the cursor in display memory.
5. **PRAM**: Defines starting addresses and lengths of the display areas and specifies the eight bytes for the graphics character.
6. **PITCH**: Specifies the width of the X dimension of display memory.

**DRAWING CONTROL COMMANDS**
1. **WDAT**: Writes data words or bytes into display memory.
2. **MASK**: Sets the mask register contents.
3. **FIGS**: Specifies the parameters for the drawing processor.
4. **FIGD**: Draws the figure as specified above.
5. **GCHR**: Draws the graphics character into display memory.

**DATA READ COMMANDS**
1. **RDAT**: Reads data words or bytes from display memory.
2. **CURD**: Reads the cursor position.
3. **LPFD**: Reads the light pen address.

**DMA CONTROL COMMANDS**
1. **DMAR**: Requests a DMA read transfer.
2. **DMAW**: Requests a DMA write transfer.

---

**Figure 5. GDC Command Summary**

---

**SR-6: Horizontal Blanking Active**: A 1 value for this flag signifies that horizontal retrace blanking is currently underway.

**SR-5: Vertical Sync**: Vertical retrace sync occurs while this flag is a 1. The vertical sync flag coordinates display format modifying commands to the blanked interval surrounding vertical sync. This eliminates display disturbances.

**SR-4: DMA Execute**: This bit is a 1 during DMA data transfers.

**SR-3: Drawing in Progress**: While the GDC is drawing a graphics figure, this status bit is a 1.

**SR-2: FIFO Empty**: This bit and the FIFO Full flag coordinate system microprocessor accesses with the GDC FIFO. When it is 1, the Empty flag ensures that all the commands and parameters previously sent to the GDC have been processed.

**SR-1: FIFO Full**: A 1 at this flag indicates a full FIFO in the GDC. A 0 ensures that there is room for at least one byte. This flag needs to be checked before each write into the GDC.

**SR-0: Data Ready**: When this flag is a 1, it indicates that a byte is available to be read by the system microprocessor. This bit must be tested before each read operation. It drops to a 0 while the data is transferred from the FIFO into the microprocessor interface data register.

---

**FIFO Operation & Command Protocol**

The first-in, first-out buffer (FIFO) in the GDC handles the command dialogue with the system microprocessor. This flow of information uses a half-duplex technique, in which the single 16-location FIFO is used for both directions of data movement, one direction at a time. The FIFO's direction is controlled by the system microprocessor through the GDC's command set. The microprocessor coordinates these transfers by checking the appropriate status register bits.

The command protocol used by the GDC requires the differentiation of the first byte of a command sequence from the succeeding bytes. This first byte contains the operation code and the remaining bytes carry parameters. Writing into the GDC causes the FIFO to store a flag value alongside the data byte to signify whether the byte was written into the command or the parameter address. The command processor in the GDC tests this bit as it interprets the entries in the FIFO.
The receipt of a command byte by the command processor marks the end of any previous operation. The number of parameter bytes supplied with a command is cut short by the receipt of the next command byte. A read operation from the GDC to the microprocessor can be terminated at any time by the next command.

The FIFO changes direction under the control of the system microprocessor. Commands written into the GDC always put the FIFO into write mode if it wasn't in it already. If it was in read mode, any read data in the FIFO at the time of the turnaround is lost. Commands which require a GDC response, such as RDAT, CURD and LPRD, put the FIFO into read mode after the command is interpreted by the GDC's command processor. Any commands and parameters behind the read-evoking command are discarded when the FIFO direction is reversed.

Read-Modify-Write Cycle

Data transfers between the GDC and the display memory are accomplished using a read-modify-write (RMW) memory cycle. The four clock period timing of the RMW cycle is used to: 1) output the address, 2) read data from the memory, 3) modify the data, and 4) write the modified data back into the initially selected memory address. This type of memory cycle is used for all interactions with display memory including DMA transfers, except for the two clock period display and RAM refresh cycles.

The operations performed during the modify portion of the RMW cycle merit additional explanation. The circuitry in the GDC uses three main elements: the Pattern register, the Mask register, and the 16-bit Logic unit. The Pattern register holds the data pattern to be moved into memory. It is loaded by the WDAT command or, during drawing, from the parameter RAM. The Mask register contents determine which bits of the read data will be modified. Based on the contents of these registers, the Logic unit performs the selected operations of REPLACE, COMPLEMENT, SET, or CLEAR on the data read from display memory.

The Pattern register contents are ANDed with the Mask register contents to enable the actual modification of the memory read data, on a bit-by-bit basis. For graphics drawing, one bit at a time from the Pattern register is combined with the Mask. When ANDed with the bit set to a 1 in the Mask register, the proper single pixel is modified by the Logic Unit. For the next pixel in the figure, the next bit in the Pattern register is selected and the Mask register bit is moved to identify the pixel's location within the word. The Execution word address pointer register, EAD, is also adjusted as required to address the word containing the next pixel.

In character mode, all of the bits in the Pattern register are used in parallel to form the respective bits of the modify data word. Since the bits of the character code word are used in parallel, unlike the one-bit-at-a-time graphics drawing process, this facility allows any or all of the bits in a memory word to be modified in one RMW memory cycle. The Mask register must be loaded with 1s in the positions where modification is to be permitted.

The Mask register can be loaded in either of two ways. In graphics mode, the CURS command contains a four-bit dAD field to specify the dot address. The command processor converts this parameter into the one-of-16 format used in the Mask register for figure drawing. A full 16 bits can be loaded into the Mask register using the MASK command. In addition to the character mode use mentioned above, the 16-bit MASK load is convenient in graphics mode when all of the pixels of a word are to be set to the same value.

The Logic unit combines the data read from display memory, the Pattern register, and the Mask register to generate the data to be written back into display memory. Any one of four operations can be selected: REPLACE, COMPLEMENT, CLEAR or SET. In each case, if the respective Mask bit is 0, that particular bit of the read data is returned to memory unmodified. If the Mask bit is 1, the modification is enabled. With the REPLACE operation, the modify data simply takes the place of the read data for modification enabled bits. For the other three operations, a 0 in the modify data allows the read data bit to be returned to memory. A 1 value causes the specified operation to be performed in the bit positions with set Mask bits.

Figure Drawing

The GDC draws graphics figures at the rate of one pixel per read-modify-write (RMW) display memory cycle. These cycles take four clock periods to complete. At a clock frequency of 5 MHz, this is equal to 800 ns. During the RMW cycle the GDC simultaneously calculates the address and position of the next pixel to be drawn.

The graphics figure drawing process depends on the display memory addressing structure. Groups of 16 horizontally adjacent pixels form the 16-bit words
which are handled by the GDC. Display memory is organized as a linearly addressed space of these words. Addressing of individual pixels is handled by the GDC’s internal RMW logic.

During the drawing process, the GDC finds the next pixel of the figure which is one of the eight nearest neighbors of the last pixel drawn. The GDC assigns each of these eight directions a number from 0 to 7, starting with straight down and proceeding counterclockwise.

Figure 8 summarizes these operations for each direction.

Whole word drawing is useful for filling areas in memory with a single value. By setting the Mask register to all 1s with the MASK command, both the LSB and MSB of the dAD will always be 1, so that the EAD value will be incremented or decremented for each cycle regardless of direction. One RMW cycle will be able to affect all 16 bits of the word for any drawing type. One bit in the Pattern register is used per RMW cycle to write all the bits of the word to the same value. The next Pattern bit is used for the word, etc.

<table>
<thead>
<tr>
<th>DIR</th>
<th>ADDRESS OPERATION(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EAD = EAD + P</td>
</tr>
<tr>
<td>1</td>
<td>EAD = EAD + P</td>
</tr>
<tr>
<td></td>
<td>If dAD.MSB = 1 then EAD = EAD + 1</td>
</tr>
<tr>
<td></td>
<td>dAD = LR(dAD)</td>
</tr>
<tr>
<td>2</td>
<td>EAD = EAD + P</td>
</tr>
<tr>
<td></td>
<td>If dAD.MSB = 1 then EAD = EAD + 1</td>
</tr>
<tr>
<td></td>
<td>dAD = LR(dAD)</td>
</tr>
<tr>
<td>3</td>
<td>EAD = EAD - P</td>
</tr>
<tr>
<td></td>
<td>If dAD.MSB = 1 then EAD = EAD - 1</td>
</tr>
<tr>
<td></td>
<td>dAD = LR(dAD)</td>
</tr>
<tr>
<td>4</td>
<td>EAD = EAD - P</td>
</tr>
<tr>
<td>5</td>
<td>EAD = EAD - P</td>
</tr>
<tr>
<td></td>
<td>If dAD.LSB = 1 then EAD = EAD - 1</td>
</tr>
<tr>
<td></td>
<td>dAD = RR(dAD)</td>
</tr>
<tr>
<td>6</td>
<td>EAD = EAD - P</td>
</tr>
<tr>
<td></td>
<td>If dAD.LSB = 1 then EAD = EAD - 1</td>
</tr>
<tr>
<td></td>
<td>dAD = RR(dAD)</td>
</tr>
<tr>
<td>7</td>
<td>EAD = EAD + P</td>
</tr>
<tr>
<td></td>
<td>If dAD.LSB = 1 then EAD = EAD - 1</td>
</tr>
<tr>
<td></td>
<td>dAD = RR(dAD)</td>
</tr>
</tbody>
</table>

WHERE
P = PITCH, LR = LEFT ROTATE, RR = RIGHT ROTATE  
CAD = CURSOR ADDRESS  
dAD = DOT ADDRESS  
LSB = LEAST SIGNIFICANT BIT  
MSB = MOST SIGNIFICANT BIT

Figure 8. Address Calculation Details

Figure 7. Drawing Directions

Figure drawing requires the proper manipulation of the address and the pixel bit position according to the drawing direction to determine the next pixel of the figure. To move to the word above or below the current one, it is necessary to subtract or add the number of words per line in display memory. This parameter is called the pitch. To move to the word to either side, the Execute word address cursor, EAD, must be incremented or decremented as the dot address pointer bit reaches the LSB or the MSB of the Mask register. To move to a pixel within the same word, it is necessary to rotate the dot address pointer register to the right or left.
For the various figures, the effect of the initial direction upon the resulting drawing is shown in figure 9.

Note that during line drawing, the angle of the line may be anywhere within the shaded octant defined by the DIR value. Arc drawing starts in the direction initially specified by the DIR value and veers into an arc as drawing proceeds. An arc may be up to 45 degrees in length. DMA transfers are done on word boundaries only, and follow the arrows indicated in the table to find successive word addresses. The slanted paths for DMA transfers indicate the GDC changing both the X and Y components of the word address when moving to the next word. It does not follow a 45 degree diagonal path by pixels.

<table>
<thead>
<tr>
<th>Dir</th>
<th>Line</th>
<th>Arc</th>
<th>Character</th>
<th>Slant Char</th>
<th>Rectangle</th>
<th>DMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td><img src="image1" alt="Line" /></td>
<td><img src="image2" alt="Arc" /></td>
<td><img src="image3" alt="Character" /></td>
<td><img src="image4" alt="Slant Char" /></td>
<td><img src="image5" alt="Rectangle" /></td>
<td><img src="image6" alt="DMA" /></td>
</tr>
<tr>
<td>001</td>
<td><img src="image7" alt="Line" /></td>
<td><img src="image8" alt="Arc" /></td>
<td><img src="image9" alt="Character" /></td>
<td><img src="image10" alt="Slant Char" /></td>
<td><img src="image11" alt="Rectangle" /></td>
<td><img src="image12" alt="DMA" /></td>
</tr>
<tr>
<td>010</td>
<td><img src="image13" alt="Line" /></td>
<td><img src="image14" alt="Arc" /></td>
<td><img src="image15" alt="Character" /></td>
<td><img src="image16" alt="Slant Char" /></td>
<td><img src="image17" alt="Rectangle" /></td>
<td><img src="image18" alt="DMA" /></td>
</tr>
<tr>
<td>011</td>
<td><img src="image19" alt="Line" /></td>
<td><img src="image20" alt="Arc" /></td>
<td><img src="image21" alt="Character" /></td>
<td><img src="image22" alt="Slant Char" /></td>
<td><img src="image23" alt="Rectangle" /></td>
<td><img src="image24" alt="DMA" /></td>
</tr>
<tr>
<td>100</td>
<td><img src="image25" alt="Line" /></td>
<td><img src="image26" alt="Arc" /></td>
<td><img src="image27" alt="Character" /></td>
<td><img src="image28" alt="Slant Char" /></td>
<td><img src="image29" alt="Rectangle" /></td>
<td><img src="image30" alt="DMA" /></td>
</tr>
<tr>
<td>101</td>
<td><img src="image31" alt="Line" /></td>
<td><img src="image32" alt="Arc" /></td>
<td><img src="image33" alt="Character" /></td>
<td><img src="image34" alt="Slant Char" /></td>
<td><img src="image35" alt="Rectangle" /></td>
<td><img src="image36" alt="DMA" /></td>
</tr>
<tr>
<td>110</td>
<td><img src="image37" alt="Line" /></td>
<td><img src="image38" alt="Arc" /></td>
<td><img src="image39" alt="Character" /></td>
<td><img src="image40" alt="Slant Char" /></td>
<td><img src="image41" alt="Rectangle" /></td>
<td><img src="image42" alt="DMA" /></td>
</tr>
<tr>
<td>111</td>
<td><img src="image43" alt="Line" /></td>
<td><img src="image44" alt="Arc" /></td>
<td><img src="image45" alt="Character" /></td>
<td><img src="image46" alt="Slant Char" /></td>
<td><img src="image47" alt="Rectangle" /></td>
<td><img src="image48" alt="DMA" /></td>
</tr>
</tbody>
</table>

Figure 9. Effect of the Direction Parameter
Drawing Parameters

In preparation for graphics figure drawing, the GDC's Drawing Processor needs the figure type, direction and drawing parameters, the starting pixel address, and the pattern from the microprocessor. Once these are in place within the GDC, the Figure Draw command, FIGD, initiates the drawing operation. From that point on, the system microprocessor is not involved in the drawing process. The GDC Drawing Processor coordinates the RMW circuitry and address registers to draw the specified figure pixel by pixel.

The algorithms used by the processor for figure drawing are designed to optimize its drawing speed. To this end, the specific details about the figure to be drawn are reduced by the microprocessor to a form conducive to high-speed address calculations within the GDC. In this way the repetitive, pixel-by-pixel calculations can be done quickly, thereby minimizing the overall figure drawing time. Figure 3 summarizes the parameters.

Graphics Character Drawing

Graphics characters can be drawn into display memory pixel-by-pixel. The up to 8-by-8 character is loaded into the GDC's parameter RAM by the system microprocessor. Consequently, there are no limitations on the character set used. By varying the drawing parameters and drawing direction, numerous drawing options are available. In area fill applications, a character can be written into display memory as many times as desired without reloading the parameter RAM.

Once the parameter RAM has been loaded with up to eight graphics character bytes by the appropriate PRAM command, the GCHRD command can be used to draw the bytes into display memory starting at the cursor. The zoom magnification factor for writing, set by the zoom command, controls the size of the character written into the display memory in integer multiples of 1 through 16. The bit values in the PRAM are repeated horizontally and vertically the number of times specified by the zoom factor.

The movement of these PRAM bytes to the display memory is controlled by the parameters of the FIGS command. Based on the specified height and width of the area to be drawn, the parameter RAM is scanned to fill the required area.

For an 8-by-8 graphics character, the first pixel drawn uses the LSB of RA-15, the second pixel uses bit 1 of RA-15, and so on, until the MSB of RA-15 is reached. The GDC jumps to the corresponding bit in RA-14 to continue the drawing. The progression then advances toward the LSB of RA-14. This snaking sequence is continued for the other 6 PRAM bytes. This progression matches the sequence of display memory addresses calculated by the drawing processor as shown in figure 9. If the area is narrower than 8 pixels wide, the snaking will advance to the next PRAM byte before the MSB is reached. If the area is less than 8 lines high, fewer bytes in the parameter RAM will be scanned. If the area is larger than 8 by 8, the GDC will repeat the contents of the parameter RAM in two dimensions.
Parameter RAM Contents

The parameters stored in the parameter RAM, PRAM, are available for the GDC to refer to repeatedly during figure drawing and raster-scanning. In each mode of operation the values in the PRAM are interpreted by the GDC in a predetermined fashion. The host microprocessor must load the appropriate parameters into the proper PRAM locations. PRAM loading command allows the host to write into any location of the PRAM and transfer as many bytes as desired. In this way any stored parameter byte or bytes may be changed without influencing the other bytes.

The PRAM stores two types of information. For specifying the details of the display area partitions, blocks of four bytes are used. The four parameters stored in each block include the starting address in display memory of each display area, and its length.

In addition, there are two mode bits for each area which specify whether the area is a bit-mapped graphics area or a coded character area, and whether a normal or wide display cycle is to be used for that area.

The other use for the PRAM contents is to supply the pattern for figure drawing when in a bit-mapped graphics area or mode. In these situations, PRAM bytes 8 through 16 are reserved for this patterning information. For line, arc, and rectangle drawing (linear figures) locations 8 and 9 are loaded into the Pattern register to allow the GDC to draw dotted, dashed, etc. lines. For area filling and graphics bit-mapped character drawing locations 8 through 15 are referenced for the pattern or character to be drawn.

Details of the bit assignments are shown on the following pages for the various modes of operation.
DISPLAY PARTITION AREA 1 STARTING ADDRESS WITH LOW AND HIGH SIGNIFICANCE FIELDS (WORD ADDRESS).

LENGTH OF DISPLAY PARTITION 1 (LINE COUNT) WITH LOW AND HIGH SIGNIFICANCE FIELDS.

THE IMAGE BIT AFFECTS THE OPERATION OF THE DISPLAY ADDRESS COUNTER IN CHARACTER MODE. IF THE IMAGE BIT IS ZERO, IT WILL INCREMENT BY ONE AFTER EACH READ CYCLE. IF THE IMAGE BIT IS SET, IT WILL INCREMENT BY ONE AFTER EVERY TWO READ CYCLES.

A WIDE DISPLAY CYCLE WIDTH OF TWO WORDS PER MEMORY CYCLE IS SELECTED FOR THIS DISPLAY AREA IF THIS BIT IS SET TO A '1'. THE DISPLAY ADDRESS COUNTER IS THEN INCREMENTED BY 2 FOR EACH DISPLAY SCAN CYCLE. OTHER MEMORY CYCLE TYPES ARE NOT INFLUENCED.

DISPLAY PARTITION 2 STARTING ADDRESS AND LENGTH

DISPLAY PARTITION 3 STARTING ADDRESS AND LENGTH

DISPLAY PARTITION 4 STARTING ADDRESS AND LENGTH

Figure 10. Parameter RAM Contents—Character Mode
Figure 11. Parameter RAM Contents—Graphics and Mixed Graphics and Character Modes
Figure 12. Command Bytes Summary

VIDEO CONTROL COMMANDS

RESET COMMAND
This command can be executed at any time and does not modify any of the parameters already loaded into the GDC.

If followed by parameter bytes, this command also sets the sync generator parameters as described below. Idle mode is exited with the START command.
82720

MODE CONTROL BITS.
SEE FIGURE 15.

ACTIVE DISPLAY WORDS PER LINE - 2. MUST
BE EVEN NUMBER WITH BIT 0 = 0.

HORIZONTAL SYNCH WIDTH -1
VERTICAL SYNCH WIDTH, LOW BITS

HORIZONTAL FRONT PORCH WIDTH -1
VERTICAL SYNCH WIDTH, HIGH BITS

HORIZONTAL BACK PORCH WIDTH -1.
VERTICAL FRONT PORCH WIDTH

ACTIVE DISPLAY LINES PER VIDEO FIELD,
LOW BITS

ACTIVE DISPLAY LINES PER VIDEO FIELD,
HIGH BITS

VERTICAL BACK PORCH WIDTH

Figure 14. Optional Reset Parameters

In graphics mode, a word is a group of 16 pixels. In
character mode, a word is one character code and its
attributes, if any.

The number of active words per line must be an even
number from 2 to 256.

An all-zero parameter value selects a count equal to
2^n where n = number of bits in the parameter field for
vertical parameters.

All horizontal widths are counted in display words.
All vertical intervals are counted in lines.

Sync Parameter Constraints

HORIZONTAL FRONT PORCH CONSTRAINTS
1. In general:
   HFP ≥ 2 words
2. If DMA is used, or the display zoom factor is greater
   than one in interlaced display mode:
   HFP ≥ 3 words
3. If the GDC is used in slave mode:
   HFP ≥ 4 words
4. If the light pen input is used:
   HFP ≥ 6 words

HORIZONTAL Sync CONSTRAINTS
1. If dynamic RAM refresh is used:
   HS ≥ 2 words
2. If interlaced display mode is used:
   HS ≥ 5 words

HORIZONTAL BACK PORCH CONSTRAINTS
1. In general:
   HBP ≥ 3 words
2. If interlaced display mode is used, or the IMAGE or
   WIDE mode bits change within one video field:
   HBP ≥ 5 words

MODE CONTROL BITS (FIGURE 15)

Repeat Field Framing: 2 Field Sequence with ½
line offset between otherwise identical fields.

Interlaced Framing: 2 Field Sequence with ½
line offset. Each field displays alternate lines.

Noninterlaced Framing: 1 field brings all of the in-
formation to the screen.

Total scanned lines in interlace mode is odd. The
sum of VFP + VS + VBP + AL should equal one less
than the desired odd number of lines.

Dynamic RAM refresh is important when high display
zoom factors or DMA are used in such a way that not
all of the rows in the RAMs are regularly accessed
during display raster generation and for otherwise
inactive display memory.

Access to display memory can be limited to retrace
blanking intervals only, so that no disruptions of the
image are seen on the screen.
<table>
<thead>
<tr>
<th>C G</th>
<th>DISPLAY MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>MIXED GRAPHICS &amp; CHARACTER</td>
</tr>
<tr>
<td>0 1</td>
<td>GRAPHICS MODE</td>
</tr>
<tr>
<td>1 0</td>
<td>CHARACTER MODE</td>
</tr>
<tr>
<td>1 1</td>
<td>INVALID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1 S</th>
<th>VIDEO FRAMING</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>NONINTERLACED</td>
</tr>
<tr>
<td>0 1</td>
<td>INVALID</td>
</tr>
<tr>
<td>1 0</td>
<td>INTERLACED REPEAT FIELD FOR CHARACTER DISPLAYS</td>
</tr>
<tr>
<td>1 1</td>
<td>INTERLACED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>DYNAMIC RAM REFRESH CYCLES ENABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO REFRESH—STATIC RAM</td>
</tr>
<tr>
<td>1</td>
<td>REFRESH—DYNAMIC RAM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F</th>
<th>DRAWING TIME WINDOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DRAWING DURING ACTIVE DISPLAY TIME AND RETRACE BLANKING</td>
</tr>
<tr>
<td>1</td>
<td>DRAWING ONLY DURING RETRACE BLANKING</td>
</tr>
</tbody>
</table>

Figure 15. Mode Control Bits

Figure 16. Sync Command
SYNC Format Specify Command

This command loads parameters into the sync generator. The various parameter fields and bits are identical to those at the RESET command. The GDC is not reset nor does it enter idle mode.

Vertical Sync Mode Command

When using two or more GDCs to contribute to one image, one GDC is defined as the master sync generator, and the others operate as its slaves. The VSYNC pins of all GDCs are connected together.

Slave Mode Operation

A few considerations should be observed when synchronizing two or more GDCs to generate overlaid video via the VSYNC INPUT/OUTPUT pin. As mentioned above, the Horizontal Front Porch (HFP) must be 4 or more display cycles wide. This is equivalent to eight or more clock cycles. This gives the slave GDCs time to initialize their internal video sync generators to the proper point in the video field to match the incoming vertical sync pulse (VSYNC). This resetting of the generator occurs just after the end of the incoming VSYNC pulse, during the HFP interval. Enough time during HFP is required to allow the slave GDC to complete the operation before the start of the HSYNC interval.

Once the GDCs are initialized and set up as Master and Slaves, they must be given time to synchronize. It is a good idea to watch the VSYNC status bit of the Master GDC and wait until after one or more VSYNC pulses have been generated before the display process is started. The START command will begin the active display of data and will end the video synchronization process, so be sure there has been at least one VSYNC pulse generated for the Slaves to synchronize to.

Example:

- VSYNC: 0 1 1 0 1 1 1 M
  - 0—ACCEPT EXTERNAL VERTICAL SYNC—SLAVE MODE
  - 1—GENERATE & OUTPUT VERTICAL SYNC—MASTER MODE

Figure 17. Vertical Sync Mode Command

Example:

- CCHAR: 0 1 0 0 1 0 1 1
  - P1: DC 0 0 LR
    - DISPLAY CURSOR IF 1
  - P2: BR L SC CTOP
    - CURSOR TOP LINE NUMBER IN THE ROW
    - 0—BLINKING CURSOR
    - 1—STEADY CURSOR
    - BLINK RATE, LOWER BITS
  - P3: CBOT BR L
    - CURSOR BOTTOM LINE NUMBER IN THE ROW
    - BLINK RATE, UPPER BITS

Figure 18. Cursor & Character Characteristics Command
Cursor and Character Characteristics Command

In graphics mode, LR should be set to 0. For interlaced displays in graphics mode, BR should be set to 3. The blink rate parameter controls both the cursor and attribute blink rates. The cursor blink-on-time = blink-off-time = \(2 \times BR\) (video frames). The attribute blink rate is always \(\frac{1}{2}\) the cursor rate but with a \(\frac{3}{4}\) on-\(\frac{1}{4}\) off duty cycle.

DISPLAY CONTROL COMMANDS

Zoom Factors Specify Command

Zoom magnification factors of 1 through 16 are available using codes 0 through 15, respectively.

Cursor Position Specify Command

In character mode, the third parameter byte is not needed. The cursor is displayed for the word time in which the display scan address (DAD) equals the cursor address. In graphics mode, the cursor word address specifies the word containing the starting pixel of the drawing; the dot address value specifies the pixel within that word.

Parameter RAM Load Command

From the starting address, SA, any number of bytes may be loaded into the parameter RAM at incrementing addresses, up to location 15. The sequence of parameter bytes is terminated by the next command byte entered into the FIFO. The parameter RAM stores 16 bytes of information in predefined locations which differ for graphics and character modes. See the parameter RAM discussion for bit assignments.

Pitch Specification Command

This value is used during drawing by the drawing processor to find the word directly above or below the current word, and during display to find the start of the next line.

The Pitch parameter (width of display memory) is set by two different commands. In addition to the PITCH command, the RESET (or SYNC) command also sets the pitch value. The "active words per line" parameter, which specifies the width of the raster-scan display, also sets the Pitch of the display memory. In situations in which these two values are equal there is no need to execute a PITCH command.

![Figure 19. Display Control Commands](image-url)
DRAWING CONTROL COMMANDS

Write Data Command

Upon receiving a set of parameters (two bytes for a word transfer, one for a byte transfer), one RMW cycle into Video Memory is done at the address pointed to by the cursor EAD. The EAD pointer is advanced to the next word, according to the previously specified direction. More parameters can then be accepted.

For byte writes, the unspecified byte is treated as all zeros during the RMW memory cycle.

In graphics bit-map situations, only the LSB of the WDAT parameter bytes is used as the pattern in the RMW operations. Therefore it is possible to have only an all ones or all zeros pattern. In coded character applications all the bits of the WDAT parameters are used to establish the drawing pattern.

The WDAT command operates differently from the other commands which initiate RMW cycle activity. It requires parameters to set up the Pattern register while the other commands use the stored values in the parameter RAM. Like all of these commands, the
WDAT command must be preceded by a FIGS command and its parameters. Only the first three parameters need be given following the FIGS opcode, to set up the type of drawing, the DIR direction, and the DC value. The DC parameter +1 will be the number of RMW cycles done by the GDC with the first set of WDAT parameters. Additional sets of WDAT parameters will see a DC value of 0 which will cause only one RMW cycle to be executed.

Figure 24. Figure Drawing Parameters Specify Command
Mask Register Load Command

This command sets the value of the 16-bit Mask register of the figure drawing processor. The Mask register controls which bits can be modified in the display memory during a read-modify-write cycle.

The Mask register is loaded both by the MASK command and the third parameter byte of the CURS command. The MASK command accepts two parameter bytes to load a 16-bit value into the Mask register. All 16 bits can be individually one or zero, under program control. The CURS command on the other hand, puts a “1 of 16” pattern into the Mask register based on the value of the Dot Address value, dAD. If normal single-pixel-at-a-time graphics figure drawing is desired, there is no need to do a MASK command at all since the CURS command will set up the proper pattern to address the proper pixels as drawing progresses. For coded character DMA, and screen setting and clearing operations using the WDAT command, the MASK command should be used after the CURS command if its third parameter byte has been output. The Mask register should be set to all ones for any “word-at-a-time” operation.

Figure Draw Start Command

On execution of this instruction, the GDC loads the parameters from the parameter RAM into the drawing processor and starts the drawing process at the pixel pointed to by the cursor, EAD, and the dot address, dAD.

Graphics Char. Draw and Area Fill Start Command

Based on parameters loaded with the FIGS command, this command initiates the drawing of the graphics character or area filling pattern stored in Parameter RAM. Drawing begins at the address in display memory pointed to by the EAD and dAD values.

DATA READ COMMANDS

Read Data Command

Using the DIR and DC parameters of the FIGS command to establish direction and transfer count, multiple RMW cycles can be executed without specification of the cursor address after the initial load (DC = number of words or bytes).

As this instruction begins to execute, the FIFO buffer direction is reversed so that the data read from display memory can pass to the microprocessor. Any commands or parameters in the FIFO at this time will be lost. A command byte sent to the GDC will immediately reverse the buffer direction back to write mode, and all RDAT information not yet read from the FIFO will be lost. MOD should be set to 00.

Cursor Address Read Command

The Execute Address, EAD, points to the display memory word containing the pixel to be addressed.

The Dot Address, dAD, within the word is represented as a 1-of-16 code.

Light Pen Address Read Command

The light pen address, LAD, corresponds to the display memory address, DAD, at which the light pen input signal is detected and deglitched.
### Figure 28. Cursor Address Read Command

**CURD:** $1, 1, 0, 0, 0, 0$

The following bytes are returned by the GDC:

- **P1**: $\text{EAD}_2, \text{EAD}_1, \text{AD}_3$
  - Execute address (EAD), low byte
- **P2**: $\text{A15}_2, \text{EAD}_2, \text{AD}_3$
  - Execute address (EAD), middle byte
- **P3**: $\text{X, X, X, X, X, EAD}_1$  
  - Execute address (EAD), high bits
- **P4**: $\text{X, X, X, X, X, X, X, X, AD}_1$
  - Dot address (OAD), low byte
- **P5**: $\text{X, X, X, X, X, X, X, X, X, X, AD}_1$
  - Dot address (OAD), high byte

**X** = Undefined

### Figure 29. Light Pen Address Read Command

**LPXD:** $1, 1, 0, 0, 0, 0$

The following bytes are returned by the GDC:

- **P1**: $\text{AD}_2, \text{EAD}_1, \text{AD}_3$
  - Light pen address, low byte
- **P2**: $\text{A15}_2, \text{EAD}_2, \text{AD}_3$
  - Light pen address, middle byte
- **P3**: $\text{X, X, X, X, AD}_1$
  - Light pen address, high bits

### Figure 30. DMA Control Commands

**DMA READ REQUEST**

**DMAR**: $1, 0, 1, \text{MOD}$

- **DATA TRANSFER TYPE:**
  - 0 0 ← Word, low then high byte
  - 1 0 ← Low byte of the word
  - 1 1 ← High byte of the word
  - 0 1 ← INVALID

**DMA WRITE REQUEST**

**DMAW**: $0, 0, 1, \text{MOD}$

- **RMW MEMORY LOGICAL OPERATION:**
  - 0 0 ← Replace with pattern
  - 0 1 ← Complement
  - 1 0 ← Reset to zero
  - 1 1 ← Set to one

- **DATA TRANSFER TYPE:**
  - 0 0 ← Word, low then high byte
  - 1 0 ← Low byte of the word
  - 1 1 ← High byte of the word
  - 0 1 ← INVALID
**ABSOLUTE MAXIMUM RATINGS**

Ambient Temperature Under Bias ............ 0°C to 70°C
Storage Temperature .......................... -65°C to 150°C
Voltage on any Pin with Respect to Ground ................ -0.5V to +7V
Power Dissipation ............................ 1.5 Watt

*COMMENT: Exposing the device to stresses above those listed in Absolute Maximum Ratings could cause permanent damage. The device is not meant to be operated under conditions outside the limits described in the operational sections of this specification. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

**DC CHARACTERISTICS**

\( T_A = 0°C \) to 70°C; \( V_{CC} = 5V \pm 10\% \); GND = 0V

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{IL} )</td>
<td>Input Low Voltage</td>
<td>Min. -0.5</td>
<td>Max. 0.8</td>
<td>V</td>
</tr>
<tr>
<td>( V_{IH} )</td>
<td>Input High Voltage Except DACK</td>
<td>Min. 2.2</td>
<td>Max. ( V_{CC} + 0.5 )</td>
<td>V</td>
</tr>
<tr>
<td>( V_{OL} )</td>
<td>Output Low Voltage</td>
<td>Min. 0.45</td>
<td>Max. V</td>
<td>( I_{OL} = 2.2 ) mA</td>
</tr>
<tr>
<td>( V_{OH} )</td>
<td>Output High Voltage</td>
<td>Min. 2.4</td>
<td>Max. V</td>
<td>( I_{OH} = -400 ) ( \mu )A</td>
</tr>
<tr>
<td>( I_{OZ} )</td>
<td>Output Leakage Current</td>
<td>Min. ( \pm 10 )</td>
<td>Max. ( \mu )A</td>
<td>( V_{SS} + 0.45 \leq V_{I} \leq V_{CC} )</td>
</tr>
<tr>
<td>( I_{IL} )</td>
<td>Input Leakage Current</td>
<td>Min. ( \pm 10 )</td>
<td>Max. ( \mu )A</td>
<td>( V_{SS} &lt; V_{I} &lt; V_{CC} )</td>
</tr>
<tr>
<td>( V_{CL} )</td>
<td>Clock Input Low Voltage</td>
<td>Min. -0.5</td>
<td>Max. 0.6</td>
<td>V</td>
</tr>
<tr>
<td>( V_{CH} )</td>
<td>Clock Input High Voltage</td>
<td>Min. 3.5</td>
<td>Max. ( V_{CC} + 0.5 )</td>
<td>V</td>
</tr>
<tr>
<td>( I_{CC} )</td>
<td>( V_{CC} ) Supply Current</td>
<td>Min. 270</td>
<td>Max. mA</td>
<td>Typical = 150 mA</td>
</tr>
<tr>
<td>( V_{IH1} )</td>
<td>Input High Voltage DACK Only</td>
<td>Min. 2.4</td>
<td>Max. ( V_{CC} + 0.5 )</td>
<td>V</td>
</tr>
</tbody>
</table>

**CAPACITANCE**

\( T_A = 25°C \); \( V_{CC} = \) GND = 0V

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{IN} )</td>
<td>Input Capacitance</td>
<td>Min. 10</td>
<td>Max. pF</td>
<td>( f_c = 1 ) MHz</td>
</tr>
<tr>
<td>( C_{IO} )</td>
<td>I/O Capacitance</td>
<td>Min. 20</td>
<td>Max. pF</td>
<td>( V = 0 )</td>
</tr>
<tr>
<td>( C_{OUT} )</td>
<td>Output Capacitance</td>
<td>Min. 20</td>
<td>Max. pF</td>
<td></td>
</tr>
<tr>
<td>( C_{O} )</td>
<td>Clock Input Capacitance</td>
<td>Min. 20</td>
<td>Max. pF</td>
<td></td>
</tr>
</tbody>
</table>

(1) Suggest pull up resistor to reduce noise sensitivity on DACK only.
(2) Sample tested initially.
### A.C. CHARACTERISTICS (TA = 0°C to +70°C, VSS = 0V, VCC = +5V ± 10%)

#### DATA BUS READ CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>82720</th>
<th>82720-1</th>
<th>82720-2</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAR</td>
<td>A₀ setup to RD</td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
</tr>
<tr>
<td>TRA</td>
<td>A₀ hold after RD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>TRR</td>
<td>RD Pulse Width</td>
<td>Tₚ₋₂₀</td>
<td>Tₚ₋₂₀</td>
<td>Tₚ₋₂₀</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRD</td>
<td>RD to Data Out Delay</td>
<td>120</td>
<td>80</td>
<td>70</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRF</td>
<td>RD to Data Float Delay</td>
<td>0</td>
<td>120</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>TRAV</td>
<td>RD Recovery Time</td>
<td>4 Tₚ₋₂₀</td>
<td>4 Tₚ₋₂₀</td>
<td>4 Tₚ₋₂₀</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

#### DATA BUS WRITE CYCLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>82720</th>
<th>82720-1</th>
<th>82720-2</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAW</td>
<td>A₀ Setup to WR</td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
</tr>
<tr>
<td>TWA</td>
<td>A₀ Hold after WR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>TWW</td>
<td>WR Pulse Width</td>
<td>120</td>
<td>100</td>
<td>90</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TWD</td>
<td>Data Setup to WR</td>
<td>100</td>
<td>80</td>
<td>70</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRV</td>
<td>WR Recovery Time</td>
<td>4 Tₚ₋₂₀</td>
<td>4 Tₚ₋₂₀</td>
<td>4 Tₚ₋₂₀</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

#### DISPLAY MEMORY TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>82720</th>
<th>82720-1</th>
<th>82720-2</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCA</td>
<td>Address/Data Delay from 2XWCLK</td>
<td>30</td>
<td>160</td>
<td>30</td>
<td>130</td>
<td>30</td>
</tr>
<tr>
<td>TAC</td>
<td>Address/Data Hold Time</td>
<td>30</td>
<td>160</td>
<td>30</td>
<td>130</td>
<td>30</td>
</tr>
<tr>
<td>TDC</td>
<td>Input Data Setup to 2XWCLK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>TCD</td>
<td>Input Data Hold Time</td>
<td>TIE</td>
<td>TIE</td>
<td>TIE</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TIE</td>
<td>2XWCLK to DBIN</td>
<td>30</td>
<td>120</td>
<td>30</td>
<td>90</td>
<td>30</td>
</tr>
<tr>
<td>TCAH</td>
<td>2XWCLK to ALE</td>
<td>30</td>
<td>125</td>
<td>30</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>TCAL</td>
<td>2XWCLK to ALE</td>
<td>15</td>
<td>100</td>
<td>15</td>
<td>80</td>
<td>15</td>
</tr>
<tr>
<td>TAL</td>
<td>ALE Low Time</td>
<td>Tₚ₋₂₀ + 30</td>
<td>Tₚ₋₂₀ + 30</td>
<td>Tₚ₋₂₀ + 30</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TAH</td>
<td>ALE High Time</td>
<td>Tₚ₋₂₀ - 20</td>
<td>Tₚ₋₂₀ - 20</td>
<td>Tₚ₋₂₀ - 20</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCD</td>
<td>Video Signal Delay from 2XWCLK</td>
<td>150</td>
<td>120</td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address Valid Hold Time After ALE</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TVAL</td>
<td>Address Valid Hold Time Before ALE</td>
<td>20</td>
<td>10</td>
<td>5</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

7-117
A.C. CHARACTERISTICS (Continued)

### OTHER TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>82720</th>
<th>82720-1</th>
<th>82720-2</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T_{PC})</td>
<td>LPEN or VSYNC Input Setup to 2XWCLK1</td>
<td>30</td>
<td>20</td>
<td>15</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(T_{PP})</td>
<td>LPEN or VSYNC Input Pulse Width</td>
<td>(T_{CY})</td>
<td>(T_{CY})</td>
<td>(T_{CY})</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### CLOCK TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>82720</th>
<th>82720-1</th>
<th>82720-2</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T_{CY})</td>
<td>Clock Period</td>
<td>250</td>
<td>2000</td>
<td>200</td>
<td>2000</td>
<td>180</td>
</tr>
<tr>
<td>(T_{CH})</td>
<td>Clock High Time</td>
<td>105</td>
<td>80</td>
<td>70</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(T_{CL})</td>
<td>Clock Low Time</td>
<td>105</td>
<td>80</td>
<td>70</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(T_{R})</td>
<td>Rise Time</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(T_{F})</td>
<td>Fall Time</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### DMA TIMING

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>82720</th>
<th>82720-1</th>
<th>82720-2</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T_{ACC})</td>
<td>DACK Setup to RD I or WR I</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(T_{CAC})</td>
<td>DACK Hold from RD I or WR I</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(T_{RD1})</td>
<td>RD I to Data Out Delay</td>
<td>1.5 (T_{CY}) + 120</td>
<td>1.5 (T_{CY}) + 80</td>
<td>1.5 (T_{CY}) + 70</td>
<td>ns</td>
<td>CL = 50pF</td>
</tr>
<tr>
<td>(T_{RD2})</td>
<td>2XWCLK1 to DREQ Delay</td>
<td>150</td>
<td>120</td>
<td>100</td>
<td>ns</td>
<td>CL = 50pF</td>
</tr>
<tr>
<td>(T_{RDK})</td>
<td>DREQ Setup to DACK I</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(T_{AKRQ})</td>
<td>DACK I to DREQ I Delay</td>
<td>(T_{CY} + 150)</td>
<td>(T_{CY} + 120)</td>
<td>(T_{CY} + 100)</td>
<td>ns</td>
<td>CL = 50pF</td>
</tr>
<tr>
<td>(T_{AKH})</td>
<td>DACK High Time</td>
<td>(T_{CY})</td>
<td>(T_{CY})</td>
<td>(T_{CY})</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(T_{AK1})</td>
<td>DACK Cycle Time, Word Mode</td>
<td>4 (T_{CY})</td>
<td>4 (T_{CY})</td>
<td>4 (T_{CY})</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>(T_{AK2})</td>
<td>DACK Cycle Time, Byte Mode</td>
<td>5 (T_{CY})</td>
<td>5 (T_{CY})</td>
<td>5 (T_{CY})</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>
A.C. TEST CONDITIONS

Input Pulse Levels (except 2XWCLK) .................................................. 0.45V to 2.4V
Input Pulse Levels (2XWCLK) ................................................................. 0.3V to 3.9V
Timing Measurement Reference Levels (except 2XWCLK) ..................... 0.8V to 2.0V
Timing Measurement Reference Levels (2XWCLK) ............................... 0.6V to 3.5V

WAVEFORMS

DATA BUS TIMING
READ CYCLE

WRITE CYCLE
WAVEFORMS (Continued)

DMA TIMING
READ

2x WCLK

DREQ

DACK

RD

DB0-7

WRITE

2x WCLK

DREQ

DACK

WR

DATA VALID

T_KQ

T_KQ

T_RD1

T_CAC

T_FF

T_TR1

T_ACC

T_ACK

T_ACK1, T_ACK2

T_RD1

T_KQ

T_KQ

T_KQ

T_KQ

T_KQ

T_KQ

T_KQ
DISPLAY MEMORY TIMING
READ/MODIFY/WRITE CYCLE

2x WCLK

ADO-15

DBIN

A16, A17

ALE

S1  S2  S3  S4

VALID  VALID  VALID

OUTPUT ADDRESS  INPUT DATA  OUTPUT DATA

TCA  TAC  TDC  TCD

TIE  TIE

TCA

TCA

TCA

TCA

TCA

TCA

TCA

TCA

TCA

TCA

TCA
WAVEFORMS (Continued)

DISPLAY MEMORY TIMING (Continued)
READ CYCLE

OTHER TIMING

CLOCK TIMING
WAVEFORMS (Continued)

DISPLAY AND RMW CYCLES (1X ZOOM)

2xWCLK

ALE.

DBIN.

ADD-15

Output Address

Output Address

Input Data

Output Data

Output Address

A16, 17

HSYNC.

BLANK:

VlEXT SYNC:
WAVEFORMS (Continued)

DISPLAY AND RMW CYCLES (2X ZOOM)

<table>
<thead>
<tr>
<th>2xWCLK</th>
<th>ALE</th>
<th>DBIN</th>
<th>ADD-15</th>
<th>A16, 17</th>
<th>Blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoomed Display Cycle</td>
<td>Zoomed Display Cycle</td>
<td>RMW Cycle</td>
<td>Display or RMW Cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

ZOOMED DISPLAY OPERATION WITH RMW CYCLE (3X ZOOM)

<table>
<thead>
<tr>
<th>2xWCLK</th>
<th>ALE</th>
<th>DBIN</th>
<th>ADD-15</th>
<th>A16, 17</th>
<th>Blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoomed Display Cycle</td>
<td>Zoomed Display Cycle</td>
<td>RMW Cycle</td>
<td>Display or RMW Cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>S6</td>
</tr>
</tbody>
</table>
WAVEFORMS (Continued)

**VIDEO SYNC SIGNALS TIMING**

- **2xWCLL:**
- **HBLANK:**
- **HSYNC:**
- **ADD-15:**
- **LC0-4:**
- **Row:**
- **Row:**
- **VBLANK:**
- **VSYNC:**

**INTERLACED VIDEO TIMING**

- **HBLANK:**
- **VBLANK:**
- **VSYNC:** (Interlace)
- **Odd Field**
- **Even Field**
- **VSYNC:** (No Interface)
WAVEFORMS (Continued)

VIDEO HORIZONTAL SYNC GENERATOR PARAMETERS

HBLANK: ____________________________

HSYNC: ____________________________________________

HFP | HBP | AW

HS

VIDEO VERTICAL SYNC GENERATOR PARAMETERS

VBLANK: ____________________________

VSYNC: ____________________________________________

VBP | AL | VFP | VBP

VS

CURSOR—IMAGE BIT FLAG

2xWCLK

HBLANK: ____________________________

HSYNC: ____________________________________________

A17

TCY

10TCY

Invalid Cursor Image
VIDEO FIELD TIMING

DRAWING INTERVALS

DMA REQUEST INTERVALS

- Drawing Interval
- Additional Drawing Interval When in Flash Mode
- Additional DMA Request Intervals When in Flash Mode
By managing tasks like graphics generation and CRT refreshing, a dedicated VLSI display controller simplifies the design of intelligent graphics work stations.

Dedicated VLSI chip lightens graphics display design load

The role of graphics is becoming increasingly important for unscrambling the communications traffic between people and computers. Thanks to microprocessors and dedicated control ICs, designing high-reliability graphics work stations is now easier and less expensive than in the days of small-scale integration and expensive discrete-circuit CRT technology. Microprocessors simplify workstation design by transferring some graphics control tasks from hardware to software. However, a dedicated VLSI controller such as the 82720—with an on-board graphics processor—can push another step forward toward fast and economical design of high-quality intelligent graphics systems.

A typical application for the controller is a graphics work station aimed at high-end business and low-end engineering systems. Since such a station usually fits on the top of a desk, all of the electronics must be contained within a single printed-circuit board. This type of system requires a resolution of about 512 by 512 pixels and is frequently called on to display three-dimensional objects in various perspectives. To minimize the distortion of rotating objects, horizontal and vertical pixels should be equally spaced.

A typical display (500 vertices) must be drawn on the screen in less than 1 second to provide satisfactory interaction with the operator. The display may consist of lines, arcs, filled areas, and colors—seven colors are acceptable (see "A Look into Graphics Fundamentals").

Serial link interfaces station

An intelligent work station usually interfaces with a mainframe host via a serial communications link, a keyboard, and a serial link with an optional graphics tablet. This type of graphics input/output subsystem is diagrammed in Fig. 1. Two 5 1/4-in. floppy disks can satisfy the mass-storage needs of the system. Disk formatting must be compatible with the requirements of an IBM personal computer. Moreover, general-purpose software written for

Gary DePalma, Field Applications Engineer
Mark Olson, Product Marketing Engineer
Roger Jollis, Design Engineer
Intel Corp.
2625 Walsh Ave., Santa Clara, Calif. 95051
this computer must also be able to run on the work station.

Two of the most basic functions of a graphics system are generating and refreshing images on the CRT screen. Information pertaining to the images is stored in the bit-map memory, where monochrome pixels are represented by single bits and color pixels by groups of bits. Lines and arcs defined in normalized screen coordinates must be converted into images of the physical object.

In a bit-mapped raster graphics system, lines described by a transformed display list are reduced to a series of dots and placed in the image memory. The selection of the dots that will be activated is achieved through a scanning conversion algorithm, which must create lines that appear very smooth, start and end as expected, and look symmetrical no matter in which direction they are drawn. The algorithm is repeated thousands of times to draw a single picture and thus must operate as quickly as possible. At the same time, the image in memory must be repainted on the screen 30 times/s for

1. A graphics I/O subsystem for an intelligent work station consists of input peripherals (a keyboard and tablet), a serial communications link, and mass storage (floppy disks). Intelligence is provided by the microprocessor and the peripheral and memory controllers (a). The three basic tasks performed—I/O, transformation processing, and CRT control—all require data in the form of display lists stored in a data base (b).
interlaced frames and 60 times/s for noninterlaced frames. Simple tasks, they nevertheless demand a high memory bandwidth.

Unlike other system control tasks, generating graphics figures requires both bit-manipulation and mathematics capabilities. Integer addition and multiplication operations calculate the coordinates of points on a line or a circle. But since pixels generally are neither complete words nor bytes, logical operations must be performed on the bits within the word that contains the selected pixel.

The inner loop of a so-called Bresenham line-drawing algorithm requires two or three addition operations, two comparisons or tests, and the masking of the correct value into the word for each pixel. Algorithms for drawing circles or filling areas are even more complex. In the inner loop of a filling algorithm, for example, the old word must be read from the bit map to determine whether some, all, or none of the pixels are within the area to be filled. If they are, the algorithm tests whether the pixels must be modified and then returns the word to the processor.

2. The 82720 graphics display controller separates the tasks of graphics generation and CRT refreshing from other system tasks. That permits much greater system bandwidth, leading to graphics work stations that not only draw sharp pictures, but also offer color.

3. Three memory planes are implemented in the interface between the bit map and the graphics display controller. Three primary colors—red, green, and blue—are provided, with the controller's upper address bits responsible for selecting the memory planes during read/modify/write cycles.
Computer Graphics: Graphics display controller

bit map. Because such algorithms are heavily exercised, they must execute at extremely high speeds to avoid an adverse impact on the system's overall efficiency.

Memory bandwidth is the most precious commodity in a graphics system. In this application, screen refreshing requires that 750,000 bits be read 60 times/s, equating to a bandwidth of almost 6 Mbytes/s. The picture refreshing, therefore, has the highest-priority access to memory because any missed readings show up as noise in the picture, a situation that sometimes occurs with simple systems possessing a single-microprocessor, single-memory scheme.

In the latter type of design, one processor handles all functions except refreshing, which is implemented by a discrete counter arrangement or a simple CRT controller chip. Nevertheless, the refresh memory bandwidth always slows down the microprocessor. That loss of speed can be eliminated simply by separating the processor's memory system from the bit map, a process that effectively doubles system memory bandwidth.

The 82720 graphics display controller can provide the means of separating graphics generation and CRT refreshing from the other tasks and also perform the two tasks quickly and concurrently with the others. Residing between the microprocessor and the bit-map memory and video logic, the controller refreshes the CRT like other CRT controllers, converts high-level commands into images by placing the proper data into the correct bit

A look into graphics fundamentals

The graphics data found in graphics display lists typically describes objects in the real-world Cartesian coordinate system conforming to the axes X, Y, and Z (see the figure). Graphics data does not take the form of one bit for every point on a line; rather it represents higher-level forms such as the end points of a line and the starting, ending, and center points of an arc.

The coordinate system handles physical measurement units such as inches, feet, or meters, which are typically represented in a computer by 16- or 32-bit integers or by floating-point formats. Ultimately, complex graphics structures are stored in a data base in a hierarchical form consisting of lists of X, Y, and Z coordinates.

The first step in designing a CRT subsystem involves selecting the resolution and scanning rates. All conventional raster-scanning monitors have a display area that is wider than it is high in the ratio of 4 to 3 (called the aspect ratio). For pixels to be square—equally spaced in both the X and the Y direction—the number of horizontal pixels must be 4/3 the number of vertical pixels. This is expressed as 4H:3V, where H and V represent the number of horizontal and vertical pixels respectively. Resolution depends on the total quantity of pixels, which must be a power of two. If it is not, the number of pixels must be rounded to the nearest highest power of two, in which case some bits will be wasted. Furthermore, the number of horizontal pixels must be organized as an even number of 16-bit words.

To prevent wasted bits, the number of vertical and horizontal pixels are chosen as large as possible without exceeding a power of two. For the display in question, 512H by 512V = 2^9 = 524,288 pixels. A screen format of 576H by 432V normally meets all requirements. The total number of pixels is then 248,832, and the ratio of horizontal to vertical pixels (576/432) is correct. Furthermore, the number of horizontal pixels makes exactly 38 16-bit words.

After figuring the aspect ratio, the format of the bit-map memory is the next item to be considered. The screen contains about 250,000 pixels, each of which can be either black or one of seven colors. These eight shades can be represented by three bits/pixel (2^3 = 8), meaning that the bit-map memory must handle about 750,000 bits. The organization of the memory, however, must be determined according to the various tradeoffs.

The entire memory must be accessed 60 times/s since that is the rate at which the image must be painted to prevent flickering. That equates to a refresh rate of 16.7 ms. As a rule of thumb, the monitor displays information 75% of the time and is blanked for refreshing operations 25% of the time. Thus the whole memory must be read and sent to the CRT during a 12.3-ms interval (16.7 × 0.75), which constitutes the active
map, and interfaces easily and simply with proprietary microprocessors.

The 82720 accepts high-level commands (such as DRAW LINE, DRAW ARC, and FILL RECTANGLE) and executes them at much faster speeds than general-purpose microprocessors, primarily because it is a dedicated graphics hardware processor. Burst drawing rates as high as 1 pixel every 800 ns can be achieved. Screen refreshing is handled directly by the controller. The displayed portion of the bit-map memory can be configured to allow the display to be scrolled through memory in any direction. The horizontal and sync periods both are fully programmable, as is the position of the sync pulse in the blanking interval. Furthermore, the controller can be programmed to refresh low-cost dynamic RAMs. In the design being considered, the 82720 offloads the microprocessor from low-level graphics tasks, as shown in Fig. 2.

For the bit-map interface, the memory is implemented as three planes, each 16 kwords by 16 bits, with each plane driving red, green, or blue (Fig. 3). The upper address bits-A16 and A17—select the memory planes during read/modify/write cycles but are ignored during screen refreshing cycles.

The graphics display controller generates the Row Address Strobe (RAS) signal for the dynamic RAMs, but the remaining timing signals must be supplied by external devices. These signals are produced by a state-machine timing generator consisting of a 4-bit counter and two flip-flops. The state machine synchronizes itself with RAS after

portion of a frame.

To meet these requirements, it is helpful to break the bit map into three planes of 432 by 676 bits. While the screen is being refreshed, data is read from the same address in each of the three planes and sent serially to the screen. The memories can then be arranged as three 16-kword-by-16-bit arrays, requiring a memory cycle time of 800 ns and consequently permitting the use of relatively slow, low-cost 16-kbit dynamic RAM chips.

When drawing graphics figures, memory can be treated as a single large plane divided into three primary colors: red, green, and blue. Thus the low-order memory could represent the color red; the middle-order memory, green; and the high-order memory, blue. Each primary color requires the setting of just 1 bit/pixel. However, a secondary color—cyan, yellow, or magenta—necessitates setting 2 bits/pixel. Therefore, drawing in a secondary color takes two memory cycles/pixel and is slower than drawing primary colors.

If this creates system problems, additional hardware can be used to draw more than one plane at a time. However, in the system example, drawing speeds are not only met, but also exceeded without relying on extra hardware.

Starting with the vertical refresh rate of 16.67 ms/frame, the basic timing can be analyzed. From the 16.67 ms figure, subtract the 1.25 ms required by the monitor for its vertical blanking. That leaves 15.42 ms for scanning the 432 lines of the active portion of the display. Dividing 15.42 ms by 432 lines gives 33.5 µs/line, equivalent to a horizontal scan rate of 28 kHertz.

Vertical refreshing requires 7 µs/line, and the active portion of each line is 35.7 µs. Subtracting 7 µs from 35.7 µs leaves 28.7 µs. During this time 576 pixels are displayed, for a pixel period of 23.1 µs/676 or 49.8 µs. This corresponds to a dot clock rate of 20.07 MHz, which is chosen as the system's basic clock rate.

Display lists and commands pass from the I/O subsystem to a unit that executes the transformation tasks. Transformations are primarily mathematical operations performed on the display units. Depending on the command, this module edits display lists, organizes them for display on the screen, or edits the display-list data base. By editing a display list, objects in the physical coordinate system can be created, destroyed, moved, or changed. Transforming a display list into a form compatible with the display is necessary, as the data base can have an unlimited real-world coordinate system in three dimensions, but the CRT screen is limited to only two dimensions.

Transformation tasks place a heavy burden on a microprocessor. For instance, in a typical transformation, a matrix multiplication is performed for every point on an object's display list. In three dimensions, each point requires multiplying a 4-element vector by a 4-by-4 matrix. Some of the elements are always zero, but the operation still takes 13 multiplications and 9 additions. A two-dimensional display list requires 4 additions and 4 multiplications. A typical display can contain hundreds or thousands of lines, each of which has two end points. Therefore the speed of matrix multiplication significantly influences system performance.

The coordinate system supported by the design example is three-dimensional and employs 32-bit integers. The system CPU executes 32-bit integer matrix multiplications at high speed. In conjunction with the graphics display controller, the drawing task is offloaded from the CPU, which in turn maximizes the central-processor time that can be allocated for those matrix multiplications. Waiting time is now much lower than in conventional systems. However, if a system requires floating-point transformations, the best high-speed performance is achieved with the addition of a numerical coprocessor.
Computer Graphics: Graphics display controller

the 82720 has been initialized. Figure 4 shows the complete schematic for each plane of the bit-map interface.

The remainder of the hardware design interfaces the graphics display processor, the processor memory, and the other peripherals with the 80186 microprocessor. The task is simplified by the processor's on-board chip-selection logic and wait-state generators. Furthermore, because of the processor's highly integrated architecture, the size of the overall hardware is quite small.

Joining processor and controller

Connecting the graphics display controller to the microprocessor is a simple task, as the processor's Data, Read, and Write signals are completely compatible with those of the 82720. However, because the controller has no chip-selection input, the Read or Write signals must be qualified through external hardware.

A number of chip-selection lines on the microprocessor can be programmed to place peripherals either in memory or in the processor's I/O space. Two gates are added to qualify the Read and Write signals. The DMA channel on the 80186 uses a second chip-select input as the Acknowledge signal, and data buffers are used to prevent bus contention at the end of a processor read cycle (Fig. 5).

Without buffers, the display controller must remove its data from the multiplexed address and data lines before the processor puts out the next address. At an 8-MHz clock rate, the processor requires that peripherals and memory vacate the bus in less than 85 ns; however, the standard speed of the controller is 100 ns. A faster version, the 82720-1, can be used, but it requires faster memory chips. A more cost-effective solution is simply adding buffers, if board space permits.

Serial communications to both the host and the optional tablet are handled by a multiprotocol serial controller (the 8274), which takes care of the host's synchronous and the tablet's asynchronous

4. The bit-map memory interface contains three address planes (one of which is shown here) to complete the graphics system. The RAS signal for the RAMs is generated by the graphics display controller.
5. The interface between the 82720 and the system microprocessor is simple to implement because all of the processor's signals are compatible with the controller. It is necessary, however, to use external gates to qualify the RD or WR signals.

6. A complete graphics control system is centered around an 80186 microprocessor and the 82720 controller. Local storage is provided by 32 kbytes of EPROM and 16 kbytes of RAM. The system comprises 85 chips and is housed on a single 12-by-12-in. printed-circuit board.
the processor has a high-speed disk interface, which loads it lightly.

To complete the graphics system illustrated in Fig. 6, 32 kbytes of EPROM and 16 kbytes of RAM support the microprocessor's program and display lists. The two EPROMs (27128s) come in 28-pin packages, thereby saving board space.

Hooking up the RAM chips is almost as straightforward. Since the microprocessor is a fully byte-addressable device, it can write bytes as well as words to the RAM. The chip-select input for the low (even) address RAM must be qualified with address A₀ at a logic zero, and the high (odd) address RAM must be qualified by the processor's Byte High Enable signal (BHE). The RAMs, designated 2186, have built-in controllers.

Since dynamic RAMs latch addresses on the leading edge of the chip-select signal, they must be qualified with the processor's Address Latch Enable signal to ensure that selection is made only after the address is valid. Then, a RAM latches the data to be written on the leading edge of the write pulse. The microprocessor's write signal must be delayed by one-half of a clock cycle to guarantee that data is valid at the correct time.

At this point, the design meets all of its performance goals. The system draws lines and circles at about 120,000 bits/s. That is approximately 82,000 pixels for a display consisting of even amounts of the three primary colors, as well as three secondary colors, and white. The 500 vectors of 25 pixels each can be drawn in about 0.15 s, six times faster than the 1-s requirement. The worst case—drawing all lines in white—can be accomplished in about one-third of a second. These specifications are satisfied when the graphics display controller is running from a 2.5-MHz clock. Drawing is performed only during retracing and the 82720 is programmed to use three memory cycles of each horizontal retrace for memory refreshing.

All of the components fit on a board measuring 12 by 12 in., so that the desktop size requirements are satisfied. The electronic components occupy about 100 in.² of the low-cost, double-sided printed-circuit board. □

Bibliography:
Graphics Chip Makes Low-Cost, High Resolution Color Displays Possible

by Mark Olson and Brad May

The making of displays that are both high-resolution and low-cost is the key to producing equipment for both the automated office and the engineering workstation. Through the introduction of 16-bit µPs such as Intel's iAPX 8088, 80186 and 80286, the processing power has been made available to perform very sophisticated functions for the user while making the human interface very simple.

That processing power can be unnecessarily drained, however, if the µP is burdened with the entire task of graphics display. Such a burden can fill up a significant part of the processor's I/O bandwidth, slow down the refresh rate of the display, and decrease the computational power of the CPU.

Intelligent peripheral ICs offload processing tasks from the CPU.

The logical way to avoid such limitations is to dedicate a specialized processor to the handling of display function. It should be capable of accepting high-level commands to minimize the burden on the CPU, as well as optimizing the execution of such commands through raster operations implemented in hardware at the device level.

Such a chip is Intel's 82720 Graphics Display Controller (GDC). It has features that give systems a fast drawing speed while reducing graphics display costs by 60% or more. It achieves these results by taking over the drawing and refresh functions from the CPU, by allowing the use of dynamic RAM's instead of static RAM's, and by reducing the overall parts count needed to create a complete graphics system.

The implementation of the drawing task is a major feature of the GDC. Other graphics chips perform only the display refresh function, leaving the more complicated drawing function entirely to the CPU. Since the CPU is doing every pixel of the drawing function on these systems, they also require faster bit map RAM than with the GDC. The GDC, on the other hand, is capable of handling the drawing function itself, drawing such objects as characters, slanted characters, points, lines, arcs, rectangles, and slanted rectangles based only upon lengths, slopes, and arc centers supplied by the CPU. The GDC's processing, moreover, takes place concurrently with the processing of the CPU.

2048 × 2048 Resolution

With its 4 Megapixel addressability, one GDC can handle a monochrome display with resolution as high as 2048 × 2048, and multiple GDCs can be linked to provide even higher resolution, such as color displays at 2048 × 2048. The chances are, however, that the GDC's full power will not be used in most applications. The typical

Figure 1: General graphics commands are translated into the VDI interface level and then into driver device commands.

Mark Olson and Brad May are Product Marketing Engineers for Peripheral Components Operation, Intel Corp., Santa Clara, CA 95051.

Reprinted from DIGITAL DESIGN © April 1983, Morgan-Grampian Publishing Company, Boston, MA 02215 231-310-001

231-310-001
product considered high resolution for office automation applications is a 512 x 512 pixel monochrome or color display.

These latter restrictions are not imposed by the GDC, but rather have more to do with the cost of display monitors, the amount of RAM memory needed to support such displays, and the adequacy of such displays for most applications. It is possible to build "super graphics" boards with a GDC, such as the 1K by 1K pixel by 8 color plane graphics display designed by Phoenix Computer Graphics (Lafayette, LA). Such a display is capable of rendering 256 different colors on a high resolution screen.

Even higher performance can be achieved through the use of multiple GDCs to support multiple display windows, increased drawing speed, or increased bits per pixel. For multiple display windows, each GDC can be used to control one window of the display. For increased drawing speed, multiple GDCs can be operated in parallel. For increased bits/pixel, each GDC can contribute a portion of the number of bits necessary for a pixel.

Although the GDC is intended primarily for raster-scan graphics, it can also be used as a character display controller. It is capable of supporting up to four screens of data containing 25 rows by 80 columns, or one screen containing up to 100 rows by 256 characters.

Office Automation Display

High performance applications can stretch the usage of the GDC from low-end to high-end engineering displays, but research has shown that for office automation products, a 512 x 512 pixel display is quite acceptable, and that color is often a requirement. These requirements mesh with a major factor in display—the cost of the CRT. In OEM quantities, for example, one could expect to find a 512 x 512 monochrome display for under $100, a 256 x 256 color display (TV quality) for about $150, a 512 x 512 color CRT in the $300 range, and a 1K x 1K color display in the $800–$1000 category.

To give an example of the type of display that can be built for new office products using the GDC, consider a 512 x 512 pixel by 3 color plane combination CPU and graphics display on a single 12" by 12" board. Such a display is capable of generating 8 colors.

The list of parts (Table 2) comes to about $175 for 85 IC’s taking up 104 square inches of board space. Even that parts count could be reduced by replacing the 48 16K DRAMs with 12 64K DRAMs—if a 4K x 16 bit DRAM were available. A very important note about the parts list is that the design is implemented with inexpensive 2118 dynamic RAMs. The design does
not require the faster, more expensive, and less dense static RAMs.

The parts count is low enough so that the processor and graphics controller can be placed together in a single 12" by 12" board. This is important because small overall size and footprint are selling points for desktop workstations. System speed is also enhanced when the graphics controller and CPU are on the same board, because their communication need not take up bus, inter-board bandwidth or experience any additional delays.

**Pipelining Transformations**

More important than putting the graphics display on the same board as the CPU is the level of communication between the CPU and graphics controller. If the burden of transformation processing is left entirely to the CPU while the graphics chip is used only as a CRT controller, then the CPU must communicate one bit per pixel to update a display. With the GDC, the CPU input takes higher level forms such as the slope and length of a line, the length and center point of an arc, or the key coordinates of a rectangle. Since the average line on a screen is about 25 pixels, that means that 25 times fewer CPU bus cycles are required to draw a graphical object with the GDC. These CPU cycles (an average of 50 μs each) to calculate the graphical object and communicate it to the GDC are the determining factor in drawing rate.

Viewed from a larger perspective, there are four tasks that must be performed by a CPU-graphics chip combination:

1. The CPU must calculate the higher-level graphics operations. This is done by the CPU and it involves the processing of macro-operations such as the CORE, GKS, PMIG or other graphics protocols. These general graphics commands are translated into an intermediate level, the VDI interface level (Figure 1) and then into device driver commands by software in the CPU.

2. Then, these lower-level graphical objects such as the key parameters for lines, arcs, characters, and rectangles, must be trans-

---

**VLSI Takes Aim At Text Processing**

The concept of co-processing is not a new one. Intended as a way of offloading computationally intensive tasks from a host CPU, it has been around at Intel since the introduction of the 8087 numerics processor and the 8089 I/O machine. A more recently developed product, the 82720 Graphics Display Controller is designed to bolster system performance by offloading graphics control chores from the CPU. The chip accepts high level commands from the CPU and, using its own drawing processor, accesses the required positions in the bit-map and handles the processing and display control functions.

Building on the success of these parts come two new co-processors designed to partition system intelligence even further. The 82586 is a communications co-processor designed to bridge the characteristics of CPU and network data rates. Its FIFO buffer and DMA facilities make it possible for a CPU to operate at the full Ethernet 10 Mbits/s transfer rate even in the face of continuous bursts of network data traffic.

Intel's most recent introduction is the 82730 text co-processor. Printers and other hard copy peripherals have supported additional text processing features such as proportional spacing and simultaneous superscript and subscript for some time. Implementing these features on the display screen has traditionally been a costly procedure. Thus, it is typically not done and screen displays often are not identical to their hard-copy printouts. Armed to solve this designer's headache, the 82730 has its own DMA capability and communicates asynchronously with the CPU via shared memory messages. It supports the generation of high quality text displays through features like proportional spacing, simultaneous superscript/subscript, dynamically reloadable fonts and user programmable field and character attributes. In addition, when coupled with the 82720 Graphics Display Controller (Figure 1) the 82730 provides flexible mixing of text and graphics simultaneously on the same display.

—Wilson

![Figure 1: Offloading system tasks is simplified by new VLSI devices.](image-url)
formed into changes in the actual bits. This function is performed in hardware in the GDC concurrently with any level one processing done by the CPU. Other graphics controllers leave this task to the CPU to execute in software. The contrast is that, in such systems, the CPU must resolve the graphical object down to every point on a line, while with the GDC it need only designate the endpoints.

(3.) With the actual bits for the bit map calculated, they must be placed in the bit map memory. This involves a read-modify-write operation that requires three CPU cycles using other methods. With the GDC these operations are not the responsibility of the CPU. The GDC pipelines its execution so that it is calculating the next bit to change while it is executing the read-modify-write cycles.

(4.) Finally, the bit map memory must be dumped into the CRT. This is the refresh function performed by other graphics chips as well as the GDC.

The summation is that other systems require the CPU to process steps one to three serially, leaving only step four for the graphics controller. Systems with the GDC require the CPU to process only step one, with the GDC concurrently processing steps two through four. The GDC has another advantage in that during the transformation process in step three, the GDC executes the algorithms in hardware while a CPU must execute the algorithms in software. The algorithms are exactly the same in both cases. They are the Bresenam algorithms from IBM, in which the next pixel to be drawn becomes a binary decision between two pixels.

The execution of these algorithms is a crucial drawing time factor, because they are invoked many times for each updated screen. Consider that, in the inner loop of Bresenam's "line drawing algorithm," there are two or three additions, two comparisons or tests, and the masking of the proper value into the word for each pixel. The algorithms for drawing circles or filling areas are even more complex. In the inner loop of a fill algorithm, the old word must be read from the bit map, then tested to see if all, some, or none of the pixels are within the area to be filled. Next, it tests whether some or all of the pixels must be modified. Finally, the word must be returned to the bit map.

These algorithms are heavily used and the speed with which they can be executed has a direct effect upon the overall system efficiency. If they must be executed by a µP, the instruction fetching process slows down the calculations to a drawing rate of 15–20 µsec per pixel. With a hardware implementation of these algorithms in the GDC, the calculations can be speeded up to achieve a drawing rate of 1600 ns (2.5 MHz version) or 800 ns (5 MHz version) per pixel.

Methods Of Refresh

In the fourth step, the dumping of bit map memory into the CRT, there are some differences between graphics controller chips. Motorola's MC6845 CRT controller, for example, uses a split-cycle refresh method in which each refresh cycle is alternated with a drawing cycle in which the µP updates the bit map. This gives the MC6845 a 50% drawing bandwidth.

With the GDC there are two drawing modes. The first is a "draw anytime" mode which replaces CRT refresh cycles with drawing cycles. This is the fastest mode, but it does result in on-screen disruptions. The second mode, which does not disrupt the on-screen display, draws only during the vertical and horizontal retrace periods. This gives the GDC about a 25%
drawing bandwidth. At first glance, drawing with the GDC a disadvantage in drawing rate, but the fact is, its pipelining and hardware execution of transformations, the GDC makes much more efficient use of its bandwidth. The critical timing factor is the amount of CPU participation in the drawing process, not the refresh bandwidth of the graphics controller. Another tradeoff is that, with its split-cycle architecture, the MC6845 requires RAM memory that is twice as fast as that required by the GDC in the same application.

### Inexpensive RAM Is Fast Enough

Applying this perspective, one can begin to build the display with parts listed in Table 2. First one notes that a square display, as indicated by the 512 × 512 pixel initial specification, is not pleasing to the eye. It is much more appealing to have an aspect ratio of about 4:3, in which the number of pixels horizontally is 4/3 the number vertically. If the resolution is such that the total number of pixels is not a power of two, it will be necessary to round up to the next power of two and waste the extra bits.

The pixel arrangement which best meets this requirement is one with a 432 × 576 pixel format. It also meets the requirement that the number of pixels horizontally be an even number of 16-bit words. With three color bits per pixel (red, blue, and green), the total display memory is then about 500 × 500 × 3, or 750k bits.

It makes the most sense to break the memory up into three planes, with each plane feeding one of the primary color guns of the CRT (Figure 2). This leads to a memory arrangement of 16K × 16 × 3, using 16K dynamic RAMs with a 1K × 16 architecture. When drawing graphics figures, the memory can be treated as one large plane, split into the three primary colors. Drawing in low-order memory could represent red, middle-order could be used for green, and high-order for blue.

One advantage of this 3D memory is that drawing with a primary color requires setting only one bit per pixel. Drawing with a secondary color such as cyan, yellow, or magenta would take two GDC cycles, and creating white from all three colors would take three GDC cycles. If this were an issue, additional hardware could be used to draw more than one plane at a time. As the results will show, however, the drawing speed requirements can be exceeded without any added hardware.

### Calculate The Drawing Rate

To see if the proposed design is practical, one should first calculate the drawing rate to see what the user interface will be like. Then one should check the refresh rate to make sure the design is uninterrupted and without flicker.

The proof of the assumption that CPU participation is the dominating factor lies in the 50 μs average time that it takes the CPU to calculate a graphical object and communicate its key parameters to the GDC. Assume that the graphical object is an average line containing...
25 pixels, and that there are about 500 vectors on the average screen display.

The GDC's normal clock rate is 2.5 MHz, giving it a 400 ns period (the average clock rate is 5 MHz, with a 200 ns period.) It takes four GDC cycles to execute a read-modify-write on a bit (because two read cycles are required), so that the GDC's normal drawing rate is one pixel per 1600 ns. To draw the 25 pixels involved in the average line, then, would take 25 x 1600 ns, or 40 µs. Since this operation is done concurrently with CPU processing, the GDC will be waiting for the next graphical object by the time the CPU is ready.

If the screen were filled with nothing but 25-pixel vectors, then the drawing rate would be determined by the 50 µs average CPU calculation and transfer cycle, averaging about 2 µs per pixel. If all the vectors were white (worst case), then it would take 1.5 secs of drawing time to update the white screen. Since, in the undisturbed-screen mode, drawing is only done during the 25% of the time that the screen is undergoing horizontal or vertical blanking, this would mean 6 secs between updates.

In reality, however, the screen will not be filled with vectors. It will have an average of 500 vectors, and the color distribution could be presumed to be evenly distributed as one-third primary colors, one-third secondary colors, and one-third white. The 500 vectors will require the drawing of 12.5K pixels in monochrome, or 25K pixels with distributed colors. At a drawing rate of 2 µs per pixel, this takes 50 ms to draw. Drawing only during blanking, the screen would be updated in 200 ms.

Under these conditions, it would not help to use the maximum clock rate GDC (5 MHz), but if in some applications the average vector length is 100 pixels, then the CPU calculation-and-bus cycle (50 µs) would remain the same and the GDC's drawing cycle (1600 ns x 100 = 160 µs) would become a limiting factor. Using the 5 MHz GDC would cut that drawing time down to 800 ns/pixel, or 80 µs/vector. The 500 vector average screen would then contain 100K pixels with distributed colors and could be drawn in 80 ms. Multiplying by four because the drawing is done during blanking (25% of the time), that is 320 ms. That is a screen update in less than one-third second for a "busy" screen.

Calculate The Refresh Rate

These calculations are of little importance if the display flickers due to lack of refresh. This exercise is actually a demonstration of how the basic GDC clock rate was derived. Assume a non-interlaced display that must be refreshed 60 times per second. That gives a screen refresh rate of 16.67 ms, but on a typical CRT some 4.27 ms of that is blanked, leaving 12.4 ms of active display time. The dot sweep period is the 12.4 ms divided by the number of pixels (432 x 576 = 248,896), or 49.8 ns. The inverse gives a 20.07 MHz dot clock.

Since the GDC dumps 16 bits from the bit map memory into the

---

**Figure 4:** Completed graphics system uses the 80186 and 82720 GDC
16-bit shift register during each read, and since the shift register then feeds these bits out serially to the CRT, it makes sense that the GDC's read period should be 16 times the dot sweep period. That gives a GDC read period of about 800 ns. With each GDC read taking two cycles, the basic GDC clock period is then 400 ns, or 2.5 MHz. This gives a rock-solid display, and one would only want to go to the 5 MHz GDC to improve drawing rate.

For those who want to examine the blanking intervals to see if the CRT is indeed "typical," the blanking can be further broken down. The vertical blanking interval is 1.25 ms, leaving 15.42 ms to scan the 432 lines on the active portion of the display. Dividing 15.42 ms by 432 lines gives a 35.7 μs period per line, or a horizontal sweep rate of 28 kHz. Time is also needed for horizontal retrace, in this case, 7 μs of horizontal blanking per line. This leaves 28.7 μs to scan the 576 pixels on each line, resulting in the dot sweep period of 49.8 ns. Using a 20 MHz CRT helps keep the costs down, but the GDC can use CRT displays as fast as 80 MHz when higher resolution is required.

**Mixed Mode**

While it is possible to generate 8 x 8 characters and slanted characters in the graphics mode, the GDC also offers a mixed mode memory organization to display both characters and graphics drawn from separate windows in the display memory. The advantage of this mode is that it allows characters to be manipulated as 8-bit entities instead of the 64 bits that each would require in graphics mode. Of necessity, the graphics window display memory is reduced in this mode (64K 16-bit words instead of 256K), but even the reduced maximum graphics memory is still a megapixel and quite sufficient for both office automation and engineering display purposes.

In the character window, the GDC operates as it does in the pure character mode, with the exception that the line counter must be implemented externally. In addition to the two windows used for graphics and characters in the mixed mode, two other windows can be supported. These can be designated as either character or graphics windows by a selection on the A17 line.

**Panning, Zooming, Light Pen**

As special features, the GDC allows both panning and zooming in either graphics, character, or mixed modes. The zoom is accomplished by effectively increasing the size of the dots on the screen. Vertically, this is done by repeating the same display line. The number of repeat times is determined by the display zoom parameter. Horizontally, zoom is accomplished by extending each display word cycle and displaying fewer words per line, according to the zoom factor.
82730
TEXT COPROCESSOR

- High Quality Display for Text and Graphics Applications
- Provides Proportional Spacing, Simultaneous Superscript/Subscript, Soft Font Support and Bit Map Graphics
- High Performance Manipulation of Text/Graphics Strings
- Programmable Bus Interface Handles 8 or 16 Bit Data and 16 or 32 Bit Addressing; iAPX 86/88/186/188 Compatible
- On-Chip Processing Unit Simplifies Software Design by Executing High Level Commands and Supporting Linked List Data Structures
- Extremely Flexible; Programmable Features Include Screen and Row Formats, Two Cursors, Character and Field Attributes and Smooth Scrolling
- Supports Multiple Windows
- High Resolution Display; Up to 200 Characters/Graphics Cells per Row and 2048 Scan Lines per Frame
- Separate Bus and Video Clocks Allow Optimization of Overall System Performance
- Provides a Complete LSI Solution for Display Control when Used in Conjunction with the 82731 Video Interface Controller

The 82730 Text Coprocessor is a high performance VLSI solution for raster scan text and graphics displays. The 82730 works as a coprocessor and has processing capabilities specifically tailored to execute data manipulation and display tasks. It provides the designer the ability to functionally partition his system thereby offloading the system CPU and achieving maximum performance through concurrent processing. The 82730 supports the generation of high quality text displays through features like proportional spacing, simultaneous superscript/subscript, dynamically reloadable fonts and user programmable field and character attributes. It supports high quality graphics with fast manipulation and display of bit map strings. An intelligent system interface and efficient software capabilities makes 82730 based systems easy to design.

Figure 1. 82730 Block Diagram
The 82730 is packaged in a 68 pin JEDEC Type A ceramic package.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin Number</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD15–AD0</td>
<td>1–8, 10–17</td>
<td>I/O</td>
<td>Address Data Bus; these lines output the time multiplexed address (TU, T1 states) and data (T2, T3, T4 and TW) bus. The bus is active HIGH and floats to 3-state OFF when the 82730 is not driving the bus (i.e. HOLD is not active or when HOLD is active but not acknowledged, or when RESET is active).</td>
</tr>
<tr>
<td>BCLK</td>
<td>59</td>
<td>I</td>
<td>Bus clock; provides the basic timing for the Memory Interface Unit.</td>
</tr>
<tr>
<td>RD</td>
<td>62</td>
<td>O</td>
<td>Read strobe; indicates that the 82730 is performing a memory read cycle on the bus. RD is active low for T2, T3 and TW of any read cycle and is guaranteed to remain high in T2 until the address is removed from the bus. RD is active low and floats to 3-state OFF when 82730 is not driving the bus. RD will return high before entering the float state and will not glitch low when entering or leaving float.</td>
</tr>
</tbody>
</table>
Table 1. 82730 Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin Number</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>WR</td>
<td>63</td>
<td>O</td>
<td>Write strobe; indicates that the data on the bus is to be written in a memory device. WR is active for T2, T3 and TW of any write cycle. It is active LOW and floats when 82730 is not driving the bus. WR will return high before entering the float state and will not glitch low when entering or leaving float.</td>
</tr>
<tr>
<td>ALE</td>
<td>61</td>
<td>O</td>
<td>Lower Address Latch Enable; provided by the 82730 to latch the address into an external address latch such as 8282/8283 (active HIGH). Addresses are guaranteed to be valid on the trailing edge of ALE.</td>
</tr>
<tr>
<td>UALE</td>
<td>68</td>
<td>O</td>
<td>Upper Address Latch Enable; it is similar to ALE except that it occurs in upper address output cycle (TU).</td>
</tr>
<tr>
<td>AEN</td>
<td>67</td>
<td>O</td>
<td>Address Enable; AEN is active LOW during the entire period when 82730 is driving the bus. It can be used to unfloat the outputs of the Upper and Lower Address latches.</td>
</tr>
<tr>
<td>DEN</td>
<td>66</td>
<td>O</td>
<td>Data enable; provided as a data bus transceiver output enable for transceivers like the 8286/8287. DEN is active LOW during each bus cycle and floats when 82730 is not driving the bus. DEN will not glitch when entering or leaving the float state.</td>
</tr>
<tr>
<td>S0, S1</td>
<td>53, 54</td>
<td>O</td>
<td>Status pins; encoded to provide bus-transaction information:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>Bus Cycle Initiated</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>- - - (Reserved)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Memory Read</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Memory Write</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Passive (No bus cycle)</td>
</tr>
</tbody>
</table>

These pins are directly compatible with iAPX 86, 186 status outputs $S_1$ and $S_0$. The status pins are floated when 82730 is not driving the bus. They will not glitch when entering or leaving the 3-state condition.

<p>| READY | 55 | I | READY; signal to inform the 82730 that the data transfer can be completed. Immediately after RESET, READY is asynchronous (internally synchronized) but can be programmed during initialization to bus synchronous. |</p>
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin Number</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOLD</td>
<td>65</td>
<td>O</td>
<td>HOLD; indicates that the 82730 wants bus access. HOLD stays active HIGH during the entire period when 82730 is driving the bus.</td>
</tr>
<tr>
<td>HLDA</td>
<td>64</td>
<td>I</td>
<td>Hold Acknowledge; indicates to 82730 that it is granted the bus access as requested. HLDA may be asynchronous to 82730 clock. If HLDA goes inactive (LOW) in the middle of an 82730 bus cycle, the 82730 will complete the current bus cycle first, then it will drop HOLD and float address and bus control outputs.</td>
</tr>
<tr>
<td>CA</td>
<td>52</td>
<td>I</td>
<td>Channel Attention; used to notify 82730 that a command in the command block is waiting to be processed. CA is latched on its falling edge.</td>
</tr>
<tr>
<td>SINT</td>
<td>56</td>
<td>O</td>
<td>Status Interrupt; used to inform the processor that an unmasked interrupt has been generated in the 82730 status register.</td>
</tr>
<tr>
<td>IRST</td>
<td>57</td>
<td>I</td>
<td>Interrupt Reset; SINT is cleared by activating the IRST pin.</td>
</tr>
<tr>
<td>RESET</td>
<td>58</td>
<td>I</td>
<td>Reset; causes 82730 to immediately terminate its present activity and enter a dormant state. The signal must be active HIGH for at least 4 BCLK cycles and is internally synchronized to the bus clock.</td>
</tr>
<tr>
<td>CCLK</td>
<td>27</td>
<td>I</td>
<td>Character clock; input used to clock row buffer data, attribute, cursor and line count out of 82730. When more than one 82730 is connected in cluster mode, CCLK is used to synchronize output from both master and slave chips. A character data word will be output at every rising edge of CCLK.</td>
</tr>
<tr>
<td>RCLK</td>
<td>25</td>
<td>I</td>
<td>Reference clock; input used to generate timings for the screen layout and to define screen columns for data formatting. All raster output signals are specified relative to the rising edge of RCLK.</td>
</tr>
<tr>
<td>DAT0–DAT14</td>
<td>36–42</td>
<td>O</td>
<td>Video data bus output; the least significant 15 bits of the character data words are passed through the 82730 row buffer and made available on the pins DAT0–DAT14. The user has the flexibility to partition the data word into character and attribute bits per his requirements. The bits that are assigned for internally generated attributes may also be available at pin DAT0–DAT14. New character data will be shifted to these output pins at every rising edge of the CCLK. Together with LC0–LC4, they may be used to address the character generator or as attribute controls.</td>
</tr>
</tbody>
</table>

<p>|        | 44–51      |      |                     |</p>
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin Number</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDEF</td>
<td>35</td>
<td>O</td>
<td>Width Defeat; is used to indicate when the character is allowed to be a variable width or must be of fixed width. WDEF is LOW if the character being output is normal, but is HIGH if it is a superscript/subscript character or visible attribute (TAB or GPA). Optionally, WDEF can be held high by user command.</td>
</tr>
<tr>
<td>LC0-LC4</td>
<td>18-22</td>
<td>O</td>
<td>Line count outputs; used to address the character generator for the line positions in a row. The line number output is a function of the display mode and character attributes programmed by the user.</td>
</tr>
<tr>
<td>CSYNC</td>
<td>28</td>
<td>O</td>
<td>CCLK synchronization output; used to synchronize external character clock generator to reference clock timing. This output is active (high) outside the display field.</td>
</tr>
<tr>
<td>CHOLD</td>
<td>32</td>
<td>O</td>
<td>CCLK Inhibit output; used by external logic to inhibit CCLK generation. This output is active (low) during the tab and end-of-row function.</td>
</tr>
<tr>
<td>SYNCIN</td>
<td>24</td>
<td>I</td>
<td>Synchronization input; used to synchronize the vertical timing counters to an externally generated VSYNC signal. Used by slave mode 82730 to synchronize to a master mode 82730 and by the master 82730 to lock the frame to an external source such as the power line frequency.</td>
</tr>
<tr>
<td>HSYNC</td>
<td>23</td>
<td>O (MASTER)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>I (SLAVE)</td>
<td>Horizontal Sync; in master mode, it is used to generate the CRT monitor's horizontal sync signal. It is active HIGH during the programmed horizontal sync interval. In interlace slave mode it is used in conjunction with SYNCIN to indicate the start of the even field for timing counter reset. At RESET, pin is set as an output in the LOW state.</td>
</tr>
<tr>
<td>VSYNC</td>
<td>29</td>
<td>O</td>
<td>Vertical Sync; active HIGH during the programmed vertical sync interval and used to generate the CRT monitor's vertical sync signal.</td>
</tr>
<tr>
<td>BLANK</td>
<td>33</td>
<td>O</td>
<td>Blanking output; used to suppress the video signal to the CRT. BLANK is clocked by CCLK.</td>
</tr>
<tr>
<td>CRVV</td>
<td>34</td>
<td>O</td>
<td>Character Reverse Video (CCLK output); used to externally invert video data output. CRVV is clocked by CCLK.</td>
</tr>
<tr>
<td>RRVV</td>
<td>30</td>
<td>O</td>
<td>Reference Reverse Video (RCLK output); to externally invert video in the field and border area if so programmed by user. It is LOW outside the border area, RRVV is clocked by RCLK.</td>
</tr>
</tbody>
</table>
Table 1. 82730 Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin Number</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPEN</td>
<td>31</td>
<td>I</td>
<td>Light Pen Input; used to latch the position of a light pen. At the rising edge of this input, the column position and the row position of the 82730 will be loaded into the LPENROW and LPENCOL locations in the Command block.</td>
</tr>
<tr>
<td>VCC</td>
<td>9, 43</td>
<td></td>
<td>Power; +5 volts nominal potential.</td>
</tr>
<tr>
<td>VSS</td>
<td>26, 60</td>
<td></td>
<td>Power; ground potential.</td>
</tr>
</tbody>
</table>

**FUNCTIONAL DESCRIPTION**

Figure 1 shows a basic block diagram of the 82730 Text Coprocessor. The chip is divided into two main sections, the Memory Interface Unit and the Display Generator. The Memory Interface Unit controls fetching of the data and commands and handles interrupts and status. The Display Generator takes the data fetched by the Memory Interface Unit and presents it to the Video Interface logic which in turn drives the CRT monitor.

**Memory Interface Unit**

The Memory Interface Unit is divided into two sections: the Bus Interface Unit and the Microcontroller Unit. The Bus Interface Unit does the actual interfacing to the memory bus. It fetches or writes data under the control of the Microcontroller Unit. The Microcontroller Unit is a microprogrammed controller which is designed to efficiently fetch data from memory (up to 4 Mbytes/sec), and decode and execute various control and data handling commands. The Bus Interface Unit may be configured for 8 or 16 bit bus operation. With 8 bit bus selection, the user may specify either 8 or 16 bit character data. It also handles address manipulation automatically after being loaded from the Microcontroller Unit.

**Display Generator**

The Display Generator takes the data fetched from memory plus the modes programmed into it at initialization and produces all the video timing and the data transfers to support the CRT monitor at the character level. The 82730 works with an external character generator and the 82731 Video Interface Controller. The data is passed to the Display Generator from the Memory Interface Unit through the dual row buffers (similar in operation to the one in the 8275 CRT controller). The row buffers allow the user to use cheaper and slower main memory for display needs, provide on-chip attribute and display function generation, and avoid the conflict of access to the display memory (that would otherwise take place) by using an ordinary DMA access mechanism.

**SYSTEM BUS INTERFACE**

The Memory Interface Unit provides communication with system processor as well as memory interactions. Communication between the processor and the 82730 is performed via messages placed in communication blocks in shared memory. The processor can issue commands by preparing message blocks and directing the 82730's attention to them by asserting a hardware channel attention. The 82730 can cause interrupts on certain conditions, if enabled by the processor by activating its System Interrupt output, with status and error reporting taking place through the communication block in memory.

**BUS INTERFACE UNIT:**

The 82730 Bus Interface Unit provides an 8086 compatible bus interface which consists of:
- a 16/32 bit multiplexed Address/Data Bus: AD₀ - AD₁₅
- A complete set of local bus control signals compatible with 8086 min mode: RD, WR, ALE, DEN and READY
- Two status signals S₀ and S₁, compatible with 8086 max mode so that a bus controller (8288) can be shared for Multibus® access.
- Local bus arbitration through HOLD/HLDA
- Two upper Address Latch controls: UALE and AEN
The BUS INTERFACE UNIT (BIU) utilizes the same Bus structure as the 80186 or basically the same bus structure as the 8086 in both Min. and Max. mode, (with the exception of RQ/GT) and it performs a bus cycle only on demand (e.g., to fetch a command from the command block, or fetch a character from display data memory). The same set of T-states (T1, T2, T3, T4 and TW) of 8086 are used to handle the time multiplexed address/data bus. However, adaptations are made to handle 32 bit addresses as explained in the following sections where specific details of the BIU operation are described. Those details not mentioned can be assumed to be the same as those of the 80186.

ADDRESS BUS

The 82730 can be programmed during initialization to operate on either 16 bit or 32 bit (including any length between 17 and 32) physical addresses. Note that the 82730 does not use memory segmentation. The programmer must calculate physical addresses from segment and offset values to manipulate data structures.

To support 32 bit physical addresses with a 16 bit physical bus, multiplexing is again used. An upper address output cycle, TU, is inserted between T4 and T1 to output the upper 16 bits of address. The upper address latch enable, UALE, is used to latch the upper addresses during TU. Figure 3 shows the configuration of a 32 bit address bus.

TU occurs only when the 32 bit mode is specified and the upper address register of BIU is reloaded by MCU. This may result from:

i) Initialization

ii) Manipulation of display data or command pointers, for example, when a new string pointer is loaded during the execution of the END OF STRING command.

iii) DMA address incrementing across a 64K byte segment boundary.

iv) Regaining the bus after losing it to a higher priority master.

Timing of UALE is identical to that of ALE. AEN is equivalent to the active period of 82730 driving the bus.

If 16 bit address mode is programmed, TU will never occur in any bus cycle since the MIU treats all display pointers as 16 bit quantities and loading of internal upper address register is bypassed during address calculation. UALE always stays inactive, but AEN still goes active to indicate the 82730 has control of the bus.

DATA BUS

The 82730 is capable of operating on either an 8 bit or a 16 bit Data bus, as programmed during initialization on the SYSBUS byte.

When an 8 bit data bus is specified, the address present on AD15 to AD8 Address/Data lines is maintained for the complete bus cycle. Therefore, compatibility with 80188, 8088, 8089 and 8085 multiplexed address peripherals is maintained. Since the internal processing of the 82730 generally operates on 16 bit data quantities, two Bus fetch cycles are performed for each 16 bit data item. The first cycle fetches the low order byte, the second cycle the high order byte. These 2 fetch cycles are always executed back to back. If HLDA drops during the first cycle, the 82730 will not respond until the second cycle is completed. An 8 bit data mode can be selected in an 8 bit bus system that requires only 8 bit character data be fetched.

In 16 bit bus system, the 82730 requires all 16 bit quantities to start on even address boundary. Word transfer to or from odd boundary is not allowed since this type of transfer not only doubles the use of bus bandwidth but also can be easily avoided in application software. All that is required is to make sure all address pointers be an even number (AO=0).

---

**Figure 3. Address Extension up to 32 Bits**
BUS CONTROLS

The 82730 BIU provides both the 8086 MIN. Mode (Local Bus Control) and MAX. mode bus control signals simultaneously in any bus cycle. By providing a complete set of Local Bus control signals, the component count of the Local processing module is minimized.

Because only two types of Bus operations, Memory Read and Memory Write, are executed in the 82730 BIU, the 8086's S2 status signal is omitted from the Max. mode controls. S2 could be set to "1" during any 82730 Bus cycle. AEN can be used to produce S2 since it stays active whenever 82730 is driving the bus. The status signals become valid at the middle of the cycle before T1 which could be either T4 or TU.

BHE is not provided on the 82730 because, the 82730 only writes words to even address boundaries and bytes to the upper byte position. For these writes BHE is always high. A pullup resistor or a three-state buffer controlled by AEN can provide this signal.

DT/R is also not provided on the 82730 because its function can be replaced with $S1$, latched by ALE.

After RESET is applied, READY is set to be an asynchronous input. An on-board synchronization circuit provides reliable operation for any type of system. During initialization, READY may be programmed to be bus synchronous. For those systems that can meet the set-up time specifications, this mode provides more efficient bus utilization.

LOCAL BUS ARBITRATION

The 82730 BIU is designed to function as a bus master in a multimaster Local bus environment using the HOLD/HLDA protocol for Bus arbitration.

In the Self Contained Arbitration scheme, one processor and one 82730 share access to the local bus. The 82730 raises its HOLD request whenever it needs bus access. After HLDA is granted from the processor, the 82730 will not start driving the bus until 2 clock cycles later. This latency allows sufficient time for the 8086 or 80186 processor to get off the bus. When 82730 completes its bus accesses, it will first float its output drivers before dropping the hold request.

In a Local bus configuration with three or more bus masters, a higher priority DMA Peripheral device can preempt the HLDA from a 82730 which is the current bus master. The 82730 will complete its current bus cycle, then float its output drivers and drop the HOLD request. However, the 82730 may raise the HOLD request again 2 clock cycles later if it still needs the bus to complete the interrupted burst DMA activities.

DMA BURST AND SPACE

Some system configurations using the 82730 would be adversely affected by the long burst data transfers which the Memory Interface Unit (MIU) may occasionally desire. Since the 82730 will normally be configured as one of the higher priority bus masters, burst lengths must be limited for these systems. For this reason, the length of a burst transfer and the number of memory cycles between burst transfers are both programmable via the mode registers:

\[
\begin{align*}
15 & \quad 14 & \quad 8 & \quad 7 & \quad 6 & \quad 0 \\
\text{MPTR} & \quad \text{BRSTLEN} & \quad \text{BRSTSPAC}
\end{align*}
\]

BRSTLEN - Burst Length. Determines the number of contiguous word-fetch cycles which may be requested. Programmable from 1 to 127. Note that in an 8 bit bus, 16 bit data system, the burst counter only increments once for the 2 bus cycles required to complete a word fetch. (Note: burst length = 0 is not defined and should not be programmed with a non-zero burst space)

BRSTSPAC - Burst Space. Determines a minimum number of bus clocks to occur between burst accesses. Programmable from 0-511 in increments of four. Zero space selects an infinite burst length.

A DMA burst could be terminated before the programmed burst length is reached in the following circumstances:

i) The MIU does not need any more bus accesses, for example, when the row buffer is filled.

ii) A datastream command is encountered and the MIU must execute the command first before it resumes data accessing.

iii) The bus is taken away by a higher priority device in multi-master bus configuration.

In these cases, the burst counter is cleared. The BIU must complete a full burst before it waits through the SPACE cycles. DMA Burst/Space will be set to zero space until the completion of the first MODESET command.
INITIALIZATION OF BIU

Upon activation of the RESET input, the 82730 BIU will stop all operations in progress and deactivate all outputs. It will stay in this quiescent state until memory access is requested by the MCU after MCU receives its first channel attention after RESET. The following table shows the state of all MIU outputs during and after reset.

**Table 2. 82730 Bus During and After Reset**

<table>
<thead>
<tr>
<th>Signals</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD15-0</td>
<td>Three-state</td>
</tr>
<tr>
<td>RD, WR, DEN</td>
<td>Driven to ‘1’ then three-state</td>
</tr>
<tr>
<td>S0, S1</td>
<td>Driven to ‘1’ then three-state</td>
</tr>
<tr>
<td>ALE, UALE</td>
<td>Low</td>
</tr>
<tr>
<td>AEN</td>
<td>High</td>
</tr>
<tr>
<td>HOLD</td>
<td>Low</td>
</tr>
<tr>
<td>SINT</td>
<td>Low</td>
</tr>
</tbody>
</table>

82730 COMPATIBILITY ISSUES
82730 Bus Clock Compatibility

The 82730 uses the 50% duty cycle output of the iAPX-186 at 8 MHz or that generated by a clock generator such as the 82285. A different duty cycle clock may be used at lower frequencies, so the 82730 is also useable with the iAPX-86, 88 family.

82730 Bus Interface Compatibility

The bus interface compatibility between the 82730 and another bus master has four main issues: data bus width, addressability, control bus structure and local bus mastership arbitration.

Data Bus

Data Bus width compatibility with all 85/86 family processors (8085, 8086, 8088, 8089, 80186, and 80286) is being supported by the 8/16 data bit programmability already discussed. This allows interfacing to the above processors either directly or through a Multibus-like interface.

Address Bus

The 82730 uses real 32-bit addresses. The user's software must calculate real addresses; this general addressing scheme allows the 82730 to be used with any microprocessor.

Control Bus

The 82730 implements both 8086 minimum and maximum mode bus control structures. This was done to maximize compatibility with the 80186 which has the same structure. This allows the 82730 to be run locally (minimum mode) with a 8085, 8086, 8088, 80188, or 80186. The 80186/188 and 82730 can run together at 8MHz because of clock duty cycle considerations. The 82730 can only communicate to an 80286 via a system bus (such as MULTIBUS), bus interface, or dual-port RAM.

INITIALIZATION SEQUENCE

The first CA (Channel Attention) after Reset causes an Initialization Sequence to be executed. The system processor must set up the appropriate initialization information in memory and set the BUSY flag in the Intermediate Block to a non-zero value prior to issuing this CA.

Initially, 32-bit addressing and 8-bit data bus width are assumed until the corresponding information is fetched during the initialization. First the SYSBUS byte is fetched from memory location FFFF FFF6. (When the address bus is less than 32 bits wide, the higher order bits are unused.) The format for SYSBUS byte is shown in Figure 4 and is the same as that used for 8089. The data bus width is specified by the least significant bit w, with w=0 indicating an 8-bit bus and w=1 signifying a 16-bit bus.

A 32-bit real address pointer is then fetched from memory locations FFFFFF FFFC through FFFFFF FFFF, with lower bytes of the pointer residing in lower addresses. This pointer is used as an Intermediate Block Pointer (IBP).

The Intermediate Block Pointer (IBP) is incremented by two and is used to locate the Command Block Pointer (CBP). Four bytes are fetched irrespective of whether a 16-bit or 32-bit addressing option is used. The System Configuration byte (SCB) is then fetched from location (IBP + 6).

The least significant bit, (U of the SCB) specifies 16 or 32-bit addressing option, with U=0 indicating 16-bit addressing and U=1 specifying 32-bit addressing. The SCB also contains information about cluster operation. Since up to four 82730's can be connected in a cluster with their respective data interleaved in memory, cluster information is needed for the data access task. The SCB specifies Cluster Number (CL NO), which is the number of 82730's connected in a cluster and Cluster Position (CL POS) which is the position...
of this particular 82730 within the cluster. CL NO = 0, 1, 2 or 3 indicates a cluster containing 1, 2, 3 or 4 82730's respectively. Similarly, CL POS = 0, 1, 2 or 3 indicates 1st, 2nd, 3rd or 4th position respectively. Each 82730 adds an offset equal to 2 * CLPOS to the SPTR fetched from memory and increments the pointer by 2 * (CL NO + 1). The programming of CL NO and CL POS is independent. No checking is done for CL POS greater than CL NO on the 82730. Note that at least one 82730, in a cluster (even if it is a cluster of one), must be assigned as cluster position zero (CL POS = 0) for Virtual Display mode to work properly.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRDY</td>
<td>DTW16</td>
<td>M/S</td>
<td>CL POS</td>
<td>CL NO</td>
<td>U</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SRDY</th>
<th>READY MODE</th>
<th>DTW16</th>
<th>Display Data Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Asynchronous</td>
<td>0</td>
<td>8-bit data</td>
</tr>
<tr>
<td>1</td>
<td>Synchronous</td>
<td>1</td>
<td>16-bit data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M/S</th>
<th>Mode</th>
<th>CL POS</th>
<th>Position in Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Slave</td>
<td>00</td>
<td>1st</td>
</tr>
<tr>
<td>1</td>
<td>Master</td>
<td>01</td>
<td>2nd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>3rd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>4th</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CL NO.</th>
<th>No. of 82730's In Cluster</th>
<th>U</th>
<th>ADDR BUS WIDTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>0</td>
<td>16-bit</td>
</tr>
<tr>
<td>01</td>
<td>2</td>
<td>1</td>
<td>32-bit</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. SYSBUS and SCB Encoding
The SCB also contains an M/S bit which specifies a master or slave mode. The M/S bit is stored internally for use by the Display Generator (DG). M/S = 1 indicates a master mode and M/S = 0 specifies a slave mode. The format for the System Configuration Byte (SCB) is shown in Figure 4. Following these actions, the BUSY flag in the Intermediate Block at address IBP is cleared and a normal Channel Attention sequence is then executed.

The last two bits in the SCB are DTW16 and SRDY. DTW16 specifies whether the display data in 8 bit bus mode (W=0) is 8 or 16 bit. If a 16 bit system is specified (W=1) then DTW16 is ignored and forced internally to a "one". SRDY specifies whether the clock synchronization circuit for the READY pin is internal (SRDY=0) or external (SRDY=1).

The Initialization Control Blocks in memory are illustrated in Fig. 5a. How these fit into the control structure of the 82730 is shown in Figure 5b.

Channel Attention Sequence

When the processor activates CA, an internal latch in 82730 is set on the falling edge of CA input and this latch is sampled by the MCU. The first CA activation after reset causes the 82730 to execute an initialization sequence. Any subsequent activation will cause the MCU to start processing the command block by fetching a channel command.

If a display is in progress, the MCU will sample CA at each end of frame, otherwise it will sample CA every cycle until it is found active. When CA is found active, the MCU will fetch the command byte from "COMMAND" location in the command block, execute the command and clear the BUSY flag upon completion. The internal CA latch is also cleared by the MCU. An invalid command code has the effect of NOP and the BUSY flag is cleared. It will also cause the Reserved Channel Command (RCC) status bit to be set.

**Figure 5a. Initialization Control Blocks**

<table>
<thead>
<tr>
<th>INTERMEDIATE BLOCK POINTER</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBP UPPER</td>
<td>FFFF</td>
<td>FFFE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBP LOWER</td>
<td>FFFF</td>
<td>FFFC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RESERVED) SYUSB</td>
<td></td>
<td>FFFF</td>
<td>FFF6</td>
<td></td>
</tr>
<tr>
<td>(RESERVED) SCB</td>
<td>IBP + 6</td>
<td>IBP + 4</td>
<td>IBP + 2</td>
<td>IBP</td>
</tr>
<tr>
<td>CBP UPPER</td>
<td></td>
<td></td>
<td></td>
<td>CBP</td>
</tr>
<tr>
<td>CBP LOWER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RESERVED) BUSY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMAND BLOCK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMAND BUSY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOW SYSTEM MEMORY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7-153
Figure 5b. Control Structure of the 82730
82730 TEST FEATURES

The 82730 has built-in Self-Test features that provide testability at the component or at the board level. These features include the test commands and the output pin force capability and are described below.

Output Pin Force Capability

A capability to force logic states (high, low, high impedance) on all output pins is provided in the 82730 Text Co-Processor. This is accomplished by providing a stimulus on pins LC0-LC2 during chip reset. This feature is used for dc parametric tests on the output pins.

The state of pins LC0-LC2 is monitored during chip reset. If no external inputs are applied during reset, the state observed will be all 1's and no action will be taken by the 82730. If any external inputs are applied to pins LC0-L2 during reset, the resulting action will depend upon the state latched on the falling edge of reset. The 82730 maintains pins LC0-LC2 in high impedance state for the duration of chip reset to avoid contention with external inputs. Also internal pull-ups ensure that a state of all 1's will be detected if no external inputs are applied.

The actions corresponding to each of the observed states of pins LC0-LC2 are summarized in Fig. 6a.

State of Pins LC0-LC2
During Chip Reset

<table>
<thead>
<tr>
<th>LC2</th>
<th>LC1</th>
<th>LC0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Action

- **Invoke Stand-Alone Self Test**
- Force all Outputs to High Impedance State
- Force all Outputs to Logic High State
- Force all Outputs to Logic Low State
- NOP

Stand-Alone Self Test

The built-in Self Test capability of the 82730 can be invoked in a stand-alone mode by applying an external stimulus through pins LC0-LC2 during chip reset. This is the same mechanism as the one used for forcing logic states on output pins. Fig. 6a.

If pin LC2 is pulled low during chip reset, the 82730 executes a built-in self test. Upon completion of the self-test, a 16-bit signature, generated internally as the test result, is output via pins WDEF, DAT14-DAT0. The completion is signalled by providing a logic “0” output on pin LC3 as a completion flag. The signature will remain on the output pins until the next chip reset. The 82730 will enter an idle state awaiting chip reset and will not respond to any external inputs until a reset signal is applied. During the process of presenting the signature onto WDEF, DAT14-DAT0, the signature will also appear briefly on the AD bus in the form of a bus cycle with two 8-bit accesses to addresses, AAAAH, AAABH. However, this phenomenon is only incidental. Pins WDEF, DAT14-DAT0 should be used for observing the signature.

The stand-alone self test includes the testing of internal address pointer registers. These registers are not tested when the self test is invoked by issuing a “Self Test” command. (See under Channel Commands below). Therefore, the signature generated during stand-alone self test will be different from that generated by the “Self Test” command.

Figure 6a. Output Pin Forcing and Stand-Alone Self Test Invocation
82730 CHANNEL COMMANDS

Table 3. Channel Commands

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>OP CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 START DISPLAY</td>
<td>0000 0001 01 H</td>
</tr>
<tr>
<td>2 START VIRTUAL DISPLAY</td>
<td>0000 0010 02 H</td>
</tr>
<tr>
<td>3 STOP DISPLAY</td>
<td>0000 0011 03 H</td>
</tr>
<tr>
<td>4 MODE SET</td>
<td>0000 0100 04 H</td>
</tr>
<tr>
<td>5 LOAD CBP</td>
<td>0000 0101 05 H</td>
</tr>
<tr>
<td>6 LOAD INTMASK</td>
<td>0000 0110 06 H</td>
</tr>
<tr>
<td>7 LPEN ENABLE</td>
<td>0000 0111 07 H</td>
</tr>
<tr>
<td>8 READ STATUS</td>
<td>0000 1000 08 H</td>
</tr>
<tr>
<td>9 LD CUR POS</td>
<td>0000 1001 09 H</td>
</tr>
<tr>
<td>10 SELF TEST</td>
<td>0000 1010 0A H</td>
</tr>
<tr>
<td>11 TEST ROW BUFFER</td>
<td>0000 1011 0B H</td>
</tr>
<tr>
<td>12 NOP</td>
<td>0000 0000 00 H</td>
</tr>
<tr>
<td>13 (RESERVED)</td>
<td>From: 0000 1100 DC H</td>
</tr>
<tr>
<td></td>
<td>To: 1111 1111 FF H</td>
</tr>
</tbody>
</table>

The system processor issues channel commands to 82730 via the Command Block. The processor first checks if the BUSY flag in the command block has been cleared. It should wait for the BUSY flag to be cleared before proceeding with the issuing of a command. When the BUSY flag is cleared, the processor places a command byte in the "COMMAND" location in command block, sets the BUSY flag to a non-zero value and asserts Channel Attention (CA), by activating the CA input to 82730. A Channel Attention should not be issued, if the BUSY flag has not been cleared.

START DISPLAY

<table>
<thead>
<tr>
<th>CMD Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0001</td>
</tr>
</tbody>
</table>

LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Position values are fetched from the Command Block and stored internally after this command is received. The BUSY flag is cleared and the normal display process is activated.

The MCU fetches strings of data from the memory, using the parameters LISTSWITCH, LBASE0 and LBASE1. The data fetched is interpreted as data-stream commands or character data to be displayed by the Display Generator. The MCU loads the data into one of the two Row Buffers in the CRT controller, while the Display Generator displays the data from the other buffer, the buffers being swapped at the end of the row. Any data-stream commands encountered during data fetch are immediately executed.

The display process is continued until it is deactivated by a STOP DISPLAY command or a Reset. Other channel commands can be issued while a display is in progress and they will be executed when CA is found active at one of the periodic samplings at each end of frame.

The DIP (Display in Progress) status bit is set and the VDIP (Virtual Display in Progress) is cleared upon receiving a START DISPLAY command. Both bits are reset upon receiving a STOP DISPLAY command or a Reset.

It is necessary to load in proper mode information through a MODESET command before activating the display. Following Reset, START DISPLAY command will not be executed, i.e., will result in a NOP until a MODESET command has been issued.
START VIRTUAL DISPLAY

0000 0010 CMD Byte

LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Positions are fetched from the Command Block and stored internally upon receiving this command. The BUSY flag is cleared and the Virtual Screen display process is activated.

The operation of the Virtual Screen display process is similar to that of a regular display process, except for following a different data access mechanism. The parameters LISTSWITCH, LBASE0 and LBASE1 in the command block represent ACCESS SWITCH, ACCESS BASE0 and ACCESS BASE1 respectively, in virtual screen display.

The VDIP (Virtual Display in Progress) status bit is set and the DIP status bit is cleared upon receiving a START VIRTUAL DISP command: Both DIP and VDIP are reset upon receiving a STOP DISPLAY command or a Reset.

START VIRTUAL DISPLAY command will not activate a display and results in a NOP until a MODESET command is issued after a Reset.

STOP DISPLAY

0000 0011 CMD Byte

The display process is deactivated upon receiving this command. The DIP and VDIP status bit are reset and the BUSY flag is cleared.

This command blanks the display. HSYNC and VSYNC are not affected.

MODESET

0000 0100 CMD Byte

The Mode Pointer contained in command block location (CBP + 30) is used to access the Mode Block and the modes are fetched sequentially and loaded into the corresponding internal registers in 82730. LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Positions are fetched from the Command Block and stored internally upon completion and the BUSY flag is cleared.

The organization of mode words in the mode block and the parameters supplied by them are shown below (See Figure 10). Some of these parameters which are critical to the operation of a text coprocessor are required to remain unchanged over most of normal operation. No provision is made to prevent MODESET from changing these parameters and it is left to the designer to insure that they are not changed.

The modes provide horizontal and vertical mode display parameters, interlace information, DMA burst and spacing specifications, cursor characteristics as well as attribute enables and bit-selects. Typically, this would be the first command issued after initialization. The Mode Block provides all the parameters needed for a complete initialization of the 82730 for display. Thus a single Modeset command can fully initialize the chip. Note that until the first Modeset command is sent, certain functions such as VSYNC and HSYNC are not enabled.

It is necessary to set up proper mode information, before activating a display. Therefore, a display activating commands should not be issued unless proper mode information has been loaded through a MODESET command. START DISPLAY and START VIRTUAL DISPLAY commands will result in a NOP if a MODESET command has not been issued since the most recent Reset.

LOAD CBP

0000 0101 CMD Byte

The address pointer "NEW CBP" contained in the command block is fetched and stored in the CBP register in the text coprocessor, replacing the old CBP. This effectively moves the command block in the memory. The Command byte from the new Command Block is fetched and the specified channel command is executed. The BUSY flag in the new Command Block is cleared upon completion.

LOAD INTMASK

0000 0110 CMD Byte

The interrupt mask contained in location "INT MASK" in the command block is fetched and stored internally in the CRT controller. When a particular mask bit is set, the interrupt is disabled for a status bit in the corresponding bit position. An interrupt is generated by the text coprocessor by activating the SINT pin, if a status bit is 1 and the corresponding bit in the interrupt mask is 0. The BUSY flag is cleared upon completion.
Interrupts can be enabled for the following status bits.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>BIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RDC</td>
<td>RCC</td>
<td>FDE</td>
<td>EOF</td>
<td>DBOR</td>
<td>LPU</td>
<td>DUR</td>
<td>STATUS WORD</td>
</tr>
</tbody>
</table>

RDC: Reserved Datastream Command Encountered
RCC: Reserved Channel Command Executed
FDE: Frame Data Error (Fetching characters past physical End of Frame)
EOF: End of "n" frames (Logical end of nth frame)
DBOR: Data Buffer Overrun (Row Buffer filled completely without encountering END OF ROW command)
LPU: Light Pen Update
DUR: Data Underrun (Buffer swap initiated before finishing Row Buf loading)

**READ STATUS**

| 0000 | 1000 | CMD Byte |

The internal status register is written to "STATUS" location in the command block. The status register is then cleared, however DIP and VDIP status bits are not cleared. LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Positions are fetched from the Command Block and stored internally. The BUSY flag is then cleared.

**STATUS WORD**

<table>
<thead>
<tr>
<th>15-9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDIP</td>
<td>DIP</td>
<td>RDC</td>
<td>RCC</td>
<td>FDE</td>
<td>EOF</td>
<td>DBOR</td>
<td>LPU</td>
<td>DUR</td>
<td></td>
</tr>
</tbody>
</table>

**LD CUR POS**

| 0000 | 1001 | CMD Byte |

The display row and column positions of cursors 1 & 2 as set in locations "CUR1 ROW," "CUR1 COL," "CUR2 ROW" and "CUR2 COL" in the command block are loaded into internal registers in the CRT controller. Also LISTSWITCH Auto Linefeed and Max DMA Count are loaded from the Command Block and the BUSY flag is cleared. This command is used to change the cursors only. Note that the cursor positions are also updated with the execution of other channel commands.

The cursor characteristics for display are defined by the mode. During the display process, a cursor will be displayed accordingly at the position specified above.

**TEST COMMANDS**

The test commands for the 82730 are issued in the same manner as the normal channel commands. However, the parameters used by test commands are different from those used by the channel commands in normal operation. Therefore, a Test Block which is similar in format to the Command Block is defined. Switching between Command Block and Test Block is accomplished using the "Load CBP" command. The Test Block differs only in the parameter locations associated
with the command. The locations for New CPB, command byte and busy flag are the same for both Command Block and Test Block. The "Test Result" location in Test Block corresponds to the "Status" location in Command Block.

The test commands can be executed, following chip reset, only until the first Modeset command is issued. Once a Modeset command has been executed following chip reset, any subsequent test commands will not be executed and will result in a NOP.

"Self Test" Command

0000 0010 CMD Byte

A built-in Self test is performed using an internal test pattern. The signature generated during the test is written to the Test Result location (TBP+18) in the Test Block. The Busy Flag in the Test Block is then cleared. The Self Test command must be immediately preceded by a chip reset in order to ensure a consistent signature.

The Test Block format for issuing the Self Test command is shown in Figure 6b.

"Row Buffer Test" Command

0000 1011 CMD Byte

The Load Pointer in Test Block is fetched. It points to the system memory area storing the test pattern to be used for testing the on-chip RAM (i.e. the Row Buffers). The Store Pointer, which points to memory area where the data read back from the RAM will be written, is also fetched from Test Block.

Successive words are fetched from memory and written to the Row Buffer, until it is completely filled. Note that three extra words beyond the maximum Row Buffer capacity will be fetched. If N = Max Row Buffer capacity, (N+3) words will be fetched from memory. The extra words fetched will be ignored. The Row Buffer contents are then read back and are written to successive locations in memory area pointed to by the Store Pointer. The test is then repeated on the second Row Buffer. Note that the (N+4)th word in the pattern stored in memory constitutes the first word written to the second Row Buffer. The data storage for the Row Buffer test patterns is illustrated in Figure 6c.

Figure 6b. Test Block Format for "Self Test" Command
(For both 16-bit and 32-bit addressing modes)
Internally, the Row Buffers are 17-bits wide, while the data path is 16-bits wide. During the writing of data to Row Buffers, a complement of bit 15 is written to bit 16 of the Row Buffer in order to test all 17 bits. During the read back, two data words are stored in system memory for each location in the Row Buffer. The first word will consist of bits 0-15 read from the Row Buffer, while the second word will consist of bits 0-14 and bit 16 from the Row Buffer. Thus a total of 4*N words will be stored back in system memory as a result of the Row Buffer Test (2*N for each Row Buffer).

A signature is generated during the test and is written to Test Result location in Test Block upon completion. The BUSY flag in the Test Block is then cleared.

The Test Block format for issuing the Row Buffer Test command is illustrated in Figures 6d.1 and 6d.2. Note that the locations for Load Pointer and Store Pointer parameters are different for 16-bit and 32-bit addressing modes.

---

**Figure 6c. Data Storage for Row Buffer Test Command**
Figure 6d.1 Test Block Format for "Row Buffer Test" Command
(32-bit addressing mode)

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>BIT 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND</td>
<td>BUSY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RESERVED)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD POINTER LOWER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD POINTER UPPER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE POINTER LOWER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE POINTER UPPER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RESERVED)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW CBP LOWER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW CBP UPPER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST RESULT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST BLOCK POINTER (TBP)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+18</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6d.2 Test Block Format for "Row Buffer Test" Command
(16-bit addressing mode)

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>BIT 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND</td>
<td>BUSY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RESERVED)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RESERVED)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD POINTER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RESERVED)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE POINTER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RESERVED)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW CBP LOWER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW CBP UPPER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST RESULT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST BLOCK POINTER (TBP)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBP+18</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOP

0000 0000 0000

LISTSWITCH, Auto Linefeed, Max DMA Count, and Cursor Positions are fetched from the command block and stored internally as in all other channel commands. The Busy flag is then cleared.
**82730 DATASTREAM COMMANDS**

**Datstream Commands**

Datstream Commands are commands embedded in the data fetched from memory by the data access task. These commands are differentiated from character data by the command bit. The most significant bit (MSB) of each data word is designated as the command bit. If the command bit is "1", the lower 15 bits of the data word are interpreted as a datstream command, while if the command bit is "0" the lower 15 bits (or 7 bits if DTW16=0) are interpreted as character data.

**Datstream Command Operation**

During the data access task, the Micro Controller Unit (MCU) examines the command bit of each data word fetched. If the command bit is 1, it executes the datstream command specified in the data word. Otherwise, it stores the lower 15 bits of the data word in the Row Buffer as character data. This process is repeated for each data word fetched.

Datstream commands can be used for changing Row Characteristics on a row by row basis, for carrying out editing functions and for formatting data into rows and frames. These commands are executed by the MCU immediately after they are encountered. As a convenience for the user, the set of all possible command codes starting with "11" in the two most significant bits has been designated as NOP commands. The user can use these command codes for any desired purpose. All other command codes which are not presently defined, are reserved for future expansion and should not be used by the user. The currently undefined codes cause the RDC (Reserved Datstream Command) status bit to be set and also generate an interrupt, if enabled. Reserved command codes should not be used.

**Datstream Command List**

Table 4. 82730 Datstream Commands

<table>
<thead>
<tr>
<th>COMMAND CODE</th>
<th>OP CODE</th>
<th>PARAMETERS</th>
<th>OP CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ENDROW</td>
<td>1000</td>
<td>0000</td>
<td>XXXX</td>
</tr>
<tr>
<td>2 EOF</td>
<td>1000</td>
<td>0001</td>
<td>XXXX</td>
</tr>
<tr>
<td>3 END OF STRING &amp; END OF ROW</td>
<td>1000</td>
<td>0010</td>
<td>XXXX</td>
</tr>
<tr>
<td>4 FULROWDESCRPT</td>
<td>1000</td>
<td>0011</td>
<td>&quot;n&quot;</td>
</tr>
<tr>
<td>5 SL SCROLL STRT</td>
<td>1000</td>
<td>0100</td>
<td>XXXX</td>
</tr>
<tr>
<td>6 SL SCROLL END</td>
<td>1000</td>
<td>0101</td>
<td>XXXX</td>
</tr>
<tr>
<td>7 TAB TO n</td>
<td>1000</td>
<td>0110</td>
<td>&quot;n&quot;</td>
</tr>
<tr>
<td>8 LD MAX DMA COUNT</td>
<td>1000</td>
<td>0111</td>
<td>COUNT</td>
</tr>
<tr>
<td>9 ENDSTRG</td>
<td>1000</td>
<td>1000</td>
<td>XXXX</td>
</tr>
<tr>
<td>10 SKIP n</td>
<td>1000</td>
<td>1001</td>
<td>&quot;n&quot;</td>
</tr>
<tr>
<td>11 REPEAT n</td>
<td>1000</td>
<td>1010</td>
<td>&quot;n&quot;</td>
</tr>
<tr>
<td>12 SUB SUP n</td>
<td>1000</td>
<td>1011</td>
<td>&quot;n&quot;</td>
</tr>
<tr>
<td>13 RPT SUB SUP n</td>
<td>1000</td>
<td>1100</td>
<td>&quot;n&quot;</td>
</tr>
<tr>
<td>14 SET GEN PUR ATTRIB</td>
<td>1000</td>
<td>1101</td>
<td>GPA OP</td>
</tr>
<tr>
<td>15 SET FIELD ATTRIB</td>
<td>1000</td>
<td>1110</td>
<td>XXXX</td>
</tr>
<tr>
<td>16 INIT NEXT PROCESS</td>
<td>1000</td>
<td>1111</td>
<td>XXXX</td>
</tr>
<tr>
<td>(Command process command)</td>
<td>10XX</td>
<td>XXXX</td>
<td>XXXX</td>
</tr>
<tr>
<td>17 (RESERVED)</td>
<td>11XX</td>
<td>XXXX</td>
<td>XXXX</td>
</tr>
</tbody>
</table>
The preceding commands are recognized as valid datastream commands. The corresponding command codes are also indicated. It should be noted that the most significant bit of the command bit is always 1, in order for the word to be interpreted as command.

The "Init Next Process" command can be issued only through a command process in Virtual Screen Display. It is included in this list because its operation is analogous to a datastream command in a virtual screen access environment. Also, in virtual screen display certain datastream commands are interpreted differently, depending upon whether they are encountered in a process datastream or as command process commands. When a command is ignored (becomes a NO-OP) in a virtual display, any parameters that are associated with it are also ignored. The command process command operation is discussed separately. The operation of all other datastream commands is described below.

**ENDROW**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000</td>
<td>000</td>
<td>XXXX</td>
<td>XXXX</td>
</tr>
</tbody>
</table>

This command signifies that no more characters will be loaded in the Row Buffer for this row and an End of Row indicator is stored accordingly. When the row currently being loaded is displayed, the Display Generator (DG) will blank the screen from the end of row character position until the physical end of row.

The Micro Controller Unit (MCU) stops fetching data and waits for DG to swap the Row Buffers. The data access task is resumed following the buffer swap. If a physical end of frame is reached while the MCU is waiting for a buffer swap the MCU ceases to wait and executes an EOF (End of Frame) command.

In virtual display, this command is interpreted as a VEOR (Virtual End of Row) if encountered in a virtual process datastream.

**VEOR**

ENDROW command in a virtual process datastream is interpreted as VEOR (Virtual End of Row) and it terminates a virtual row. The current LPTR is stored in the process header addressed by the "Process Addr" register. The Max Count register is also stored in the Max DMA Count location in the process header. Similarly, the Field Attribute Mask is also stored in the header. In addition, in auto linefeed mode (ALF = 1) other parameters characterizing the process state are also saved in the header. The "Process Addr" register is loaded with the address of the header of the next process fetched from the Access table. The "Access Tab Addr" register is post-incremented by two if a 16-bit addressing option is used and by four if 32-bit addressing is used. The data access task is then resumed for the next process.

**EOF**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000</td>
<td>0001</td>
<td>XXXX</td>
<td>XXXX</td>
</tr>
</tbody>
</table>

This command (End of Frame) signifies that no more characters will be loaded in the Row Buffers for this frame. The Micro Controller Unit (MCU) stops fetching data words and waits for the physical end of frame. If a virtual display is in progress, this command is interpreted as VEOS (Virtual End of Frame), if encountered in a virtual process datastream.

The Display Generator (DG) swaps the row buffers at the end of the current display row and starts displaying the row containing the EOF command. When the character preceding the EOF command is displayed, the DG blanks the screen until the physical end of frame. The MCU fetches the Status Row data then waits until its display is completed. It then performs the actions described below.

If LPEN has been enabled and a rising edge on the LPEN input has been detected, the LPENROW and LPENCOL positions in the command block are updated and the LPU status bit is set. If a Channel Attention has occurred, i.e., if CA has been activated, the command byte is fetched from command block and the specified channel command is executed. If the command issued is a "Stop Display" command, the MCU will terminate the display process and wait for the next channel attention. Otherwise, the MCU resumes the data access task by reinitializing pointers for the new frame and continues to fill the Row Buffers.

**VEOF**

EOF command in a virtual process datastream is interpreted as VEOF (Virtual End of Frame). It provides for reinitialization of LPTR using LISTSWITCH, LBASE0 and LBASE1 for each process, analogous to the automatic reinitialization of LPTR at each end of frame in a Normal Display.
LPTR for the current process is reinitialized using LISTSWITCH, LBASE0 and LBASE1 contained in the process header. The End of Display (EOD) bit in the header is set to 1. The current process is terminated as in a VEOR and the next process in Access Table is accessed.

The EOL (End of Line) command has a combined effect of NXTROW and NXTSTRG commands. All the actions performed in an END OF ROW command are carried out. In addition a END OF STRING command is executed before resuming the data access task. Thus, following, the end of row, the data access is continued with the next data string. In virtual process datastream, this command has the combined effect of VEOR and END OF STRING.

<table>
<thead>
<tr>
<th>Upper Byte</th>
<th>Lower Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>RVV BLK DBL W ROW HGT DEF NRMSTRT SUPSTRT SUBSTRT CUR1 STRT CUR2 STRT UL2 LINE SEL UL1 LINE SEL</td>
</tr>
</tbody>
</table>

RVV ROW, when this bit is set the CRVV pin will be inverted for the next full row. BLK ROW, when this bit is set the row will be blanked (BLANK high). DBLHGT, when the double height bit is set, all character are displayed with twice the scan lines per row. WDEF, when the width defeat bit is set, the WDEF pin is activated for the entire row.

The following can be programmed from 0 to 31 yielding a range of 1 to 32 lines. LPR specifies number of lines per row.

NRMSTRT, SUPSTRT, SUBSTRT specify line numbers in a display row which mark the start of normal, superscript and subscript characters respectively.

NRMSTOP, SUPSTOP, SUBSTOP specify line numbers in a row where normal, superscript and subscript characters end respectively.

CUR1 STRT, CUR2 STRT specify the starting line numbers in a row for cursor 1 and cursor 2 respectively.

ULINE1 SEL, ULINE2 SEL specify the line numbers in a row where underline 2 will appear respectively.

All FULROWDESCRIPT parameters affect the row in which they are programmed and stay in effect until changed by another FULROWDESCRIPT command.

The next "n" words fetched from memory are loaded into the Row Characteristics holding registers. "n" is specified by the lower order byte of the command word and should be between 0 and 7.

The parameters loaded by this command will be used to define the row characteristics at the time the row currently being loaded is displayed. The data words defining these characteristics which follow the FULROWDESCRIPT command must be ordered and organized in memory in a specific format. The format for FULROWDESCRIPT parameters is shown below in Figure 6e starting with "Lines Per Row" as the first parameter loaded.

This command will be ignored if encountered in a virtual process datastream. The MSB of all the parameters must be zero for proper operation in virtual display.
The Slow Scan register in 82C3 is loaded with the scroll line specified by the five least significant bits of the command word. When the row currently being loaded is displayed, the line count for that row will start with the value specified by the Slow Scan register. A "Margin" (MGN) parameter, loaded by MODESET, specifies the number of blank lines plus one to be added at the top of the slow scroll field on the screen. This ensures the availability of sufficient DMA time for fetching the next row, when only a small number of scan lines are displayed in the top row of slow scroll window. This command is used for starting a slow scroll. (Note: MGN = 0 results in no margin buffer lines)

This command will be ignored if encountered in a virtual process datastream or if a SL SCROLL END command is encountered later on the same row.

The scroll location in row characteristics holding registers is loaded with the number of lines specified by the five least significant bits of the command word. This number specifies the number of lines to be displayed when the row currently being loaded is displayed. This is used instead of the regular LPR (Lines Per Row) characteristics, for this particular row. This command is used in the last row of a slow scroll for terminating a slow scroll. The Margin (MGN) parameter, loaded by MODESET, is used in the same way as in slow scroll start except that the specified number of blank lines are inserted at the bottom of the slow scroll in this case. This command will be ignored if encountered in a virtual process datastream or if followed by a SL SCROLL STRT on the same row.

The lower byte of the command word specifies the column (RCLK count) after SYNCSTRG at which a Tab should occur. At display time, after the character preceding the Tab command is displayed, the screen is blanked until the RCLK count specified by the command ("n") is reached. After reaching the specified count, display is resumed by displaying the character following the TAB command.

If the RCLK count specified by the Tab command has already occurred before beginning the blanking for Tab, the display will be blanked until the end of the row.

This command is ignored, if encountered in a virtual display process datastream.

The Max Count register in 82730 is loaded with the Max DMA Count specified by the lower byte of the command word. The DMA Counter is also reinitialized with the Max Count value in the Command Block after all channel commands.

MAX DMA Count is programmable in the range of 1 to 256 (MAX COUNT value 0 equals 256). However, counts greater than the row buffer capacity will cause row buffer overruns if the data strings depend on MAX DMA to terminate the fetching.

The DMA counter is decremented for each data word as the Row Buffer is being loaded. Datastream commands and words supplying parameters for datastream commands as in FULROW-DESCRPT, are not counted. Superscript/Subscript characters are counted in pairs, i.e., a pair of characters causes only one count.

In virtual screen display, every time a new process is accessed, the DMA counter is initialized with the Max DMA Count contained in the process header. This value is also stored in a Max Counter register.

At virtual end of row (VEOR) the Max Count register is written to the process header. The "LD Max DMA Count" command is ignored if encountered in a virtual process datastream.

The SPTR register in the 82730 is loaded with a new String Pointer (SPTR) value fetched from the memory location indexed by the List Pointer (LPTR), which is stored in the LPTR register. The
L PTR register is incremented by two if a 16-bit addressing option is used and by four if 32-bit addressing is used. When more than one 82730 is connected in a cluster, each of them adds an offset, determined by its position in the cluster, to the pointer fetched from memory, before storing it in its SPTR register.

This command directs the data access to the next data string in the list of strings indexed by L PTR. The operation of this command is identical for a Virtual or Normal Display. In virtual display, the next data string within the current display process is accessed.

**SKIP n**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000</td>
<td>1001</td>
<td>n</td>
<td></td>
</tr>
</tbody>
</table>

The next "n" data words fetched from memory are ignored. "n" is specified by the lower byte of the command word and is programmable from 0 to 255. If n equal to 0 is specified, no words are skipped. Any datastream commands encountered in the data fetch are not counted towards these n words. Also parameters following the datastream command as in FUL ROW DESCRIPT are not counted. All embedded datastream commands are executed with the following exceptions.

If a Tab To N data stream command is encountered during the execution of a Skip N command, the Tab command will result in a NOP, i.e. a Tab embedded in the data to be skipped will be ignored.

If an EOL (End Of Line) data stream command is encountered during the execution of a Skip N command, it will be executed with the following effect. In non-auto line feed mode, (ALF = 0) the EOL command has the combined effect of End Of Row and End Of String commands. In auto line feed mode, (ALF = 1) the EOL command has the effect of an End Of String command only.

If the data words skipped include any superscript-subscript characters, they are skipped in pairs and a pair of characters is counted as only one count in "n". If another skip command is encountered it's value of "n" is added to the present skip count and skipping continues.

**REPEAT n**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000</td>
<td>1010</td>
<td>n</td>
<td></td>
</tr>
</tbody>
</table>

The next data word (byte, if DTW16=0) fetched from memory is stored in the Row Buffer "n" times, where "n" is specified by the lower byte of the command word. "n" is programmable from 0 to 255. If n equal to 0 is specified no repetitions will occur, and the word following the Repeat n command will be ignored. This character will eventually be displayed n times. The DMA counter is also made to count n times. In non-auto linefeed mode (ALF = 0), reaching Max DMA Count before the n repetitions are completed will result in a termination of the Repeat n command. This command will also be terminated if the Row Buffer gets filled completely before the n repetitions are completed.

It should be noted that the data word immediately following the Repeat n command is treated as character data, irrespective of the value of its command bit.

**SUP/SUB n**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000</td>
<td>1011</td>
<td>n</td>
<td></td>
</tr>
</tbody>
</table>

The next "n" pairs of data words (bytes, if DTW16 = 0) fetched from memory are treated as superscripts or subscript characters. "n" is specified by the lower byte of the command word. These n pairs are assumed to be ordered with the superscript preceding the subscript.

No datastream commands are permitted in the 2n words following this command. All of these words are interpreted as superscript-subscript pairs. The DMA counter is made to count only once for each pair of characters. In non-auto line feed mode (ALF=0), reaching the Max DMA Count will result in a termination of this command. If n equal to zero is specified, no action will result.

**RPT SUB/SUP n**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000</td>
<td>1100</td>
<td>n</td>
<td></td>
</tr>
</tbody>
</table>

The operation of this command is similar to that of the "Repeat n" command except that the pair of characters following the "RPT SUB/SUP n" command is repeated n times. "n" is specified by the lower byte of the command word and is programmable from 0 to 255. If n equal to zero is specified, no repetitions will occur, and the two data words following the "RPT Sub/Sup n" command will be ignored. The two data words (bytes, if DTW16=0) immediately following the command word are interpreted as a superscript-subscript pair and are repeated. The DMA counter is made to count only once for each repetition of the pair. In non-auto linefeed mode (ALF=0), reaching Max DMA Count prior to completion of n repetitions will cause a termination of this command.
SET GEN PUR ATTRIB

<table>
<thead>
<tr>
<th>GPA OPERAND</th>
<th>GPA OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPA DATA</td>
<td>GPA DATA</td>
</tr>
<tr>
<td>GPA EN</td>
<td>GPA EN</td>
</tr>
</tbody>
</table>

This command provides control over the output pins assigned to General Purpose Attributes, GPA1 through GPA4. The GPA in the Process Header is updated each time a SET GPA command is executed. Thus the GPA state in the header is updated to reflect any changes caused by the "Set Gen Pur Attrb" command. The GPA command occupies a character space on the screen. Consequently, a GPA command is counted as a character towards MAX DMA count. However, a GPA command nested in a Skip N or a TAB to N command is skipped, i.e., it has no effect.

The encoding of the operand, specifying GPA operation, is shown below.

```
SET FIELD ATTRIB

<table>
<thead>
<tr>
<th>GPA OPERAND</th>
<th>GPA OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPA DATA</td>
<td>GPA DATA</td>
</tr>
<tr>
<td>GPA EN</td>
<td>GPA EN</td>
</tr>
</tbody>
</table>
```

The word following this command is fetched. This word is used as a Field Attribute Mask in storing all subsequent display data words in row buffer. The bits in the data words fetched from memory corresponding to the bit positions containing a "1" in Field Attribute Mask are all set to 1 before storing the data word in row buffer. The Field Attribute Mask is used on all display data words fetched from memory. The mask register will contain all 0's upon reset and is cleared at the beginning of each frame.

```
NOP

<table>
<thead>
<tr>
<th>GPA OPERAND</th>
<th>GPA OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPA DATA</td>
<td>GPA DATA</td>
</tr>
<tr>
<td>GPA EN</td>
<td>GPA EN</td>
</tr>
</tbody>
</table>
```

No action is taken. The data access task is resumed by fetching the next data word.

---

Datastream Command Conventions

The reaching of Max DMA Count, encountering of terminating commands such as ENDROW, EOF, etc. and occurrences of these while executing a "skip n" command give rise to various possible combinations of events. The behaviour of 82730 under these circumstances is described below:

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPA GPA GPA GPA GPA GPA GPA GPA</td>
</tr>
<tr>
<td>OPERAND EN EN EN EN EN EN EN</td>
</tr>
</tbody>
</table>

The GPA in the Process Header is updated each time a SET GPA command is executed. Thus the GPA state in the header is updated to reflect any changes caused by the "Set Gen Pur Attrb" command. The GPA command occupies a character space on the screen. Consequently, a GPA command is counted as a character towards MAX DMA count. However, a GPA command nested in a Skip N or a TAB to N command is skipped, i.e., it has no effect.

The encoding of the operand, specifying GPA operation, is shown below.

```
SET FIELD ATTRIB

<table>
<thead>
<tr>
<th>GPA OPERAND</th>
<th>GPA OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPA DATA</td>
<td>GPA DATA</td>
</tr>
<tr>
<td>GPA EN</td>
<td>GPA EN</td>
</tr>
</tbody>
</table>
```

The word following this command is fetched. This word is used as a Field Attribute Mask in storing all subsequent display data words in row buffer. The bits in the data words fetched from memory corresponding to the bit positions containing a "1" in Field Attribute Mask are all set to 1 before storing the data word in row buffer. The Field Attribute Mask is used on all display data words fetched from memory. The mask register will contain all 0's upon reset and is cleared at the beginning of each frame.

```
NOP

<table>
<thead>
<tr>
<th>GPA OPERAND</th>
<th>GPA OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPA DATA</td>
<td>GPA DATA</td>
</tr>
<tr>
<td>GPA EN</td>
<td>GPA EN</td>
</tr>
</tbody>
</table>
```

No action is taken. The data access task is resumed by fetching the next data word.

---

The behaviour of 82730 under these circumstances is described below:

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPA GPA GPA GPA GPA GPA GPA GPA</td>
</tr>
<tr>
<td>OPERAND EN EN EN EN EN EN EN</td>
</tr>
</tbody>
</table>

When Max DMA Count is reached, it has the effect of a VEOR command if a Virtual Display is in progress or a ENDROW command if a Normal Display is in progress. It also causes an automatic end of string i.e., the effect of a NXTSTRG command in non-auto linefeed mode (ALF = 0).

In non-auto linefeed mode, "Repeat n", "Sub/Sup n" and Rpt Sub/Sup n" commands are terminated upon reaching a max DMA count, even if "n" is not reached.

"Skip n" command is terminated if EOF command is encountered. It is also terminated upon encountering an ENDROW command in non-auto linefeed mode (ALF = 0).

"Repeat n" "Sub/Sup n" and "RPT Sub/Sup n" commands can be nested within a "Skip n" command. If superscript-subscript characters are skipped, each pair of characters counts as one skipped character. If the above commands are encountered during a "skip n" and if the specified count (n) in these commands is not reached by the end of execution of the "skip n" command, the execution of the nested command is continued beyond the termination of "skip n" command until the remaining portion of the count specified in the nested command is completed.
VIRTUAL SCREEN MODE

Command Process Commands

In Virtual Screen Display, 82730 accesses display processes and command processes through the Access table. The command processes enable the I/O Driver process to direct 82730 to execute certain data stream commands by inserting an appropriate command process address in the Access table. This capability enables the preservation of uniformity and consistency of operation between normal and virtual environments, by assigning different interpretations to the command according to the access environment. It is especially useful for termination and initialization commands. The operation of command process commands is analogous to that of data stream commands except for a different access environment.

Command Process Command List

The commands allowed in command processes can be divided into two subsets. The first subset consists of commands that can be issued only through a command process, while the second one consists of normal data stream commands that can also be issued through a command process. The command code for a data stream command issued through a command process is the same as that for the normal data stream command embedded in the data. However, certain data stream commands are interpreted differently when they are issued through a command process as opposed to embedding in the data stream of a virtual display process. The most significant bit (MSB) of the command word must be a "1". In the data stream, this bit distinguishes a command word from character data. In the process environment, this bit distinguishes a command process from a display process. The commands permitted in command processes are listed below. No other commands will be recognized if encountered in a command process and will result in a NOP. All undefined command codes apart from those designated as NOP are reserved and should not be used. Encountering an illegal command code causes the RDC (Reserved Datastream Command) status bit to be set and will generate an interrupt, if enabled.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 INIT NEXT PROCESS</td>
<td>NOP</td>
<td>1000</td>
<td>1111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command Process or Datastream Commands:</th>
<th>Command Code</th>
<th>Command Code</th>
<th>Op Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 ENDRW</td>
<td>VEOR</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>3 EOF</td>
<td>VEOR</td>
<td>1000</td>
<td>0001</td>
</tr>
<tr>
<td>4 EOL</td>
<td>VEOR + NXTSTRG</td>
<td>1000</td>
<td>0010</td>
</tr>
<tr>
<td>5 FULROWDESCRIPT</td>
<td>NOP</td>
<td>1000</td>
<td>0011</td>
</tr>
<tr>
<td>6 SL SCROLL STRT</td>
<td>NOP</td>
<td>1000</td>
<td>0100</td>
</tr>
<tr>
<td>7 SL SCROLL END</td>
<td>NOP</td>
<td>1000</td>
<td>0101</td>
</tr>
<tr>
<td>8 TAB TO n</td>
<td>NOP</td>
<td>1000</td>
<td>0110</td>
</tr>
<tr>
<td>9 LD MAX DMA COUNT</td>
<td>NOP</td>
<td>1000</td>
<td>0111</td>
</tr>
<tr>
<td>10 (RESERVED)</td>
<td>RESERVED</td>
<td>10XX</td>
<td>XXXX</td>
</tr>
<tr>
<td>11 NOP</td>
<td>NOP</td>
<td>11XX</td>
<td>XXXX</td>
</tr>
</tbody>
</table>

Table 5. Command Process Command List
This command can be used only in a command process to initiate a virtual display "window".

Upon receiving this command, the command process is terminated and the next process in Access Table is accessed by fetching the new process address. However, the LPTR register is not directly loaded from the LPTR location in the process header. Instead, LISTSWITCH in the process header is examined and LPTR is initialized with the value LBASE 0 or LBASE 1 depending upon whether LISTSWITCH is 0 or 1 respectively. Both LBASE0 and LBASE1 are contained in the header.

The process header format is shown in Figure 7. Also the End of Display Bit (EOD) in the header is reset.

The data access task for a virtual display is then resumed, with this value of LPTR.

---

**Figure 7. Process Header for Display and Command Process**
### ENDROW

**Command:**
```
15 14 8 7 0
  1 000 0000 XXXX XXXX
```

The actions performed by a ENDROW data-stream command in a Normal Display are carried out. The next process in Access Table is accessed and the data access task is resumed, after the next Row Buffer swap.

### EOF

**Command:**
```
15 14 8 7 0
  1 000 0001 XXXX XXXX
```

The actions performed by an EOF (End of Frame) data stream command in a Normal Display are carried out.

### EOL

**Command:**
```
15 14 8 7 0
  1 000 0010 XXXX XXXX
```

This command is identical to ENDROW command in Virtual Display in Command Process environment. ENDSTRG, which is strictly a data operation within a display process is meaningless in the command process environment.

### FULROWDESCRPT

**Command:**
```
15 14 8 7 0
  1 000 0011 "n"
```

The actions performed by the FULROWDESCRPT datastream command are carried out. The data access task is resumed by accessing the next process in the Access Table.

### SL SCROLL STRT

**Command:**
```
15 14 8 7 5 4 0
  1 000 0100 XXX "SCR LINE"
```

The same actions as the SL SCROLL STRT datastream command. The data access is resumed with the next process in Access Table.

### SL SCROLL END

**Command:**
```
15 14 8 7 5 4 0
  1 000 0101 XXX "END LINE"
```

The actions performed by a SL SCROLL END datastream command, in a Normal display, are carried out. The data access task is resumed with the next process in Access Table.

### TAB TO n

**Command:**
```
15 14 8 7 0
  1 000 0110 "n"
```

The effect of this command process command is identical to that of the TAB TO n datastream command. The TAB can be used to establish the left edge of a virtual display “window”.

### LD MAX DMA COUNT

**Command:**
```
15 14 8 7 0
  1 000 0111 MAX COUNT
```

The Max Count register on 82730 is loaded with the value specified by the lower byte of the command word. The DMA counter is also initialized with this Max Count Value.

The next process in the Access Table is accessed. However, the Max DMA Count value in the process header is not used for initializing the DMA counter. Instead, the DMA counter as initialized by the LD Max DMA Count command is used for this process. The virtual display data access task is then resumed normally. When the process is terminated, the new Max Count value is written to the process header. Thus the Max Count value in the header is updated as a result of this command.

### NOP

**Command:**
```
15 14 8 7 0
  1 1XX XXXX XXXX XXXX XXXX
```

No action is taken. Data access task is resumed by fetching the next process address from Access Table.

### ERROR AND STATUS HANDLING

#### Error Conditions

Since the MCU and DG function asynchronously with respect to each other, different relative timings in MCU and DG operation are possible, some of which result in error conditions. The lack of appropriate termination commands for row or frame data in the datastream also gives rise to certain error conditions. These types of situations occurring in display process operation are described below.

In normal operation, DG initiates a buffer swap at the physical end of a display row. If the MCU has not finished loading its row buffer by that time, a “Data Underrun” occurs. This results in
blanking of the screen until physical end of frame by DG and execution of an EOF (End of Frame) command by MCU. Data underrun also occurs when the first row of the frame has not finished loading by the start of the character field. The entire frame will be blanked in this case.

If a physical end of frame is reached prior to encountering an EOF datstream command, a “Frame Data Error” occurs, which results in the execution of an EOF command by MCU. (Note that this does not disrupt the visible display action, and may not constitute an error for certain data structures. The error indication is included as a flag where knowledge of this condition is desired.) Similarly, when the MCU fills up a row buffer completely, without encountering a END-ROW command, the “Data Buffer Overrun” flag is set.

All of the above conditions result in the setting of an appropriate status bit and generation of an interrupt if the corresponding interrupt has been enabled.

Status and Interrupt Handling

A status word is maintained in an internal register by 82730 and it is written to the “STATUS” location in command block when the “Read Status” channel command is executed. The processor can thus read status information by issuing this command. The processor can also enable interrupts for certain status bits by specifying an interrupt mask which is loaded in 82730 as a result of a “Load Int Mask” channel command. This establishes a communication mechanism between 82730 and the processor for error and status reporting.

Status Word

The format for the status word is shown below. The function of each of the status bits is described below.

<table>
<thead>
<tr>
<th></th>
<th>VDIP</th>
<th>DIP</th>
<th>RDC</th>
<th>RCC</th>
<th>FDE</th>
<th>EOF</th>
<th>DBOR</th>
<th>LPU</th>
<th>DUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RESERVED)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VDIP: Virtual Display In Progress
DIP: Display In Progress
RCC: Reserved Channel Command
RDC: Reserved Datasream Command
FDE: Frame Data Error

DUR: Data Under Run

This status bit is set by Display Generator if the Microcontroller Unit (MCU) has not finished loading its Row Buffer when the DG initiates a buffer swap at the physical end of a display row. This condition is defined as data underrun and causes the MCU to execute an EOF command and the DG to blank the screen until the physical end of frame.

LPU: Light Pen Update

This status bit is set by the MCU after updating the LPENROW and LPENCOL locations in command block. The detection of LPEN input is enabled by the LPEN ENABLE channel com-

EOF: End of Frame
DBOR: End of Row
LPU: Light Pen Update
DUR: Data Under Run

mand. The detection of a rising edge on the LPEN input causes the current row and column position to be stored internally. The MCU updates the LPEN ROW and LPENCOL locations in command block at the next end of frame and sets the LPU status bit. Further updates of these command block locations are inhibited until another LPEN ENABLE command is issued.

DBOR: Data Buffer Over Run

This status bit is set when the MCU tries to fill a row buffer beyond its capacity. The MCU will stop fetching characters after this point and the display is blanked following the completion of the row currently being displayed.

All status bits are cleared by a Reset.
EOF: End of Frame
This bit is set by the DG at the physical end of the nth frame, where 'n' is specified by the MODESET parameter FRAME INTERRUPT COUNT. This provides the means for timing frame related events such as slow scrolls.

FDE: Frame Data Error
This status bit is set by the DG at the physical end of frame if no EOS datastream command has been encountered until then. This also results in the execution of the EOS command by the MCU.

RCC: Reserved Channel Command
This bit is set by the MCU upon encountering an illegal datastream or command process command. This can be used to trap software errors during program development.

RDC: Reserved Datastream Command
This bit is set by the MCU upon encountering an illegal datastream or command process command. This can be used to trap software errors during program development.

DIP: Display In Progress
This bit is set by the MCU immediately after receiving a "Start Display" channel command. It remains set as long as the display process is active and is reset upon receiving a "Start Virtual Display" or "Stop Display" command or a Reset. Interrupts cannot be enabled for this status bit.

VDIP: Virtual Display In Progress
This bit is set by the MCU immediately after receiving a "Start Virtual Display" channel command and is reset upon receiving a "Start Display" or "Stop Display" command or a Reset. This bit remains active as long as the virtual display process is active. Interrupts cannot be enabled for this status bit.

Interrupt Processing
The system processor can enable interrupts on any of the status bits, with the exception of DIP and VDIP bits, by specifying an interrupt mask. A "1" in a bit position in the interrupt mask disables (masks out) interrupts on the status bit located in the corresponding bit position in the status word. The format for Interrupt Mask is shown below. The Int Mask can be loaded into 82730 from the INTMASK location in command block by a "Load Int Mask" channel command.

If the interrupt is enabled for a particular status bit by programming a "0" in the corresponding bit position in INTMASK and if the status bit gets set during the course of the display, an interrupt will be generated by 82730 at the next end of frame. At the end of frame, the 82730 will first perform the tasks of updating LPEN position (if required) and servicing the Channel Attention (if CA was activated). Then the status word in the internal register will be written to the INT GENERATION CODE location in the Command Block and the SINT output will be activated. The SINT pin is not deactivated until an interrupt reset signal is received at the IRST pin.

82730 continues to perform its normal display task after activating the SINT pin. If no interrupt reset is received until the next end of frame then any new interrupts that might have been generated at that end of frame will be lost. Therefore, it is essential for the system processor to issue an interrupt reset within a frame time after an interrupt is generated.

When the display is not activated, the only interrupt that can occur is the Reserved Channel Command interrupt. Upon receiving an invalid channel command, 82730 will write the status word to INT Generation Code location in the Command Block and activate SINT output, if that interrupt is enabled.

The processor can use the interrupt capability to get status information from 82730. A possible interrupt service routine for the system processor is shown in flow chart form in Figure 9.
### Interrupt Mask

<table>
<thead>
<tr>
<th>15</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDC</td>
<td>RCC</td>
<td>FDE</td>
<td>EOF</td>
<td>DBOR</td>
<td>LPU</td>
<td>DUR</td>
<td>INT</td>
<td>INT</td>
</tr>
<tr>
<td>INT</td>
<td>INT</td>
<td>INT</td>
<td>INT</td>
<td>INT</td>
<td>INT</td>
<td>INT</td>
<td>MASK</td>
<td>MASK</td>
</tr>
<tr>
<td>MASK</td>
<td>MASK</td>
<td>MASK</td>
<td>MASK</td>
<td>MASK</td>
<td>MASK</td>
<td>MASK</td>
<td>MASK</td>
<td>MASK</td>
</tr>
</tbody>
</table>

INT MASK = 0 Enables the corresponding interrupt.
INT MASK = 1 Masks or disables the corresponding interrupt.

**Figure 8. Interrupt Mask**

### Interrupt Service Routine For System Processor

1. **INTERRUPT**
2. **READ STATUS FROM**
   **"INT GENERATION CODE" LOCATION IN CMD BLOCK**
3. **PERFORM APPROPRIATE SERVICE TASKS**
4. **ISSUE INT RESET (IRST) SIGNAL TO 82730**
5. **END**

**Figure 9. Interrupt Service Routine For System Processor**
82730 VIDEO INTERFACE

The Mode Pointer in the Command Block points to a parameter block containing the Mode information required for the display. The organization of the mode words in the Mode Block is shown below.

**Figure 10. Mode Block Organization**

<table>
<thead>
<tr>
<th>FROM MODE POINTER IN COMMAND BLOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 LOCATION</td>
</tr>
<tr>
<td>DMA</td>
</tr>
<tr>
<td>HORIZONTAL MODES</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CHAR ROW CHARACTERISTICS</td>
</tr>
<tr>
<td>(FULROWDESCRIPTION)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>VERTICAL MODES</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>BLINK CONTROL</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE BIT SELECTS</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
CAM ARRAYS

Three Content Addressable Memory arrays are used for generating timing parameters to control the video display: the HORIZ MODE CAM, the VERT MODE CAM and the CHAR ROW CAM. The user has the flexibility to define his own timing parameters by loading them into the CAM arrays via the MIU. All of these parameters can be modified at the end of every frame. All the parameters in the CHAR ROW CAM, except MARGIN, are changeable on a row by row basis. Each of the three CAM arrays is described separately below:

Timing Sources

RCLK and CCLK inputs are provided by the external video logic to the 82730. The RCLK is used to increment the HORIZ COL CNTR and hence generates all horizontal timing parameters. CCLK is used to clock the character and attribute data output from the 82730 to the external display dot logic. Data changes on the positive going edge of RCLK or CCLK.

Initialization

Upon activation of the RESET input, the 82730 display generator will stop all operations in progress and deactivate all outputs. It will stay in this quiescent state until the MIU executes the MODESET command. The following table shows the states of all the Display Generator outputs during and after RESET.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAT0-14</td>
<td>Low</td>
</tr>
<tr>
<td>WDEF</td>
<td>Low</td>
</tr>
<tr>
<td>LCO-4</td>
<td>High</td>
</tr>
<tr>
<td>BLANK</td>
<td>Low</td>
</tr>
<tr>
<td>CSYNC</td>
<td>High</td>
</tr>
<tr>
<td>HOLD</td>
<td>High</td>
</tr>
<tr>
<td>HSYNC</td>
<td>Low</td>
</tr>
<tr>
<td>VSYNC</td>
<td>Low</td>
</tr>
<tr>
<td>CRVV</td>
<td>Low</td>
</tr>
<tr>
<td>RRVV</td>
<td>Low</td>
</tr>
</tbody>
</table>

After reset of the 82730, the CAM arrays are in undetermined states. The CAM arrays are set upon the execution by the MIU of the MODESET command. The HORIZ and VERT MODE CAM contents are especially critical since they are used to generate timing control signals to the external video logic. Without the generation of the timing signals, no display process can take place. Hence, START DISPLAY command cannot be executed before the first MODESET command after the device reset. The START DISPLAY command will be ignored if it precedes the MODESET command.

The row buffers also contain unknown information after power up and reset. In executing the START DISPLAY command, the MIU would first load the two row buffers with the first two rows of character data to be displayed. Upon completion of loading of both buffers, it will signal the DG to begin the display process. In this way, only valid character data will be output to the external video logic.

Timing Parameters

The timing parameters read from the MODESET Block and stored in the VERT MODE CAM and HORIZ MODE CAM are used to control the video display and they can be best illustrated in the Map of Timing Parameters shown below. All of these timings have to be defined after power up and reset and can be changed on a frame by frame basis during display.
Row Timing Parameters

The row timing parameters are stored in HORIZ MODE CAM and are programmable from 0 to 255 RCLK times. These parameters are:

(a) HSYNCSTRT - Horizontal Sync Start. The RCLK count on each scan line where HSYNC pin is activated. This parameter is not programmable. The RCLK period that follows the rising HSYNC edge is defined as column zero. It is used as the reference for all other horizontal timing parameters.

(b) HSYNCSTP - Horizontal Sync Stop. The RCLK count on each scan line where the HSYNC pin is deactivated. The falling edge of HSYNC occurs at the leading edge of the programmed RCLK period.

(c) LINELEN - Line Length. This parameter defines the total number of RCLK's in each scan line including display time, border and horizontal retrace time. There are LINELEN + 1 RCLK periods per horizontal line scan.

(d) HBDRSTRT - Horizontal border start. The RCLK count on a scan line where the border begins. The border begins at the leading edge of the programmed RCLK period.

(e) HBDRSTP - Horizontal Border Stop. The RCLK count on a scan line where the border ends. The border terminates at the leading edge of the programmed RCLK period.

(f) HFLDSTRT - Horizontal Field Start. The RCLK count on a scan line where the character display field begins. If the row buffer is ready to be displayed, the CSYN pin will be deactivated at this point. This field begins at the leading edge of the programmed RCLK period.

(g) HFLDSTP - Horizontal Field Stop. The RCLK count on a line where the character display field stops. When this timing point is reached, CSYN will be activated. This field ends at the leading edge of the programmed RCLK period.

There is also one pseudo parameter, SYNCDLY. It is fixed at one half LINELEN and is used as the start and end timing for VSYNC in odd frames in interlaced displays. VSYNC starts at HSYNCSTRT in even frames for interlaced displays and all frames for non-interlaced displays.
There are certain restrictions in the programming of HFLDSTRT and HFLDSTP and those restrictions are best illustrated below. There has to be at least 4 RCLKS in between HFLDSTRT and HFLDSTP of the same scan line and 15 RCLKS in between HFLDSTP of one line and HFLDSTRT of the next. The minimum delay of 15 RCLKS is for the charging of the pipeline from the row buffer to the character data output DAT0-DAT14 as well as the setting of the correct value for the scan line output LC0-LC4.

**Figure 12. Horizontal Timing Restrictions**

**Frame Timing Parameters**

Frame timing parameters are stored in the VERT MODE CAM and are programmable from 0-2047 scan lines. These parameters are:

(a) VSYNCSTRT - Vertical Sync Start. The line count where the VSYNC is activated. This occurs at the end of a field automatically. This parameter is not programmable. The rising edge of VSYNC occurs with the rising edge of HSYNC for all non-interlace fields and for odd fields in the interlace mode.

(b) VSYNCSTP - Vertical Sync Stop. The line count at which the VSYNC pin is normally deactivated. VSYNC changes at the rising edge of HSYNC normally. However it occurs at SYNCDLY at the beginning of odd fields of an interlaced display.

(c) FRAMELEN - Frame Length. This parameter defines the total number of scan lines per frame. It is used to reset the FRAME LINE CNTR. In an interlaced display, FRAMELEN must be an even number. If an odd number is programmed, one additional line will occur automatically.

There will be FRAMELEN + 1 scan lines per frame. (Note that interlace mode contains two fields per frame).

(d) VFLDSTRT - Vertical Field Start. Programs the scan line count where the character display field begins.

(e) VFLDSTP - Vertical Field Stop. Programs the scan line count where the regular character display field ends. VFLDSTP times the beginning of the Status Row. The channel attention sequences, interrupt handling, row buffer swap and initialization for the next frame are started after the display of the Status Row is completed. See * below.

(Character Field Boundry definition: The starting or ending event is defined to occur at HFLDSTP on the scan line following the programmed value. Thus the visible character field effectively begins two scan lines below the programmed start value and ends one scan line below the programmed stop value.)
Status Row
The Vertical Frame Timing Parameters have no border controls, unlike the Horizontal Row Timing Parameters. The top and bottom borders can be replaced with regular display rows that are video-reversed and contain no data. The top border is easily timed from VFLDSTRT. The bottom border is more difficult without help from the Vertical Timing generators. If there were no help, the user would have to keep track of the number of scan lines used in each row to know when to stop regular display and create the bottom border. This would also preclude his ending his regular display with an EOF command before the border.

The 82730 provides this help with the Status Row feature. The display of the Status row is timed from VFLDSTP and allows the user to display a row in a fixed position at the bottom of the screen that is independent of the regular data and any display errors (display ended by an EOF command or the DURN, DBOR, or FDE errors). (There is one dependency on the regular display data: the row format. The last FULROWDESCRIPRT (FRD) set in the regular data will be used on the Status Row unless a new command is issued for the row. It is recommended that the user include a new FRD command in the Status Row data to eliminate this dependency).

Status Row display starts SCROLL MARGIN plus one scan line after VFLDSTP. This margin is provided to insure enough DMA time if the regular display runs up to VFLDSTP. The user can create a bottom border or any end-of-display row that he chooses. A display status or system status line, or special programmable key function definition line can be implemented with this feature.

CHARACTER ATTRIBUTES
The 15 bits of the character word can be partitioned into character address and attribute bits. Some common attributes may be individually defined and enabled or disabled by fields in the attribute parameter registers. Each attribute has two means of being enabled. The enable bits defined below are set during the MODESET channel command and are used as a global enable. The user does not have to enable the provided attributes. He may free more data bits for his own use this way. The second enable bit is contained in each character loaded to the row buffer to enable the attribute on a character by character basis. They are individually described in detail in the following sections.

Reverse Video
When a character with the reverse video attribute is displayed, the CRVV pin will be inverted during the time the character is being displayed. The reverse video affects the entire height of the row for that character space. For superscript/subscript pairs, the reverse video effect is controlled by superscript until SUBSTRNT when the subscript attribute bit takes control. The parameter for this attribute is:

RVBS - Reverse Video Bit Select. This parameter selects one of the 15 bits of a character data word. Values 0 through 14 select the corresponding bit. Value 15 disables the Reverse Video attribute.

Blinking Character
When a character with the blinking character attribute is displayed, the BLANK pin will be activated and deactivated during the character display time according to programmable rate and duty cycle. The parameters for this attribute are:

(a) BCBS - Blinking Character Bit Select. Selects one of the 15 bits of a character data word as the blinking character attribute control. As with Reverse Video above, the value of the select determines the controlling bit or disables the attribute.

(b) CHAR BLNK FREQ - Selects one of the 32 blinking frequencies available for the blinking character and blinking underline. The character blink rate is calculated as below:

Frame Refresh Rate
Blink Rate = 4 x CHAR BLNK FREQ

(c) CHAR DUTY CYCLE - A 2-bit register to select 4 duty cycles available for blinking character and blinking underline. The selection logic is defined to be as follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>100% always on</td>
</tr>
<tr>
<td>11</td>
<td>75% on</td>
</tr>
<tr>
<td>10</td>
<td>50% on</td>
</tr>
<tr>
<td>01</td>
<td>25% on</td>
</tr>
</tbody>
</table>

Underline #1
When a character with underline is displayed, the BLANK Pin will be activated and the CRVV pin will be inverted during the time the scan line specified...
by the underline select register is displayed. The parameters used to define underline #1 are:

(a) ULS1 - Underline Line Select 1. It determines which scan line of a character row will be used for the underline #1. This parameter is modifiable on a row by row basis by the FULROWDESCRIPT command.

(b) ULBS1 - Underline Bit Select 1. This parameter can only be changed by MODESET. It selects one of the 15 bits of a character data word as the underline #1 attribute control. Again, a value of 15 in the select field disables this attribute.

Underline #2 (Blinking)

Underline #2 can be made to blink. When its blinking feature is deactivated, its visual effect is exactly the same as underline #1. When it is enabled to blink, its blink rate and blinking duty cycle are the same as those defined for blinking character. The parameters used to define this attribute are:

(a) UL2SEL - Underline Line Select 2. This parameter determines which scan line of a character will be the 2nd underline. It is changeable on a row by row basis by the FULROWDESCRIPT command.

The next two parameters can only be modified by the MODESET Command.

(b) ULBS2 - Underline Bit Select 2. Selects one of the 15 bits of a character data word or GPA1 as the second underline attribute control. A bit select value of 15 disables the second underline.

(c) BUE - Blinking Underline Enable. Activation of this bit will cause the second underline attribute to start blinking.

Invisible

A character with this attribute will occupy its character position on the screen but will not be displayed (i.e. BLANK will be active). This attribute does not affect the Reverse Video attribute if they are programmed together. The parameter that is used to implement this attribute:

IBS - Invisible Bit Select. Selects one of the 15 bits of a character data word as the invisible attribute control. Value 15 disables the invisible attribute.

Absolute Line Cntr Attribute

This character attribute allows the display of special graphic characters, or may be used to upshift normal characters to implement displays with overlapping superscript and subscript fields. When a character with this character attribute enabled is being displayed, its LC0-LC4 pins will reflect the output from the CHAR ROW LINE CNTR which counts the absolute line count of a row. The activation of this attribute overrides the line count mode of both normal and subscript/superscript characters. The parameter used to select the attribute is:

ABS LINE BIT SEL. This four bit register selects one of the 15 bits of a character data word as the absolute line counter output attribute control. Select value 15 disables the ABS Line attribute.

Cursor Generation

The cursor characteristic parameters are changeable on a frame by frame basis by MODESET.

(a) CUR FREQ - Cursor frequency. Selects the blinking frequency for both cursors. The selection logic is similar to CHAR BLNK FREQ.

(b) CUR DUTY CYCLE - Cursor duty cycle. Selects the blinking duty cycle for both cursors. Its selection logic is similar to CHAR DUTY CYCLE.

(c) CR1RVV - Cursor 1 Reverse Video Enable selects a reverse video type cursor as opposed to a solid (blanking) cursor.

(d) CR1BE - Cursor 1 Blink Enable changes the cursor 1 block or underline to a blinking block or underline. Enabling this bit also causes DAT 14 pin to "blink" as well, if the CR10E bit is set.

(e) CR10E - Cursor 1 Output Enable reconfigures the DAT 14 pin to indicate when cursor 1 is active. CR20E enabled directs the cursor 2 signal to DAT 13 pin in a similar fashion.

(f) CR1CD - Cursor 1 Light Pen Cursor Detect directs the CCLK cursor #1 position to be translated to its nearest equivalent RCLK position through the LPEN facility.

An identical set of parameters (c) through (f) is available for the generation of CURSOR 2. The two cursors share the same FREQ and DUTY CYCLE parameters.
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ........ 0°C to 70°C
Storage Temperature ............. -65°C to +150°C
Voltage on Any Pin with
  Respect to Ground .......... -1.0V to +7V
Power Dissipation .............. 3 Watts

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS TA = 0°C to 70°C, VCC = 5V ± 10%

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>+0.8</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td>VCC + 0.5</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>VLO</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td></td>
<td>Volts</td>
<td>IOL = 2 mA [1], IOH = -400 μA</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td></td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>Power Supply Current</td>
<td>400</td>
<td></td>
<td>mA</td>
<td>@ TA = 0°C</td>
</tr>
<tr>
<td>ILI</td>
<td>Input Leakage Current</td>
<td>10</td>
<td></td>
<td>μA</td>
<td>VIN = 0 - VCC</td>
</tr>
<tr>
<td>ILO</td>
<td>Output Leakage Current</td>
<td>±10</td>
<td></td>
<td>μA</td>
<td>VOUT = 0.45 - VCC</td>
</tr>
<tr>
<td>ILOCL</td>
<td>LC0, LC1, LC2 Input Low Current</td>
<td>-125</td>
<td></td>
<td>μA</td>
<td>VIN = 0 Volts, Reset = &quot;1&quot; (2)</td>
</tr>
<tr>
<td>VBLI</td>
<td>Bus Clock Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>VBLH</td>
<td>Bus Clock Input High Voltage</td>
<td>2.0</td>
<td>VCC + 1.0</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>VCLI</td>
<td>Character Clock Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>VCHI</td>
<td>Character Clock Input High Voltage</td>
<td>2.2</td>
<td>VCC + 0.5</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>VRLI</td>
<td>Reference Clock Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>VRHI</td>
<td>Reference Clock Input High Voltage</td>
<td>2.2</td>
<td>VCC + 0.5</td>
<td>Volts</td>
<td></td>
</tr>
</tbody>
</table>

NOTE:
1. IOL = 2.6 mA on the S1 and S0 pins.
2. Measured after at least 5 BCLK cycles after RESET = High

A.C. CHARACTERISTICS

TA = 0°C to 70°C, VCC = 5V ± 10%. All timings in nanoseconds. CL = 50 pF.

82730 Bus Interface Input Timing Requirements

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLCL</td>
<td>BCLK Cycle Period</td>
<td>125</td>
<td>2500</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCLCH</td>
<td>BCLK Low Time</td>
<td>52</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCHCL</td>
<td>BCLK High Time</td>
<td>52</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCH1CH2</td>
<td>BCLK Rise Time</td>
<td>30</td>
<td></td>
<td>ns</td>
<td>0.45V → 2.4V (1)</td>
</tr>
<tr>
<td>TCH1CL2</td>
<td>BCLK Fall Time</td>
<td>30</td>
<td></td>
<td>ns</td>
<td>2.4V → 0.45V (1)</td>
</tr>
<tr>
<td>TDVCL</td>
<td>Data in Set-Up Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>
### A.C. CHARACTERISTICS (Continued)

#### 82730 Bus Interface Input Timing Requirements (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLDX</td>
<td>Data in Hold Time</td>
<td>5</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TARYHCH</td>
<td>Async. READY Active Set-Up Time</td>
<td>35</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TSRYHCL</td>
<td>Sync. READY Active Set-Up Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRYLCL</td>
<td>READY Inactive Set-Up Time</td>
<td>10</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCLRYX</td>
<td>READY Hold Time</td>
<td>20</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCTVCL</td>
<td>HLDA, RESET Set-Up Time</td>
<td>35</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCLCTX</td>
<td>HLDA, RESET Hold Time</td>
<td>10</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCAVCAK</td>
<td>CA Pulse Width</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRIVRIX</td>
<td>IRST Width</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRLLCH</td>
<td>LCx Input Hold Time</td>
<td>5TCLCL</td>
<td></td>
<td>ns</td>
<td>(2)</td>
</tr>
</tbody>
</table>

#### 82730 Bus Interface Output Timing Response

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLAV</td>
<td>Address Valid Delay</td>
<td>0</td>
<td>55</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCLKAX</td>
<td>Address Hold Time</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TAVAL</td>
<td>Address Valid to ALE/UALE Inactive</td>
<td></td>
<td>TCLKCH - 30</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address Hold to ALE Inactive</td>
<td>TCHCL - 10</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCLAZ</td>
<td>Address Float Delay</td>
<td>TCLAX</td>
<td>45</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TAZRL</td>
<td>Address Float to RD Active</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TLHLL</td>
<td>ALE/UALE Width</td>
<td>TCLKCH - 10</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCLKLH</td>
<td>ALE/UALE Active Delay</td>
<td>0</td>
<td>45</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCHLL</td>
<td>ALE/UALE Inactive Delay</td>
<td>0</td>
<td>45</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCVCTV</td>
<td>Control Active Delay (DEN,WR,AEN)</td>
<td>0</td>
<td>70</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCVCTXW</td>
<td>Control Inactive Delay (WR,AEN)</td>
<td>0</td>
<td>80</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCVCTXD</td>
<td>Control Inactive Delay (DEN)</td>
<td>5</td>
<td>80</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCDLOV</td>
<td>Data Out Valid Delay</td>
<td>0</td>
<td>50</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCDLOX</td>
<td>Data Out Hold Time</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>THWDX</td>
<td>Data Out Hold Time After WR</td>
<td>TCLKCL - 60</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCLKHV</td>
<td>Hold Output Delay</td>
<td>0</td>
<td>85</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRLRH</td>
<td>RD Width</td>
<td>2TCLKCL - 50</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCLRL</td>
<td>RD Active Delay</td>
<td>0</td>
<td>95</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCLRRL</td>
<td>RD Inactive Delay</td>
<td>5</td>
<td>70</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRHADV</td>
<td>RD Inactive to Next Address Active</td>
<td>TCLKCL - 40</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:**
2. Applies only to test mode invocation.
A.C. CHARACTERISTICS (Continued)

82730 Bus Interface Output Timing Response (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLSIN</td>
<td>SINT Valid Delay</td>
<td>0</td>
<td>70</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRIHSIL</td>
<td>RINT Active to SINT Inactive</td>
<td></td>
<td>250</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCHSV</td>
<td>Status Active Delay</td>
<td>0</td>
<td>75</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCLSH</td>
<td>Status Inactive Delay</td>
<td>0</td>
<td>70</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TWLWH</td>
<td>WR Width</td>
<td>2TCLCL − 40</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TFLHL</td>
<td>Bus Float to HOLD Inactive</td>
<td>0</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

82730 Display Generator Input Timing Requirements

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRCHRCH</td>
<td>RCLK Cycle Period</td>
<td>100</td>
<td>2500</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRCHRCL</td>
<td>RCLK High Time</td>
<td>40</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRCLRCH</td>
<td>RCLK Low Time</td>
<td>40</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRRCK</td>
<td>RCLK Rise Time</td>
<td>30</td>
<td></td>
<td>ns</td>
<td>0.45V − 2.4V (1)</td>
</tr>
<tr>
<td>TFRCK</td>
<td>RCLK Fall Time</td>
<td>30</td>
<td></td>
<td>ns</td>
<td>2.4V − 0.45V (1)</td>
</tr>
<tr>
<td>TCCHCCH</td>
<td>CCLK Cycle Period</td>
<td>100</td>
<td>None</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCCHCCL</td>
<td>CCLK High Time</td>
<td>30</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCCLCCCH</td>
<td>CCLK Low Time</td>
<td>40</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRCCK</td>
<td>CCLK Rise Time</td>
<td>30</td>
<td></td>
<td>ns</td>
<td>0.45V − 2.4V (1)</td>
</tr>
<tr>
<td>TFCCCK</td>
<td>CCLK Fall Time</td>
<td>30</td>
<td></td>
<td>ns</td>
<td>2.4V − 0.45V (1)</td>
</tr>
<tr>
<td>TVCVR</td>
<td>HSYNC, SYNCIN Set-Up Time</td>
<td>30</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCRVCX</td>
<td>HSYNC, SYNCIN Hold Time</td>
<td>10</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TLPVCVF</td>
<td>LPEN Set-Up Time</td>
<td>30</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TCFLPX</td>
<td>LPEN Hold Time</td>
<td>10</td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TRCHCCH</td>
<td>CCLK/RCLK Skew During CSYNC</td>
<td>−10</td>
<td>10</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

82730 Display Generator Output Timing Response

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCCHDV</td>
<td>Data, Line Count and Attribute and Output Valid Delay from the Delay from the Rising Edge of CCLK</td>
<td>60</td>
<td></td>
<td>ns</td>
<td>C_L = 100 pF</td>
</tr>
<tr>
<td>TCCHDX</td>
<td>Data, Line Count and Attribute and Output Hold Time</td>
<td>5</td>
<td></td>
<td>ns</td>
<td>C_L = 100 pF</td>
</tr>
<tr>
<td>TRCHCV</td>
<td>Delay of Outputs CSYNC, VSYNC, HSYNC or RRV from the Rising Edge of RCLK</td>
<td>70</td>
<td></td>
<td>ns</td>
<td>C_L = 100 pF</td>
</tr>
<tr>
<td>TCCHCL</td>
<td>CCLK Rising to CHOLD Low</td>
<td>75</td>
<td></td>
<td>ns</td>
<td>C_L = 50 pF</td>
</tr>
<tr>
<td>TRCLCH</td>
<td>RCLK Falling to CHOLD High</td>
<td>60</td>
<td></td>
<td>ns</td>
<td>C_L = 50 pF</td>
</tr>
</tbody>
</table>

NOTE:
1. Clock maximum rise and fall times are for functionality only. AC timings are not tested at this condition.
2. Applies only to test mode invocation
WAVEFORMS

BUS TIMING DIAGRAM

T4  T3  T2  T1  T4  T3  T2  T1

BCLK  TCVCTV
AEN  TCHSV  TCHSV
S5, S1  TCHLL  TCHLL
UALE  TCHLL  TCHLL
ALE  TCLA2  TCLA2
AD15-AD0  TAWAL  TAWAL
AD15-AD0  DATA IN  DATA IN
RD  TCVCTV  TCVCTV
DEN  TCLRH  TCLRH
DEN  TCVCTXD  TCVCTXD
AD15-AD0  DATA OUT  DATA OUT
WR  TCVCTV  TCVCTV
DEN  TCLDOX  TCLDOX
DEN  TCLRYX  TCLRYX
ASYNC
READY
SYNC
TSRYHCL  TCVCTXW  TCVCTXD
TCLRYX  TCLRYX
TCLRYX
WAVEFORMS (Continued)

HOLD, RESET, SINT AND CA TIMING

BCLK

HOLD

HLDA

ADDRESS

DATA

CONTROL

RESET

SINT

IRST

CA

LCX

TCLSIN

TCLCTX

TCLBV

TCLBV

TCTVCL

TCTVCL

TFLHL

TFLHL

TRIVRIX

TRIVRIX

TRIHSIL

TRIHSIL

TCAVCA

TCAVCA

TRLLCH

TRLLCH
DISPLAY GENERATOR INTERFACE TIMING

WAVEFORMS (Continued)
WAVEFORMS (Continued)

DISPLAY GENERATOR INTERFACE TIMING

<table>
<thead>
<tr>
<th>Signal</th>
<th>Timing Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCLK</td>
<td></td>
</tr>
<tr>
<td>HSYNC, SYNC IN</td>
<td></td>
</tr>
<tr>
<td>LPEN</td>
<td></td>
</tr>
<tr>
<td>CCLK</td>
<td></td>
</tr>
</tbody>
</table>

A.C. TESTING INPUT, OUTPUT WAVEFORM

INPUT OUTPUT

A.C. TESTING INPUTS ARE DRIVEN AT 24V FOR A LOGIC 1 AND 0.2V FOR A LOGIC 0. TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC 1 AND 0V FOR A LOGIC 0.

A.C. TESTING LOAD CIRCUIT

DEVICE UNDER TEST

C1 INCLUDES JIG CAPACITANCE
82731 VIDEO INTERFACE CONTROLLER

- Parallel to Serial Data Conversion
- On-Chip Clock Generator
- High Video Dot Rates
  80 MHz—82731-2
  50 MHz—82731
- Character up to 16 Dots Wide
- Proportional Character Spacing
- On-Chip Character Attribute Processing
- Control Functions to Provide Screen Reverse Video, Video Clock, Synchronization and Tab Function
- Single 5V Power Supply
- 40 Pin DIP
- All Inputs and Outputs TTL Compatible Except Video Output which is ECL

The 82731 is a general purpose video interface which generates a serial video signal output from parallel character and attribute information coming from the character generator and the 82730 Text Coprocessor. With a character generator and minimal hardware, the 82731 will comprise a complete video interface system for the 82730 Text Coprocessor and the CRT monitor.

Figure 1. 82731 Block Diagram

Figure 2. 82731 Pin Configuration
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin Number</th>
<th>Type</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0-D15</td>
<td>8-1, 39-32</td>
<td>I</td>
<td>Character data parallel inputs.</td>
</tr>
<tr>
<td>PROG</td>
<td>9</td>
<td>I</td>
<td>Program control input; used to program default width values of CCLK and RCLK; these are latched into the 82731 via D0-D7 at the rising edge of CCLK (PROG is active high).</td>
</tr>
<tr>
<td>VIDEO</td>
<td>10</td>
<td>O</td>
<td>Video output; provides the dot information clocked by the internal dot clock.</td>
</tr>
<tr>
<td>RCLK</td>
<td>11</td>
<td>O</td>
<td>Reference clock output; used to generate timings for the screen columns for data formatting and video signals. The period of RCLK is programmable from 6 to 21 times the period of the internal dot clock.</td>
</tr>
<tr>
<td>CCLK</td>
<td>12</td>
<td>O</td>
<td>Character clock output; used to clock character and attribute information out of the CRT controller. The period of CCLK is programmable from 3 to 18 times the period of the internal dot clock.</td>
</tr>
<tr>
<td>HDOT</td>
<td>13</td>
<td>I</td>
<td>Half dot shift input; the video signal at the video output will be delayed by half dot clock for character rounding (active high).</td>
</tr>
<tr>
<td>CBLANK</td>
<td>14</td>
<td>I</td>
<td>Character blank attribute input, the video output is blanked (active high).</td>
</tr>
<tr>
<td>WDEF</td>
<td>15</td>
<td>I</td>
<td>Width defeat attribute input; the CCLK period is set to a preprogrammed default value (active high).</td>
</tr>
<tr>
<td>CRVV</td>
<td>16</td>
<td>I</td>
<td>Character reverse video attribute input, inverts the character data from D0-D15 (active high).</td>
</tr>
<tr>
<td>DW</td>
<td>17</td>
<td>I</td>
<td>Double width attribute input, the internal dot clock frequency and the CCLK frequency are divided by two (active high). The RCLK frequency remains unchanged.</td>
</tr>
<tr>
<td>W0-W3</td>
<td>18, 19, 21, 22</td>
<td>I</td>
<td>Clock width inputs, they are used for programming the CCLK clock width on a character by character basis.</td>
</tr>
<tr>
<td>CHOLD</td>
<td>23</td>
<td>I</td>
<td>CCLK inhibit input, this signal inhibits CCLK generation and is used for TAB function (active low).</td>
</tr>
<tr>
<td>CSYN</td>
<td>24</td>
<td>I</td>
<td>CCLK synchronization input, CCLK will be synchronized to RCLK and the video output signal is defined by RRVV (active high).</td>
</tr>
<tr>
<td>RRVV</td>
<td>25</td>
<td>I</td>
<td>Field reverse video input; the video signal at the video output will be inverted (active high).</td>
</tr>
<tr>
<td>DCLK</td>
<td>26</td>
<td>O</td>
<td>Dot clock output; ECL-level signal; must be connected to a 3.3k resistor to ground if used.</td>
</tr>
<tr>
<td>X1-X2</td>
<td>27, 28</td>
<td>I</td>
<td>Inputs for fundamental mode crystal; its frequency must be 1/8 of the required dot clock frequency.</td>
</tr>
<tr>
<td>VT</td>
<td>29</td>
<td>O</td>
<td>Tuning voltage for PLL-VCO; this output is used to tune the LC-circuit and thus control the oscillator frequency of the internal dot clock.</td>
</tr>
<tr>
<td>T1-T2</td>
<td>30, 31</td>
<td>I</td>
<td>LC-circuit inputs for PLL-VCO. T1 can be used to provide the 82731 with an external TTL-level clock at twice the dot clock frequency.</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td>-</td>
<td>+5V power supply</td>
</tr>
<tr>
<td>6ND</td>
<td>20</td>
<td>-</td>
<td>Ground (OV)</td>
</tr>
</tbody>
</table>
FUNCTIONAL DESCRIPTION

The Video Interface Controller, 82731, in a typical CRT system shown in Figure 3, interfaces the Text Coprocessor to the CRT video terminal. It receives the parallel data along with the attribute and control information from the Text Coprocessor, processes it into a serial video signal which can be fed to a video CRT terminal. It also generates the basic dot clock (DCLK), character clock (CCLK) and the reference clock (RCLK) signals required by the Text Coprocessor. CRT terminals requiring very high resolution, extremely stable and absolutely flicker-free picture place special demands on the dot rate generator. In such applications dot rates up to 80 MHz are necessary. This allows 12.5 ns per dot (pixel) for converting data, attribute and control information into serial form for the video terminal. The functionality of the 82731 is largely determined by the complexity and the demands of the CRT controller it supports. Figure 1 shows the block diagram of the Video Interface Controller. The dot clock is generated by voltage controlled LC circuit connected at T1 and T2. Another clock is generated which is crystal controlled and has frequency 1/8 of the dot clock. This is used to stabilize the dot clock using an on-chip phase locked loop (PLL). This two-oscillator concept enables the use of low cost, fundamental mode crystals even for generating frequencies up to 80 MHz.

The 16 bit shift register receives parallel inputs from pins D0-D15. This allows a maximum character width of 16 dots. The minimum width is 3 dots. The character width is programmable through pins W0-W3 for proportional character spacing. This also determines the character clock (CCLK) frequency. Programming of the default character width and the reference clock (RCLK) is done through inputs D0-D7 and PROG. Signal WDEF can be used to switch between the default character width and the one specified dynamically through the lines W0-W3. When using variable character width, for example, in generating tables on the screen, it is essential that every entry in a column starts at the same dot distance (and not the character distance) from the start of line. The 82731 supports this requirement by providing a tab function using CSYN and CHOLD signals to synchronize with the reference clock (RCLK).

It is possible to shift any scan line of any character by half a dot using the HDOT signal. This feature, known as character rounding, further enhances the quality of high resolution character displays. Other features, like character blinking, reverse video etc., which improve the readability of text on screen are directly supported by the 82731 using signals CRVV and RRVV from the Text Coprocessor, processing them and affecting the final video signal to show the characters with the desired attributes.

Figure 3. CRT System Block Diagram
Clock Generation

The most fundamental clock required to run the CRT display is the dot clock which provides the reference for the dot data to be shifted serially to the CRT. In addition, it is the basis for the character clock (CCLK) and the reference clock (RCLK) required by the 82730.

Dot Clock

The dot clock is derived from an on-chip oscillator which runs at twice the normal dot clock (DCLK) frequency. A voltage-controlled LC circuit is connected to the T1, T2 pins, to create a voltage-controlled oscillator (VCO). The 82731 compares the phase of this oscillator with another on-chip oscillator controlled by a crystal attached to the X1, X2 pins. This oscillator runs at 1/8 the normal DCLK frequency to allow using inexpensive low-frequency crystals. The on-chip PLL circuit produces an error voltage via the VT pin which locks the VCO to the 16th harmonic of the crystal frequency (see Figure 4a).

Alternatively, the 82731 can be supplied with an external TTL-level clock at twice the normal DCLK frequency via the T1 pin, as shown in Figure 4b.

When the Double Width (DW) input is active, the DCLK frequency is divided to 1/2 its normal value. This affects the DCLK, CCLK, and VIDEO outputs, but not RCLK.

Designing the Oscillator Circuit

The whole external oscillator circuit consists of three parts:
- the crystal circuit,
- the voltage controlled LC-circuit, and
- the loop filter for the PLL.

Figure 5a shows the general crystal circuit. The crystal must be a fundamental mode series resonant type with a resonant frequency of 1/8 of the desired dot clock frequency. The capacitor Cx is necessary if a fine adjustment of the dot clock rate must be
Figure 5. Designing the Oscillator Circuit
made. Figure 5b shows an example how the dot clock frequency can vary with different values of Cx. The capacitors C1 and C2 may be necessary to suppress overtone oscillations if the crystal frequency is below 6 MHz. The exact values depend on the crystal used and must be determined empirically. The recommended ranges are 0 to 10 pF for C1 and 0 to 100 pF for C2.

The voltage controlled LC-circuit is shown in Figure 5c. The effective resonant circuit consists of the inductance L, the capacitance Cd of the varactor diode and the parasitic capacitance Cp. Its resonant frequency is

\[ f_R = \frac{1}{2\pi \sqrt{L \cdot (Cd + Cp)}} \]

where \( f_R \) must be \( 2 \times f_{DCLK} \). The value of \( Cp \) depends on many factors (e.g., layout, single/multi-layer board, etc.), thus it changes from application to application. However a value of 5 to 15 pF seems to be a good approximation.

The value of DC (varactor diode) should be determined at a control voltage of 2.5 V to get the lock-in-range as wide as possible. The variation of VT ranges from 1 V to VCC-1 which results in a minimum frequency shift of about 6-8% in relation to the center frequency at 2.5 V.

The value of the inductance L must be determined in such a way that the resulting center frequency lies as near as possible to the needed frequency \( f_R = 2 \times f_{DCLK} \) to guarantee a stable dot clock under all operation conditions. Figure 5f shows a diagram that will help to find the needed inductance L. It is based on the use of a varactor diode (Siemens BB 505G) that has a capacitance of 12 pF at a control voltage of 2.5 V. The use of other diodes will of course lead to other diagrams.

At dot clock frequencies higher than 50 MHz the needed inductance becomes lower than 100 \( \mu \)H. In these cases it is better to integrate the inductance into the board layout. Figure 5e shows a possible layout for the external oscillator circuit and approximate (measured) values of the inductance of the printed coil (trace width and trace spacing 20 mils).

The loop filter converts the current pulses at the VT pin into the control voltage VT for the VCO. It is an essential part of the PLL and affects the lock-in-range and stability of the PLL. A second order filter that was found to work well under all operation conditions and over the full frequency range is shown in Figure 5d.

Reference Clock (RCLK)

RCLK is the reference clock output used to generate video timing and to define screen columns for data formatting and tabular locations. In addition, it is used to clock the field attribute signals into the 82731. The period of RCLK is programmable from 6 to 21 times the period of the dot clock, i.e., the RCLK high-time is 3 dot clock periods and the RCLK low-time is programmable from 3 to 18 dot clock periods. It is programmed via D4-D7 at the rising edge of CCLK, when PROG is active (see Table 1 and Figure 6).

The RCLK clock width should be programmed only once after a system reset.

<table>
<thead>
<tr>
<th>Table 1. Programming RCLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Character Clock (CCLK)

CCLK is the fundamental character clock output used to clock character and attribute information from the 87730.

It is a rising edge triggered clock and inside the active character field its period is programmable from 3 to 18 times the period of the dot clock, i.e. the
CCLK high time is 2 dot clock periods and the CCLK low time is programmable from 1 to 16 dot clock periods.

When CSYN is active (normally outside the active character field) CCLK is forced to match RCLK. In this case the CCLK high time is 3 dot clock periods instead of 2.

In order to support proportional spacing, the period of CCLK can be reprogrammed at the beginning of each CCLK cycle (i.e. at the beginning of each character) if PROG is inactive.

Programming the character width is done via the clock width inputs W0-W3 according to Table 2. The W0-W3 input data is clocked into the 82731 at the rising edge of CCLK and defines the width of the currently displayed character (see Figure 7).

If the width defeat attribute (WDEF) is active, the period of CCLK will be set to the programmed default value ignoring the clock width inputs W0-W3. This value is programmable from 3 to 18 times the period of the dot clock via the 00-03 inputs, when the PROG input is active (see Figure 6).

The default CCLK width should be programmed only once after a system reset.

The CCLK clock period will be doubled if the double width attribute (DW) is asserted at the rising edge of CCLK.

**NOTE**

If width of CCLK is programmed to 17 or 18, zeros are shifted out from the internal shift register after the 16 data bits and displayed according to the attribute signals.

**Clock Initialization Sequence (PROG)**

After power on the width of RCLK is a random value between 6 and 21 and the width of CCLK is a random value between 3 and 18.

The 82731 should be initialized in the following way:

- Activate the CSYN signal. CCLK is forced to match RCLK, which has a minimum clock width of 6 dot clock periods.
- Apply the clock width informations to D0-D3 and D4-D7 according to tables.
- Activate the PROG signal. The default width of CCLK and the width of RCLK are programmed at the next rising edge of CCLK (see Figure 6).
- Remove the PROG signal.

CSYN can be removed at the beginning of the next active data field.

### Table 2. Programming CCLK

<table>
<thead>
<tr>
<th>PROG</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>CCLK Period (dot clocks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>15</td>
</tr>
</tbody>
</table>

**Note.**

PROG = 1: Programming the CCLK default clock width during the initialization phase via D0-D3 at the rising edge of CCLK.

PROG = 0: Programming the clock width of the current CCLK cycle via W0-W3 at the rising edge of CCLK.

### Character Data Signals

The character data signals are normally provided by the character ROM and clocked into the 82731 at the rising edge of CCLK.

The character data signals consist of:

- the character data lines (D0-D15),
- the character width information (W0-W3), and
- the half dot shift signal (HDOT).

### Dot Data (D0-D15)

The dot data signals will be clocked into the 82731 via the D0-D15 inputs at the rising edge of CCLK. The actual character width is defined by the W0-W3 inputs or the default width information previously programmed. The dot data will be displayed dependent on the control signals and on the corresponding attribute information. The data bits are serially shifted out at the video output starting with D0.
Figure 6. Clock Initialization

Figure 7. Action of Clock Width Inputs W0-W3 on CCLK
If CCLK width is greater than 16, zeros are shifted out for the rest of the dot clocks and displayed according to the attribute signals.

**Character Width (W0-W3)**

The W0-W3 inputs are clocked into the 82731 at the rising edge of CCLK and determine the width of the currently displayed character.

**Half Dot Shift (HDOT)**

The half dot shift signal is clocked into the 82731 at the rising edge of CCLK. When the half dot shift signal is active (high), the output of the video data will be delayed by half a dot time. The first dot of the character dot line is transmitted for one and a half dot clock period while the last dot of this character dot line is displayed for half a dot clock period. The remaining character dots are transmitted for one dot clock period and thus are shifted by half a dot.

The HDOT signal is not a character attribute signal, because it can change from scan line to scan line of a character. Thus it is reasonable to generate it from the character ROM, together with the dot data and the width information.

**Character Attribute Signals**

These signals are clocked into the 82731 at the rising edge of CCLK. Thus they are valid for the next character only.

The character attribute signals consist of:
- character blanking (CBLANK)
- character reverse video (CRVV)
- double width (DW)
- width defeat (WDEF)

Outside the active character field (which is defined by the CSYN signal) all character attribute signals are ignored.

**Character Blanking (CBLANK)**

If CBLANK is active (high), the blank attribute will produce the effect of blanking the display of the character. When the CBLANK attribute is active, the corresponding dot data information D0-D15 will be as if all zeros were forced at the inputs. The video output can be inverted to all ones by simultaneously activating the CRVV attribute. Independent of these character oriented operations the video output signal is also affected by the RRVV field attribute signal.

---

![Diagram](image_url)

**Figure 8. Function of HDOT on VIDEO**
Character Reverse Video (CRVV)

CRVV is an active high signal. In the character field, the CRVV attribute will produce the effect of reversing the polarity of the display during the transmission of the current character. CRVV is also effective together with the CBLANK attribute (see CBLANK description) and the RRVV signal. Outside the character field, the CRVV attribute is ignored.

Although the CBLANK signal is normally a character attribute, it may change from dot line to dot line of a character. Thus one or more underlines or cursors can be generated by the CRT controller activating CBLANK and CRVV.

Double Width (DW)

The dot clock frequency and the CCLK frequency will be halved when the double width attribute is active (high), producing characters that are twice as wide. The period of RCLK is not changed (see Figure 9).

Width Defeat (WDEF)

The WDEF attribute signal is clocked into the SAB 82731 at the rising edge of CCLK. When the width defeat attribute is active (high), the width of CCLK will be set to a default width value previously programmed (see figure 10).

Field Attribute Signals

The field attribute signals are clocked into the 82731 with the rising edge of RCLK. Thus the attributes are valid for a specific part of the screen independent of how many characters are displayed within this part.

The 82731 supports two field attributes:
- field reverse video RRVV, and
- clock synchronization CSYN.

Row Reverse Video (RRVV)

RRVV control signal is clocked into the 82731 at the rising edge of RCLK. It immediately affects the display by the polarity of the video output in both the character field and the border of the display. It is an active high signal.

Clock Synchronization (CSYN)

CSYN is a field attribute signal, because it defines the active character field in addition to its function of synchronizing CCLK and RCLK.

CSYN must be inactive (low) during the display of characters. At the first rising edge of RCLK after CSYN is deactivated (low), character data is latched into the 82731, beginning the display of the active character field (see Figure 11). At the next rising edge of RCLK after CSYN is activated (i.e. at the end of the character field), the video output is forced to zero or, if the RRVV control signal is active, to a high level. The currently transmitted character will be truncated at this location. At the same time, CCLK will be forced to match RCLK starting with the next rising edge of RCLK (see Figure 11). While CSYN is active all character attribute and data signals are ignored and only the field reverse video signal (RRVV) affects the video output.

Before the deactivation of CSYN, the data and attribute pipeline has to be filled by the CRT controller with the information of the first character.

---

Figure 9. Function of DW on DCLK and CCLK
Tabulator Function

The 82731 supports tabulator functions by providing the CHOLD (character clock inhibit) input.

CCLK Inhibit (CHOLD)

When the CHOLD signal is activated (low) it inhibits CCLK and thus freezes the information pipeline between CRT-controller and 82731 until the next tabulator location is reached. CHOLD has to be activated simultaneously with the display of the TAB-character. If the TAB-character doesn't consist of all zeros, it must be blanked by activating CBLANK.

The width of the TAB-character can be determined by W0-W3 or by activating WDEF.

The CHOLD signal is provided by the 82730 and it is assumed to be triggered with the rising edge of CCLK (Figure 12). With the same edge of CCLK, the TAB-character will be latched into the 82731. Thus the TAB-character will be displayed completely and the CCLK will be inhibited until reaching the specified tabulator location, which is defined by CHOLD inactive (high) at the rising edge of RCLK.

In the timing diagrams it is assumed that CHOLD is deactivated by the falling edge of RCLK. Figure 12 shows the normal case where the display of the TAB-character is finished before deactivation of CHOLD. The gap between the TAB- and the following character is normally blanked. In this scheme the TAB-character will be handled by the 82731 like each other character (attributes operate normally).

In case of CHOLD active width less than the TAB-character width the TAB-character will be also displayed completely. However, we have to distinguish three different cases:

1) TAB-character is terminated before reaching TAB-location. The next character will be displayed as described before. In the gap the video output is normally blanked.
2) TAB-character is finished exactly at the TAB-location. The next character will be displayed immediately without delay.
3) TAB-character is not terminated when reaching the TAB-location (see Figure 13). The following character will be displayed subsequently after the display of the TAB-character (i.e. the start of the following character is not at the TAB-location).

If the CHOLD signal is not deactivated the video output will be continuously blanked. In the gap between the end of the TAB-character and the TAB-location all character attribute signals will have no effect on the video output signal. If the RRV control signal is active the video output signal is inverted.
Figure 12. Function of CHOLD (Normal Case)

(1) TAB character is displayed completely - video output is blanked
(2) Video output is blanked
(3) Next character
(4) Default width: 7, TAB character width defined by WDEF

Figure 13. Function of CHOLD with CHOLD Width Less than Character Width (Case 3)

NOTE:
(1) TAB character is displayed completely - video output is blanked
(2) Next character is displayed (not on TAB location)
(4) Default width = 11.
Video Output

The video output provides an ECL-oriented signal (see Figure 14) and is matched to drive a 50 Ohm coax cable (see Figure 15). In case of external attribute processing the external logic can be ECL- or STTL-compatible.

![Figure 14. Video Output Stage](image)

![Figure 15. A Video Output Load](image)
Figure 15.B Proposed Converter for Video Output to TTL Level Output

Figure 16. TTL-Level-Output Test Load
ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias ........... 0°C to 70°C
Storage Temperature ........... -65°C to +125°C
All Output and Supply Voltages ........... -0.5V to +6V
All Input Voltages ........... -0.6V to +5.5V
Power Dissipation ........... 1.75 Watt

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (Vcc = 5V ± 10%, TA = 0°C to 70°C)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC</td>
<td>Input Clamp Voltage</td>
<td>-1</td>
<td>V</td>
<td>IC = -5 mA</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>Forward Input Current</td>
<td>-0.7</td>
<td>mA</td>
<td>IF = 0.5V</td>
<td></td>
</tr>
<tr>
<td>IR</td>
<td>Reverse Input Current</td>
<td>50</td>
<td>µA</td>
<td>IR = Vcc</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage CLK, RCLK, VIDEO</td>
<td>Vcc - 1.2V</td>
<td>0.5 V</td>
<td>IOL = 8 mA</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage CLK, RCLK, VIDEO</td>
<td>Vcc - 0.2V</td>
<td>2.4 V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>0.8</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Icc</td>
<td>Power Supply Current</td>
<td>2.0</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zo</td>
<td>Output Impedance VIDEO</td>
<td>40</td>
<td>70 Ω</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cin</td>
<td>Input Capacitance</td>
<td>15</td>
<td>pF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A.C. CHARACTERISTICS

$T_A = 0$ to $70^\circ C$; $V_{CC} = 5V \pm 10\%$. All timings measured at 1.5V unless otherwise noted.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limit Values</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>82731</td>
<td>82731-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
</tr>
<tr>
<td>$T_{DHDL}$</td>
<td>DCLK cycle period</td>
<td>20</td>
<td>125</td>
</tr>
<tr>
<td>$T_{CHCH}$</td>
<td>CCLK cycle period</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>$T_{CLCH}$</td>
<td>CCLK low time</td>
<td>$T_{DHDL}$</td>
<td>$T_{DHDL}$</td>
</tr>
<tr>
<td>$T_{CLCH}$</td>
<td>CCLK high time</td>
<td>$2 T_{DHDL} - 5$</td>
<td>-</td>
</tr>
<tr>
<td>$T_{RHRH}$</td>
<td>RCLK cycle period</td>
<td>6</td>
<td>21</td>
</tr>
<tr>
<td>$T_{RLRH}$</td>
<td>RCLK low time</td>
<td>$3 T_{DHDL} - 10$</td>
<td>$18 T_{DHDL}$</td>
</tr>
<tr>
<td>$T_{RHRL}$</td>
<td>RCLK high time</td>
<td>$3 T_{DHDL} - 5$</td>
<td>$3 T_{DHDL} - 5$</td>
</tr>
<tr>
<td>$T_{DRCH}$</td>
<td>Data and attribute input set up time</td>
<td>25</td>
<td>20</td>
</tr>
<tr>
<td>$T_{CHDX}$</td>
<td>Data and attribute input hold time</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$T_{HLTE}$</td>
<td>CHOLD active before end of TAB-character</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$T_{HLHH}$</td>
<td>CHOLD pulse width</td>
<td>25</td>
<td>20</td>
</tr>
<tr>
<td>$T_{HHRH}$</td>
<td>CHOLD inactive set up before rising edge of RCLK</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$T_{HLRH}$</td>
<td>CHOLD inactive hold time after rising edge of RCLK</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_{CHVV}$</td>
<td>Video output valid after rising edge of CCLK</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>$T_{OLLOH}$</td>
<td>TTL-output rise time</td>
<td>-</td>
<td>10</td>
</tr>
<tr>
<td>$T_{OHOL}$</td>
<td>TTL-output fall time</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$T_{VLVH}$</td>
<td>Video output rise time</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
Figure 17. TTL-Level-Output Test Load

RCLK: $R_L = 700\Omega$
CCLK: $R_L = 350\Omega$

Figure 18. ECL-Level-Output Test

Figure 19. TTL-Level-Output Load Circuit

Figure 20. ECL-Level-Output Load Circuit

Figure 21. Basic Timing
Figure 22. Timing on CHOLD

Figure 23. Example Interface to 8275
COMPONENTS SPECIAL
• Capacitors • Precision resistors • Shielding materials
• Selecting electrolytics • Power MOSFETs for switchers

Coprocessor chip dedicates itself to text
The first chip dedicated to text manipulation, the 82730 operates as a coprocessor to a host CPU and executes many high-level commands that reduce the software needed for processing text.

Text coprocessor brings quality to CRT displays

The quality of text in raster-scanning CRT displays has always been a tradeoff against the complexity, performance, and cost of the associated video system. By allocating many of the complex display functions to firmware, a dedicated text coprocessor chip, the first of its kind, replaces printed-circuit boards that contain more than 100 ICs while increasing system performance by relieving many of the host processor's text manipulation tasks. The chip thus makes possible the high quality and high performance sought, without the need to compromise because of high design complexity and high cost of text-processing hardware.

Though its speed makes the 82730 text coprocessor beneficial on its own, its utility can be enhanced considerably when working with the 82731 video interface controller. Together they provide proportional spacing, simultaneous subscript and superscript displays, dual cursors, dynamically reloadable character fonts, and user-programmable field and character attributes. By adding still another chip, the 82720 graphics display controller, the device can display high-resolution graphics and text at the same time.

Housed in a 68-pin package, the 82730 text coprocessor combines a direct memory access channel and a processor bus interface that permit it to fetch its own instructions and data from the host system's memory, independent of and in parallel with the host CPU.

The two processors communicate through messages—commands, parameters, and status words—which are placed in a communication block inside a shared memory. The host issues commands by preparing messages, storing them in the communication block, and directing the coprocessor's attention to them by activating a Channel Attention signal, which is implemented in hardware. In return, the coprocessor sets a flag in the shared memory that notifies the host when it has executed the command.

The 29 high-level commands built into the 82730 break down into two groups: channel commands, which work at the system level to start and stop the display and to communicate status and similar information, and data-stream commands, which are incorporated directly into the display-data strings to govern the DMA process and control row...
Text coprocessor

characteristics, character attributes, and so on.

The 82730 resides on a local system bus with the host microprocessor, such as the 80186 CPU, and therefore provides the same address, data, and control signals as the main processor. By handling several of the tasks typically done by the host processor—like DMA control and display formatting—it leaves the host free for other tasks.

For example, when the coprocessor is configured to share the CPU bus, a portion of the host microprocessor bus bandwidth must be devoted to the DMA process that refreshes the CRT. However, the 82730's high-speed intelligent DMA controller (operating at a maximum data rate of 4 Mbytes/s) helps minimize the time spent executing the refresh operation, while simultaneously handling the formatting of the display data. A different approach involves a dual-ported memory architecture, which places the memory between the CPU and the coprocessor. That completely frees the processor bus of the refresh activity, allowing the host more time to execute other tasks. It has become a more cost-effective method, as some dynamic memory controllers now contain dual-ported arbitration logic on chip.

Inside the chip

The basic architecture of the coprocessor is divided into two main parts: a memory interface and a display generator section (Fig. 1). The memory interface lets the coprocessor and the system processor communicate via the shared memory. The display generator, in turn, responds to the data provided by the memory interface and carries out the display operations.

The memory interface actually comprises two smaller subsections, a bus interface unit and a microcontroller unit. The bus interface provides an intelligent connection from the 82730 to the host processor bus and also buffers the data transfer requests from the microcontroller. Upon initialization, the bus interface can be programmed for 8- or 16-bit data and 16- or 32-bit addresses. Furthermore, the host interface can be configured for 8- or 16-bit-wide data buses, making the coprocessor compatible with 8- or 16-bit host processors, like the 8088/80188 and the 8086/80186. Running at 8 MHz maximum in 16-bit systems, the 82730 handles the maximum DMA rate of 4 Mbytes/s.

The microcontroller unit stores the microinstructions for the 82730's high-level operations. The microcontroller's internal processor manages the memory transfers, interprets the commands embedded in the data stream, and executes those commands by sending data to the appropriate control registers or display data buffers. To optimize the transfer of data between the system and the CRT interface, the coprocessor uses three clocks—one for the host interface, the other two for video data. The memory interface section runs from the bus clock, the CRT interface operates from a reference and a character clock.

1. Divided into two main sections—a memory interface unit and a display generator—the 82730 text coprocessor can operate at optimum speed since each section can function independently at a different clock speed.
Although the coprocessor packs a considerable amount of processing power on a single NMOS chip, it cannot handle the high video dot rate needed to deliver high character counts to the CRT display. For that, it needs the 82731 video interface controller, which gains its high speed and drive capability from bipolar technology. In addition, the combination of the 82730 and 82731 succeeds in reducing the video interface to just a few latches and a software character generator residing in RAM or ROM (Fig. 2).

Inside the 82731 are the reference- and character-clock generators, a video shift register, and all attribute logic (Fig. 3). Housed in a 40-pin package, the circuit offers TTL-compatible inputs and outputs except for the video output, which is ECL-compatible and provides a dot-shift clock rate of 50 MHz maximum on characters up to 16 dots wide. The circuit proportionally spaces characters by accepting the width sent from the character generator and sending an appropriate character-clock output whose period determines the variable width of the character to be displayed.

The video interface controller can employ an inexpensive, low-frequency crystal and internally multiply that frequency to generate the high-frequency dot clock. It also supports control functions such as screen reverse video, synchronized character field, and tabbing operations. The dot clock drives the internal video shift register, the character clock controls the unloading of data from the row buffers in the 82730, and the reference clock establishes the raster and screen formats. The reference clock also supplies the basic timing for the horizontal sync, blanking, border, and active display time. The corresponding vertical attributes—except border—are driven by the horizontal line time. All seven of these screen parameters are programmable by the system designer with the 82730.

System interfaces are simple

As a coprocessor, the 82730 has the same bus-control signals as an 80186 host processor and thus can share the system-bus controllers, drivers, and latches. The host processor and the 82730 arbitrate for control of the local bus through the Hold and Hold Acknowledge lines (HLD/HLDA). The Channel Attention (CA) and System Interrupt (SINT) control lines complete the wired interface. With this configuration, the 82730 has access to all the memory that the 80186 CPU has available.

Anytime the CPU wants to send a message to the 82730, it writes the command and busy flag into the 82730 command block and then pulses the coprocessor's CA input to inform it that a message is waiting. The 82730 then raises the HOLD output and waits for access to the bus. When the CPU relinquishes the bus, it raises the HLDA input of the 82730.

Once the 82730 becomes active, it transmits the command block address that was stored in its...
Text coprocessor

registers during initialization. That address, in conjunction with the appropriate memory control signals—including read or write strobes, lower or upper address latch enables, upper address output, or data enable output—will either read the command block or write to it. All these signals are coordinated by the bus clock.

Whenever the 82730 must send status information to the host CPU, it gains control of the bus and places the data into the status location in the command block. The bus is then released and the coprocessor notifies the CPU through the SINT signal. When the coprocessor is using a dual-ported memory to communicate with the 82730, the HOLD and HLDA signals are not employed. Instead, the 82730 accesses the dual-ported memory directly rather than acquiring the bus from the CPU.

When the display process is activated, the coprocessor uses its built-in DMA capability to fetch display data from the memory. The data consists of character data mixed with data-stream commands; embedded data-stream commands provide the flexibility to manipulate data on the fly.

Soft fonts loaded

The 82730 also permits soft fonts to be automatically loaded into RAM-based character generators. Addresses and data stored in the system memory are then loaded into the row buffers of the coprocessor. During blanked rows (generally during the vertical retrace), address information is loaded into a latch and data is written to the character generator.

The 82730’s dual row buffers help reduce the bandwidth and access time requirements for the system memory. The data stored in one buffer is being used to display a row on the screen while the second buffer is being loaded, by the microcontroller, with the next display row from the system memory. Up to 200 characters can be stored and displayed by each row buffer. Furthermore, since the display generator section operates asynchronously with the microcontroller unit, each can operate at optimal speed. Processing is synchronized by internal flags and shared internal storage, and data that will be displayed is exchanged through the row buffers.

The coprocessor’s display generator handles the data that defines the timing and the operation of the CRT interface. That data, which is stored in the display characteristics registers of the chip, controls every aspect of the display—from the screen’s format to the blink rates of the characters and cursors. All the parameters that define the initial display characteristics can be set by one command—MODEST—thus reducing the time and effort required to establish a screen format.

Beneath the simplicity of the hardware shown in Fig. 2 are the high-level instructions—channel commands—and the data-stream commands. When combined with a table-driven linked-list data structure, they ease software development.

Central to the software is the command block, through which all channel commands are transferred between the coprocessor and the host. This block is located within the shared memory, and its exact position is set during the 82730’s initialization routine (Fig. 3a). Once established, it contains all the information needed to start the display-data fetch; to communicate status, interrupt, and cursor position information; and to give the location of the mode block, which contains all the parameters for setting up the display. The START DISPLAY channel command begins the sequence (Fig. 3b).

Since the display data is set up within linked lists, the coprocessor can rapidly change any of the lists without shifting huge amounts of data. The display fetch starts with the value of the list-switch bit which selects one of two list-base pointers in the command block: The pointer points to its string pointer list; the pointers in that list direct the on-chip DMA to the data strings containing the desired display data and data-stream commands. The programmer can modify one pointer list while
displaying from the other, and can also switch screens merely by changing the list-switch bit, thus eliminating time-critical data manipulations.

Two data-stream commands—End of String (EOS) and End of Row (EOR)—are key to the linked list and DMA activities. Strings are a logical concept: they contain blocks of contiguous data stored anywhere in memory. In contrast, rows are a physical concept and represent a block of characters that make up a physical row on the screen. Many strings can exist in a display row, or many rows in a string. (Only the extra DMA overhead of fetching the new string pointer sets a practical limit on the number of strings in each row.)

The actions of the two commands are independent. End of String tells the 82730 to get the next string pointer from the list, and from there, the next data string. End of Row suspends the DMA until the row buffers are swapped at the end of the current row. The DMA then takes over, into the new row buffer.

**String manipulation fosters high speed**

Strings are commonly the next level of text organization above single characters. With the 82730, a string can be as small as a character or it can be a word, row, sentence, paragraph, or a page of characters. These high-level entities can be moved merely by manipulating a small string pointer table (Fig. 5). The heart of the algorithm for word wraparound, a common feature in text processors, can easily be accommodated by a single command such as the String Compare command of the 80186. Word wraparound is then achieved by scanning the data (not moving it) and manipulating a few pointers. Earlier system designs would have required a multiple-instruction software loop that scanned and moved every individual character.

An extension of the linked list allows programmers to set up several independent data windows on the CRT screen in a virtual screen mode. That feature is especially helpful if a user wants both a menu window and one or more work-space windows. Such a scheme saves a lot of time for the end user—eliminating the back-and-forth movement between menus and working text. To set this up, several data structures, each with its own command block, can be accessed in a table-driven sequence to put data in a given window on the screen (Fig. 6).

The string list and data strings are the same for regular or virtual modes; only the structure of their command blocks differs. Thus, each virtual window can be an independent software entity in the system, and the 82730 can present these independent data bases simultaneously.

4. Both the host CPU and the coprocessor go through an initialization sequence when the computer system is reset (a). The coprocessor then looks for a START DISPLAY command so that it can load the various data strings from the system memory into the display generator section, attach attributes, and display the data on the CRT (b).
Text coprocessor

Multiple 82730s can also be used in a single system. Up to four devices can be clustered in a single system, with one serving as a system master and the others as slaves. The data for this cluster can be interleaved, permitting the cluster to work from one data base to get more characters per screen or more bits per character. Also, in the slave mode, the 82730's video outputs can be synchronized to an external video signal, giving the system such capabilities as mixed text and graphics, broadcast subtitling (text overlay), and overlays for video recording.

Attributes enhance display quality

The designers of the 82730 have given it the ability to highlight various areas of an on-screen document through the use of character and field attributes. In the 16-bit data word, for example, only the most significant bit is committed; it serves as the command or data designator. If set to 1, the word is a data-stream command, with the remaining 15 bits becoming one of the predefined instructions. However, if the MSB is 0, the other bits are at the discretion of the designer, who may choose which and how many are needed for character codes, attributes, or user-defined functions.

The 82730's six predefined attributes—reverse video, invisible, blinking character, two underlines, and a special graphics character—can be programmed to respond to any of the 15 bits, or they can be completely disabled. In addition, they can be set character by character or through a field-attribute mask. All can be attached to any character. The blinking character can be programmed for a wide range of duty cycles and blink rates. The two underlines can be independently positioned anywhere in the row height, and the position can be changed from row to row. Thus the underline can be doubled or serve as a strike-through line, a fraction line, or an overbar. One of the underlines can also be programmed to blink at the same rate as a blinking character.

The graphics character is relatively important, since it permits character information to be displayed to the full height of the row. It causes the chip's line-counter output to count from zero at the top of the display row continuously through to the bottom of the row. When used with special characters, this attribute allows business forms and graphs to be easily constructed.

5. If a character or word must be inserted near the beginning of a screen of text, only the list pointers must be changed to add the item. In older systems, all the characters following the insertion or deletion were shifted in the memory to revise the display.
Text coprocessor

Another capability of the 82730 is subscript and superscript characters, done by manipulating the line-counter outputs. The SUB SUP N data-stream command declares which and how many pairs of characters will be displayed simultaneously as subscripts and superscripts.

Proportionally spaced displays could cause some subscript and superscript characters to have different widths and thus disrupt the vertical alignment of a character pair. A special output of the 82730 called Width Defeat prevents that misalignment by causing the 82731 video interface controller to enforce a predefined width—programmed upon system initialization—during the display of subscript and superscript characters.

The proportional spacing is performed by the reference and the character clock. Used to shift out the character and attribute data, the character clock operates during the display field. Its frequency can vary character by character, up to a rate of 10 MHz, to set the width of the character currently being displayed. The video interface controller takes the character width information that has been supplied by the character generator and produces a variable width character clock that supports the proportional spacing. This approach also greatly reduces system complexity and cost compared with previous designs.

**Screen and row formats are flexible**

The reference clock signal in a system that contains the 82730 and 82731 chips is a constant-frequency clock that forms the time base to generate the horizontal scan lines and vertical frame periods. One scan line can last for 256 reference clock periods, and one frame can contain up to 2048 scan lines. Within these periods, the respective synchronization pulses and the border and character fields can be set anywhere within that range. All these timing relationships, including the scan and frame periods, can be changed on a frame-by-frame basis to suit changing applications.

The screen format is flexible all the way down to the row level. For instance, the height of a row (up to 32 scan lines) and the vertical position of the characters within that row can be changed from row to row by a single data-stream command called FULROWDESCRIP. In addition, the command lets the programmer set the starting and ending scan lines within the row for the normal, subscript, and superscript character fields and the two cursors.

The same data-stream command that defines the row characteristics can also be used to blank the row, display it as reverse video, double its height (for up to 64 scan lines per row), or eliminate the proportional spacing.

Graphics, too, can be handled by the 82730, although flexibility and resolution are not as high as with the 82720 graphics display processor. Business applications typically need graphics that are no more complex than two- or three-dimensional charts or business forms. These formats can be stored as special characters in a standard font set for the character generator. Even more complex graphics can be handled through the use of mosaic graphic cells, which can be stored in RAM to permit alterations. Of course, as in most systems using floppy-disk systems for main storage, the desired fonts or graphics forms can be saved on the disks and downloaded as needed for the display.

There are many applications that also require a simple graphic display along with text—signature verification on banking terminals and general-purpose credit verification, for example. 

6. The virtual window capability of the 82730 lets the user arrange independent areas in the system memory that can be displayed simultaneously on the CRT monitor.
VLSI Coprocessor Delivers High Quality Displays

Many microprocessor-based systems today use VLSI technology in processing and memory components. However, designers of subsystems have, up until now, not been able to incorporate this technology into their products because of the lack of available ICs. When, in 1981, NEC introduced the 7220 graphics display controller, users found that they could bolster system performance by off-loading graphics control chores from the system CPU. Second-sourced by Intel as the 82720, the chip uses its own drawing processor to access the required positions in the bitmap and handles both processing and display functions.

Now, Intel is poised to introduce a text coprocessor, the 82730, which is specifically tailored to execute data manipulation and display tasks. Lucio Lanza of Intel explains, "In an intelligent terminal or workstation, the CPU spends a lot of its time manipulating both graphics and text. We have identified these areas in terms of CPU use and we have distributed these blocks so that the CPU is not overburdened."

Coprocessors fall into two categories based on their architecture and operation. One type expands the microprocessor's own architecture by adding additional hardware and instructions. This type of tightly coupled coprocessor can be thought of as a transparent expansion of the microprocessor's architecture and works in synchronization with the CPU. Intel's first such coprocessor, the 8087, was designed for numerics processing and increased the microprocessor's math performance as much as 100 times.

The second type of coprocessor independently fetches its own data and sends instructions in parallel to the microprocessor. It therefore allows the microprocessor to process the tasks it handles best and delegate to the coprocessor the task it is best equipped to handle. In this cate-

Andrew Wilson
Technical Editor
Reprinted from ELECTRONIC IMAGING © April 1983, Morgan-Grampian Publishing Company, Boston, MA 02215

FIGURE 1: Block diagram of the 82730.
gory are I/O channel coprocessors and others that deal with communications and text processing tasks.

"The 82720 is not yet at this level," Lanza said, "since it does not have the capability of going to memory and extracting its own instruction and executing it—it needs something to spoon feed it."

Coprocessors of the second category do not monitor the CPU instruction stream. Instead, they are linked to the CPU via messages prepared and stored in shared memory. The CPU will prepare data and high level directives and then place them in memory. Upon completion of this control block, the CPU will alert the coprocessor by signaling it through a common channel attention line. From that point on, the coprocessor works on its own, fetching required data and instructions and then executing those instructions.

It is not synchronized with the CPU but works asynchronously and independently. When the coprocessor completes its task, it informs the CPU by signaling on the CPU's interrupt line.

The rationale for designing a coprocessor with one or the other architectures depends on the application requirement. Tightly coupling the coprocessor with the CPU gives the advantage of a short coprocessor preparation time but has the drawback of consuming the CPU's bus bandwidth.

In the case of numeric processing, the speed of executing the floating point algorithm is of paramount importance. Reducing the preparation time of the coprocessor task is the key because of the number of microseconds it takes to execute the task. Rapid algorithmic execution requires tight coupling. In the application of the I/O related coprocessor, the task execution time is much longer and the requirement for bus time can be much higher. And, for I/O operations the preparation time is not critical. A shared memory coupling is preferred for those types of applications because it provides greater flexibility in the design of the bus structure.

Text coprocessing

"In the design of the 82730," said Lanza, "we have tried to eliminate all the known differences between what is visible on the screen and what is obtained on the printed page. In word processing systems today, even the length of a row on the CRT is sometimes not the same as the length seen in print. Clearly, when you are editing text this becomes a major problem."

The 82730 supports the generation of text displays through features which include proportional spacing, simultaneous superscript/subscript, dynamically reloadable fonts and user programmable field and character attributes. Editing capabilities are further enhanced by features such as split screen, virtual windows, smooth scrolling and table-driven linked lists.

Figure 1 shows a block diagram of the 82730. The chip is divided into two main sections—the memory interface unit and the display generator. The memory interface unit provides the communication between the 82730 and the system processor, while the display generator acts on the display data and carries out the display operation.

Communication between the 82730 and the CPU takes place through messages placed in communication blocks in shared memory. The processor issues channel commands by preparing these message blocks and directing the 82730's attention to them by activating a hardware channel attention signal (CA). The memory interface unit fetches and executes these commands. When the display process is activated, the 82730 repeatedly fetches display data and embedded datastream commands from memory utilizing its built-in DMA capability, executes any datastream commands as encountered on the fly, and loads the row buffers with the display data. After executing these commands, the 82730 clears a busy flag in memory, to inform the host CPU that it is ready for the next command.

The memory interface unit is divided into two sections—the bus unit and the microcontroller unit. The bus interface unit provides the electrical interface to the system bus and the timing signals required for the microcontroller unit operations, making these operations transparent to the microcontroller unit. The 82730 can be programmed during initialization to provide 8 or 16 bit data, and 16 or 32 bit addressing.

The microcontroller unit contains the microinstruction store and the associated circuitry required for the execution of all channel and datastream commands. It uses the bus interface unit in carrying out its memory access tasks such as loading the row buffers with display data.

The interaction between the microcontroller unit and the display generator takes place through shared internal storage. The microcon-
controller unit fetches data from memory and writes it in the internal storage, while the display generator reads from the internal storage and carries out the display operation. The microcontroller unit and display generator operate asynchronously with respect to each other. Synchronization is accomplished through communication via internal flags and display timing signals generated by the display generator. The internal shared storage consists of row buffers which store the display data and internal registers which store display parameters. There are two row buffers each capable of storing up to 200 characters. The data in one row buffer is used by the display generator to display one complete character row on the screen, while the microcontroller unit is loading the second row buffer with display data fetched from memory. At the end of the row being displayed, the buffers are swapped and the microcontroller unit and display generator resume their tasks.

The display characteristics registers contain all the information used to control every aspect of display characteristics from screen size to blink rates. A major portion of this register set is the three content addressable memory (CAM) arrays that allow flexible timing control for row and screen characteristics. The user has the power to set the parameters for the entire screen by invoking a single high-level command.

By separating the video interface clocks from the bus interface clock, the 82730 provides the designer with the ability to independently maximize the performance of the CPU and video sections of the system.

The video interface consists of two independent clocks: the Reference Clock (RCLK) and the Character Clock (CCLK). While the RCLK controls the raster timing and defines the screen layout, the CCLK independently shifts character and attribute information out of the 82730, which allows proportional spacing to be achieved.

Combining text and graphics
A major requirement in the design of engineering workstations is the simultaneous display of both text and graphics. In terms of graphics requirements, the designer of such systems needs a drawing processor for fast geometric primitives, a math processor for fast transformations and a general purpose processor for access to the graphics database.

For text, string processing is needed for manipulation of text primitives and database processing is needed for access to the document files. The solution to this problem can be solved by using both the 720 graphics coprocessor and the 730 text coprocessor (Figure 2).

Both coprocessors work with Intel's new 82586 communications coprocessor. This works in conjunction with a CPU and the appropriate software to provide local area network (LAN) control capabilities. Message data to be placed on the network by a microprocessor-based workstation is stored in transmit blocks. Pointers (starting address information) to these blocks are stored along with processing instructions in other shared memory blocks. Status information and overall directives are stored in system control blocks which serve as the mailbox between the CPU and the 82586.

When alerted by a channel attention signal, the 82586 will perform a host of tasks involved in accessing data to be transmitted from its location in memory, framing the message packets containing the data and seeing to the transmission on the network medium. In addition, the 82586 receives and buffers incoming data which it then stores in shared memory for the CPU to collect. It is the CPU's job to allocate the blocks of memory for the LAN coprocessor to store the received packet data.
Something exciting is going on. But like most significant events, it is not happening quickly. Spurred on by developments in integrated circuit technology, a new generation of personal computers is taking shape, and the IBM PC and its clones are at the forefront.

As IBM PC users, it's sometimes hard to remember that the inanimate metal boxes in front of us are susceptible to evolution. But occasionally a product is introduced that forces the complete redesign of our personal computers.

Integrated circuits (ICs), the devices that bring intelligence to our machines, have reached a new level of technological achievement, and now the computers that use them must advance as well. Strange as it seems, these small silicon chips are setting the diskless computers running a wide array of software exactly as it will be printed, with superscripts and subscripts, and bold and italic typefaces on screen; and systems with greater, more accessible graphics.

As computer design is simplified by these advanced ICs, product differentiation will become greater. This portends the death of those PC clones capable only of basic spreadsheet and word processing operations. Instead, to survive in the increasingly cost-competitive, standardized personal computer market, small-system manufacturers will tailor their products for niche markets.

THE CHIP MAKERS

Now that personal computers have caught on, the semiconductor manufacturers who make ICs are eying the swelling market for personal computer ICs.

Dozens of newly developed semiconductor chips are being aimed at the personal computer market. These chips range from hard disk controllers that speed access time to linear predictive coding processors for speech recognition. With these new ICs driving personal computer design, we'll soon see machines we once only reasoned would exist: diskless computers running a wide array of software loaded over telephone lines; computers that display text exactly as it will be printed, with justified margins, superscripts and subscripts, and bold and italic typefaces on screen; and systems with greater, more accessible graphics.

As computer design is simplified by these advanced ICs, product differentiation will become greater. This portends the death of those PC clones capable only of basic spreadsheet and word processing operations. Instead, to survive in the increasingly cost-competitive, standardized personal computer market, small-system manufacturers will tailor their products for niche markets.

BIG BLUE

Intel Corporation, located in Northern California's renowned Silicon Valley, is one of the largest and most innovative chip manufacturers in the industry. IBM has been committed to Intel products for years; the PC is built around Intel's 8088 microprocessor and, as recently as late 1982, IBM invested $225 million in a minority share of Intel stock. A commitment this size is a good indicator of IBM's faith in Intel products. IBM's good faith and multimillion-dollar investment is guaranteed by Intel's long-standing promise that software written for the 8088 will run on all its future processors.

By taking a close look at the Intel ICs, we can gain valuable insight into the capabilities of the IBM PCs that will be built around them. The design philosophy of Intel's IC family differs radically from that of competitors Motorola, National Semiconductor, and Zilog. Diverse chip designs mean that the system designs of the IBM PC and its competitors, such as Apple's Lisa (based on the Motorola 6800 microprocessor), will also be radically different.

THE MICROPROCESSORS

Of the many Intel chips being produced, some will have a greater impact on the computer industry than others. In the vanguard will be the new microprocessors.

Design of the PC was shaped by IBM's surprising selection of the 8088. This choice caught most industry observers off guard since IBM, also the world's largest semiconductor manufacturer, had traditionally used its own designs for computer logic. Once Big Blue settled on the 8088, Intel's design philosophy was firmly implanted in the PC—from the 8088's segmented memory scheme to its 16-bit registers and 8-bit bus.

Like the 8088, each of the four microprocessors Intel is now readying for production could dramatically influence the design and performance of tomorrow's PCs.

The 80186. The recipe for putting an entire central processing unit (CPU) board on one chip is easy. Take an; 8086 (the 16-bit bus big brother of the 8088), speed it up, and then add most of the support chips essential to making the 8086 run in a personal computer. Reduce the size with the help of computer-aided design until all the chips fit onto one sliver of silicon, and voila, you have the 80186 (186), an entire motherboard on a chip.

While firming up plans for full-scale production of the 186, Intel is currently providing samples of the chip to computer manufacturers, including MAD Computer and Durango Systems. The rewards for using this newest chip are many: manufacturing costs are cut since a single IC is less expensive to buy than a boardful of them; physical CPU size is reduced, opening the way to shrink overall computer size or to put more power in the same box; and development time is cut for computer designers, which means considerable savings for system makers.

The 80188. If the 186 is too rich for your taste, the 80188 (188) may be more suitable. As with the 186; the 188's core CPU and support chips are melded on a single IC; like the 8088, however, the 188 has an 8-bit interface to the outside world (the 186 has a 16-bit interface). The 188 decreases costs by allowing computer manufacturers to use less expensive 8-bit peripherals. Although the 186 has received more publicity so far, the 188, aimed squarely at the massive 8-bit computer market, is expected to be used in greater numbers, at least in the short term.

The 80286. Powerful multiuser systems will benefit the most from the 80286 (286), possibly the most powerful microprocessor commercially available to date. Squeezing 150,000 transistors on a chip, the 286's designers have integrated a pair of HMOS-III (Intel's own proprietary process technology) 8086s and numerous other very large scale integration (VLSI) components. The resultant chip is two to three times faster than the Motorola 68000 even though both chips can address about the same amount of
memory. The 286 has very high speed (1.5 million instructions per second, five to six times faster than the 8086), about 16 megabytes worth of addressable physical memory, the ability to address a virtual memory of 1 gigabyte per task (equal to the capacity of 100 IBM XT Winchester drives), and the ability to provide several layers of multiuser security on chip.

The 80386. Not yet built, the 80386 (386) is promised for 1984, but the release date may slide to 1985. If the 286 is vastly more powerful than the 8088 or 8086, then the 386's potential is staggering. Complementary metallic oxide semiconductor (CMOS) process technology, which lowers power consumption, is being used to build this 32-bit chip. Intel, Motorola, and National Semiconductor are already jockeying for position in what will be an intense competition for the 32-bit market. Motorola is claiming that its 68020 will be the first widely available 32-bit microprocessor when it is introduced later this year, although NCR has already scooped the industry with its 32-bit chip. Hewlett-Packard, not to be outdone, has put 450,000 transistors on a single proprietary 32-bit microprocessor, which is used in the $20,000 to $30,000 HP 9000 workstation.

How will these processors impact the personal computers that use them? The most obvious effect will be faster performance. Even the budget model 188 boasts two to five times the instruction and execution speed of the 8088 in today's PC. A 286 is about twice again as fast as the 188, and next year's data-gobbling 386 will have more speed than anyone can immediately use. Since the 188 is ideal for low-priced portable computers, it ceases the ironic probability that a PC-compatible portable may soon be available that will run the IBM PC's full line of software and run it faster than the full-sized PC.

SOFTWARE ON SILICON

One chip ready to plug into the next generation of personal computers is the 80150 (150) CP/M software-in-silicon operating system. A complete CP/M-86 operating system is stored in ROM on this chip, along with drivers for input and output devices.

Use of a 150 CP/M chip will eliminate the traditional booting up procedure of loading an operating system disk and reading its contents into operating RAM. Instead, the user will simply turn on the computer and press a CP/M-86 button. Again and more importantly, this chip lowers overall computer production costs since a disk drive and attendant control circuits are replaced by a solitary chip.

Another chip, similar to the 150, has Intel's proprietary RMX operating system in silicon. This little-known RMX chip is also suitable for present and future IBM PCs.

Many people question the wisdom of putting software in silicon. "Software should be soft," says Bill Gates, chairman of the board at Microsoft. He points out that operating systems are constantly updated; for instance, Microsoft will soon offer a revised version of MS-DOS that supports networking. Such updates can't readily be added to a hardware production line and certainly won't help the ROM chips already in users' computers.

BLOCK DIAGRAM OF INTEL'S 80150 CP/M ON A CHIP WITH THE 8088 OR 8086 MICROPROCESSOR
Still, Intel argues that its choice of CP/M makes the 150 practical. "We picked CP/M because it is a mature operating system," Says Intel's product marketing engineer for software on silicon, Carl Buck. "We'd have more difficulty with a less developed product." The many versions of MS-DOS helped eliminate that operating system from consideration. Yet according to Digital Research President John Rowley, Intel left some room on the 150 chip to add to CP/M in the future.

Also, use of the 150 CP/M chip doesn't preclude the use of other operating systems. PC-DOS could still be loaded into a system and run, making use of the 150's input/output drivers.

PORTABLES

Having software on silicon opens the way for very powerful diskless portable computers. The minimum configuration for a 188-based unit with the 150 CP/M operating system could include one or two BASIC applications programs in ROM, providing spreadsheet and word processing power in a unit the size of a keyboard with a small flip-up screen. Intel Product Marketing Engineer Tony Zingale suggests we may soon see truly usable portables selling for around $500.

More ambitious and expensive portables could accept applications software over telephone lines, loading them into a variety of media. Several memory technologies will compete for room in portable computers, including magnetic bubble memories, already being used in the Grid and Teleram computers. Commercially available bubbles have 4 megabit capacity, while 10- to 16-megabit bubbles are projected for the near future. Japan's NEC reported a major breakthrough that within 5 years will allow bubbles to store 1 gigabit of data. Of course, 8 of those bits are needed to store 1 byte of data.

Vying with bubbles in some applications and complementing them in others are electronically programmable read-only memories, or EPROMs. Like ROM, EPROMs are nonvolatile chips. Unlike ROM, EPROMs can be reprogrammed. Intel now offers 256K EPROMs, and it is anticipated that other companies will offer 256K EPROMs before the year's end.

GRAPHICS

The space created on the motherboard by the 186 and friends will enable computer designers to add more graphics capability to their systems. Like the 150 there are co-processor chips ready for the task.

A pair of Intel ICs, the 82720 (720) graphics display controller and the 82730 (730) text co-processor, are touted as providing vastly enhanced and simplified displays. With the 730, text can be displayed on the computer screen as it will be printed out. Italics can be mixed with straight text, and superscripts and subscripts are shown without the annoying and often misleading arrows common in today's software.

Editing can be speeded up by the 730's support for split screens, multiple windows, dual cursors, smooth scrolling, and table-driven linked lists. Displays of up to 200 characters per row and 128 lines per screen can be supported, and unique character sets, such as Arabic or Japanese, can be built.

Even more capability can be added though the 720, an IC that works with or without the 730. Introduced in September 1982, the 720, a joint effort between Intel and NEC, is said to be integral to graphics plans for NEC's 8086-based Advanced Personal Computer.

One application in which the 720 and 730 will shine is opening windows on-screen. Most computer users are familiar with the ability of Apple's Lisa to link spreadsheets, graphics, and word processing through multiple displays, or windows, on one screen. Lisa uses memory-hungry software and dedicated hardware. Apple's initial release uses 1 full megabyte of RAM, and Lisa will soon be offered with 4M of internal memory in addition to a mandatory 5M hard disk.

For comparison, the IBM PC, limited by the range of the 8088, can address 1M tops. VisiCorp's Visi/ON promises Lisa-like graphics and program-linking capabilities for the IBM PC, with lower memory demands and no dedicated hardware other than a mouse. Although Visi/ON supposedly runs faster with an 8087 math co-processor, VisiCorp will not comment on whether its software will make use of the 720 or the 730.

BIT-MAPPED GRAPHICS

Both Lisa and Visi/ON use bit-mapping, a process that the 720 and the 730 are said to simplify. In plain words, to create an image on-screen, the electron gun that illuminates the screen must be positioned and then turned on and off. Data to do this is stored in RAM as a bit-map memory corresponding to positions of pixels lit on the screen. For one-level monochrome displays, 1 memory bit describes each pixel; for color and levels of grey, several bits must be used to describe each pixel.

Creating images is a lengthy chain of simple operations. In a system that uses the 8088 alone, the microprocessor is heavily burdened and the software runs slowly. Using complementary chips to take up part of the processing chore will speed up the process considerably. This is where the 720 and the 730 come in. By doing tasks such as looking up and manipulating a library of commonly used figures, quickly accessing the bit-map memory, and rewriting the bit map, both chips speed text and graphics operations.

FLAT VS. SEGMENTED MEMORY

Use of the 720 and the 730 demonstrates Intel's design philosophy and how this philosophy impacts the IBM PC. Computers such as Lisa that are based on the Motorola 68000 have a flat memory, while computers based on the 8088 or 8086 use segmented memory. According to Intel,
segmented memory (see “How the PC thinks,” *PC World*, Vol. 1, No. 1) works better for text and graphics manipulation than its flat counterpart. Ordinarily in processing any string of characters, changing a single letter in a string of text means repositioning every character in a document. But since segmentation uses pointers to locate data in memory, only the pointers locating the beginning and the end of a passage of text have to be changed. Similarly, pointers in memory can be used to position bit-map data corresponding to multiple windows on-screen, eliminating the need to recalculate and manipulate the entire bit map. Segmented vs. flat memory has become somewhat of a religious issue in the semiconductor industry.

Intel and Motorola also differ on how much burden to put on the CPU. Motorola’s 68000 is faster than the 8088 and the 8086 and can address more memory than either of those chips or the 188 or the 186. But the 186 and the 286 are substantially faster than the 68000. Also, the 286’s ability to address 16M opens the way to using large memory segments, strengthening Intel’s case for segmented memory.

In many 68000-based high-end systems the computer designers have decided to use a co-processor, either bit slice, or in one case, an 8086, to do graphics. Many people are skeptical of Intel’s graphics approach, but Intel maintains that its approach will allow computer designers greater flexibility. In an ultimate system, multiple 720s and 730s could be combined to handle interactive windows under the direction of a 286 processor, while more complex imagery (beyond the practical ability of bit-mapping) could be managed by an 80287 math co-processor, the next generation cousin of the 8087. The creation of three-dimensional graphics that can be rotated on-screen for advanced computer-aided design and manufacturing systems, for instance, is best handled by Vector Graphics rather than bit-mapping.

**SOFTWARE DEMANDS**

Yet there is more to computer design than hardware. Software must be written to take advantage of the new IC’s promise. In the case of the 286, no operating system yet exists that can take full advantage of its operating capabilities. Plug-ins currently on the market that add the 286 to the IBM PC provide little more than a faster 8086. Only new operating software will use the new chips to their fullest potential.

One solution on the horizon is a 286 version of XENIX due to be introduced mid-1983. XENIX, a multiuser operating system with a visual shell similar to MS-DOS, is a takeoff on Bell Labs’ UNIX operating system. A licensing agreement among Intel, Bell Labs, and Microsoft, the author of XENIX and MS-DOS, is reported being negotiated. Negotiations between Intel and Digital Research to provide a CP/M variant for the 286 have been underway for some time but have reportedly stagnated.

For lower-end systems Microsoft is said to be upgrading MS-DOS to accommodate networking. This advance comes at the right time, as the 188 and 186 open up sockets that could be used for local area network chips such as the programmable Ethernet chip from Intel.

As long as software and hardware keep growing rapidly together, PC users will be offered a continuing stream of improved computers and ever more capable plug-in boards. The variety seems endless and next year’s crop exciting.
### DOMESTIC SALES OFFICES

<table>
<thead>
<tr>
<th>State</th>
<th>Address Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALABAMA</td>
<td>Intel Corp., 3300 Bedford Drive, Suite E, Haltomville, TX 76118. Tel: (817) 830-4010</td>
</tr>
<tr>
<td>ARIZONA</td>
<td>Intel Corp., 11225 N 28th Drive, Suite 2140, Phoenix, AZ 85029. Tel: (602) 869-4980</td>
</tr>
<tr>
<td>CALIFORNIA</td>
<td>Intel Corp., 2725 Vanowen Street, Suite 700, Burbank, CA 91505. Tel: (818) 704-6500</td>
</tr>
<tr>
<td>COLORADO</td>
<td>Intel Corp., 4440 Cervantes Park Drive, Suite 100, Colorado Springs, CO 80907. Tel: (719) 594-6262</td>
</tr>
<tr>
<td>CONNECTICUT</td>
<td>Intel Corp., 26 Mill Plain Road, Danbury, CT 06810. Tel: (203) 748-3130. TWX: 212-514-1169</td>
</tr>
<tr>
<td>FLORIDA</td>
<td>Intel Corp., 2402 Rainbow Drive, Suite 105, Allentown, PA 18337. Tel: (610) 889-5858</td>
</tr>
<tr>
<td>GEORGIA</td>
<td>Intel Corp., 3100 Holcomb Park Drive, Suite 170, Atlanta, GA 30308. Tel: (404) 869-5569</td>
</tr>
<tr>
<td>ILLINOIS</td>
<td>Intel Corp., 2330 North 38th Street, Suite 875, Decatur, IL 62526. Tel: (217) 447-0517</td>
</tr>
<tr>
<td>INDIANA</td>
<td>Intel Corp., 17777 Blockade Road, Suite 200, Indianapolis, IN 46268. Tel: (317) 875-0623</td>
</tr>
<tr>
<td>IOWA</td>
<td>Intel Corp., 6400 W 10th Street, Suite 170, Des Moines, IA 50321. Tel: (515) 642-0690</td>
</tr>
<tr>
<td>KANSAS</td>
<td>Intel Corp., 3121 Parkway Drive South, Suite C, Herndon, VA 22070. Tel: (703) 766-7000. TWX: 703-862-5444</td>
</tr>
<tr>
<td>LOUISIANA</td>
<td>Intel Corp., 7321 Parkway Drive South, Suite C, Herndon, VA 22070. Tel: (703) 766-7000. TWX: 703-862-5444</td>
</tr>
<tr>
<td>MARYLAND</td>
<td>Intel Corp., 7321 Parkway Drive South, Suite C, Herndon, VA 22070. Tel: (703) 766-7000. TWX: 703-862-5444</td>
</tr>
<tr>
<td>MASSACHUSETTS</td>
<td>Intel Corp., 38 Industrial Avenue Chelsea, MA 02150. Tel: (617) 256-1500. TWX: 703-343-6330</td>
</tr>
<tr>
<td>MICHIGAN</td>
<td>Intel Corp., 9701 Orchard Lake Road, Suite 100, West Bloomfield, MI 48303. Tel: (813) 851-9900</td>
</tr>
<tr>
<td>MINNESOTA</td>
<td>Intel Corp., 3244 W 80th Street Suite 360 Bloomington, MN 55431. Tel: (612) 835-4722. TWX: 612-950-3687</td>
</tr>
<tr>
<td>MISSOURI</td>
<td>Intel Corp., 4020 Citi Expressway Suite 131 St. Louis, MO 63145. Tel: (314) 291-1990</td>
</tr>
<tr>
<td>MICHIGAN</td>
<td>Intel Corp., 9701 Orchard Lake Road, Suite 100, West Bloomfield, MI 48303. Tel: (813) 851-9900</td>
</tr>
<tr>
<td>NEW JERSEY</td>
<td>Intel Corp., 9550 Park Boulevard, Suite B-250, Albuquerque, NM 87112. Tel: (505) 282-8066</td>
</tr>
<tr>
<td>NEW MEXICO</td>
<td>Intel Corp., 12425 Park Boulevard, Suite B-250, Albuquerque, NM 87112. Tel: (505) 282-8066</td>
</tr>
<tr>
<td>NEW YORK</td>
<td>Intel Corp., 280 Vanderbilt Avenue, Suite 300, New York, NY 10017. Tel: (212) 841-8087. TWX: 910-901-5657</td>
</tr>
<tr>
<td>OHIO</td>
<td>Intel Corp., 8500 Main Street, Suite 200, Beaverton, OR 97005. Tel: (503) 641-8066. TWX: 503-641-0741</td>
</tr>
<tr>
<td>TEXAS</td>
<td>Intel Corp., 3300 West Freeway, Suite 1400, Houston, TX 77074. Tel: (713) 877-8068. TWX: 910-682-9999</td>
</tr>
<tr>
<td>COLORADO</td>
<td>Intel Corp., 1161 North Eldorado Place, Suite 301, Tucson, AZ 85715. Tel: (602) 299-6815</td>
</tr>
<tr>
<td>TEXAS</td>
<td>Intel Corp., 1350 Shorebird Way, Suite 101, Cheyenne, WY 82001. Tel: (307) 613-8022</td>
</tr>
<tr>
<td>TEXAS</td>
<td>Intel Corp., 4300 Citi Expressway Suite 131, St. Louis, MO 63145. Tel: (314) 291-1990</td>
</tr>
<tr>
<td>VIRGINIA</td>
<td>Intel Corp., 8520 Sunset Road Suite 120, Fairfax, VA 22030. Tel: (703) 524-2020</td>
</tr>
<tr>
<td>WASHINGTON</td>
<td>Intel Corp., 11310 11th Avenue N.E. Suite 510, Seattle, WA 98125. Tel: (206) 404-2056. TWX: 910-443-3002</td>
</tr>
<tr>
<td>WISCONSIN</td>
<td>Intel Corp., 450 N. Sunnyslope Road Suite 132, Chippewa Falls, WI 54729. Tel: (715) 344-2736. TWX: 715-457-3008</td>
</tr>
</tbody>
</table>

*Field Application Location
UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Japan

FRANCE
Intel
5 Place de la Balance
Silic 223
94528 Rungis Cedex
France

UNITED KINGDOM
Intel
Piper's Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel
Seidstrasse 27
D-8000 Munchen 2
West Germany