LITERATURE

To order Intel Literature or obtain literature pricing information in the U.S. and Canada call or write Intel Literature Sales. In Europe and other international locations, please contact your local sales office or distributor.

INTEL LITERATURE SALES
P.O. BOX 7641
Mt. Prospect, IL 60056-7641

CURRENT HANDBOOKS

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

<table>
<thead>
<tr>
<th>TITLE</th>
<th>LITERATURE ORDER NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET OF 11 HANDBOOKS (Available in U.S. and Canada only)</td>
<td>231003</td>
</tr>
<tr>
<td>EMBEDDED APPLICATIONS</td>
<td>270648</td>
</tr>
<tr>
<td>8-BIT EMBEDDED CONTROLLERS</td>
<td>270645</td>
</tr>
<tr>
<td>16-BIT EMBEDDED CONTROLLERS</td>
<td>270646</td>
</tr>
<tr>
<td>16/32-BIT EMBEDDED PROCESSORS</td>
<td>270647</td>
</tr>
<tr>
<td>MEMORY</td>
<td>210830</td>
</tr>
<tr>
<td>MICROCOMMUNICATIONS (2 volume set)</td>
<td>231658</td>
</tr>
<tr>
<td>MICROCOMPUTER SYSTEMS</td>
<td>280407</td>
</tr>
<tr>
<td>MICROPROCESSORS</td>
<td>230843</td>
</tr>
<tr>
<td>PERIPHERALS</td>
<td>296467</td>
</tr>
<tr>
<td>PRODUCT GUIDE (Overview of Intel's complete product lines)</td>
<td>210846</td>
</tr>
<tr>
<td>PROGRAMMABLE LOGIC</td>
<td>296083</td>
</tr>
<tr>
<td>ADDITIONAL LITERATURE (Not included in handbook set)</td>
<td></td>
</tr>
<tr>
<td>AUTOMOTIVE SUPPLEMENT</td>
<td>231792</td>
</tr>
<tr>
<td>COMPONENTS QUALITY/RELIABILITY HANDBOOK</td>
<td>210997</td>
</tr>
<tr>
<td>INTEL PACKAGING OUTLINES AND DIMENSIONS (Packaging types, number of leads, etc.)</td>
<td>231369</td>
</tr>
<tr>
<td>INTERNATIONAL LITERATURE GUIDE</td>
<td>E00029</td>
</tr>
<tr>
<td>LITERATURE PRICE LIST (U.S. and Canada) (Comprehensive list of current Intel Literature)</td>
<td>210620</td>
</tr>
<tr>
<td>MILITARY (2 volume set)</td>
<td>210461</td>
</tr>
<tr>
<td>SYSTEMS QUALITY/RELIABILITY</td>
<td>231762</td>
</tr>
</tbody>
</table>

ORDER NUMBER
231003
270648
270645
270646
270647
210830
231658
280407
230843
296467
210846
296083
231792
210997
231369
E00029
210620
210461
231762
# U.S. and CANADA LITERATURE ORDER FORM

**NAME:** ________________________________________________

**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>
<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>

Subtotal __________________

**Must Add Your Local Sales Tax** ____________________

Postage: add 10% of subtotal __________________

Postage __________________

**Total** __________________

Pay by check, money order, or include company purchase order with this form ($100 minimum). We also accept VISA, MasterCard or American Express. Make payment to Intel Literature Sales. Allow 2-4 weeks for delivery.

- [ ] VISA  - [ ] MasterCard  - [ ] American Express  Expiration Date __________________

Account No. ____________________________________________

Signature ________________________________________________

**Mail To:** Intel Literature Sales  
P.O. Box 7641  
Mt. Prospect, IL 60056-7641

**International Customers** outside the U.S. and Canada should use the International order form or contact their local Sales Office or Distributor.

For phone orders in the U.S. and Canada  
Call Toll Free: (800) 548-4725  
Prices good until 12/31/90.

Source: HB
INTERNATIONAL LITERATURE ORDER FORM

NAME: _________________________________________________________

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>
<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>

Subtotal ________

Must Add Your Local Sales Tax ________

Total ________

PAYMENT

Cheques should be made payable to your local Intel Sales Office (see inside back cover.)

Other forms of payment may be available in your country. Please contact the Literature Coordinator at your local Intel Sales Office for details.

The completed form should be marked to the attention of the LITERATURE COORDINATOR and returned to your local Intel Sales Office.
i860™
64-BIT
MICROPROCESSOR
HARDWARE
REFERENCE
MANUAL

1990
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:

376, 386, 387, 486, 4-SITE, Above, ACE51, ACE96, ACE186, ACE196, ACE960, BITBUS, COMMputer, CREDIT, Data Pipeline, DVI, ETOX, FaxBACK, Genius, i, t, i486, i750, i860, ICE, ICEL, ICEVIEW, iCS, IDBP, IDIS, iICE, ILBX, IMDDX, IMMX, Inboard, Insite, Intel, intgld, Intel386, intgI386OS, Intel Certified, Intelelevision, intelligent Identifier, intelligent Programming, Intellec, Intellink, IOSP, iPAT, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iXM, Library Manager, MAPNET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, PRO750, PROMPT, Promware, QUEST, QueX, Quick-Erase, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, TooITALK, UPI, Visual Edge, VLSICELL, and ZapCode, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iXM, 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.

CHMOS and HMOS are patented processes of Intel Corp.

Intel Corporation and Intel’s FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

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

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641
PREFACE

The Intel i860™ 64-bit microprocessor is a general purpose microprocessor integrating an integer RISC core unit, a floating-point unit, a paged memory management unit, instruction and data caches, and 3-D graphics software assist logic in a single VLSI component. The versatile 64-bit design of the i860 microprocessor balances performance across integer, floating point, and graphics processing capability. Its parallel architecture achieves high throughput with RISC design techniques, pipelined processing units, wide data paths, large on-chip caches and fast one micron CHMOS® IV silicon technology.

This manual provides the basic information required to implement an i860 microprocessor based system. It explains the external hardware details of the processor.

Although the main users of this manual are hardware design engineers, it contains basic hardware information which is of value to software engineers and programmers. These readers should reference the first three chapters only.

RELATED PUBLICATIONS

In this manual, the i860 microprocessor is presented from a hardware perspective. Information on the software architecture, instruction set and programming can be found in these related Intel publications:

- i860™ 64-Bit Microprocessor Programmers Reference Manual, order number 240329

Information on the device specification for the i860 microprocessor is available in the i860™ 64-Bit Microprocessor Data Sheet, order number 240296. Always refer to the most recent version of the device specification.

ORGANIZATION OF THE MANUAL

- Chapter 1, “Introduction to i860™ 64-Bit Microprocessor”, provides an overview of the features of the i860 microprocessor and the advantages to system designers. It also provides the insight to i860 microprocessor applications.
- Chapter 2, “Internal Architecture”, describes the internal architecture of the i860 microprocessor.
- Chapter 3, “Local Bus Interface”, discusses the i860 microprocessor local bus interface. This includes the signal descriptions, bus operation and local bus interface guidelines.
- Chapter 4, “Memory Interfacing”, discusses techniques for designing memory subsystems for the i860 microprocessor. The schematics are given in Appendix C.
- Chapter 5, “I/O Interfacing”, discusses techniques for connecting I/O devices to an i860 microprocessor system.
- Chapter 6, “Graphics Subsystem Example”, discusses a design example for implementing an i860 microprocessor based graphics subsystem.
- Chapter 7, “MULTIBUS® II and the i860™ Microprocessor”, provides a design example for the MULTIBUS II board built around the i860 microprocessor.
• Chapter 8, “Physical Design and Debugging”, contains recommendations for constructing and debugging the i860 microprocessor based systems.

• Chapter 9, “Testability”, covers the testability issues on the i860 microprocessor.

• Appendix A provides tested schematics of an i860 microprocessor Graphics Frame Buffer Board and is to used in conjunction with Chapter 6.

• Appendix B contains an untested example for the i860 microprocessor based MULTIBUS II design that is explained in Chapter 7.

• Appendix C provides the tested schematics and PLD codes for i860 microprocessor memory design.

Signal mnemonics convey whether a signal state is active high or active low. An active-low signal mnemonic is suffixed with a pound character (#); an active-high signal mnemonic does not have this suffix.
<table>
<thead>
<tr>
<th>CHAPTER 1</th>
<th>INTRODUCTION TO THE i860™ 64-BIT MICROPROCESSOR</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>PROCESSOR CHARACTERISTICS</td>
<td>1-1</td>
</tr>
<tr>
<td>1.2</td>
<td>PROCESSOR OVERVIEW</td>
<td>1-3</td>
</tr>
<tr>
<td>1.2.1</td>
<td>Pipelining and Parallelism</td>
<td>1-3</td>
</tr>
<tr>
<td>1.2.2</td>
<td>Instruction Set Architecture</td>
<td>1-4</td>
</tr>
<tr>
<td>1.2.3</td>
<td>Registers</td>
<td>1-4</td>
</tr>
<tr>
<td>1.2.4</td>
<td>Address Space</td>
<td>1-4</td>
</tr>
<tr>
<td>1.2.5</td>
<td>Floating-Point Operations</td>
<td>1-5</td>
</tr>
<tr>
<td>1.2.6</td>
<td>Graphics Support</td>
<td>1-5</td>
</tr>
<tr>
<td>1.2.7</td>
<td>Caches</td>
<td>1-5</td>
</tr>
<tr>
<td>1.2.8</td>
<td>Paging Unit</td>
<td>1-6</td>
</tr>
<tr>
<td>1.2.9</td>
<td>Debugging Support</td>
<td>1-6</td>
</tr>
<tr>
<td>1.2.10</td>
<td>External Interface</td>
<td>1-6</td>
</tr>
<tr>
<td>1.2.10.1</td>
<td>TWO-LEVEL BUS PIPELINING OF LOCAL BUS</td>
<td>1-6</td>
</tr>
<tr>
<td>1.2.10.2</td>
<td>BUS ARBITRATION SUPPORT</td>
<td>1-6</td>
</tr>
<tr>
<td>1.2.10.3</td>
<td>DRAM INTERFACE SUPPORT (PAGE-MODE AND STATIC COLUMN)</td>
<td>1-7</td>
</tr>
<tr>
<td>1.2.10.4</td>
<td>CACHE CONTROL</td>
<td>1-7</td>
</tr>
<tr>
<td>1.2.10.5</td>
<td>LOCKED MEMORY CYCLES</td>
<td>1-7</td>
</tr>
<tr>
<td>1.2.10.6</td>
<td>EIGHT-BIT BUS ACCESS FOR BOOTSTRAPPING OPERATION (CS8)</td>
<td>1-8</td>
</tr>
<tr>
<td>1.2.10.7</td>
<td>BOUNDARY SCAN TO SIMPLIFY BOARD LEVEL TESTING OR BOOTUP CODE</td>
<td>1-8</td>
</tr>
<tr>
<td>1.2.11</td>
<td>Clock Requirements</td>
<td>1-8</td>
</tr>
<tr>
<td>1.2.12</td>
<td>i860™ Microprocessor Packaging and Power Requirements</td>
<td>1-8</td>
</tr>
<tr>
<td>1.3</td>
<td>SYSTEM CONFIGURATION</td>
<td>1-9</td>
</tr>
<tr>
<td>1.3.1</td>
<td>Private and Shared Memory Configuration</td>
<td>1-9</td>
</tr>
<tr>
<td>1.4</td>
<td>APPLICATION OVERVIEW</td>
<td>1-9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 2</th>
<th>INTERNAL ARCHITECTURE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>CORE EXECUTION UNIT</td>
<td>2-1</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Core Unit Registers</td>
<td>2-1</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Instruction Execution</td>
<td>2-3</td>
</tr>
<tr>
<td>2.1.2.1</td>
<td>RISC OPTIMIZATIONS</td>
<td>2-3</td>
</tr>
<tr>
<td>2.1.2.2</td>
<td>DUAL-INSTRUCTION MODE</td>
<td>2-4</td>
</tr>
<tr>
<td>2.2</td>
<td>FLOATING-POINT UNIT</td>
<td>2-4</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Floating-Point Register Bank</td>
<td>2-6</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Pipelined and Scalar Operations</td>
<td>2-6</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Floating-Point Adder Unit</td>
<td>2-8</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Floating-Point Multiplier Unit</td>
<td>2-8</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Dual Operation Feature</td>
<td>2-8</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

2.2.6 Floating-Point Computation Throughput .............................................. 2-9
2.3 PAGING UNIT .................................................................................. 2-10
  2.3.1 Paging Algorithm ........................................................................... 2-11
2.4 ON-CHIP CACHES AND BUS CONTROL ............................................. 2-13
  2.4.1 Instruction Cache Unit .................................................................... 2-13
  2.4.2 Data Cache Unit ........................................................................... 2-14
  2.4.2.1 WRITE OPERATIONS AND THE DATA CACHE ......................... 2-14
  2.4.3 Bypassing Instruction and Data Caches ........................................ 2-15
  2.4.4 Flushing Instruction Cache, Data Cache, and TLB ......................... 2-15
  2.4.5 Bus and Cache Control Unit ......................................................... 2-15
  2.4.6 Write Buffers ............................................................................... 2-16
2.5 GRAPHICS UNIT .............................................................................. 2-16

CHAPTER 3
LOCAL BUS INTERFACE
3.1 i860™ MICROPROCESSOR EXTERNAL INTERFACE AND BUS SIGNALS ............................................. 3-1
  3.1.1 i860™ Microprocessor Buses ........................................................ 3-1
  3.1.2 i860™ Microprocessor Output Signals .......................................... 3-1
  3.1.3 i860™ Microprocessor Input Signals ............................................ 3-2
  3.1.4 Power and Ground Pins ............................................................... 3-4
  3.2 BUS CHARACTERISTICS .................................................................. 3-4
3.3 BUS TRANSFER OPERATIONS ............................................................. 3-5
  3.3.1 64-bit Bus and Byte Alignment of Data ......................................... 3-5
  3.3.1.1 MEMORY ADDRESSABILITY AND ALIGNMENT REQUIREMENTS ............................................................................ 3-5
  3.3.1.2 DATA ALIGNMENT DURING READ OPERATIONS ................. 3-5
  3.3.1.3 DATA ALIGNMENT DURING WRITE OPERATIONS ............... 3-6
  3.3.1.4 LITTLE AND BIG ENDIAN MODES AND BUS OPERATION ....... 3-6
  3.3.1.5 ENDIAN MODE FOR CODE ACCESSES ................................ 3-10
  3.3.1.6 SYSTEM OPERATION AND ENDIAN MODE ............................ 3-10
  3.3.2 Basic Bus Operation ..................................................................... 3-10
  3.3.3 Nonpipelined Bus Operations ....................................................... 3-10
  3.3.3.1 NONPIPELINED READ OPERATIONS .................................. 3-11
  3.3.3.2 NONPIPELINED WRITE CYCLES ......................................... 3-11
  3.3.3.3 WRITE CYCLES FOLLOWING READ CYCLES ....................... 3-13
  3.3.4 Pipelined Operations ..................................................................... 3-14
  3.3.4.1 PIPELINING AND INTERLEAVED MEMORY BANKS ............ 3-15
  3.3.4.2 ORDERING OF DATA DURING PIPELINED OPERATIONS .... 3-15
  3.3.4.3 BUS STATE MACHINE FOR PIPELINING ............................... 3-16
  3.3.4.4 PIPELINED READ AND WRITE CYCLES ............................... 3-18
  3.3.5 8-Bit Bus Transfers for Bootstrapping (CS8 Mode) ....................... 3-19
3.4 BUS CONTROL OPERATIONS ............................................................... 3-20
  3.4.1 Page Mode, Static Column DRAMs and Next Near Operation .......... 3-20
  3.4.2 Bus Hold, Hold Acknowledge, Bus Request ................................... 3-21
  3.4.3 Bus Lock ..................................................................................... 3-22
TABLE OF CONTENTS

3.4.3.1 SUPERVISOR-MODE ACTIVATION OF LOCK# .......................... 3-24
3.4.3.2 USER-MODE ACTIVATION OF LOCK# .................................. 3-24
3.4.3.3 BUS LOCK DURING PAGE TABLE UPDATE .......................... 3-25
3.5 CACHE CONTROL OPERATIONS ........................................ 3-25
3.6 TRAPS AND INTERRUPTS .................................................. 3-26
3.7 TEST SUPPORT FUNCTIONS ............................................... 3-27
3.7.1 Normal Mode Operation .................................................. 3-29
3.7.2 Serial Mode Operation .................................................. 3-29
3.8 RESET AND CLOCK CIRCUIT ............................................. 3-30

CHAPTER 4
MEMORY INTERFACING
4.1 INTRODUCTION .......................................................... 4-1
4.2 CPU FEATURES ........................................................... 4-1
4.2.1 The KEN# Input ..................................................... 4-1
4.2.2 Bus Pipelining ......................................................... 4-2
4.2.3 The Next Near Pin .................................................... 4-3
4.2.4 Write Data Function ................................................. 4-4
4.3 DRAM SUBSYSTEM OVERVIEW ......................................... 4-5
4.3.1 Address Path Logic ................................................... 4-5
4.3.2 Data Path Logic ....................................................... 4-6
4.3.3 Parity Logic ............................................................ 4-7
4.3.4 Control Logic .......................................................... 4-7
4.4 DRAM SUBSYSTEM FUNCTION ........................................... 4-9
4.4.1 Signal Description ..................................................... 4-9
4.4.1.1 PROCESSOR INTERFACE ........................................ 4-9
4.4.1.2 DATA PATH LOGIC CONTROL .................................. 4-9
4.4.1.3 ADDRESS PATH LOGIC CONTROL ................................ 4-10
4.4.1.4 CONTROLLER SIGNALS .......................................... 4-10
4.4.2 Basic Read Cycle ....................................................... 4-11
4.4.3 Pipelined Read Cycles ............................................... 4-12
4.4.4 Basic Write Cycle ..................................................... 4-14
4.4.4.1 CONSECUTIVE WRITE CYCLES .................................. 4-15
4.4.5 Consecutive Bus Cycles .............................................. 4-15
4.4.5.1 WRITE FOLLOWED BY READ CYCLES .......................... 4-16
4.4.5.2 READ CYCLES FOLLOWED BY WRITE CYCLES ............... 4-18
4.4.6 Page Miss Cycles ....................................................... 4-19
4.4.7 Refresh Cycles .......................................................... 4-20
4.5 PARITY CIRCUIT ........................................................... 4-21
4.5.1 Dedicated Signals ..................................................... 4-21
4.5.2 Parity Function ......................................................... 4-21
CHAPTER 5
I/O INTERFACING
5.1 OVERVIEW .......................................................... 5-1
5.1.1 i860™ Microprocessor I/O Subsystem .......................... 5-1
5.2 GENERATING I/O CONTROL SIGNALS ............................ 5-3
5.2.1 I/O Control Logic .................................................. 5-3
5.2.2 Address Decode and Chip Select ................................ 5-4
5.2.3 IORD#/IOWR# ...................................................... 5-5
5.2.4 READY# .............................................................. 5-5
5.2.5 Recovery and Bus Contention ..................................... 5-5
5.3 I/O CYCLES .............................................................. 5-6
5.3.1 Read Cycle Timing .................................................. 5-6
5.3.2 Write Cycle Timing .................................................. 5-8
5.4 DESIGN EXAMPLES .................................................... 5-9
5.4.1 82510 Interface ..................................................... 5-9
5.4.2 Eprom Interface ..................................................... 5-12
5.4.2.1 DOUBLE COPY LOAD ........................................... 5-16
5.4.2.2 EPROM TIMINGS .................................................. 5-17
5.5 DMA INTERFACE RECOMMENDATIONS ........................... 5-18

CHAPTER 6
GRAPHICS SUBSYSTEM EXAMPLE
6.1 INTRODUCTION ......................................................... 6-1
6.2 GRAPHICS AND THE i860™ MICROPROCESSOR .................... 6-1
6.2.1 Processor Bus Bandwidth ......................................... 6-2
6.3 3-D GRAPHICS EXAMPLE ............................................. 6-2
6.3.1 Features ............................................................. 6-2
6.3.2 Testing ............................................................... 6-2
6.4 SYSTEM OVERVIEW ..................................................... 6-3
6.4.1 Expansion Bus Interface ............................................ 6-3
6.4.2 Data Transceiver/Latch Control .................................. 6-4
6.4.3 Address Transceiver/Latch ....................................... 6-4
6.4.4 VRAM Control ....................................................... 6-4
6.4.5 Serial Row/Column Address Generation .......................... 6-4
6.4.6 Double Buffering .................................................... 6-4
6.4.7 Expansion Interrupt/Buffer Switch ................................ 6-4
6.4.8 CRT Timing Generation ............................................. 6-5
6.4.9 Pixel Serializer ...................................................... 6-5
6.4.10 Video DACs ......................................................... 6-5
6.5 OPERATION .............................................................. 6-5
6.5.1 VRAM Control ......................................................... 6-5
6.5.1.1 SPEED MODE ..................................................... 6-5
6.5.1.2 PROCESSOR-INITIATED CYCLES ............................. 6-5
6.5.1.3 REFRESH/RAM-TO-SAM TRANSFER CYCLES ................. 6-7
6.5.2 Expansion Bus Interface ............................................ 6-9
### TABLE OF CONTENTS

6.5.2.1 EXPANSION SELECT ........................................... 6-9
6.5.2.2 DATA TRANSCEIVER/LATCH SHARING .......................... 6-9
6.5.3 CRT Timing Logic ........................................... 6-10
6.5.4 Schematics Description ....................................... 6-11

### CHAPTER 7

**MULTIBUS® II AND i860™ MICROPROCESSOR**

7.1 i860™ MICROPROCESSOR CPU BOARD .................................. 7-1
7.2 MULTIBUS® II SYSTEM BUS STANDARD ................................ 7-2
7.2.1 Parallel System Bus (PSB) ...................................... 7-2
7.2.2 Message Passing Coprocessor .................................... 7-3
7.2.2.1 MPC INTERFACE TO PSB .................................... 7-3
7.2.2.2 MPC LOCAL BUS INTERFACE .................................. 7-6
7.2.2.3 MPC DMA INTERFACE ........................................ 7-6
7.3 i860™ MICROPROCESSOR BUS INTERFACE .......................... 7-6
7.4 DRAM SYSTEM ....................................................... 7-7
7.5 LOCAL I/O SYSTEM ................................................ 7-7
7.5.1 82380 Integrated Systems Peripheral .......................... 7-7
7.5.2 iSBX™ Bus Connector ............................................ 7-8
7.6 DMA CONTROL AND THE SRAM MESSAGE SYSTEM ................. 7-8
7.6.1 DMA Channels .................................................. 7-8
7.6.2 SRAM Message Area .............................................. 7-9
7.7 EXPANSION CONNECTOR ............................................. 7-11
7.7.1 Memory Expansion .............................................. 7-11
7.7.2 Intelligent Expansion ............................................ 7-12

### CHAPTER 8

**PHYSICAL DESIGN AND DEBUGGING**

8.1 GENERAL DESIGN GUIDELINES ...................................... 8-1
8.2 POWER DISSIPATION AND DISTRIBUTION ............................ 8-1
8.2.1 Power and Ground Planes ....................................... 8-2
8.3 DECOUPLING CAPACITORS ......................................... 8-4
8.4 HIGH FREQUENCY DESIGN CONSIDERATIONS ....................... 8-7
8.4.1 Transmission Line Effects ...................................... 8-8
8.4.1.1 TRANSMISSION LINE TYPES ................................ 8-9
8.4.1.1.1 Micro Strip Lines ....................................... 8-10
8.4.1.1.2 Strip Lines .............................................. 8-10
8.4.2 Impedance Mismatch ............................................ 8-11
8.4.2.1 IMPEDANCE MATCHING ...................................... 8-14
8.4.2.1.1 Need for Termination .................................... 8-14
8.4.2.1.2 Series Termination ....................................... 8-15
8.4.2.1.3 Parallel Terminated Lines ............................... 8-15
8.4.2.1.4 Thevenin's Equivalent Termination ..................... 8-16
8.4.2.1.5 AC Termination .......................................... 8-17
8.4.2.1.6 Active Termination ...................................... 8-18
TABLE OF CONTENTS

8.4.2.1.7 Impedance Matching Example ........................................ 8-19
8.4.2.2 DAISY CHAINING .......................................................... 8-21
8.4.2.3 90 DEGREE ANGLES ...................................................... 8-21
8.4.2.4 VIAS (FEED-THROUGH CONNECTIONS) ............................... 8-21
8.4.3 Interference ................................................................. 8-21
8.4.3.1 ELECTROMAGNETIC INTERFERENCE (CROSSTALK) ................ 8-22
8.4.3.2 MINIMIZING CROSSTALK .............................................. 8-23
8.4.3.3 ELECTROSTATIC INTERFERENCE .................................... 8-25
8.4.4 Propagation Delay ......................................................... 8-25
8.5 LATCH-UP ............................................................................ 8-26
8.6 CLOCK CONSIDERATIONS .................................................... 8-26
8.6.1 Requirements .................................................................... 8-26
8.6.2 Routing ............................................................................. 8-27
8.7 Thermal Characteristics ....................................................... 8-28
8.8 DERATING CURVE AND ITS EFFECTS ....................................... 8-30
8.9 BUILDING AND DEBUGGING THE i860™ MICROPROCESSOR-BASED SYSTEM .................................................. 8-31
8.9.1 Debugging Features of the i860™ Microprocessor ................. 8-33
8.9.2 Certain Gotchas when Debugging with i860™ Microprocessor .... 8-33
8.9.3 Debugging ....................................................................... 8-34
8.9.4 Simple Diagnostic Programs .............................................. 8-34
8.9.5 Other Simple Diagnostic Software ..................................... 8-35

CHAPTER 9
TESTABILITY
9.1 INTRODUCTION ................................................................. 9-1
9.2 BOUNDARY SCAN MODE ..................................................... 9-1
9.2.1 Shift Mode Operation ...................................................... 9-3
9.2.2 Normal Mode ................................................................. 9-5
9.2.3 A Test Sequence Example ............................................... 9-6
9.3 USING THE TESTABILITY FEATURES IN A SYSTEM .............. 9-7
9.3.1 Component-level Testing .................................................. 9-7
9.3.2 System-level Testing ....................................................... 9-7

APPENDIX A
GRAPHICS FRAME BUFFER SCHEMATICS AND PLD CODE

APPENDIX B
MULTIBUS® II SCHEMATICS

APPENDIX C
MEMORY INTERFACE SCHEMATICS AND PLD CODE
### Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Block Diagram of i860™ Microprocessor</td>
<td>2-2</td>
</tr>
<tr>
<td>2-2</td>
<td>Dual-Instruction Mode</td>
<td>2-5</td>
</tr>
<tr>
<td>2-3</td>
<td>Pipelined Instruction Execution</td>
<td>2-7</td>
</tr>
<tr>
<td>2-4</td>
<td>Dual-operation Data Paths</td>
<td>2-10</td>
</tr>
<tr>
<td>2-5</td>
<td>Paging Algorithm Implementation</td>
<td>2-12</td>
</tr>
<tr>
<td>3-1</td>
<td>Byte Enable Control Signals</td>
<td>3-7</td>
</tr>
<tr>
<td>3-2</td>
<td>Little and Big Endian Data Access</td>
<td>3-9</td>
</tr>
<tr>
<td>3-3</td>
<td>Nonpipelined Bus State Machine</td>
<td>3-11</td>
</tr>
<tr>
<td>3-4</td>
<td>Fastest Read Cycles</td>
<td>3-12</td>
</tr>
<tr>
<td>3-5</td>
<td>Fastest Write Cycles</td>
<td>3-13</td>
</tr>
<tr>
<td>3-6</td>
<td>Fastest Read/Write Cycles</td>
<td>3-14</td>
</tr>
<tr>
<td>3-7</td>
<td>Memory Operation Pipelining</td>
<td>3-15</td>
</tr>
<tr>
<td>3-8</td>
<td>Pipelined Bus State Machine</td>
<td>3-17</td>
</tr>
<tr>
<td>3-9</td>
<td>Pipelined Read Followed by Pipelined Write</td>
<td>3-18</td>
</tr>
<tr>
<td>3-10</td>
<td>Pipelined Write Followed by Pipelined Read</td>
<td>3-20</td>
</tr>
<tr>
<td>3-11</td>
<td>CS8 and RESET Activity</td>
<td>3-21</td>
</tr>
<tr>
<td>3-12</td>
<td>Pipelined Bus State Machine Including Hold State</td>
<td>3-23</td>
</tr>
<tr>
<td>3-13</td>
<td>HOLD, HLDA, and BREQ</td>
<td>3-24</td>
</tr>
<tr>
<td>3-14</td>
<td>Locked Cycles</td>
<td>3-26</td>
</tr>
<tr>
<td>3-15</td>
<td>Boundary Scan Chain</td>
<td>3-29</td>
</tr>
<tr>
<td>3-16</td>
<td>Circuit for Clock, RESET and CS8 Generation</td>
<td>3-31</td>
</tr>
<tr>
<td>4-1</td>
<td>KEN# Timing</td>
<td>4-2</td>
</tr>
<tr>
<td>4-2</td>
<td>Bus Pipelining</td>
<td>4-3</td>
</tr>
<tr>
<td>4-3</td>
<td>Read - Write Timing</td>
<td>4-4</td>
</tr>
<tr>
<td>4-4</td>
<td>Maximum Example Configuration</td>
<td>4-6</td>
</tr>
<tr>
<td>4-5</td>
<td>Address Map</td>
<td>4-8</td>
</tr>
<tr>
<td>4-6</td>
<td>Basic Read Cycles</td>
<td>4-11</td>
</tr>
<tr>
<td>4-7</td>
<td>Pipelined Read Cycles</td>
<td>4-13</td>
</tr>
<tr>
<td>4-8</td>
<td>Basic Write Cycles</td>
<td>4-15</td>
</tr>
<tr>
<td>4-9</td>
<td>Consecutive Write Cycles</td>
<td>4-16</td>
</tr>
<tr>
<td>4-10</td>
<td>Write Followed by Read Cycles</td>
<td>4-17</td>
</tr>
<tr>
<td>4-11</td>
<td>Pipelined Read Followed by Write Cycles</td>
<td>4-18</td>
</tr>
<tr>
<td>4-12</td>
<td>Page Miss Cycles</td>
<td>4-20</td>
</tr>
<tr>
<td>4-13</td>
<td>Parity Block Diagram</td>
<td>4-22</td>
</tr>
<tr>
<td>5-1</td>
<td>i860™ Microprocessor System</td>
<td>5-2</td>
</tr>
<tr>
<td>5-2</td>
<td>I/O Control Logic</td>
<td>5-4</td>
</tr>
<tr>
<td>5-3</td>
<td>Read Cycle Timings</td>
<td>5-6</td>
</tr>
<tr>
<td>5-4</td>
<td>I/O Write Timings</td>
<td>5-7</td>
</tr>
<tr>
<td>5-5</td>
<td>i860™ Processor Interface to 82510</td>
<td>5-10</td>
</tr>
<tr>
<td>5-6</td>
<td>i860™ EPROM Interface</td>
<td>5-13</td>
</tr>
<tr>
<td>5-7</td>
<td>Circuit for Booting from EPROM</td>
<td>5-14</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-1</td>
<td>Block Diagram of the i860™ Microprocessor Based Graphics</td>
</tr>
<tr>
<td></td>
<td>Frame Buffer Board</td>
</tr>
<tr>
<td>6-2</td>
<td>Read/Write/Read Operation (1-Wait-State-Write Mode)</td>
</tr>
<tr>
<td>6-3</td>
<td>Write/Read/Write Operations (1-Wait-State-Write Mode)</td>
</tr>
<tr>
<td>6-4</td>
<td>Refresh/Read/Write Operations (1-Wait-State-Write Mode)</td>
</tr>
<tr>
<td>6-5</td>
<td>Expansion Space</td>
</tr>
<tr>
<td>6-6</td>
<td>Blank and Sync Signals</td>
</tr>
<tr>
<td>6-7</td>
<td>Serial Data Clocking</td>
</tr>
<tr>
<td>6-8</td>
<td>Blank Signals</td>
</tr>
<tr>
<td>7-1</td>
<td>System Block Diagram of i860™ Microprocessor Based</td>
</tr>
<tr>
<td></td>
<td>MULTIBUS® II Board</td>
</tr>
<tr>
<td>7-2</td>
<td>PSB Bus Cycle Timing</td>
</tr>
<tr>
<td>7-3</td>
<td>MPC Signal Groups</td>
</tr>
<tr>
<td>7-4</td>
<td>Fly-by Transfer with SRAM Read and I/O Write</td>
</tr>
<tr>
<td>7-5</td>
<td>SRAM Message Area using 32-bit Bus</td>
</tr>
<tr>
<td>7-6</td>
<td>SRAM Message Area using 64-bit Bus</td>
</tr>
<tr>
<td>7-7</td>
<td>DMA System Arbitration and Control</td>
</tr>
<tr>
<td>8-1</td>
<td>Reduction in Impedance</td>
</tr>
<tr>
<td>8-2</td>
<td>Typical Power and Ground Trace Layout for Double-layer Boards</td>
</tr>
<tr>
<td>8-3</td>
<td>Orthogonal Arrangement</td>
</tr>
<tr>
<td>8-4</td>
<td>Circuit without Decoupling</td>
</tr>
<tr>
<td>8-5</td>
<td>Decoupling Chip Capacitors</td>
</tr>
<tr>
<td>8-6</td>
<td>Decoupling Leaded Capacitors</td>
</tr>
<tr>
<td>8-7</td>
<td>Micro Strip Lines</td>
</tr>
<tr>
<td>8-8</td>
<td>Strip Lines</td>
</tr>
<tr>
<td>8-9</td>
<td>Overshoot and Undershoot Effects</td>
</tr>
<tr>
<td>8-10</td>
<td>Loaded Transmission Line</td>
</tr>
<tr>
<td>8-11</td>
<td>Series Termination</td>
</tr>
<tr>
<td>8-12</td>
<td>Parallel Termination</td>
</tr>
<tr>
<td>8-13</td>
<td>Thevenin’s Equivalent Circuit</td>
</tr>
<tr>
<td>8-14</td>
<td>A.C. Termination</td>
</tr>
<tr>
<td>8-15</td>
<td>Active Termination</td>
</tr>
<tr>
<td>8-16</td>
<td>Impedance Mismatch Example</td>
</tr>
<tr>
<td>8-17</td>
<td>Use of Series Termination to Avoid Impedance Mismatch</td>
</tr>
<tr>
<td>8-18</td>
<td>Daisy Chaining</td>
</tr>
<tr>
<td>8-19</td>
<td>Avoiding 90 Degree Angles</td>
</tr>
<tr>
<td>8-20</td>
<td>Typical Layout</td>
</tr>
<tr>
<td>8-21</td>
<td>Closed Loop Signal Paths are Undesirable</td>
</tr>
<tr>
<td>8-22</td>
<td>Typical Clock Circuit</td>
</tr>
<tr>
<td>8-23</td>
<td>Clock Timings</td>
</tr>
<tr>
<td>8-24</td>
<td>Clock Routing</td>
</tr>
<tr>
<td>8-25</td>
<td>Star Connection</td>
</tr>
<tr>
<td>8-26</td>
<td>Typical Heat Sinks</td>
</tr>
<tr>
<td>8-27</td>
<td>Deraturing Curves for the i860™ Processor</td>
</tr>
<tr>
<td>8-28</td>
<td>Typical i860™ Processor-based System</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

8-29 Simple Diagnostic Program .................................................. 8-34
8-30 Read/Write Diagnostic Program ................................................. 8-35
9-1 Entering and Exiting the Boundary Scan Mode .............................. 9-2
9-2 Order of Boundary Scan Chain .................................................. 9-4
9-3 The Shift Mode of the Boundary Scan Mode .................................. 9-5
9-4 The Normal Mode of the Boundary Scan Mode ............................... 9-6
9-5 A Typical System with Diagnostics Capabilities ............................ 9-7
9-6 A Typical Timing for Serial Scan Mode ........................................ 9-9

Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Cacheability Based on CD, WT and KEN#</td>
<td>2-15</td>
</tr>
<tr>
<td>3-1</td>
<td>Pin Summary</td>
<td>3-3</td>
</tr>
<tr>
<td>3-2</td>
<td>Types of Traps</td>
<td>3-27</td>
</tr>
<tr>
<td>3-3</td>
<td>Test Mode Selection</td>
<td>3-28</td>
</tr>
<tr>
<td>3-4</td>
<td>Test Mode Latches</td>
<td>3-28</td>
</tr>
<tr>
<td>3-5</td>
<td>Output Pin Status During Reset</td>
<td>3-30</td>
</tr>
<tr>
<td>5-1</td>
<td>Valid Addresses for SPACED copying</td>
<td>5-17</td>
</tr>
<tr>
<td>8-1</td>
<td>Comparison of Various Termination Techniques</td>
<td>8-19</td>
</tr>
<tr>
<td>9-1</td>
<td>Test Mode Selection</td>
<td>9-3</td>
</tr>
<tr>
<td>9-2</td>
<td>Test Mode Latches</td>
<td>9-4</td>
</tr>
</tbody>
</table>
CUSTOMER SUPPORT

INTEL'S COMPLETE SUPPORT SOLUTION WORLDWIDE

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, consulting services and network management services. For detailed information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It can start with assistance during your development effort to network management. 100 Intel sales and service offices are located worldwide—in the U.S., Canada, Europe and the Far East. So wherever you're using Intel technology, our professional staff is within close reach.

HARDWARE SUPPORT SERVICES

Intel's hardware maintenance service, starting with complete on-site installation will boost your productivity from the start and keep you running at maximum efficiency. Support for system or board level products can be tailored to match your needs, from complete on-site repair and maintenance support to economical carry-in or mail-in factory service.

Intel can provide support service for not only Intel systems and emulators, but also support for equipment in your development lab or provide service on your product to your end-user/customer.

SOFTWARE SUPPORT SERVICES

Software products are supported by our Technical Information Service (TIPS) that has a special toll free number to provide you with direct, ready information on known, documented problems and deficiencies, as well as work-arounds, patches and other solutions.

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and; COMMENTS Magazine). Basic support consists of updates and the subscription service. Contracts are sold in environments which represent product groupings (e.g., iRMX® environment).

CONSULTING SERVICES

Intel provides field system engineering consulting services for any phase of your development or application effort. You can use our system engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training and customizing an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, BITBUS™ and LAN applications.

NETWORK MANAGEMENT SERVICES

Today's networking products are powerful and extremely flexible. The return they can provide on your investment via increased productivity and reduced costs can be very substantial.

Intel offers complete network support, from definition of your network's physical and functional design, to implementation, installation and maintenance. Whether installing your first network or adding to an existing one, Intel's Networking Specialists can optimize network performance for you.
Introduction to the i860™
64-Bit Microprocessor
CHAPTER 1
INTRODUCTION TO THE i860™ 64-BIT MICROPROCESSOR

This chapter provides an overview of the i860™ microprocessor, its characteristics, possible system configurations and applications.

The i860 microprocessor uses pipelining, parallelism and reduced instruction set computer (RISC) design techniques for high performance. With over one million transistors, it integrates an integer unit, a floating-point unit, a graphics unit, a memory management unit, and separate data and instruction caches.

The i860 microprocessor executes up to two instructions in parallel and makes extensive use of pipelining. High integration and wide data paths eliminate bottlenecks in fetching instructions or data from its on-chip caches.

Designing with the i860 CPU is like designing with any other microprocessor. Its high-performance can be easily delivered using standard interface logic, DRAMs, EPROMs, and I/O devices. An existing 32-bit microprocessor-based system can be easily upgraded with the 64-bit i860 CPU by widening the memory bus.

1.1 PROCESSOR CHARACTERISTICS

The i860 microprocessor’s 1 million transistors offer a high level of integration. A single board design can be completed with four address latches, eight data tranceivers, a clock generator circuit, one EPROM for bootstrapping, a 64-bit wide memory and a few PLDs.

The i860 microprocessor contains the following features:

- Integer processing unit
- Floating-point and graphics unit
- Memory management unit
- 8 Kbyte data cache with 128-bit internal data path
- 4 Kbyte instruction cache with 64-bit internal data path

The i860 microprocessor uses RISC design techniques such as:

- Single clock instruction execution
- Load and store architecture
- 32 general purpose 32-bit registers
- Delayed branching
• Register scoreboarding
• Register bypassing
• Single-cycle loop instruction
• Branch taken/not-taken instruction fetching optimization

The on-chip graphics and floating-point units of the i860 microprocessor support simulation and 3-D displays. The integer instructions can be used to feed data to the graphics and floating-point units. These units are pipelined to produce up to two floating-point results per clock. Some of the features related to the graphics and floating-point unit are:

• Separate adder and multiplier units
• Pipelined/parallel floating-point hardware
• Graphics instructions and hardware optimized for 3-D
• 8-, 16- or 32-bit color or black and white pixel data types
• Single- and double-precision IEEE floating-point standard
• Parallel integer and floating-point operation

The i860 microprocessor's high-performance design uses wide buses and pipelined logic to sustain many parallel operations. The level of performance of the i860 microprocessor is difficult to achieve in multichip systems due to the need for many wide buses. The high-performance design applies to the external bus as well, maximizing the performance that can be obtained from DRAM memory. Some of the performance related features are:

• 64-bit internal and external data bus
• Three operations per clock
• Dual-instruction mode
• 33/40 MHz operating frequency
• 80 million floating-point operations per second peak in single-precision
• 60 million floating-point operations per second peak in double-precision
• 320 Mbyte/sec instruction cache bandwidth at 40 MHz
• 640 Mbyte/sec data cache bandwidth at 40 MHz
• 160 Mbyte/sec external bus bandwidth at 40 MHz
• Fast data movement with 128-bit load and store instructions
• Three pipelined cycles on the external bus
The i860 microprocessor can support virtual memory and multiprocessor systems. All memory references are fully restartable in case of virtual memory faults. Full support is provided for synchronizing operation between multiple CPUs. Features related to virtual memory and multiprocessor support are:

- 32-bit (4 gigabyte) address space with on-chip paging unit
- Support for demand paging
- 4 Kbyte page size
- On-chip 64-entry translation lookaside buffer (TLB)
- User level/supervisor level protection
- Bus locking across multiple instructions
- Cache control
- Trap mechanism for interrupts and faults
- Multiprocessor operating systems

Use of the i860 microprocessor design simplifies system design. It has a conventional microprocessor bus with a standard READY# signal used to allow accesses to slow memories. Software development is assisted by specialized hardware debugging capabilities. Some of the additional implementation features to facilitate hardware design and software development are:

- 1× clock
- Optimizations for use of page mode and static mode dynamic RAMs
- Pin boundary scanning for component or board testing
- CHMOS* IV 1-micron technology, TTL compatible
- 3 watt power dissipation at 40 MHz
- Single 8-bit EPROM can boot system
- On-chip debugging support

1.2 PROCESSOR OVERVIEW

This section provides a quick overview of the i860 microprocessor. The processor architecture will be discussed in greater detail in Chapter 2, and the external interface in Chapter 3.

1.2.1 Pipelining and Parallelism

Pipelining is a technique that divides tasks into a series of smaller subtasks which can be performed quickly and concurrently with one another. By means of pipelining, several data items can be acted on simultaneously. Although several clock cycles are needed to complete an operation, a new result is produced each clock. Pipelining is used throughout the i860 microprocessor to achieve maximum performance. Simple scalar operations can be used to hide details of the pipelining. All scalar floating-point and integer operations are fully interlocked.
Parallelism allows two or more operations to execute simultaneously. The i860 microprocessor supports parallelism, allowing multiplication and addition to execute simultaneously within the floating-point unit. Parallelism also allows the integer and floating-point units to execute simultaneously. The i860 microprocessor can execute an integer instruction and two floating-point operations in the same clock to achieve three operations per clock cycle.

1.2.2 Instruction Set Architecture

The instruction set architecture can implement any portable operating system, language or application. The instruction set includes integer, floating-point, and graphics instructions. All future products of the i860 CPU line will support this architecture.

The core unit executes the basic instruction set including arithmetic, logical, shift and program control instructions. The core unit, floating-point unit and graphics unit are all implemented using RISC principles. All instructions execute in only one clock cycle.

The i860 microprocessor uses a load and store architecture. Only load and store instructions can access memory data. All other instructions use only registers to perform operations. Integer and floating-point instructions use two operand registers and a destination register for the result.

The i860 microprocessor employs delayed branching techniques to avoid interruptions to the flow of data through pipelines. An optimized set of conditional branch and loop instructions minimizes pipeline breaks. Use of a branch-taken or branch-not-taken version of the conditional branch instruction selects whether to pre-fetch the upcoming sequential instruction or the branch target address.

1.2.3 Registers

The i860 microprocessor provides 32 32-bit integer registers for the core unit and another 32 32-bit floating-point registers that are used in the floating-point unit and the graphics unit. The floating-point registers can also be used in pairs as double-precision registers. Quadword memory load instructions load four floating-point registers. Register r0 of the integer registers and f0 and f1 of the floating-point registers return a zero value when read and are treated as a null destination when a value is stored into them.

1.2.4 Address Space

The i860 microprocessor can address up to four gigabytes of memory and memory-mapped I/O locations. Programmers can access memory space as 8-, 16-, 32-, 64- and 128-bit quantities. Operands of 8 and 16 bits are automatically aligned to the low order bits of a 32-bit register on a load. An improperly aligned data access causes a trap.
1.2.5 Floating-Point Operations

The floating-point unit supports the ANSI/IEEE 754-1985 Standard for Binary Floating-point Arithmetic and supports both single- and double-precision operands. Traps detect all floating-point exceptions. Hardware implements all rounding modes in order to support the standard with minimal overhead. The floating-point unit provides a full complement of operations to permit efficient implementation of all low-level and high-level functions defined by the standard.

The parallel floating-point adder unit and multiplication unit have been designed to efficiently perform matrix manipulation, series expansion and signal processing algorithms. The integer unit along with the pipelined and parallel operation of the floating-point unit can emulate the capability of vector processing instructions found in supercomputers, but with added flexibility. The scalar floating-point operations hide all details of the pipeline.

The floating-point load instructions provide 32-, 64- and 128-bit operands for the floating-point unit while it operates in dual-instruction mode. Integer operations can be performed in parallel with all floating point operations.

1.2.6 Graphics Support

The graphics unit supports instructions for 3-D color or black and white algorithms. Pixel formats of 8-, 16- or 32-bit are supported. The i860 microprocessor supports efficient implementation of Phong and Gouraud shading operations. A Z-buffer check instruction is provided to detect the closest surface of a 3-D image. Either 16- or 32-bit Z-buffer can be used. The 3-D graphics unit uses much of the floating-point point hardware.

1.2.7 Caches

Two on-chip caches help to sustain the i860 microprocessor’s high performance. The 8 Kbyte data cache is a two-way set-associative memory with a 32-byte block size and a 128-bit data path. It uses a write-back policy on memory write operations. This technique delays the external write operations needed to maintain consistency between the cache and external memory. With this approach, multiple write operations to the same location, do not result in needless multiple bus operations. When write operations to external memory are performed, two 128-bit wide write buffers are used which post the write operations and delay them until the memory subsystem is not in use.

The 4 Kbyte instruction cache is also implemented as a two-way set associative memory. It provides a 64-bit wide internal datapath. The instruction cache is read-only; writes to memory do not update the code cache. A separate flush instruction is available to invalidate the contents of code cache, if the need arises.

External caches are supported via cache control pins, cache control bits in page table entries, and a two clock bus. Multiprocessing systems can be built using external caches and preventing shared data from being cached on-chip.
Since the data and instruction caches map virtual addresses to data, the data and instruction caches can operate in parallel with the translation lookaside buffer (TLB). Data can be obtained in one clock cycle when there is a hit in the cache.

### 1.2.8 Paging Unit

The on-chip paging unit converts 32-bit virtual memory addresses to 32-bit physical addresses. Each page is 4 Kbytes in size. Supervisor and user level read and write memory protection is provided on a per page basis. Supervisor pages can be write-protected to perform copy-on-write operations for supervisor data.

A translation lookaside buffer (TLB) acts as a 64-entry cache for the virtual memory tables. These tables map virtual page addresses to physical page addresses and provide protection rights. The TLB makes memory management more efficient by operating in parallel with the data and instruction caches. TLB cache misses and updates are handled automatically in hardware.

### 1.2.9 Debugging Support

The i860 microprocessor provides a debug hardware support trap which can be activated when reading and/or writing at an address stored in the data breakpoint register. The address can refer to data inside the caches or off-chip.

### 1.2.10 External Interface

This section outlines the functions provided by the i860 microprocessor’s external interface. These functions are detailed in Chapter 3.

The external interface pin-out consists of a 29-bit address bus, an 8-bit byte enable control bus, a 64-bit data bus, 19 control signals and 48 power and ground pins. The external bus is timed relative to a clock. All outputs are valid before the end of a clock period. All inputs are synchronous to the clock.

#### 1.2.10.1 TWO-LEVEL BUS PIPELINING OF LOCAL BUS

The i860 microprocessor permits up to two levels of pipelining in external memory operations, providing throughput beyond the cycle or access time of the components used. A two level configuration allows three operations to occur simultaneously and triples memory throughput. While the total cycle time is six clocks, 64 bits of data are transferred every two clocks. Pipelining permits a high-performance memory system while using low-cost DRAMs. Even with a single bank of DRAMs, pipelining allows overlapping of accesses to the same DRAM page.

#### 1.2.10.2 BUS ARBITRATION SUPPORT

The i860 microprocessor provides three signals to control bus and control line arbitration: the input line HOLD (bus hold request) and the output lines HLDA (hold acknowledge) and BREQ (bus request). HOLD and HOLDA provide a handshake with
other processors or an arbitration circuit to allow several processors to share the external buses. The processor can operate out of its internal cache for periods of time and uses the BREQ signal to indicate that it is waiting to use the external bus.

1.2.10.3 DRAM INTERFACE SUPPORT (PAGE-MODE AND STATIC COLUMN)

The i860 microprocessor provides support for page mode and static column DRAMs. These provide faster memory cycles when sequential reads or writes take place from the same row address in a DRAM component. This occurs often, because cache updates typically involve four 64-bit sequential read cycles and the write buffers often group together four sequential write cycles.

The i860 microprocessor provides a next near signal (NENE#) to reduce the amount of external circuitry needed to support page mode and static column DRAMs. The signal is asserted when successive current cycles are in the same DRAM page. The size of the DRAM can be programmed.

1.2.10.4 CACHE CONTROL

The i860 microprocessor supports memory mapped I/O, external caches and multiple CPUs. All of these require correct use of caching. The cache enable input signal (KEN#) and page table bit output signal (PTB) control and monitor the data and instruction caches. The KEN# signal input is used by external logic to tell the processor when read data should not be cached. This prevents the processor from caching shared memory in a multiprocessor system and alleviates inconsistencies when two or more processors are accessing the same area of memory. All read operations from memory mapped I/O locations must deassert KEN#.

The page table bit (PTB) output signal operates in two modes. In one mode it indicates whether the software has disabled updates to the data or code cache during the current read cycle. In its other mode of operation it indicates whether the software has disabled the use of an external cache for the current cycle. This kind of software control of caching is done on a page by page basis, only when paging operation is enabled. The software controls the use of the cache in order to guarantee the consistency of shared data in a multiprocessor environment.

1.2.10.5 LOCKED MEMORY CYCLES

The external LOCK# pin indicates that the i860 microprocessor is performing a set of memory cycles that should not be interrupted. The memory subsystem should be designed so that the memory subsystem cannot be accessed by other processors while this pin is active. Multiple instructions can be executed while the lock signal is active. All semaphore operations, like compare and swap, can be implemented.
1.2.10.6 EIGHT-BIT BUS ACCESS FOR BOOTSTRAPPING OPERATION (CS8)

The INT/CS8 input pin can establish 8-bit code accesses upon hardware reset. This feature allows bootstrapping from an 8-bit external EPROM or ROM device and reduces the number of parts needed to complete a board. Only code accesses are affected in this mode. This mode only works immediately after activation of RESET. A control register allows software to disable this mode, but not to reenable it.

The entire code and initial data of an application or bootup code can be placed into one EPROM and copied to DRAM for execution.

1.2.10.7 BOUNDARY SCAN TO SIMPLIFY BOARD LEVEL TESTING OR BOOTUP CODE

The i860 microprocessor allows all the input pins to be serially sampled. Likewise, the state of all the output pins can assume a value set by serial input data. This precludes the need for complex programs that are normally needed to manipulate pins and permits the testing of board logic circuitry with the i860 microprocessor installed.

1.2.11 Clock Requirements

The i860 microprocessor uses an external clock that runs at 33.3 or 40 MHz. This clock synchronizes the internal functional blocks of the processor, and synchronizes the external signals. Most logic connected to the i860 CPU will also use this clock.

1.2.12 i860™ Microprocessor Packaging and Power Requirements

The i860 microprocessor is available in a 168-pin pin-grid array (PGA) package with 120 signal pins and 48 power and ground pins. All the power and ground pins must be connected. Low-inductance bypass capacitors should be used around the i860 CPU to handle current surges when all the address and data lines change state simultaneously. Direct current power dissipation when running at 33 MHz is two watts at normal operation and three watts at peak operation. A heat sink may be used to keep the case temperature within specification depending on airflow.

1.3 SYSTEM CONFIGURATION

The i860 microprocessor is suitable as a central processor in any high performance microprocessor application. It will usually be configured as a stand-alone processor with private memory and a memory mapped I/O subsystem. The i860 uses standard DRAMs and an EPROM for operation. A small system consists of i860 CPU, clock generator, 4 address latches, 8 data transceivers, 1 EPROM, 64-bit memory of DRAM/SRAM, several PLDs and I/O devices.

Other configurations employ the i860 microprocessor as a dedicated application processor. In this case, a communications bus will exist between the i860 and the host. This bus should allow quick data transfers.
The processor may also be used in a multiprocessor system. The i860 CPU may work in a loosely-coupled fashion, communicating through shared memory or through a link of independent memories. The LOCK# signal must be used to guarantee atomic multiple-cycle memory accesses.

The i860 microprocessor provides several functions for multiprocessing support. Bus granting logic (HOLD, HOLDA, BREQ) eases the interface with other processors and DMAs. The cache enable/disable logic is used to maintain consistency between internal caches and shared memory. The capability for locked external cycles allow implementation of semaphores for use with shared memory.

1.3.1 Private and Shared Memory Configuration

Peak performance for a multiprocessor i860 system requires minimal contention for the use of memory. Typically, this is achieved by providing each i860 microprocessor with cache memory, its own private memory, memory that can be shared, or some combination.

Multiprocessor systems not requiring peak performance can use shared memory only. This results in a simpler and lower-cost implementation. The memory subsystem can prioritize access to the shared memory between the processors to maximize efficiency.

1.4 APPLICATION OVERVIEW

The i860 microprocessor is designed for use in a wide range of applications. The processor's support of demand-paged virtual memory and the IEEE Floating Point Standard makes it especially suitable as a main processing engine for high-performance engineering workstations, mainframe computers and supercomputers. Its high integration allows a desktop form factor for capabilities previously associated with supercomputers and high-end graphics workstations.

The i860 power can support computation intensive applications such as electronics or mechanical system simulations. The power of the hardware 3-D graphics support graphics applications. High-performance 3-D graphics can now become a standard feature of any workstation.

The i860 microprocessor eliminates the need for special-purpose signal, graphics or floating-point coprocessors. The processor can also be used for high-performance embedded controller applications, or as an applications accelerator for existing systems.
Internal Architecture
CHAPTER 2
INTERNAL ARCHITECTURE

The i860 microprocessor architecture obtains its performance by a combination of large data paths, high bandwidth data and instruction access (caches and registers), a large number of important functions included on-chip, and high levels of pipelining and parallelism. This architecture allows up to three operations per clock to be executed.

The programmer controls the parallelism to manage data flow to and from the floating-point unit. The on-chip caches provide storage for instructions and data. Wide buses can transfer an instruction pair and two double-precision floating-point operands each clock. A programmer can use the data cache as a large bank of vector floating-point registers.

The i860 microprocessor architecture consists of nine units:
- Core execution unit
- Floating-point control unit
- Floating-point adder unit
- Floating-point multiplier unit
- Paging unit
- Data cache unit
- Instruction cache unit
- Bus and cache control unit
- Graphics unit

The arrangement of these nine units is shown in Figure 2-1. This chapter describes how these nine units interact to interpret the instructions.

2.1 CORE EXECUTION UNIT

The core execution unit is the center of intelligence for the i860 microprocessor and is responsible for its overall operation. It fetches both integer and floating-point instructions. It decodes and executes integer, logical, control-transfer, load/store, exception handling, and cache flushing instructions. It can perform loads and stores to and from the integer register file and the floating-point register file. It also includes a special pixel store instruction that facilitates implementation of the Z-buffer hidden-surface elimination algorithm.

2.1.1 Core Unit Registers

The core execution unit includes a register file containing thirty-two 32-bit integer registers, a 32-bit ALU, a barrel shifter, two 32-bit processor status registers, a data breakpoint register, a fault instruction register and control logic.

The integer registers, labeled r0 through r31, are accessible by arithmetic operations and load/store instructions. These registers are used for address computation and scalar integer computations. All the registers can be read and written except r0, which always
reads a value of zero. Writes to r0 do not store a value in r0. r0 works in combination with a number of instructions to modify and extend their function. For example, to check if a register contains a zero, the or instruction can be used with the other source operand and destination as r0. The register bank remains unmodified as a result, and the condition code CC indicates if the given register contains a zero.

The processor status register (PSR) and extended processor status register (EPSR) are 32-bit read/write registers which contain various information on the status of the current process. They provide information such as condition code bit status, loop condition code, interrupt control and status, trap flags, data breakpoint control, pixel information, processor identification, etc. Refer to the i860™ 64-Bit Microprocessor Programmer's Reference Manual, order number 240329, for more information.
The extended processor status register (epsr) is a 32-bit read/write register which contains additional state information beyond what is contained in the psr. This information includes the processor type (value of one for the i860 microprocessor), step number to distinguish among different revisions, data cache size field, and five flags: the overflow flag, big-endian mode bit, page table bit mode, write-protect mode bit, and the interlock bit. Refer to the i860™ 64-Bit Microprocessor Programmer’s Reference Manual for more information.

The data breakpoint register (db) contains a breakpoint address. It is used to generate traps when loads or stores are made from or to this address, and is thus useful for debugging. Refer to the i860™ 64-Bit Microprocessor Programmer’s Reference Manual for details.

The core execution unit contains logic for handling exceptions and external interrupts. When an exception condition or an external interrupt occurs, the processor transfers control to the trap handler. Again, refer to i860™ 64-Bit Microprocessor Programmer’s Reference Manual for details.

2.1.2 Instruction Execution

Instructions are fetched into the core execution unit from the instruction cache. If this address location is not in the cache (a cache miss), the instruction is fed to the core execution unit from the external memory, while the corresponding Instruction Cache block is simultaneously filled.

2.1.2.1 RISC OPTIMIZATIONS

The core execution unit is designed according to RISC principles, as explained in Chapter 1. It uses a pipelined organization that maximizes performance. The instructions are made purposefully simple using a load/store architecture. Emphasis is placed on minimizing circuit delays and economizing chip space in order to include the other processing units that are essential to overall high performance—the floating-point unit, graphics unit, paging unit, caches, and register banks. Pipelining is coupled with register bypassing, scoreboarding, and delayed branching to further enhance performance. Some integer operations are performed by the floating-point unit. Integer multiply and divide are implemented via a code sequence which use floating-point instructions.

Execution pipelining is transparent for arithmetic, logic and shift instructions. Core instructions appear to operate in one clock cycle with the destination register already loaded by the time the next instruction begins executing. However, this is not actually the case. Due to the delay required in storing to and reading from a register, the processor detects if the last instruction’s destination is used as an operand in the current instruction. If it is, the operand is returned to the ALU at the same time the register file is updated. This technique, known as register bypassing, is invisible to the programmer.

Unlike arithmetic and logic operations, load operations require a minimum of two clock cycles to provide a valid result for the destination register. Because load instructions require a minimum of two cycles, the integer core uses scoreboard to detect if the register operand of the current instruction is the destination of a preceding load.
The use of register bypassing and scoreboarding allow load and store instructions to be 
executed at an effective rate of 1 instruction per clock cycle, assuming the data and 
instructions are found in their respective caches. When a cache miss occurs, the hard-
ware will automatically resolve potential problems by freezing execution if the data is 
needed.

Branch instructions can also have the effect of locking the pipeline for one or more clock 
cycles. To avoid this waste, the i860 microprocessor uses delayed branching. The branch is 
delayed in the sense that the i860 microprocessor executes one additional instruction 
following the control-transfer instruction before actually transferring control. During the 
time used to execute the additional instruction, the i860 microprocessor refills the in-
struction pipeline by fetching instructions from the new instruction address. This avoids 
breaks in the instruction execution pipeline.

By using the above techniques, it is possible to execute core unit instructions at the rate 
of one per clock quite consistently, thus providing a rate of 40 MIPs (40 MHz clock) of 
native integer operation performance.

2.1.2.2 DUAL-INSTRUCTION MODE

The i860 microprocessor provides a form of parallelism in the ability to execute a core 
instruction and a floating-point instruction simultaneously. This parallel instruction exe-
cution is referred to as dual-instruction mode. When executing in this mode, the instruc-
tion sequence consists of 64-bit aligned instruction pairs with a floating-point instruction 
in the lower 32 bits and a core instruction in the upper 32 bits.

Enabling and disabling dual- and single-instruction mode is controlled by software. The 
\texttt{d.fp-op} in Figure 2-2 indicates the instructions responsible for enabling the dual-
instruction mode. As shown in the figure, there is a one-instruction delay between the 
instruction that does the enabling or disabling and the instruction which performs the 
operation.

Note that when a 64-bit dual-instruction pair directly follows a delayed branch instruc-
tion in dual-instruction mode, both 32-bit instructions are executed.

Further details regarding the use of dual-mode instructions are provided in the \textit{i860™} 
64-Bit Microprocessor Programmer’s Reference Manual, order number 240329.

2.2 FLOATING-POINT UNIT

In addition to the core unit, which executes integer instructions, is the floating-point 
unit, which processes floating-point instructions. The pipelined floating-point unit, along 
with the on-chip cache, enables the 40 MHz i860 microprocessor to achieve a peak 
execution rate of up to 80 MFLOPs for single-precision and up to 60 MFLOPs for 
double-precision operations.
The floating-point unit consists of the floating-point register bank, floating-point adder, floating-point multiplier, floating-point status register and floating-point control unit. The floating-point unit has two types of operations: scalar and pipelined. Scalar operations are like those used in most computers. Pipelined operations deliver the highest performance but require more programming expertise. Intel offers a library of common math functions already implemented using pipelined code.

Floating-point data types, floating-point instructions, and exception handling all support the IEEE standard for binary floating-point arithmetic (ANSI/IEEE Std 754-1985) for both single- and double-precision data types. The floating-point status register holds information about the result of the operation. A complete set of traps includes tests for invalid source operands such as NaN (Not a Number), denormalized numbers, and infinities, as well as tests for errors in the result, such as overflow and underflow. The cause of the traps can be determined by examining the value in the floating-point status register. The floating-point traps permit implementation of the IEEE Standard in a very efficient manner. For more information, see the i860™ 64-Bit Microprocessor Programmer's Reference Manual.

Due to the low-level instruction set philosophy of the i860 microprocessor architecture, high-level functions defined by the Standard, such as square root, sine, and cosine are...
not implemented directly by the hardware. The facilities of the floating-point unit, how­
ever, allow for a very efficient implementation of these functions that actually out­
performs dedicated floating-point coprocessors.

Intel offers an IEEE trap handler program, as well as a software library, that provides
i860 microprocessor programs with the full set of functions supported by the IEEE
standard.

2.2.1 Floating-Point Register Bank

The floating-point unit is provided with its own register bank. It contains 32 floating­
point registers, each 32-bits wide, labeled f0 through f31. The registers can also be
accessed in pairs for 64-bit double-precision values or 64-bit integer values. For this
purpose, only even registers are used, e.g. f2, f4, etc. Load and store instructions also
support the transfer of 128-bits worth of data (e.g. two double-precision operands). The
registers are used in groups of four for this purpose, e.g. f4, f8, f12, etc. Registers f0 and
f1 are special in that, when read, they always provide a value of zero, and writing into
them has no effect. These registers modify and extend the function of the floating-point
instructions. Two null registers are required in order to provide a zero operand and a
null destination when using double-precision operations.

The floating-point register bank (refer to Figure 2-1) allows multiple operations to occur
in parallel. It contains two read ports, one write port, and two bidirectional ports. All
these ports are 64-bits wide and can be used concurrently.

The two 64-bit source operands provided by the floating-point registers are used as data
input to the floating-point multiplier unit (FPMU), the floating-point adder unit (FPAU)
or the graphics unit. A 64-bit input port to the floating-point registers transfers
the result of the operations. The 64-bit integer instructions and graphics instructions also
use this register bank for their source and destination operands.

Two 64-bit bidirectional ports between the data cache and the floating-point register
bank allow transfers of up to 128 bits. A 64-bit bus can connect either of these two buses
to the data bus on the bus cache control unit. This bus allows 64-bit transfers to and
from external memory. The transfers are performed by the various floating-point load
and store instructions. These transfers are controlled by the core unit and can occur in
parallel with the floating-point instructions, as explained in Section 2.1.2.2, “Dual­
Instruction Mode”.

2.2.2 Pipelined and Scalar Operations

The floating-point unit uses parallelism to increase the rate of operations performed by
the unit. One type of parallelism used in the floating-point unit is known as “pipelining”.
A pipelined architecture treats each operation as a series of more primitive operations
called stages. These can be executed in parallel. Consider the floating-point adder unit
as an example. Let a represent the operation of the adder, and let the stages be repre­
sented by A[1], A[2], and A[3]. The stages are designed such that the A[i + 1] stage for one
add instruction can execute in parallel with the A[i] stage for the next add instruction.
Since each A[i] stage can perform its task in a single clock, and three instructions can be
in executing in parallel, one add operation per clock is achieved.
Pipelining within the floating-point multiplier unit can be described similarly, except that it requires two clocks per stage on double-precision operations. The resulting status bits in the fsr reflect the result of the last completed operation within the pipeline. Pipelined instruction execution is shown in Figure 2-3.

The functions performed by each stage of the pipeline are not documented. Programs should not rely on specific actions, only that the pipeline is of fixed length.

![Figure 2-3. Pipelined Instruction Execution](image)
In addition to pipelined execution, the i860 microprocessor can also execute floating-point operations in scalar mode. In this mode, the floating-point unit does not initiate a new operation until the previous floating-point operation is completed, so that the scalar operation passes through all the stages of its pipeline before a new operation is started. Scalar mode is used when the next operation depends upon the result of the previous floating-point operations, or when the compiler or assembly language programmer wishes to avoid the added complexity of pipelining. Integer operations can be performed in parallel to scalar floating-point operations.

### 2.2.3 Floating-Point Adder Unit

The floating-point adder unit of the i860 microprocessor supports both double- and single-precision IEEE 754 format and operates in two modes: scalar mode and pipelined mode. In scalar mode, three clocks are required to complete an add, subtract or compare operation. In pipelined mode, one result per clock for either a single- or double-precision operation is obtained.

The adder unit supports the following precision combinations between inputs and results: single to single, double to double, and single to double. For this reason, the adder is also used to perform data precision conversions. Some of the instructions executed exclusively by the adder unit are:

- Floating-point add (fadd)
- Floating-point subtract (fsub)
- Pipelined floating-point comparisons: (pfgt.p, pfeq.p)

### 2.2.4 Floating-Point Multiplier Unit

The floating-point multiplier unit performs floating-point multiplication in accordance with the IEEE standard. It is organized as a three-stage pipeline. In pipelined mode, the multiplication throughput is one single-precision instruction per clock and one double-precision instruction each two clocks.

The multiplier unit also supports a reciprocal instruction which is used to implement division and square-root operations by means of an iterative process. A small macro (or function) can be developed based on these instructions to perform the full division or square root.

Certain instructions allow the multiplier unit to operate in parallel with the adder unit in a variety of flexible ways, thereby doubling the number of operations per clock.

### 2.2.5 Dual Operation Feature

Dual Operation is a special feature of the i860 microprocessor which allows the floating-point adder and multiplier unit to work in parallel, thus doubling the number of floating-point operations performed. Both add-and-multiply and subtract-and-multiply operations are supported.
The instruction formats for add-and-multiply and subtract-and-multiply allow specification of only two source operands and one destination. However, when operating the adder and multiplier in parallel, two pairs of operands and two destinations are needed for the general case. To overcome this limitation, the adder and multiplier can be configured in a variety of ways that are especially suitable for problems such as:

- matrix manipulation (e.g., solving linear equations)
- series calculations (e.g., sine function calculation)
- signal processing applications (e.g., Fast Fourier Transform)
- graphics (e.g., coordinate transformations)

For this purpose, three special registers are used—KR, KI, and T. Both KR and KI can be used to hold constants or temporary values. These values can be loaded when used as operand inputs to the multiplier, and can later supply the value, without the need for an explicit instruction operand. T can act as a transfer register to hold the value of the result of a multiplication, which can be passed on as an operand to the adder on a later instruction.

The data paths available are shown in Figure 2-4. Possible configurations can be selected as follows:

- Operand 1 of the multiplier can be KR, KI, or scr1.
- Operand 2 of the multiplier can be scr2 or the last stage of the adder pipeline.
- Operand 1 of the adder can be scr1, the T-register, or the last stage result of the adder pipeline.
- Operand 2 of the adder can be scr2, the last-stage result of the multiplier pipeline, or the last-stage result of the adder pipeline.

In addition to the selection of operands, the instruction can choose whether to load KI, KR, or T as part of its operation. The possible operand data path selections and loading options allow a large number of possible combinations. Many of these combinations are functionally redundant or of no interest. Each instruction format for add-and-multiply and subtract-and-multiply supports 16 different instructions, and each of these instructions provides a different configuration of operand data path and KI, KR, or T loading selection. The configurations have been specially selected to streamline the implementation of the applications previously mentioned. Refer to the *i860™ 64-Bit Microprocessor Programmer's Reference Manual* for further details.

### 2.2.6 Floating-Point Computation Throughput

The combination of the dual-instruction mode feature with pipelined dual operation allows the i860 microprocessor to achieve a sustained 80 MFLOPs in single-precision and 60 MFLOPs in double-precision for inner loops of common computations. Assuming the code is in the cache, no visible memory cycles are needed to fetch the instructions.

The dual-instruction mode allows the loading and storing of operands and the updating of array indices and loop control information to be performed in parallel with floating-point execution. A load or a store (core unit instruction) can transfer up to 4 single-precision operands or 2 double-precision operands, assuming these operands are adjacent to each other in memory within some data array. Loads and stores take one
Figure 2-4. Dual-operation Data Paths

clock cycle if the data is in the cache, and two clock cycles if fetched from external memory. If the operands are from memory, instructions can continue to be executed in the pipeline as long as they don’t access the registers being loaded. Thus, indexing, loop-control, and operand loading can typically take place in parallel with the floating-point computation, maintaining the sustained rate.

In pipelined mode, two single-precision floating-point operations can be executed per clock cycle resulting in a rate of \((2 \text{ operations/clock}) \times (40 \text{ MHz/sec}) = 80 \text{ MFLOPs}.\) For double-precision, addition requires one clock cycle, while multiplication requires two. For algorithms that require two floating-point additions and one multiplication for each iteration, two adds and one multiply can be done in parallel in two clock cycles. This results in three operations in two clock cycles, or \((3 \text{-operations/2-cycles}) \times (40 \text{ MHz}) = 60 \text{ MFLOPs}.\) Algorithms requiring a double-precision multiply and add for every iteration execute at two operations per two clock cycles (due to the multiply two-clock rate), resulting in an execution rate of 40 MFLOPs.

2.3 PAGING UNIT

The paging unit provides the i860 microprocessor with the capability of supporting an efficient implementation of demand-paged virtual memory. Demand-paged virtual memory allows programs to use a larger, virtual memory space, which is actually supported by
a smaller real (or physical) memory space. The paging unit, in combination with the appropriate memory management software, automatically allocates physical pages of memory to virtual page addresses as they are needed (on demand). Typically, when all of physical memory is used up, physical memory is swapped out to disk, and pages are reallocated.

The paging unit provides the ability to translate virtual addresses used by the processor to physical addresses that correspond to locations in the external memory. It also provides page-level protection based on access rights, as well as two levels of privilege—user and supervisor.

Address translation and memory protection are optional. They are enabled by the address translation enable (ATE) bit in the directory base register. If this bit is not set, the physical address is the same as the virtual address, and no translation or access-rights checking is performed. The ATE bit is cleared upon reset. The i860 microprocessor paging unit functions the same and uses the same page table entry formats as the Intel 386™ and i486™ microprocessors.

2.3.1 Paging Algorithm

A virtual address is mapped to a physical address according to a set of tables called page tables. The address is divided into a page address and an offset. A page is a collection of data that occupies the space of a page frame in main memory, or some location in secondary storage when there is insufficient space in main memory. A page frame consists of 4K bytes of contiguous physical memory starting on a 4K-byte boundary. The upper 20 bits of the 32-bit address of a page frame is referred to as the page address. In the i860 microprocessor, both virtual and physical addresses are 32 bits wide. They both consist of a 20-bit page address and a 12-bit offset. The concatenation of the two provides a complete 32-bit byte address.

The address translation algorithm uses two levels of page tables. The two levels are referred to as the page directory and the page tables. Both levels of page tables are a page (4096 bytes) in size, consisting of 1024 32-bit entries.

The directory base register, dirbase, contains a 20-bit field which points to the page directory. Only one page directory table is active at any given time. The entries of the page directory contain the physical addresses of all the page tables used for the mapping process, or contain entries indicating that the given page tables (corresponding to virtual segments of address space) are not present in physical memory. The page tables themselves contain physical page addresses for all the valid virtual pages, or entries indicating that the given virtual memory address is not present in physical memory.

The algorithm mapping virtual memory to physical memory is fully implemented by the hardware and is depicted in Figure 2-5. The most-significant 10 bits of the virtual address are used as an index into the page directory, which selects a specific page table. The next 10 bits are used as an index into the selected page table, selecting a page frame address. The last 12 bits act as an offset into the page frame address, building up a full 32-bit physical byte address. This completes the virtual to physical address conversion.
If during a memory transfer, the page directory or page table indicates that a selected page table or page frame is not present (by means of a zero in the Present bit of the table entry), a trap occurs, allowing the software to validate the page by reading it from secondary storage. The page table entries also provide the write, user, cache disable, accessed and dirty bits, as well as 3 user-definable bits. These bits along with the write protect bit in the extended processor status register are used to provide page-level protection rights, page cacheability information, and information needed to implement an efficient replacement algorithm for swapping out page frames when main memory is full. More details are provided in the *i860™ 64-Bit Microprocessor Programmer’s Reference Manual*, order number 240329.

To avoid accessing the page directory and page table for every address translation, the i860 microprocessor implements an on-chip translation look-aside buffer (TLB), which is a cache that directly translates a virtual page address to a physical address, and provides the additional bits from the page tables needed to provide protection and information for replacement algorithms. TLB translation requires one clock cycle and is typically invisible because of the processor’s pipelining.

The TLB is implemented as a 4-way, set-associative cache, mapping a total of 64 page table entries. Because each page table entry maps 4 Kbytes of address space, a total of $4K \times 64$ or 256 Kbytes of memory are mapped at any one time by the TLB. When there

![Figure 2-5. Paging Algorithm Implementation](image-url)
is a miss in the TLB and the page tables in memory are used, an entry in the TLB is automatically replaced by the new mapping.

**2.4 ON-CHIP CACHES AND BUS CONTROL**

The i860 microprocessor contains both an instruction and data cache. Being integrated so close to the processor, and having been carefully designed for minimum delay, cache storage can operate much faster than external memory. In addition, having both caches on-chip allows them to operate in parallel. Thus, the processor can simultaneously read instructions from the instruction cache, read or write data to and from the data cache, as well as translate virtual addresses through the TLB. The caches also provide wide data paths—the instruction cache is 64-bits wide, and the data cache is 128-bits wide. Also, on-chip caches reduce the need for external caches, which reduces the total system cost and makes available valuable PC-board real estate.

Both the data and instruction caches are virtually addressed. This not only provides for faster operation, but allows the TLB to perform its virtual address to physical address translation in parallel with the operation of the cache. If one of the caches determines that it does not have the contents of the required virtual address (a cache miss), the TLB will be ready with a physical address with which to begin an external memory cycle.

Both the instruction and data caches are implemented as a 2-way set associative memory that maps a virtual address to a 32-byte block of data. These blocks correspond to 32 consecutive bytes loaded from an address having zero for the least-significant 5 bits. When transferring data to or from the cache, the processor uses the desired set of bytes from this 32-byte group. Allocation and replacement for both caches is always performed using blocks of 32 bytes.

The i860 microprocessor uses a wraparound technique which makes the process of filling a cache block more efficient. Since a block consists of four 64-bit (8 bytes) entries, the processor will first read the 64-bit entry that contains the data item, instruction, or instruction pair that is needed by the processor. The processor continues processing while the entry read is simultaneously stored in the cache. Next, the processor sequentially reads the remaining three, 64-bit entries of the block and stores them in the cache. Since the first entry read may lie in the middle of the block, the processor wraps around to read the first 64-bit entry of the block after the last one is read. In this manner, the cache block is loaded and the processor gets its data as quickly as possible.

The use of the caches when accessing memory can be bypassed by means of the CD (Cache Disable) bit in the page tables or the KEN# (Cache Enable) pin, as explained in Chapter 3. This is required for special cases such as I/O references or shared data in a multiprocessor system.

**2.4.1 Instruction Cache Unit**

The instruction cache size is 4 Kbytes. With a 64-bit wide data path, the cache can provide two 32-bit instructions in each fetch cycle: two core instructions, or one core instruction, and one floating-point instruction. Thus, the transfer rate of the cache is 64 bits/clock (320MB/sec at 40 MHz).
The instruction cache is intended to be used to map read-only memory (i.e., it does not support self-modifying code). System programs that modify code should invalidate the instruction cache via the IT! bit in the directory base register.

2.4.2 Data Cache Unit

The data cache size is 8 Kbytes. Using a 128-bit wide bus for reading data, it can transfer up to 128 bits/clock. Thus with a 40 MHz clock, 640 Mbytes/sec can be transferred. Unlike the instruction cache, the data cache supports both read and write operations (corresponding to load and store instructions).

The data cache with its 128-bit internal bus can supply up to two 64-bit operands to the floating-point unit per cycle. Alternatively, it can supply one 32-bit operand per load cycle, or a 16- or 8-bit, right-aligned, signed-extended value to the core unit.

The data read must be aligned in memory according to its size. The table below shows the restriction on the addresses for different size data items:

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Number of least-significant bits in byte address that are 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit value</td>
<td>1</td>
</tr>
<tr>
<td>32-bit value</td>
<td>2</td>
</tr>
<tr>
<td>64-bit value</td>
<td>3</td>
</tr>
<tr>
<td>128-bit value</td>
<td>4</td>
</tr>
</tbody>
</table>

The data cache can be used as a large set of floating-point registers for vector operations. The ICS field of the extended processor status register allows a program to determine the cache size and use it appropriately.

2.4.2.1 WRITE OPERATIONS AND THE DATA CACHE

Writes to memory locations not present in the cache are sent directly to the memory write buffers and do not affect the cache. A write operation to a location that is already in the cache is written to the cache, but is not immediately written to memory. In this scheme, known as write-back, the blocks that have been written to the cache but not to memory are marked as “dirty”. When the replacement algorithm chooses to replace a block containing a dirty block, or when a cache block is flushed, these dirty blocks are written to memory.

The write-back scheme provides better performance than the write-through approach, in which data written to the cache is immediately written to memory. This is because accesses and stores to variables or indices that are in the data cache require no external memory cycles and hence reduce bus traffic.

When the data cache writes a block to memory, it uses two 128-bit wide write buffers. These buffers delay the actual memory writes until an opportune time, if possible, as explained in Section 2.4.6.
2.4.3 Bypassing Instruction and Data Caches

There are two pins on the i860 microprocessor that relate to cache enabling during any external memory cycle. They are the Cache Enable input pin (KEN#) and the Page Table Bit (PTB) output pin. They are used for controlling access to shared memory, I/O buffers, and memory-mapped I/O devices.

When the i860 microprocessor detects that the KEN# input is not asserted during a bus cycle, the processor inhibits use of the data cache and instruction cache for this cycle.

When paging is enabled, the CD (cache disable) bit of the secondary page table used during a memory transfer determines whether or not to enable the use of the instruction or data caches. A similar bit, WT (write through) is also available that disables data caching. The value of the OR of these bits is reflected on the PTE output pin, depending on the PBM bit in the extended processor status register. When paging is disabled, the PTB pin remains not asserted.

KEN# is internally NORed with CD OR WT bits to determine whether or not to enable use of the cache, as shown in Table 2-1.

<table>
<thead>
<tr>
<th>CD OR WT</th>
<th>KEN#</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Cacheable access</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Noncacheable access</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Noncacheable page</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Noncacheable page</td>
</tr>
</tbody>
</table>

2.4.4 Flushing Instruction Cache, Data Cache, and TLB

Setting the ITI (Instruction TLB Invalidate) bit in the dirbase register, invalidates the contents of the instruction cache, as well as the Translation Lookaside Buffer.

The data cache is flushed by software using the cache flush instruction. This instruction flushes one cache block at a time. A loop of code is executed to clear the entire data cache. For a copy of this code, see the i860™ 64-Bit Microprocessor Programmer's Reference Manual.

2.4.5 Bus and Cache Control Unit

The bus and cache control unit interfaces to the external bus, performing instruction and data accesses for the execution core unit. The control unit transfers data to and from the external code memory, and controls TLB translation, including normal translation, miss replacement and fault processing. It receives cycle requests and specifications from the execution core unit. It performs instruction or data cache accesses and handles data or instruction cache miss processing (cache block replacement). Its pipelined structure supports up to three outstanding bus cycles. The three-level bus cycle pipelining is explained in Chapter 3.

The bus and cache control unit can fetch one, 64-bit instruction from the instruction cache and 128-bits worth of data from the data cache on every clock cycle, as long as the accessed data resides in the cache.
The bus control unit also performs the physical address comparison for the generation of the Next Near (NENE#) signal. The NENE# signal is asserted by the i860 microprocessor if the currently issued address falls on the same DRAM page as the previously issued address. The NENE# pin allows the external memory system to take advantage of the static column and page-mode DRAMs. The size of the DRAM page is programmable by three bits in the DIRBASE register.

The i860 microprocessor can operate in two instruction fetch modes: normal or CS8 (code size 8) mode. When the CS8 bit in the DIRBASE register is set, external memory cycles are processed as 8-bit cycles. When this bit is clear, instruction cache misses are processed as 64-bit bus cycles. This bit cannot be set by software. To enter the CS8 mode, the INT signal is asserted prior to the falling edge of the RESET signal. This allows the instruction bytes to be fetched on the eight least-significant bits of the external data bus. Data may be transferred as 64, 32, 16, or 8 bits values using all 64 bits of the data bus.

The CS8 mode allows the i860 microprocessor processor to be bootstrapped from an 8-bit EPROM. In the CS8 mode, the signals BE2, BE1 and BE0 are redefined to correspond to the three least-significant bits of the address so that a complete byte address is available (i.e., 32 address pins can be used).

Once the bootstrap code has been loaded into the 64-bit memory, a 64-bit fetch can be initiated. This is accomplished by clearing the CS8 bit in the dirbase register via software (one time only). Once this bit is disabled, it can not be enabled until a new hardware reset occurs.

### 2.4.6 Write Buffers

The bus and cache control unit also supports the use of two 128-bit write buffers. These buffers are designed to delay any write operations to memory until memory is not being used (i.e., instructions and data are being read from the caches). This optimizing delay is not always possible, because the memory operation in question may itself be read. The bus control logic forces memory write operations to insure proper functionality.

The two 128-bit write buffers can operate on independent memory cycles. When write operations of 128 bits are performed, each write buffer is written in two memory cycles. When writes are made that are smaller than 128 bits (64, 32, 16, or 8 bits), the write buffer is written in a single cycle. Proper alignment and the selection of the correct byte enables is made for cycles smaller than 64 bits.

### 2.5 GRAPHICS UNIT

The graphics unit executes instructions designed to support high-performance 3-D graphics applications. Support for packed pixels of 8-, 16-, and 32-bit data are supported. The precise pixel formats are given in the i860™ 64-Bit Microprocessor Programmer's Reference Manual, order number 240329.
The graphics unit executes simple but powerful instructions that can be applied to the following graphics functions:

- Hidden surface elimination
- Distance interpolation
- 3-D shading using intensity interpolation

Based on these instructions, the i860 microprocessor processor can perform real-time, shaded graphics without the need for an external graphics processor.

The instructions operate properly for the various pixel formats. With the 8-bit format the operations only affect the intensity. For 16- and 32-bit color pixels, the operations affect the isolated color vectors, e.g., red, green, or blue. Operations are performed on 64-bit entities, which can contain the values of multiple pixels in parallel. A special Pixel Store instruction implemented by the core unit can work in parallel with the graphics instructions.

The interpolation operations of the processor support graphics applications in which a set of points on the surface of a solid object is represented by polygons. The distance and color intensities of the vertices of the polygons are known, but the distance and intensities of other points must be calculated by interpolation between these points. Graphics instructions, just as floating-point instructions, can be used in dual-instruction mode to achieve greater computation rates.
Local Bus Interface
CHAPTER 3
LOCAL BUS INTERFACE

The local bus is designed to provide high data throughput among the processor, memory and I/O subsystems. It provides a flexible interface that is suitable for a wide variety of system environments.

This chapter describes the local bus interface, its basic function, operation, and timing for related signals.

3.1 i860™ MICROPROCESSOR EXTERNAL INTERFACE AND BUS SIGNALS

The external interface of the i860™ microprocessor consists of a 64-bit data bus, 29-bit address bus, eight-bit byte-enable control bus, 19 status and control signals, and 48 power and ground pins. This section provides an overview of the services provided by the external interface of the i860 microprocessor.

3.1.1 i860™ Microprocessor Buses

The i860 microprocessor communicates with external memory and I/O through a synchronous bus interface that includes a separate data and address bus as follows:

D63–D0 These 64 pins make up the bidirectional data bus external interface. Either 8, 16, 32, or 64 bits of data can be transferred during a bus cycle.

A31–A3 The address bus consists of 29 address pins which address one of $2^{29}$ 64-bit memory locations.

BE7#–BE0# The byte-enable bus consists of eight pins that specify which bytes to access within a 64-bit location. These pins are used to enable writing in one, two, four or eight-bytes of the double-word involved in the current write cycle. Read operations should always return 64-bits of data. See Section 3.3.1.3 for details. BE2#, BE1#, BE0# are used as address bits A2, A1, A0 respectively while in CS8 mode. See Section 3.3.5 for details.

3.1.2 i860™ Microprocessor Output Signals

The i860 microprocessor output signals provide control and status information. The output signals are as follows:

ADS# The i860 microprocessor asserts the address status signal to indicate the beginning of a bus cycle. It identifies the clock period during which it provides a valid address and the other signals required to perform a memory cycle.
The write/read# signal indicates whether the current cycle is a write (high state) to or read (low state) from the memory or I/O subsystem.

The lock signal is generated by the processor to indicate locked cycles to external circuitry (Section 3.4.3 provides further explanation).

The next near signal tells the memory subsystem that a cycle is on the same DRAM row as a previous cycle. This allows the memory subsystem to use page mode or static-column mode features of DRAMs (Section 3.4.1 provides further explanation).

The page table bit signal reflects either the value of the cache disable (CD) bit, or the write through (WT) bit of page table entry during the current cycle. The PBM (page-table bit mode) bit of the EPSR indicates which. If PBM is clear, PTB reflects CD; otherwise, it reflects WT (Section 3.5 provides further explanation).

The hold acknowledge signal indicates that the bus has been released (Section 3.4.2 provides further explanation).

The bus request signal is asserted when an internal bus request is pending. This signal is used to assist external bus arbitration. Its value is independent of the state of HOLD and HOLDA (Section 3.4.2 provides further explanation).

BREQ is also used as serial output for the boundary scan chain while in boundary scan mode (Section 3.7 provides further explanation).

### 3.1.3 i860™ Microprocessor Input Signals

Input signals control various i860 microprocessor actions:

**CLK**

The clock input provides basic timing information for the processor to synchronize internal and external operations. All other signals are sampled relative to the rising edge of CLK. The internal operating frequency is the same as the clock frequency.

**READY#**

The ready signal indicates to the processor that a bus cycle is finished. For read cycles, the READY# signal indicates that data being read is valid and that the processor can latch the contents of the data bus. For write cycles, it indicates that the data being output to the data bus is being latched by the memory subsystem and is no longer needed to finish the bus cycle. READY# must be synchronous to CLK; it is sampled on every clock after the clock which follows the sampling of ADS#.

**NA#**

The next address signal allows external data transfers to request pipelining. The signal indicates to the processor that the memory or I/O subsystem is ready to receive a new address and begin a pipelined cycle. NA# is sampled during the second clock after ADS# (Section 3.3.4 provides further explanation).
INT/CS8

The interrupt and code size signal serves two functions. When the RESET signal is asserted, the CS8 signal can be used to set the code size eight mode to indicate whether the bus performs instruction fetches on the low-order byte of the bus instead of the 64 bit wide bus. This feature allows booting from a single EPROM. Section 3.3.5 provides further details. At all other times, this pin serves as the INT signal and functions as the i860 microprocessor's maskable external interrupt (Section 3.6 provides further explanation). The state of the INT input is sampled on every clock.

KEN#

The cache enable signal enables updates to the processor's instruction and data caches. When paging is enabled, this signal works in combination with the WT, CD, and PTB bits of the current bus cycle. KEN# is sampled on every bus cycle (Section 3.5 provides details).

HOLD

The bus hold signal floats all output signals except HOLDA and BREQ and causes the processor to relinquish control of the bus. The HLDA signal indicates that the bus has been granted. Instruction execution continues unless required instructions and data cannot be read from the on-chip cache (Section 3.4.2. provides further explanation). The state of this pin is sampled every clock.

SHI

The boundary scan shift input signal is used to read boundary scan chain serial data when in boundary scan mode (Section 3.7 provides further explanation).

BSCN

The boundary scan enable signal enables boundary scan mode for board or component testing (Section 3.7 provides further explanation).

SCAN

The shift scan is used in conjunction with boundary scan mode to set normal mode (when SCAN is deasserted) or shift mode (when SCAN is asserted) (Section 3.7 provides further explanation).

CC1, CC0

These pins are reserved by Intel and must be strapped low.

The i860 microprocessor bus interface pins are summarized in Table 3-1.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Function</th>
<th>Active State</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>CLocK</td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>RESET</td>
<td>System reset</td>
<td>High</td>
<td>I</td>
</tr>
<tr>
<td>HOLD</td>
<td>Bus hold</td>
<td>High</td>
<td>I</td>
</tr>
<tr>
<td>HLDA</td>
<td>Bus hold acknowledge</td>
<td>High</td>
<td>O</td>
</tr>
<tr>
<td>BREQ</td>
<td>Bus request</td>
<td>High</td>
<td>O</td>
</tr>
<tr>
<td>INT/CS8</td>
<td>Interrupt, code-size</td>
<td>High</td>
<td>I</td>
</tr>
</tbody>
</table>

Table 3-1. Pin Summary
Table 3-1. Pin Summary (continued)

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Function</th>
<th>Active State</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A31–A3</td>
<td>Address bus</td>
<td>High</td>
<td>0</td>
</tr>
<tr>
<td>BE7#–BEO#</td>
<td>Byte Enables</td>
<td>Low</td>
<td>0</td>
</tr>
<tr>
<td>D63–D0</td>
<td>Data bus</td>
<td>High</td>
<td>I/O</td>
</tr>
<tr>
<td>LOCK#</td>
<td>Bus Lock</td>
<td>Low</td>
<td>0</td>
</tr>
<tr>
<td>W/R#</td>
<td>Write/Read bus cycle</td>
<td>Hi/Low</td>
<td>0</td>
</tr>
<tr>
<td>NENE#</td>
<td>NExt NEar</td>
<td>Low</td>
<td>0</td>
</tr>
<tr>
<td>NA#</td>
<td>Next Address request</td>
<td>Low</td>
<td>I</td>
</tr>
<tr>
<td>READY#</td>
<td>Transfer Acknowledge</td>
<td>Low</td>
<td>I</td>
</tr>
<tr>
<td>ADS#</td>
<td>ADdress Status</td>
<td>Low</td>
<td>0</td>
</tr>
</tbody>
</table>

Cache Interface Pins

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Active State</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEN #</td>
<td>Cache ENable</td>
<td>Low</td>
<td>I</td>
</tr>
<tr>
<td>PTB</td>
<td>Page Table Bit</td>
<td>High</td>
<td>0</td>
</tr>
</tbody>
</table>

Testability Pins

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Active State</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHI</td>
<td>Boundary Scan Shift Input</td>
<td>High</td>
<td>I</td>
</tr>
<tr>
<td>BSCN</td>
<td>Boundary Scan Enable</td>
<td>High</td>
<td>I</td>
</tr>
<tr>
<td>SCAN</td>
<td>Shift Scan Path</td>
<td>High</td>
<td>I</td>
</tr>
</tbody>
</table>

Intel-Reserved Configuration Pins

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Active State</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1–CC0</td>
<td>Configuration</td>
<td>High</td>
<td>I</td>
</tr>
</tbody>
</table>

Power and Ground Pins

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>System power</td>
</tr>
<tr>
<td>VSS</td>
<td>System ground</td>
</tr>
</tbody>
</table>

A # after a pin name indicates that the signal is active when at the low voltage level.

3.1.4 Power and Ground Pins

The i860 microprocessor has 24 ground pins and 24 power pins. The i860™ 64-Bit Microprocessor Data Sheet provides pin number assignments, detailed electrical characteristics, and decoupling requirements.

3.2 BUS CHARACTERISTICS

The fully-synchronous local bus provides 64-bit data transfers to and from memory or I/O devices. Minimum read and write cycles can be done in two clock cycles. The bus is capable of pipelining bus cycles two levels deep (three stages). I/O is memory mapped. An external address decoder can map address ranges to correspond with the I/O subsystem and the memory subsystem. Also, memory-mapped devices should drive KEN# high during reads to prevent data caching.

To simplify explanation, the term memory subsystem refers to the I/O subsystem and the memory subsystem.
3.3 BUS TRANSFER OPERATIONS

This section discusses all bus transfer operations including data alignment issues, pipelined and nonpipelined bus transfers, and 8-bit mode operation for bootstrapping.

3.3.1 64-bit Bus and Byte Alignment of Data

The i860 microprocessor performs external data transfers by means of its 64-bit data bus. This section discusses the details of how the bus control unit performs bus operations on data of various sizes.

3.3.1.1 MEMORY ADDRESSABILITY AND ALIGNMENT REQUIREMENTS

Memory is addressable down to each byte within a paged virtual address space of $2^{32}$ bytes. The i860 microprocessor instructions can operate on data of various sizes including bytes (8-bytes), half-words (16-bits), words (32-bits), double-words (64-bits) and quadwords (128-bits).

Data may be located anywhere within the byte-addressable space. However accesses to data not following the alignment requirements given below cause a trap. Load or store operations to unaligned data must be handled by a software routine and are very inefficient. The alignment requirements for data are as follows:

- 128-bit values are aligned on 16-byte boundaries when referenced in memory (the four least-significant address bits must be zero).
- 64-bit values are aligned on 8-byte boundaries when referenced in memory (i.e. the two least-significant address bits must be zero).
- 32-bit values are aligned on 4-byte boundaries when referenced in memory (i.e. the two least-significant address bits must be zero).
- 16-bit values are aligned on 2-byte boundaries when referenced in memory (the least-significant address bit must be zero).

Misaligned instructions are not allowed. Instruction alignment requirements are:

- All instructions are 32-bits long and must be aligned on 4-byte boundaries (i.e. the two least-significant address bits must be zero).
- Dual mode instruction pairs must be aligned on 8-byte boundaries with the floating-point instruction first.

3.3.1.2 DATA ALIGNMENT DURING READ OPERATIONS

The i860 microprocessor performs aligned read operations in the following manner. 64- and 128-bit transfers are handled as one and two 64-bit memory transfers, respectively.

8-, 16-, and 32-bit memory read operations are accomplished by first reading 64-bits of data and extracting the data bytes needed. The 64-bit data is typically stored in the cache in parallel with the extraction of the data. The data is extracted by shifting the 64-bit value so that the least-significant byte of the data item is aligned with bits D7–D0. For 8- and 16-bit integer data, the value is sign-extended to 32 bits, and the value is loaded into the appropriate register.
During read cycles, the byte enable signals BE0#–BE7# reflect the bytes on the data bus that are involved in the read operation. During read operations, however, the byte enable signals are not used to enable memory for specific bytes. Instead, all 8 bytes of the data bus must be read by the processor, if caching is enabled, to properly update the corresponding 64-bit entry in the cache. This type of operation is illustrated by the simplified circuit diagram in Figure 3-1.

3.3.1.3 DATA ALIGNMENT DURING WRITE OPERATIONS

Write operations require the reverse aligning process of a read operation. 64- and 128-bit write operations are handled as one and two 64-bit data transfers, respectively. 8-, 16- and 32-bit memory write operations are performed by properly aligning the data to be written onto the data bus and writing only on the bytes involved within the 64-bit word addressed.

The byte enable lines, BE0#–BE7# are used to determine which bytes of the addressed 64-bit word to write to.

For 8-bit, 16-bit and 32-bit data, the data bus outputs the register data shifted to the left by an appropriate number of bytes. The shift aligns the least-significant byte of the data with the least-significant byte of the destination in memory.

The properly-aligned data bytes in the data bus are written on the bytes of the 64-bit word addressed, as selected by the active BE7#–BE0# lines. The circuit shown in Figure 3-1 illustrates how the byte enable signals operate during a write operation.

The number of bytes by which to left shift the data and the set of byte enable signals that are activated is determined by the least three significant bits of the byte address, the size of the data and the endian mode used, as discussed in Section 3.3.1.4. The 64-bit word involved is selected by the upper 29 bits of the physical address.

3.3.1.4 LITTLE AND BIG ENDIAN MODES AND BUS OPERATION

The i860 microprocessor can store data in memory in one of two formats, little endian or big endian. In little endian, all multibyte data items are stored so that the least significant byte is at the smallest address of the bytes allocated for the data item. In big endian format all multibyte data items are stored so that the most significant byte is at the smallest address. The i860 microprocessor can operate in either mode.

In the i860 microprocessor, multiple-byte data values are normally stored in little endian format (with the least significant byte at the lowest memory address). The processor, however also provides the capability of operating in big endian mode (with the most significant byte is at the lowest address).
Figure 3-1. Byte Enable Control Signals
The following examples illustrates the use of the two modes of operation. Suppose a C program has the following global declaration for a structure:

```c
struct {
    byte A, B;  /* byte */
    short C;   /* 16-bit integer */
    long D;    /* 32-bit integer */
} rec;
```

The fields of rec are addressed as follows: A at rec + 0, B at rec + 1, C at rec + 2, D at rec + 4. If rec is double word aligned, the rec data structure for big endian mode will have the following form in memory, within the double word address corresponding to the address of rec:

```
0 2 3 4 5 6 7
A B C:msb C:lsb D:msb D:2msb D:2lsb D:1lsb
```

and in little endian the following form:

```
7 6 5 4 3 2 1 0
D:msb D:2msb D:2lsb D:1lsb C:msb C:lsb B A
```

In the above example msb stands for most-significant byte, lsb for least significant byte, 2msb for second most significant byte, and 2lsb for second least significant byte.

In both cases the addressability of each data item is the same, but the order of the byte addresses for the multibyte data item is different.

The bytes are organized as follows:

<table>
<thead>
<tr>
<th>Item</th>
<th>Address</th>
<th>Little Endian</th>
<th>Big Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>3,2</td>
<td>2,3</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>7,6,5,4</td>
<td>4,5,6,7</td>
</tr>
</tbody>
</table>

The default mode of operation for the i860 microprocessor is little endian. As an option that may be dynamically selected in supervisor mode, the i860 can operate in big endian mode. Figure 3-2 illustrates how the i860 can operate in big endian mode. Figure 3-2 illustrates how the i860 handles little endian and big endian operations. When loading a register from memory, the i860 internally byte shifts the incoming data bus to the right as required to align the least significant byte (right side) of the addressed data with the least significant byte of the register.

When storing register data in memory, the i860 shifts left the register data by the appropriate number of bytes required to align the data with the memory bytes addressed. This data is presented on the bus, and an external write cycle is performed. The bus enable lines that correspond to the data bytes being written are made active.
Figure 3-2. Little and Big Endian Data Access

The determination of how many bytes to shift right during the bus operation, or left during a store operation, as well as the selection of the appropriate byte enable signals is determined by the least-significant 3 bits of the memory address involved, by the size of the data, and by the endian-mode selected.

The big-endian little-endian example given earlier and Figure 3-2 illustrate the operation under both endian-modes. The reading or writing the A data byte in little endian shifts the data by zero bytes (no shift). For a write operation, activation of byte enable signal BE0#, stores the memory byte corresponding to bits d7–d0 of the 64-bit memory entry addressed. In big endian, reading A shifts the data bus right by seven bytes, and writing shifts the register data left by seven bytes and activates BE7#, to store it in the byte corresponding to bits d63–d56.

Big endian mode in the i860 essentially inverts the byte offset addresses, converting address 7 into 0, 0 into 7, 6 into 1, 1 into 6 and so on.
3.3.1.5 ENDIAN MODE FOR CODE ACCESSES

Code accesses are always done with little endian addressing. This implies that code will appear differently than documented here when accessed as big endian data. Intel recommends that disassemblers running in a big endian system convert instructions which have been read as data back to little endian form and present them in the format documented here.

3.3.1.6 SYSTEM OPERATION AND ENDIAN MODE

Systems based on the i860 microprocessor can be designed for normal operation in little endian or big endian mode. The natural endian-mode operation is established by the way that external devices address the byte offsets within the 64-bit aligned memory locations. Typically, the system will be used only in the natural mode of operation. The i860, however, allows for dynamically changing the endian-mode by software executing in supervisor mode.

3.3.2 Basic Bus Operation

The i860 microprocessor’s fully-synchronous external bus may operate without pipelining or with up to two levels of pipelining to boost memory subsystem throughput. All control signals that affect bus operations are sampled relative to the rising edge of the clock.

A bus cycle begins when ADS# is sampled active and ends when READY# is sampled active. READY# is sampled on every cycle after ADS# is sampled active. New bus cycles can be started on any clock cycle after a one clock cycle delay following the beginning of the prior bus cycle. Thus, new cycles can start as often as every other cycle. Pipelining allows up to three outstanding cycles to exist concurrently. A bus cycle is considered outstanding while its associated READY# has not been sampled active.

The processor can generate pipelined and nonpipelined read and write bus cycles, as requested by the memory subsystem. A pipelined cycle starts while one or two other bus cycles are outstanding. Pipelined cycles are started under control of the NA# signal as explained in Section 3.3.4.

3.3.3 Nonpipelined Bus Operations

Bus cycles require at least two clock cycles to complete. The state diagram in Figure 3-3 illustrates how the bus operates in nonpipelined mode.

The state machine assumes the idle T1 state when there are no processor requests for external bus cycles (indicated by REQ in Figure 3-4). When the processor requests a bus cycle, the bus controller transitions to the T1 state and the ADS# signal is asserted. ADS# can be sampled by the memory subsystem at the end of the T1 clock. ADS# assertion indicates the beginning of a bus cycle. The T1 clock is always followed by the T11 clock during which the bus cycle is allowed to complete. The address bus (A32–A3) and the signals W/R#, NENE# and PTB are all made valid and stable prior to the end
of the first $T_{11}$ clock cycle. During read operations, the data bus (D63-D0) is floated to allow the memory subsystem to drive the data. During write operations, the processor drives the data bus and makes it valid and stable before the end of the first $T_{11}$ clock cycle.

The memory subsystem does not assert the READY# signal unless the read and write cycles finish within the first $T_{11}$ clock. The diagram in Figure 3-3 shows that the state machine repeats the $T_{11}$ state to add wait-states as long as NA# is not asserted. Wait-states can be added as needed by leaving READY# unasserted. The signals involved in the memory cycle remain valid during wait-states. Assertion of NA# allows pipelined cycles to take place (Section 3.3.4. provides further explanation). When the processor samples the READY# signal, indicating completion of a bus cycles, the state machine transitions to the $T_1$ state, if there is no new processor bus requests at the time. If new requests are present, the state machine assumes the $T_1$ state and a new cycle begins.

### 3.3.3.1 NONPIPELINED READ OPERATIONS

Figure 3-4 shows that read operations can complete at end of the first $T_{11}$ state to produce two-clock read cycle. To achieve this level of efficiency, the memory subsystem must provide read data, assert READY# and allow for processor sampling before the read cycle completes. This calls for an extremely fast address to data access time (the i860 microprocessor Data Sheet provides detailed timing information). Required memory access times can be relaxed by adding wait-states or by using pipelining. Each wait-state eases the access time requirement by one clock period.

### 3.3.3.2 NONPIPELINED WRITE CYCLES

Figure 3-5 shows a timing diagram of back-to-back write cycles. To perform a nonpipelined write cycle, the processor asserts the ADS# signal during $T_1$ and drives the signals needed to perform the bus cycle. These include not only the same signals needed for a
read, such as the address bus, W/R#, etc; but also the write data on the data bus, and the byte enable lines BE7#–BE0#. Write operations differ from read operations in that the processor does not need to wait for memory to finish its cycle in order to continue computing. The memory subsystem can merely latch the address bus, data bus, byte enable lines and other bus related signals, while simultaneously asserting READY#, allowing the processor to continue computing. The memory subsystem can perform the write operation while the processor is starting a new cycle. Thus, in Figure 3-5, cycle 1 can physically write to memory while T1, and T11 of cycle 2 sets up the next write cycle. In this way, cycle 1 has two full clock cycles to operate and there is no need for wait-states. This situation is different for read cycles, where the required RAM access time needed to perform a two-clock nonpipelined operations is very small.

Unlike cachable read operations, write operations make use of the processor-driven byte enable signals BE7#–BE0# driven by the i860 microprocessor (Section 3.3.1.3 provides further explanation).
3.3.3.3 WRITE CYCLES FOLLOWING READ CYCLES

The timing diagram in Figure 3-6 shows that even though the data is not guaranteed valid until the later part of the $T_{11}$ cycle, the processor may begin driving the data bus early in the $T_1$ cycle. If a read cycle precedes a write cycle, there could be contention for the data bus between read data being held by the memory subsystem past the beginning of the write’s $T_1$ cycle, and write data being driven early in $T_1$ by the processor. As shown in Figure 3-6, the processor avoids this problem by delaying a write operation by one clock cycle when it follows a read operation. The processor does not start driving the write data for the write cycle (cycle 2 in Figure 3-6) until the beginning of $T_{11}$, as opposed to the write cycle in Figure 3-5, which starts driving the data as early as $T_1$.

Memory subsystem design must add a wait-state to accommodate the special handling of write cycles that follow read cycles. Although it is possible for the sake of simplicity to add a wait-state to all write operations, this is undesirable due to the degradation in performance.
3.3.4 Pipelined Operations

The i860 microprocessor provides up to two levels of pipelining for 64-bit read and write operations. Two levels of pipelining implies the presence of three outstanding cycles, one level implies two outstanding cycles, and no pipelining implies no more than one outstanding cycle.

Pipelining of the external bus is controlled by the next address signal, NA#. By use of this signal the memory subsystem allows, zero, one, or two levels of pipelining. After a clock during which the processor has asserted ADS# to start a memory cycle, the memory subsystem can assert NA# to indicate that even though the outstanding cycle is not finished, the processor can, if it needs to, start a new bus cycle. The NA# signal needs to be asserted for only one clock cycle, since it is latched internally. Once an asserted ADS# is latched, a new level of pipelining is permitted even if the processor’s bus request comes at a later time.

An m-n read or write cycle has a cycle time of m clocks and a cycle-to-cycle time of n clocks (m ≥ n). Total cycle time is calculated from the clock in which ADS# is asserted to the clock in which READY# becomes active. Cycle-to-cycle time is calculated from the time that READY# is sampled active for the previous cycle to the time that it is sampled active for the current cycle.
Pipelining may begin whenever a bus cycle in progress requires more than two clock cycles to finish \( (m > 2) \). A new cycle may begin while another cycle is still in progress, if assertion of NA# requests it. NA# is only recognized in a clock where ADS# is inactive.

Figure 3-7 shows how pipelining takes advantage of memory operations with four interleaved memory banks. Pipelined memory reads achieve 6-2 cycles. That is, a total cycle time of six clocks and a cycle to cycle throughput of two clocks. The “A” in the diagram indicates when the address is valid. Total access time in this example is six clock cycles.

### 3.3.4.1 PIPELINING AND INTERLEAVED MEMORY BANKS

Memory subsystems will typically use as many interleaved memory banks as there are stages in the pipeline. Two levels of pipelining are most effective when a memory subsystem with four interleaved banks. (Using three banks is not feasible because of the difficulty in making an address decoder of this kind.) Interleaved memory banks are designed so that each supports one of several sequential 64-bit addresses. In a two bank system, for example, one memory bank handles odd 64-bit addresses while the other handles the even addresses. In a four bank system, each bank handles one of four consecutive 64-bit addresses. When memory address cycles require a bank currently in use by an outstanding bus cycle, the memory subsystem adds wait-states while the specific outstanding cycle completes. It then resumes with pipelined operations.

In another approach to the use of bus pipelining, the system initiates one more memory operation than the interleaved memory banks can accommodate. When the extra memory cycle is started, the memory subsystem stops issuing cycles until the accessed memory

![Figure 3-7. Memory Operation Pipelining](image-url)
bank becomes free. The memory subsystem has all the information needed to start a new cycle. When the new cycle starts, the memory system queues up the next cycle. This eliminates the extra delay required to get a valid address for any queued bus cycle.

3.3.4.2 ORDERING OF DATA DURING PIPELINED OPERATIONS

In typical implementations, the memory subsystem indicates completion of pipelined write operations by asserting READY# when the processor provides valid data for writing. When pipelined bus operations are performed, the ordering of data driven onto the bus during read or write cycles must correspond to the order in which the outstanding cycles were initiated. If two read operations and a write operation are outstanding, the processor will not provide the write data until the outstanding read operations are completed. The memory subsystem must have a state machine that tracks when to drive the read data, and when to latch the write data.

3.3.4.3 BUS STATE MACHINE FOR PIPELINING

Operation of pipelined bus cycles is illustrated by the bus state machine shown in Figure 3-8. Transitions are made on every clock cycle according to the state of the signals provided with the transition paths.

The state machine is divided into sections that indicate zero, one, two and three outstanding cycles. Sections that indicate zero and one outstanding cycle are small extensions of the nonpipelined bus state machine shown in Figure 3-3.

Two three-state machines have been added to the nonpipelined operation of Figure 3-3. They indicate state machines for two and three outstanding pipelined cycles. The states are labeled with subscripts in the form of $T_j$ or $T_{ji}$. The $j$ subscript indicates the number of outstanding cycles, and the $i$ subscript indicates substates 1 and 2.

There are three processor states for each of the $j$-states. The states labeled with a single subscript $T_j$ correspond to the initial state entered to begin a new cycle, given $j - 1$ or no outstanding cycles. A $T_j$ state is entered only once for each external bus cycle and asserts the ADS# signal. A $T_j$ state for $j > 1$ is not entered unless there is a request from the processor and the NA# has been or is in the process of being asserted.

Notes:
- RDY# in the figure corresponds to the READY# signal
- NA# is not sampled during ADS# active clock
- ADS# is made active in $T_1$, $T_2$ and $T_3$
- REQ corresponds to an internal processor bus request

$T_{j1}$ is an auxiliary state which helps to complete memory cycles. $T_{j1}$ states operate as the $T_{11}$ state does for in nonpipelined cycles. During the first $T_{j1}$ cycle in a given operation, signals that are relevant to the operation (such as address bus, W/R# and NENE#) are made valid and kept valid during subsequent $T_{j1}$ wait-states. A $T_{j1}$ state can be repeated on subsequent clock cycles as wait-states are introduced (that is, READY# remains unasserted).
If the processor is not requesting a new cycle, $T_{j2}$ can be entered from $T_{j1}$ when synchronous sampling detects an asserted NA# while a memory cycle is still active. A cycle request is indicated by the REQ internal signal. The $T_{j2}$ state performs like the $T_{j1}$ state but remembers that an NA# signal has been asserted and that a further level of pipelining can be introduced at the processor's request.

As mentioned, pipelined cycles occur when the memory subsystem asserts NA# while the CPU is processing an outstanding cycle. (The memory subsystem should not assert NA# when there are no outstanding cycles.) The NA# signal tells the processor that the
memory subsystem is prepared to receive another cycle while the previous cycle completes. In Figure 3-9, for example, NA# is asserted by the memory subsystem before the end of the T1 cycle while the READY# signal remains unasserted. In this case, the memory subsystem has stored all the information it needs to complete the outstanding cycles. As soon as the processor is ready to request a new memory cycle (as indicated by the REQ signal) and starting on the clock following the sampling of NA#, the processor provides a new ADS# which starts the memory cycle.

3.3.4.4 PIPELINED READ AND WRITE CYCLES

Figures 3-9 and 3-10 illustrate pipelined memory cycles. Figure 3-9 shows four memory cycles. Nonpipelined cycles begin when there are no outstanding cycles. The digits in parentheses (such as 5-2 for cycle 2 in Figure 4-5) characterize the number of clocks needed to perform a pipelined cycle. The first digit indicates the total number of clocks between cycle initiation (ADS# assertion) and completion (READY# assertion). The second digit indicates the throughput rate (the number of clocks between READY# signals).

Figure 3-9 illustrates the sequence of events in a group of pipelined cycles. The first operation is the cycle 1 memory read which begins when T1 ends while asserting ADS#. At the completion of the T11 state, address lines A31–A3, W/R#, NENE# and PTB# are valid and latched by the memory subsystem. During this T11 clock cycle, the memory subsystem asserts NA# to request a pipelined cycle. The state machine transitions to T2 initializing the read cycle, cycle 2, by the assertion of ADS#. During T21, the memory subsystem latches the various signals needed to perform the cycle, and asserts NA# to start another pipelined cycle. As a result, the processor requests another cycle. The state

![Diagram](image-url)

**Figure 3-9. Pipelined Read Followed by Pipelined Write**
machine transitions to $T_3$ and a third cycle begins by another assertion of ADS#. During the same $T_3$ clock, READY# is asserted indicating that cycle 1 has completed and that the data bus has valid data. The state machine transition from $T_3$ to $T_{21}$ should be understood to gain insight as to how the pipelined bus state machine operates. Cycle 3 starts with a $T_3$ state and ends with a $T_{21}$ state (instead of $T_{31}$), since the sampling of an active READY# signal reduces the number of outstanding cycles back to two. Cycle 3 is a write cycle, so during $T_{21}$, all the previously mentioned bus cycle related signals in addition to the byte enable lines (BE7#–BE0#) and the data bus are made valid and latched by the memory subsystem. During $T_{21}$ a new NA# is asserted, allowing the state machine to start cycle 4 and transition into $T_3$ again. Another READY# is asserted and cycle 2 completes, leaving cycles 3 and 4 outstanding. For the remaining cycles, the state machine oscillates between $T_3$ and $T_{31}$ thus maintaining two levels of pipelining (three outstanding bus cycles).

The processor does not drive cycle-3 data onto the Data Bus until two READY# assertions indicate completion of pipelined cycles 1 and 2. Note that since the cycle-3 write operation follows a read operation (cycle 2), the processor waits an additional clock cycle before driving cycle-3 data. The memory subsystem must detect the read/write sequence and delay by one clock cycle the assertion of READY# and the sampling of write data.

Figure 3-10 shows a nonpipelined write operation followed by a pipelined write and two pipelined reads. The write operations are not preceded by a read operation, and the memory subsystem is not required to add a wait-state to complete the cycle.

### 3.3.5 8-Bit Bus Transfers for Bootstrapping (CS8 Mode)

Eight-bit code size mode is enabled when the INT/CS8 signal is sampled active at the beginning of the period in which RESET is deasserted (Figure 3-11). In eight-bit code size mode, instruction cache misses are transferred as single byte reads instead of 8-byte reads (using bits D7–D0 of the data bus). This allows the i860 microprocessor to be bootstrapped with an 8-bit EPROM. For these single byte code reads, byte enables BE2#–BE0# are redefined to be the three low order bits of the address bus, A2–A0. While KEN# is asserted, these code byte reads are used to update the contents of the instruction cache. (Section 3.5 explains the function of the KEN# signal). Pipelined memory cycles are not started in this mode, even if NA# is asserted.

In CS8 mode, all program code resides in 8-bit memory (ROM) while data are in 64-bit memory (RAM). A reset operation traps to the 0xFFFFFFF0 standard trap handler starting address. Hardware must disable RAM address space covered by the 8-bit bootstrap ROM and enable the ROM or EPROM over this address space. When exiting 8-bit code size mode, programs must output to a special I/O port which tells the processor to unmap ROM or EPROM from memory and map the normal 64-bit memory. Once code is loaded in 64-bit memory, initialization code initiates 64-bit code fetches by clearing the CS8 bit in the DIRBASE register. Once 8-bit code-size mode is disabled by software, it cannot be reenabled until the next hardware reset.
3.4 BUS CONTROL OPERATIONS

This section explains bus control operations including: page-mode and static-column DRAM support, bus arbitration, and bus locking.

3.4.1 Page Mode, Static Column DRAMs and Next Near Operation

The external bus interface facilitates high-performance designs using low-cost DRAMs. The next near signal (NENE#) facilitates designs using page mode and static column DRAMs.

Page mode and static column DRAMs perform best when multiple reads or writes access closely-situated areas of memory (as in the same row address of a given DRAM). This occurs frequently because memory accesses are often sequential.

The NENE# signal indicates that a memory cycle is using the same RAM page address as the previous cycle. The processor ignores a number of lower bits of the address that corresponds to the size of a page in the DRAM. NENE# provides information to the memory subsystem that it can use to enable page mode or the static-column mode operation of the DRAMs. NENE# eliminates the need for external circuitry and eliminates the difficult timing problems that are involved in implementing this capability.
The i860 microprocessor determines page size by interpreting the DPS field (bits 3, 2 and 1) of the dirbase register. The value in the field indicates the number of least-significant bits to ignore when comparing a cycle address to previous addresses. The number ignored equals 12 plus DPS. Zero is is an appropriate value for 256K x n RAMs, 1 for 1M x n RAMs, and so on.

NENE# is never asserted on a bus cycle immediately following the deassertion of HLDA. NENE# is not asserted for TLB miss cycles.

### 3.4.2 Bus Hold, Hold Acknowledge, Bus Request

The i860 microprocessor provides the input line HOLD and the output lines HOLDA (hold acknowledge) and BREQ (bus request) to control arbitration of the buses and control lines.

The hold request signal (HOLD) is driven by an external device or bus arbiter to request control of the bus from the processor. The HOLD signal can be driven asynchronously. The processor has an internal synchronizer to prevent metastable problems when sampling for HOLD. When hold is asserted, the processor blocks any new cycles. Once all outstanding bus cycles are completed, the processor relinquishes bus control by floating all output lines except HOLDA and BREQ. HOLDA indicates to the requesting device that bus control has been relinquished. The device can then use the signals and buses it needs until it relinquishes control to the processor.
The i860 microprocessor can continue executing from the cache once it has relinquished control of the bus. The BREQ signal indicates to any external bus arbiter that the processor is waiting for an external bus cycle. BREQ provides a means to implement a more efficient system for arbitrating between bus contenders such as a DMA device or another processor.

The bus state machine in Figure 3-12 illustrates the process that brings bus activity to a halt when the HOLD input signal is asserted. This diagram is similar to the bus state machine in Figure 3-8, except it includes the hold state and labels the REQ signals as REQNH. REQNH corresponds to REQ ANDed with NOT HOLD. ANDing all REQ signals with NOT HOLD prohibits the bus control unit from initiating new cycles while a hold is pending. Once all outstanding cycles are done, the state machine assumes the T1 idle state where the SHOLD signal (synchronized version of HOLD) causes a transition to the hold state TH. When the processor detects that the SHOLD signal is no longer active, the state machine transitions back to T1, if no internal requests are pending (NOT REQ), or to T1 to start a new bus cycle.

Figure 3-13 timing diagrams illustrate the operation of HOLD, HOLDA, and BREQ. HOLD is asserted within a T21 cycle. Since external HOLD must be synchronized internally, SHOLD is asserted one clock cycle after the assertion of HOLD. This explains why the T2, T21 bus cycle was started. This new bus cycle would not have started if HOLD had been asserted one clock earlier. The state machine completes all outstanding cycles and ignores internal bus requests indicated by the assertion of BREQ. Once the T1 idle state is reached, the hold state TH is entered. HOLDA is then asserted, and the microprocessor floats all output signals except HOLDA and BREQ. Within the second hold cycle, the HOLD signal deasserts. The state machine exits the hold state and transitions to T1 to perform another bus cycle request. Unlike the assertion of HOLD, deassertion of HOLD resets the SHOLD signal so that, on the next clock cycle, the bus can be recovered and HOLDA made inactive. The recommended set-up and hold times are required to ensure release of the hold state on the clock cycle following the deassertion of HOLD.

BREQ is activated by the processor’s internal bus requests independent of the state of HOLD and HOLDA.

Notes:
- RDY# in the table corresponds to the READY# signal
- NA# is not sampled during ADS# active clock
- ADS# is made active in T1, T2 and T3
- REQNH is the internal bus request ANDed with synchronized HOLD
- HLDA is made active in TH
- HOLD is synchronized internally.

3.4.3 Bus Lock

The bus lock signal (LOCK#) provides indivisible read-modify-write memory operation sequences for use in multiprocessor systems. Multiprocessor systems with an external arbiter should also use the LOCK# signal to prevent granting the bus to other masters.
Programmers need not track synchronicity between the bus and the instruction that sets the BL bit. LOCK# is asserted along with ADS# on the first data bus cycle following the setting of the BL bit. LOCK# and ADS# are deasserted on the first data bus cycle after clearing of the BL bit.

The bus is not locked until after the first data access cache miss causes the assertion of the LOCK# signal. Multiprocessor system software should therefore ensure that the first load instruction in a locked sequence references noncachable memory.
3.4.3.1 SUPERVISOR-MODE ACTIVATION OF LOCK# 

Supervisor mode software can set or clear the BL bit in the dirbase register directly.

3.4.3.2 USER-MODE ACTIVATION OF LOCK# 

User mode software can set or clear the BL in the dirbase with the lock and unlock instructions. These instructions support generalized interlocked sequences. The lock instruction sets the BL bit in the dirbase, and the next load or store operation locks the bus. Interrupts are disabled until the bus is unlocked. The unlock instruction clears the BL bit in the dirbase, and the next load or store operation unlocks the bus.

Some restrictions apply to this method. An interlocked instruction sequence must not branch or execute outside the 30 sequential instructions that follow a lock instruction. The interlocked sequence must be restartable from the lock instruction if a trap occurs, as in read-modify-write sequences. In sequences with more than one store instruction, software must prevent traps following the initial nonrestartable store instruction. This ensures that the sequence will not be restarted beyond this point. To meet this condition, the software must read and write the unmodified values of the locations used by the
instructions following the first store, to prevent the possibility of a data access page fault. Software must also ensure that executed code is within the same page to prevent instruction fetch page faults.

The processor will ignore a second lock instruction issued before the bus is unlocked. A trap occurs when from 30 to 33 instructions have executed following a lock instruction, if a load or store which follows an unlock instruction has not been executed.

If a trap occurs between a lock instruction and the first load or store following an unlock instruction, the interlock bit (IL) in the psr is set and BL is cleared. IL indicates to the trap handler that a lock sequence has caused the trap. It searches backward for the lock instruction and restarts from that point. The trap handler can scan up to 33 instructions; if no lock instruction is found, it signals an error to the user code.

### 3.4.3.3 BUS LOCK DURING PAGE TABLE UPDATE

The i860 microprocessor also asserts LOCK# in TLB miss processing to update the accessed bit within a page-table entry. The maximum time that LOCK# may be asserted for this case is the time needed for hardware to perform a read-modify-write in the page-table entry.

The LOCK# signal is asserted with an appropriate setup time relative to the clock cycle that samples the assertion of ADS#. This ensures that this bus cycle will be part of a locked sequence.

Figure 3-14 illustrates locked cycles. The LOCK# signal is asserted prior to the beginning of the T11 state of cycle 1 to initiate a locked cycle. The LOCK# signal is unasserted after the end of T11 to indicate that cycle 2 is not locked. This particular example is not of practical use, since it locks a single cycle, but it illustrates the LOCK# signal timing.

### 3.5 CACHE CONTROL OPERATIONS

The KEN# (cache enable) input signal enables or disables cache updates. The PTB output signal is used when paging is enabled to provide the memory subsystem with cache control page table information relating to the current cycle.

The i860 microprocessor prevents updates of both the data and instruction caches unless KEN# is sampled active. Enabling cache updates for a given cycle requires that KEN# be sampled active on the clock period after ADS# through the clock in which NA# or READY# is asserted. Updating either cache involves the generation of four 64-bit read operations. If KEN# is found deasserted for the first fetch cycle, the data being read is not cached.

Accesses to data or instructions already cached are not inhibited by KEN#. To prevent such accesses the caches must be flushed.

When paging is enabled, the values in the cache disable (CD) and write through (WT) page table bits are used to determine the caching strategy. During memory transfers, the secondary page table entry's CD bit indicates whether accessed data should be stored in an external cache. CD in the first-level page table is ignored. The WT bit allows software to determine the internal caching strategy. If WT is set in a page table entry (PTE), on-chip caching for data is inhibited. If WT is clear, normal write-back policy applies to
Figure 3-14. Locked Cycles

page table data. The WT bit of the page directory entries is reserved and is not referenced by the processor. The WT bit is independent of the CD bit; data that is not stored in the on-chip caches may be placed in an external cache.

The internal cache is disabled when a CD bit is set or when an external KEN# signal is unasserted. The CD and WT bits do not function when paging is disabled.

When the PBM bit in the epsr is set, the PTB output signal reflects the value of the WT bit. When clear, PTB reflects the value of the CD bit. The PTB output signal remains unasserted while paging is disabled.

**3.6 TRAPS AND INTERRUPTS**

Traps are caused by external conditions or by exceptional conditions detected in the course of a program’s execution. A trap causes the instruction being executed to abort and transfers control to a trap handler program stored in the hexadecimal virtual address 0FFFFFF00.
The trap handler reads the psr and fsr registers to identify the cause of the trap and branches to the portion of code that processes the trap (or traps if more than one occurred simultaneously). Once the trap handler has executed code to service the trap conditions involved, it restores the state of the processor, restarts the aborted instruction and resumes normal program execution. The i860 microprocessor is designed to ensure that all program instructions can be restarted. The i860th 64-Bit Programmer's Reference Manual explains how to write trap handling software. Table 3-2 summarizes the causes and indications of various traps.

The reset trap and interrupt trap are caused by external conditions. Reset traps are caused by hardware resets. (Section 3.8 provides further explanation.) Interrupts are caused when an external source asserts the INT signal while interrupts are enabled (i.e. while the IM interrupt mask bit is set in the psr). The trap handler immediately saves and clears the IM to inhibit further incoming interrupts.

The i860 microprocessor does not provide an interrupt acknowledge signal. After the interrupt takes place, the trap handler can read from the external I/O system to determine the source of the interrupt. The processor can then indicate to a port that the interrupt has been processed.

<table>
<thead>
<tr>
<th>Type</th>
<th>Indication</th>
<th>Caused by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Fault</td>
<td>IT, OF</td>
<td>Software traps Missing unlock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>trap, intovr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any</td>
</tr>
<tr>
<td>Floating Point</td>
<td>FT</td>
<td>Floating-point source exception</td>
</tr>
<tr>
<td>Fault</td>
<td></td>
<td>Floating-point result exception</td>
</tr>
<tr>
<td></td>
<td></td>
<td>overflow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>underflow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inexact result</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any M- or A-unit except fmlow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any M- or A-unit except fmlow, pflo, any pfeq.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reported on any F-P instruction plus pst, fst, and sometimes flid, pfdl, lxfr</td>
</tr>
<tr>
<td>Instruction</td>
<td>IAT</td>
<td>Address translation exception</td>
</tr>
<tr>
<td>Access Fault</td>
<td></td>
<td>during instruction fetch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any</td>
</tr>
<tr>
<td>Data Access</td>
<td>DAT</td>
<td>Load/store address translation</td>
</tr>
<tr>
<td>Fault</td>
<td></td>
<td>exception</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Misaligned operand address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operand address matches dh register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any load/store</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any load/store</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any load/store</td>
</tr>
<tr>
<td>Interrupt</td>
<td>IN</td>
<td>External interrupt</td>
</tr>
<tr>
<td>Reset</td>
<td>No trap bits set</td>
<td>Hardware RESET signal</td>
</tr>
</tbody>
</table>

### 3.7 TEST SUPPORT FUNCTIONS

The i860 microprocessor has a boundary scan mode for component or board level testing of signals and logic. Special test logic or instrumentation needs only to connect to the CLK, BSCN, SCAN, SHI, BREQ, RESET and HOLD signals to sample and drive the external processor signals. Table 3-3 indicates the signals for the test mode selections.
Sampling of an active boundary scan (BSCN) input signal initiates the boundary scan mode within the following clock cycle. Boundary scan mode may be activated while RESET is active. The processor exits boundary scan mode on the clock cycle following deassertion of BSCN. The internal state is undefined when the processor exits boundary scan mode. RESET should be asserted to reinitialize the processor.

<table>
<thead>
<tr>
<th>Table 3-3. Test Mode Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSCN</td>
</tr>
<tr>
<td>LO</td>
</tr>
<tr>
<td>LO</td>
</tr>
<tr>
<td>HI</td>
</tr>
<tr>
<td>HI</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

While in boundary scan mode, the processor may operate in normal mode or shift mode. Shift mode is a submode that is entered on the clock in which SCAN input is asserted. The normal mode is entered on the clock in which SCAN input is deasserted.

For testing purposes, each signed pin has associated with it an internal latch. Table 3-4 identifies these latches by name and classifies them as input, output or control.

<table>
<thead>
<tr>
<th>Table 3-4. Test Mode Latches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Latch</td>
</tr>
<tr>
<td>SHI</td>
</tr>
<tr>
<td>READY#</td>
</tr>
</tbody>
</table>
Here is a typical test sequence:

1. Enter shift mode and assign a value to all the latches.
2. Enter normal mode to force values onto desired output pins, and read all pin values.
3. Enter shift mode to read the new values of the input pins.

**3.7.1 Normal Mode Operation**

Normal mode begins with a clock cycle that samples a deasserted SCAN input signal. Within normal mode, the contents of the output latches are driven onto the output pins or buses if the corresponding control latch bits are set. If the corresponding control latch is not set, the output pins are not driven. RESET and HOLD must be asserted to float the outputs.

**3.7.2 Serial Mode Operation**

Shift mode begins with a clock cycle that samples the SCAN input in the active state. The value of all pins are immediately loaded into latches. Input pin latches load externally driven values. Floated output or bidirectional pins also read externally driven values. Latches with forced output pin values during normal mode, load their own values and do not change. Immediately after the clock cycle that sets the shift mode, the output of the BREQ output signal reflects the value of the BREQ latch. On subsequent clocks, the value of all the latches are shifted to the BREQ pin. At the same time, new values are being shifted in via the SHI pin. Figure 3-15 shows the order in which boundary scan latch chains are serially read and written. All outputs except HOLDA and BREQ are floated in shift mode to avoid glitches on the output lines. However, some glitches may be evident in the HOLDA output pin.

![Figure 3-15. Boundary Scan Chain](image-url)
3.8 RESET AND CLOCK CIRCUIT

i860 microprocessor RESET input must remain high for at least 16 clock periods to initialize the processor. Table 3-5 shows the status of all output pins while RESET is asserted. Once the RESET signal goes low, the processor traps and begins execution at hexadecimal address 0FFFFFF00 to execute the central trap handling routine. The i860™ 64-Bit Programmer's Reference Manual details processor status following a reset operation.

The circuit in Figure 3-16 shows the simple system needed to generate the clock and the RESET signal. The clock is a standard TTL circuit rated for the i860 microprocessor configuration used (e.g. 40 MHz).

The RESET signal is triggered when power is first turned on or when the reset button is pushed. Voltage in the capacitor falls to zero, creating a high value for the inverters output. The value is clocked by two D flip-flops to synchronously assert a RESET signal. The RC delay is at least 16 clock periods, raising the inverter input to a voltage level that moves the output to a low state. The low-level signal is clocked through two flip-flops and causes the RESET to deassert. The inverter and the flip-flops are used to clean up the RC signal and synchronize the resulting RESET signal. Two flip-flops help to prevent a metastable state that could be derived when synchronizing the RC network pulse.

Table 3-5. Output Pin Status During Reset

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Pin Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HOLD Not Acknowledged</td>
</tr>
<tr>
<td>ADS#, LOCK#</td>
<td>HIGH</td>
</tr>
<tr>
<td>W/R#, PTB</td>
<td>LOW</td>
</tr>
<tr>
<td>BREQ</td>
<td>LOW</td>
</tr>
<tr>
<td>HLDA</td>
<td>LOW</td>
</tr>
<tr>
<td>D63-D0</td>
<td>Tri-State OFF</td>
</tr>
<tr>
<td>A31-A3, BE7# - BE0#, NEN#</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

The circuit shown to generate CS8 consists of a driver (e.g. two inverters) to delay RESET and provide the hold time relative to the falling edge of RESET. This signal is ORed with INT. CS8 is inactive after being sampled by the falling edge of RESET, and the INT signal remains unaffected.
Figure 3-16. Circuit for Clock, RESET and CS8 Generation
4.1 INTRODUCTION

The on-chip caches of the i860™ 64-bit microprocessor contribute to its extremely high level of performance. Depending on the locality of memory references, the processor can usually operate from these caches. As a result, fewer accesses are made to the external bus allowing instructions and data to be supplied to the CPU at a very fast rate.

The need for a high-performance cache system stems from the i860 microprocessor’s access performance, which is double that of DRAMs. Because of this disparity, the cache is employed to decouple the processor from its external bus. In this way, the effect of a slower DRAM subsystem is greatly reduced.

The locality of memory references is key in determining memory system performance. Software with widely dispersed memory references increase the number of DRAM accesses. The effect of a slower DRAM subsystems on processor performance is much more pronounced in applications characterized by widely dispersed references. These applications, such as Linpack, and graphics matrix operations or file handling applications, such as Troff, warrant the effort and cost needed to design efficient DRAM subsystems. Inoptimal designs can easily reduce performance in these applications by 60 to 70 percent. Poor performance in these applications has a disproportionate detrimental effect on overall system performance, especially if important system functions such as page swapping or display are involved.

This chapter examines a DRAM subsystem design. The example has been optimized to reduce the number of clocks in which the processor waits for the bus. Controller function, processor features, and details such as timing and power consumption are analyzed. The example cannot account for all processor applications, and many applications will implement optimizations not found here. This design has been built and tested, however. If desired, it may be implemented as shown.

The example assumes a working knowledge of computer system design. Items that are discussed but not explained include DRAM operation, PLD programming and operation, worst case analysis techniques, and i860 microprocessor bus operation. The complete schematics and PLD equations are included in Appendix C.

4.2 CPU FEATURES

Certain i860 microprocessor features are specially devised to facilitate optimum DRAM subsystem performance. This section briefly describes these special features.

4.2.1 The KEN# Input

External logic, which is usually part of the DRAM controller, generates the KEN# input. KEN# indicates that the current read cycle is cacheable. As discussed in earlier sections, the processor generates three read cycles (in addition to the current cycle) in
response to KEN#. These cycles provide 32 bytes—enough data to fill one cache line. The handling of these cycles impacts memory system performance and will be discussed in the next section.

KEN# is generated for every cache block fill cycle. It is usually generated by decoding the processor address. Since the processor has no I/O space, memory mapped I/O addresses must be non-cacheable.

Figure 4-1 shows the timing requirements for generating KEN#. The most stringent case is a pipelined zero-wait-state cycle where KEN# must be generated in one clock. Once the address is valid, KEN# must be generated in time to meet the setup time \( t_{10} \) dictating that KEN# be generated by combinatorial logic.

### 4.2.2 Bus Pipelining

Obviously, the i860 microprocessor’s bus pipelining feature must be incorporated in DRAM subsystem design. This feature allows the processor to overlap bus cycles.

Pipelining is controlled by the processor’s NA# input signal. It allows the processor to begin a new bus cycle while another cycle is in progress. It can be activated twice before READY# is activated. In this way, three bus cycles can be activated before the first is completed.

Although the process is not demonstrated in this example, write cycles can be pipelined. In this design, writes are buffered or “posted” and can run without wait-states. As such, write cycle performance does not benefit from the use of pipelining.

Read cycle performance does benefit from pipelining, however, especially if read cycles are due to a floating point vector load. Vector loads can occur in long sequences. Since most of these cycles do not cross a DRAM page boundary, they can be executed with a minimum number of wait-states. Figure 4-2 demonstrates how pipelining facilitates floating-point vector loads.

![Figure 4-1. KEN# Timing](image-url)
The first read cycle in Figure 4-2 occurs from an idle bus state. In this case, the DRAM controller must add five wait-states to the cycle. These wait-states satisfy the RAS access time of the DRAMs. They are added by suppressing READY#. Ordinarily, waiting for READY# is a waste of processor time. Activating NA#, however, allows the next bus cycle to start. NA# is sampled active in the third clock, and ADS# is driven active. The address for the second cycle is valid one clock later.

NA# is activated again before returning READY# to the CPU, and the third vector load is performed. At this point, READY# must be returned to the processor before NA# is activated again. Data for the first read cycle must be valid at the processor data input pins when NA# is activated.

In this example, the pattern can be repeated for an unlimited number of bus cycles. When ADS# is not activated in response to NA#, the pattern is interrupted. Clearly, the benefit of pipelining in this example is that read cycles overlap. Consecutive pipelined read cycles return data to the processor every two clocks.

4.2.3 The Next Near Pin

The next near (NENE#) feature directly supports the DRAM subsystem. (The next near pin is described fully in Chapter 3.) Its function optimizes page or static column mode designs. It indicates that the current row address is the same as the previous address.

This DRAM design example uses static column mode memories. These memory devices contain a row address register. The register simplifies access to memory if the access is within the same DRAM page (row address). In the example, RAS# can remain active from the previous cycle. For read cycles, CS# can also remain active. This type of cycle
saves clocks: the required number of clocks doubles if RAS# must be activated to latch the row address. These extra clocks are required to meet RAS# precharge time. Implementation of this function will be detailed later in this chapter.

The NENE# function optimizes memory subsystem design. If implemented in logic, the function requires at least one register and a comparator. Another benefit is that NENE# is available along with the address, and no additional time is needed to generate this signal.

4.2.4 Write Data Function

Data bus contention could result when a write cycle is immediately preceded by a read cycle. The i860 CPU includes a bus feature to prevent this problem. When a write cycle occurs in the clock following a read cycle, write data valid timing changes. Figure 4-3 illustrates this change.

<table>
<thead>
<tr>
<th>CLK</th>
<th>ADS#</th>
<th>W/R#</th>
<th>DATA</th>
<th>READY#</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram A: Zero-Wait State Write Cycles

<table>
<thead>
<tr>
<th>CLK</th>
<th>ADS#</th>
<th>W/R#</th>
<th>DATA</th>
<th>READY#</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram B: Read Cycle Followed By Write

Figure 4-3. Read - Write Timing
Diagram A shows two successive write cycles. Diagram B shows a read cycle followed by a write cycle. The write data in Diagram B is driven one clock later than it is in Diagram A. This feature allows data bus transceivers to tristate the processor side of the bus before the CPU drives the write data.

4.3 DRAM SUBSYSTEM OVERVIEW

The features just described help make the DRAM subsystem more efficient. These functions are included in the processor to save logic needed to implement the DRAM controller. Timing constraints have also been eliminated, allowing fewer clocks per bus cycle.

The DRAM subsystem example described in this chapter illustrates how to use these features and describes the interaction of the processor with the memory system. The subsystem has been designed to use a 33.33 MHz clock. The example uses one Mbit memories which are organized 256K x 4. This memory is selected because of its minimum configuration. With a 64-bit bus, the minimum memory configuration is 2 Mbytes (16 devices). Using 1M x 1 memory devices, the minimum memory configuration is 8 Mbytes (64 devices).

Figure 4-4 is a block diagram of the memory subsystem example. It comprises five parts: address path logic, data path logic, parity logic, controller and DRAMs.

4.3.1 Address Path Logic

Address path logic performs several functions. Its primary function is to drive the processor address to the DRAM bank. To perform properly, it requires two paths for the row and column address.

In this example, the row address consists of processor address bits 12 through 20. These bits are buffered by the 74BCT29827 buffer. The buffer also disables the row address when the column address is selected. The device’s OE pin is used for this function.

The output enable of the column address logic is also used for this function, but the column address must be latched for pipelining. For this reason, an AS821 register with output enable function is used.

The register and buffer chosen to implement the address path logic are 10 bits wide. The memory uses nine of these bits. The remaining bit drives the parity DRAMs. The parity logic is described later.

Outputs of row and column address devices drive every DRAM. Restrictions on the drive capabilities of these devices require additional drivers every 4 Mbytes. Two sets of address logic devices are shown in Figure 4-4. This configuration allows up to eight Mbytes of memory. Additional drivers can easily be added for larger configurations.
4.3.2 Data Path Logic

The data path consists of eight 74AS646 bidirectional latching transceivers. These devices fulfill two critical memory system requirements. These transceivers hold read data
when the processor performs pipelined read cycles. Data for two reads has been accessed when the processor begins the third read cycle in a pipelined sequence. The data registers hold the first cycle’s data while the second cycle’s data is held at the transceiver input.

The register feature of the transceivers is also used during write cycles. The memory system posts all write cycles. During a posted write, NRDY# is returned to the processor before data is written to the DRAM. Without posting, write cycles would require at least two wait-states. Registers in the data transceivers hold write data after NRDY# has been activated. Section 4.4.4 provides a detailed description of the posted write function.

### 4.3.3 Parity Logic

Parity logic generates parity information and checks for parity errors. Eight 74AS280 9-bit parity generator/checker devices are used.

One parity bit is generated by each device. These bits are either written to the parity DRAMs or used by logic to generate a parity error signal. The parity error (PARERR) signal generates an interrupt which cannot be processed in time to stop the current bus cycle. The bus cycle which caused the error cannot be restarted.

The parity DRAM devices are 1 Mbit chips organized as $1M \times 1$. These DRAMs are used to store parity information for all eight Mbytes of main memory. The tenth address bit mentioned in Section 3.1 supports these devices. They divide the parity DRAMs into four sections which correspond to appropriate banks of main memory.

Odd-parity was chosen because of implementation restrictions. If even-parity is used, “I” inputs of the parity generators must be pulled down. Pull-ups are preferable in light of noise immunity.

### 4.3.4 Control Logic

The controller circuit contains the bus tracking state machine which is implemented with one PLD. The output signals of the PLD control three other PLD devices, each of which implements a specific control function.

The state machine PLD generates five state-variable outputs. These outputs define 32 states; designated sequences of these states implement various bus cycles. The state variable signals are decoded by the other control PLDs to create control signals. These signals, such as RAS0#, CAE#, and WEL, are activated to control the DRAMs, addresses and data logic.

Because of timing restraints, the state machine PLD (DSTAT in Appendix C) generates control signals NA# and RDY#. These constraints will be explained in the functional description (Section 4.3).

The DSTAT PLD also generates the BSY# signal. It serves as an arbiter between different system units, allocating processor bus priority to other subsystems that support bus pipelining.
The Control-A PLD generates row address strobe (RAS#) signals for all DRAM banks. Address bits are decoded along with state variable outputs of the DSTAT PLD to generate these signals. Each signal drives every DRAM in a 2 Mbyte array. The PLD is not registered because it primarily performs a decode function.

The RAS# signals select the active DRAM bank. Address inputs of this PLD are A21-A23. Figure 4-5 shows the address map for four banks of memory. The program for this PLD is listed in Appendix C. It shows the states decoded for the write, read and refresh functions. This PLD can be duplicated if a larger main memory is required, in which case address decode must be modified to select more memory banks.

This PLD also generates RAS# for the parity DRAMs. RASP0# enables the 1M x 1 DRAMs containing parity information for 8 Mbytes of main memory. Decode of this signal requires fewer address inputs.

Together, the Control-B and Control-C PLDs generate all address and data path control signals. They both have as inputs the state variables from the DSTAT PLD. Control-B also operates from a delayed clock, DCLK.

The delayed clock allows the Control-B PLD to sample state variables in the clock they are generated. Control-B outputs can be generated in the same clock.

The Control-B PLD controls the column address path. The PLD also generates the column address enable (CAE#) signal, which determines data access time, and the CSX# signal, which generates CS# signals to the DRAMs. Because the early write function is used, CSX# determines when data is written to the DRAMs. Control-B also generates signals which control the transceiver registers. The remaining transceiver control signals are generated by Control-C.

Figure 4-5. Address Map
Control-C also generates parity logic control signals. These signals latch parity data during writes and error indication during reads. Another important Control-C signal is RAE# which controls the row address path. Control-C also generates OEX# which generates output enable signals for the DRAMs.

The Control-D PLD does not use state variable inputs. Its only function is to generate write enable signals for the DRAMs. The WEL signal from Control-B is the clock input for this PLD. The signal is generated at the beginning and end of write cycles. By monitoring LADS# and W/R#, this PLD can determine the type of cycle being run when WEL is activated. The WE# signals are activated during writes and deactivated during reads.

4.4 DRAM SUBSYSTEM FUNCTION

This section defines the function of DRAM subsystem bus cycles.

4.4.1 Signal Description

The following list describes the purpose of all signals generated for the DRAM subsystem. Subsequently, the signals will be implemented in a functional discussion.

4.4.1.1 PROCESSOR INTERFACE

NRDY#  NRDY# is directly connected to the processors READY# input. It signals termination of a bus cycle and can be tristated.

NNA#  NNA# is connected to the processors NA# input. It is activated to enable pipelining and can also be tristated.

KEN#  KEN# indicates when caching can be enabled. It generates a cache line fill when activated in the same CLK as NA# and RDY#.

4.4.1.2 DATA PATH LOGIC CONTROL

RAE#  RAE# enables the row address drivers. It disables the row address before column address is driven.

CAE#  CAE# enables the column address register outputs. It disables the column address when the row address is driven.

CAL#  CAL# drives the clock input of the column address registers.

RDL  RDL activates the transceiver registers during read cycles.
### MEMORY INTERFACING

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDL</td>
<td>WDL activates the transceiver registers during write cycles.</td>
</tr>
<tr>
<td>DRMDIR</td>
<td>DRMDIR controls the direction of the data transceivers.</td>
</tr>
<tr>
<td>DRMEN#</td>
<td>DRMEN# controls the time at which data is driven to the CPU or memory.</td>
</tr>
</tbody>
</table>

#### 4.4.1.3 ADDRESS PATH LOGIC CONTROL

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSX#</td>
<td>CSX# generates CS# signals for four banks of memory and is generated from the rising edge of DCLK.</td>
</tr>
<tr>
<td>CS#0–3</td>
<td>These signals are buffered versions of CSX#. Each signal drives one bank of memory.</td>
</tr>
<tr>
<td>OEX#</td>
<td>OEX# generates OE# signals for four banks of memory and is generated from the rising edge of CLK.</td>
</tr>
<tr>
<td>OE#0–3</td>
<td>These signals are buffered versions of OEX#. Each signal drives one bank of memory.</td>
</tr>
<tr>
<td>RAS#0–3</td>
<td>These signals are decoded separately. Each enables access to a specific memory bank. They are decoded from address and state variable outputs.</td>
</tr>
<tr>
<td>RASPO#</td>
<td>RASPO# is generated during any access to the first four DRAM banks DRAMs. It latches the row address in the parity DRAMs.</td>
</tr>
<tr>
<td>WE0–WE7</td>
<td>These signals are generated by the Control-D PLD. Each signal corresponds to a single byte in each DRAM Bank.</td>
</tr>
</tbody>
</table>

#### 4.4.1.4 CONTROLLER SIGNALS

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEL</td>
<td>WEL is connected to the clock input of the Control-D PLD. It latches the BE#0–7 inputs and is activated only during write cycles.</td>
</tr>
<tr>
<td>LADS#</td>
<td>LADS# is used by the state machine to determine when a bus cycle has started. It also indicates that the processor address is valid.</td>
</tr>
<tr>
<td>CPEN#</td>
<td>CPEN# indicates that a bus cycle has started and is waiting to be completed. It signals to the state machine that LADS# has been active and that a bus cycle is pending.</td>
</tr>
<tr>
<td>DCLK</td>
<td>DCLK drives the clock input of the Control-B pal. It is produced by delaying CLK 20ns.</td>
</tr>
<tr>
<td>S0–S4</td>
<td>These signals are the state variable outputs of the state machine. They are decoded by other control PLDs to produce control signals.</td>
</tr>
</tbody>
</table>

4-10
BSY#  BSY# is used to prevent other subsystems from driving NA#, READY# or the data bus before the DRAM subsystem has completed pipelined bus cycles.

DRAMSEL  DRAMSEL is a decode output which indicates that the current bus cycle is a DRAM access.

CLRCYC  This signal clears the cycle pending register.

4.4.2 Basic Read Cycle

The basic read cycle is shown in Figure 4-6. This cycle is performed when the processor requests a single read while the bus is idle. To simplify discussion, pipelining is not used in this example, so the NA# signal is not shown.

The read cycle begins when the CPU activates ADS#. ADS# is latched by controller logic and held for one clock. ADS# is latched with discrete logic because of processor bus timing constraints. Note that the processor address is not valid in this clock.

Figure 4-6. Basic Read Cycles
The LADS# register output is sampled active at the next clock edge. The trailing edge of LADS# activates CPEN# if DRAMSEL is active. The row address from the processor becomes valid in this clock. Since RAE# is already active, the row address is driven to the DRAMs and is valid in the following clock. RAS# is driven active following this CLK edge.

The falling edge of RAS# latches the row address in the DRAMs. The row address drivers must be tristated at this point, but not until the row address hold time is met. For this reason, RAE# is not deactivated until one clock after RAS# is activated.

At this point, the column address must be driven as soon as possible. The address valid delay adds directly to data valid time, but row address buffers must be tristated before CAE# is activated. The delayed clock, DCLK, allows CAE# to be activated in the same clock in which RAE# is deactivated. In this way, CAE# is activated in the minimum time possible.

CAL# latches the column address for pipelined reads in the same clock in which CAE# is activated. CSX# is also activated in this clock and is buffered to produce four CS# signals. These signals enable DRAM data buffers for the read cycle while the column address stabilizes.

Data becomes valid two clocks after CAE# is activated. Read data is not valid long enough to meet hold time requirements of CLK or DCLK and must be latched and synchronized to CLK. This function is performed by the RDL signal.

RDL enables the data transciever registers. It is driven from the same PLD as CAE# and is exactly in synch with CAE#. RDL is activated two DCLKs after CAE#, however, to ensure it is active in time to latch the read data.

NRDY# is generated in the next clock. The processor samples read data held in the transceivers at this point. If no other cycle is pending, RAS# and CS# are deactivated in the next clock. The cycle ends when the state machine has waited three clocks and met RAS precharge time.

### 4.4.3 Pipelined Read Cycles

The basic read cycle just described only occurs if the processor generates a single read cycle to a noncacheable address. Since most read cycles are cacheable, they occur as cache block fills.

A cache block fill generates four consecutive read cycles. Using the pipelining feature of the bus, these cycles may be overlapped as shown in Figure 4-7. Through pipelining, addresses of subsequent read cycles are driven before the previous cycle completes. As a result, the latency of the last three cycles of a fill block is reduced. A cache block fill begins as a basic read cycle. It is converted to a cache block fill by the KEN# signal, which is sampled active when NA# or RDY# is active. Three additional read cycles occur subsequently to fetch the entire cache block, and these cycles can be pipelined.
NA# controls bus pipelining and is generated by DRAM control logic. NA# is driven active by logic during any read cycle. If another bus cycle is pending, the processor activates ADS# in the following clock. Once this bus cycle has started, NA# is activated a second time. The processor then begins the next cycle by issuing ADS#.

Up to three bus cycles may be pending at one time. NRDY# must be returned for the first read cycle before NA# can be activated a third time. This function must be enforced by the DRAM control logic. The state machine PLD is programmed to activate NA# at the appropriate time. During the first read, it activates NA# three clocks prior to the clock in which NRDY# is active. It activates NA# again one clock before the clock in which NRDY# is active. Figure 4-7 shows signal timing for this bus cycle sequence.

Column address and data registers are needed to implement this function. As seen in Figure 4-7, the column address path remains enabled throughout the sequence. CAE#, RAS# and CSX# remain active from the first read. An address driven by the processor in response to NA# must be latched; otherwise, the column address changes before

![Figure 4-7. Pipelined Read Cycles](240330-29)
DRAM read data is valid. Once the data becomes valid, it must also be latched. Without data registers, the column address would be held for an additional DCLK to allow the processor to sample the data. This requirement would add an additional wait-state.

NA# is activated again once NRDY# is returned to the processor for the first read cycle. As in the previous cycles, ADS# begins the fourth read one clock later. At this time, NRDY# is returned to the processor, completing the second read cycle.

At this point, NA# is activated a fourth time. If another bus cycle is pending, ADS# will be activated. Otherwise, DRAM logic completes the next two bus cycles. If no bus cycles have been started, it then deactivates RAS# and CSX#.

Another bus cycle may begin before the final bus cycles have completed, but not on the clock after which NA# is activated. RAS# and CSX# signals remain active in this case.

Pipelined sequences usually occur as described here. These cycles must all occur within the same DRAM page, however. If the row address changes, RAS# must be deacti­vated, and the pipelined sequence is interrupted. Read cycles generated for a cache block fill are always within the same page, and the sequence shown in Figure 4-7 illustrates this case. Note that all cycles of subsequent cache block fills can be pipelined.

4.4.4 Basic Write Cycle

In this example, pipelining is not used with write cycles. A method called posting is employed instead.

Posting is similar to pipelining because it improves performance by allowing consecutive bus cycles to overlap. In posting, however, NRDY# is returned to the processor before a cycle is completed at the DRAM. This function is better suited to write cycles because data is not returned from the DRAMs. The processor may complete a cycle and begin another while the DRAM subsystem completes the write cycle.

A write data register is needed to implement posted write cycles. This register holds write data after NRDY# is returned to the processor. In the example, the registers are contained in the data transceivers. This register is controlled by WDL.

The row address function of read and write cycles are the same. Figure 4-8 illustrates the function of a write cycle, which begins from an idle bus state. LADS# starts the cycle one CLK after ADS# is activated. The row address propagates to the DRAM, and RAS# is activated. As with read cycles, CAE# is activated one clock later.

CAL# is activated in this clock to latch the column address. The write data latch signal (WDL) is also activated. It is connected to the transceiver clock input. Here, data is latched and driven to memories.

CS#-controlled (early) writes are used in this example. When writes are performed in this way, data must be valid before CS# is activated. The WE# DRAM inputs must also be active prior to the falling edge of CS#.
Figure 4-8. Basic Write Cycles

CS# is activated one clock after NRDY# is returned to the processor. The write enable signals are driven in the clock after LADS# is activated. Data becomes valid in the same clock that CS# is activated. If another bus cycle is pending, ADS# is active at the next clock edge.

4.4.4.1 CONSECUTIVE WRITE CYCLES

Posting is most beneficial when write cycles are consecutive. Figure 4-9 shows three consecutive write cycles. The first is identical to that shown in Figure 4-8. The addresses of the following two cycles are in the same DRAM page as the first. These cycles can be completed without wait-states on the processor’s bus. Timing becomes more critical at zero wait-states. The WE# signals, for example, are no longer valid three clocks before CS#. They are activated in the same clock as CS#. In addition, write data is only valid in the last clock of the bus cycle.

4.4.5 Consecutive Bus Cycles

Additional wait-states are needed when different types of bus cycles occur consecutively. These are added to the second cycle of the sequence and allow the first bus cycle to complete after the second cycle begins.
Figure 4-9. Consecutive Write Cycles

The state machine performs this function. It tracks the current bus sequence. If a read cycle immediately follows a write, or if the opposite occurs, the controller adds the needed number of wait-states. Bus cycles proceed normally but are delayed to allow the previous bus cycle to complete and to reverse the direction of the data bus.

These functions are different for read and write cycles. The specific sequence for each is described in the following sections.

4.4.5.1 WRITE FOLLOWED BY READ CYCLES

Figure 4-10 shows a write cycle followed by a read cycle. This sequence occurs when a read cycle is pending before a write cycle has completed. A timing conflict would occur if the read is to the same DRAM page as the write.

The state machine enters a special sequence to handle this event. It begins as if a normal write cycle had occurred. After it returns NRDY# for the write cycle, however, the processor asserts ADS#. This signal starts the read cycle.
Figure 4-10. Write Followed by Read Cycles

In the first clock, data is written to the DRAM. Next, WDL is deactivated to prepare for another write cycle. Because the state machine has no way to determine the next cycle type, it prepares for the next write.

The process continues into the next clock. Here, CSX# is also deactivated to prepare for the next write. The write enables are still active, and WDL is activated. Control and address signals for the read become valid in this CLK, and the state machine samples them at the next CLK edge.

Once the state machine determines that a read cycle has started, it enters a special state sequence. It activates the OE# signal and activates CSX# to enable the DRAM for a read cycle. The Control-D PLD samples W/R# low and deactivates the WE# signals.
At this point, the DRAM controller has added one extra wait-state to the normal read process. It adds another for the next clock to allow data to propagate to the latching transcievers. RDL is then activated to latch the read data, and the cycle completes normally.

### 4.4.5.2 READ CYCLES FOLLOWED BY WRITE CYCLES

Figure 4-11 illustrates a read cycle followed by a write cycle. Both cycles access the same DRAM. The write cycle begins while the read cycle is in progress.

Pipelined reads add a level of complexity to the problem. When NA# is activated, a write cycle can start. This cycle can start before NRDY# is returned to the processor.
Figure 4-11 illustrates a worst-case scenario in which two read cycles must complete after the write cycle has begun.

ADS# for the write cycle is active in the clock in which NRDY# is returned for the first of two read cycles. LADS# is active in the next clock. LADS# and W/R# indicate to the state machine that the cycle is a write. The state machine then enters a two clock sequence to complete the second read.

The cycle pending signal is important in this sequence. LADS# is not active when the second read is finished. The CPEN# signal is the only indication that that another cycle has started. CPEN# is sampled by the state machine in the clock that NRDY# is active for the last read. In response, the state machine enters a special state sequence. Here, it performs the functions necessary to prepare for a write cycle.

Two clocks after NRDY# completes the last read, OE# is deactivated. In the next clock, WDL latches the write data. NRDY# is then returned to complete the write cycle. CSX# is deactivated in this CLK so that data can be written in the next clock. At this point, the cycle continues as described in Section 3.4.

4.4.6 Page Miss Cycles

A page miss cycle is a memory access which changes the row address. This type of access is important in modern memory systems using page mode or static column mode DRAMs. These DRAMs have an internal register which holds the row address. Access can be made by simply changing the column address. This feature reduces the average access time by several clocks.

The DRAM controller must be designed to take advantage of this feature. In doing so, memory system performance is improved dramatically. To use this feature, control logic need only activate RAS when the row address changes.

The NENE# signal indicates when the row address has changed. It is available along with the address during any bus cycle. The state machine samples this pin at the beginning of every bus cycle.

When NENE# is sampled inactive, the state machine enters a special state sequence. Here, it performs the functions needed to latch the new row address. Figure 4-12 shows a typical page miss sequence.

The RAS# signal is immediately deactivated when a page miss is detected and is held inactive for four clocks. The time RAS# is held inactive is determined by the RAS# precharge time. This timing is a requirement of the DRAMs and varies depending on the manufacturer and the speed selection used.

CAE# is deactivated in the same clock as RAS#. The column address register outputs must be tristated before driving the row address. RAE# is activated two clocks later. At this time, the new row address is asserted at the DRAM address inputs. The cycle can then be completed as described earlier.
4.4.7 Refresh Cycles

Refresh cycles must be performed at regular intervals of approximately 15 microseconds. Each cycle refreshes one row of DRAM data.

Most DRAMs provide their own refresh address. This address is maintained in an internal counter and is updated if CS# is activated before RAS#. The address from the counter also refreshes the memory array. This function is called CS# before RAS# refresh.

Although not illustrated in this example, the refresh address can be provided by an external counter. This method is called RAS#-only refresh.

The RFRQ signal is generated by another counter and indicates that a refresh cycle must be performed. The counter is set to a count which ensures the proper refresh interval.

The RFRQ signal is sampled by the state machine during any idle cycle and at the end of every bus cycle. If the signal is active, the state machine enters a page miss cycle state sequence. Once the RAS# precharge time is satisfied, the state machine enters the refresh sequence.

A new row address must be latched after a refresh cycle is performed. After RAS# is activated for the refresh cycle, it is again deactivated. Once the precharge time has been met, RAS# can be activated to complete a pending bus cycle.

CPEN# is the only indication that a bus cycle has begun during the refresh cycle. If this signal is active after the second RAS# precharge, a bus cycle begins immediately. Only one bus cycle may be started during the refresh sequence.
MEMORY INTERFACING

4.5 PARITY CIRCUIT

Parity logic provides error detection capability for each byte of data. One bit is stored in the parity memory for each data byte. The value of this bit depends on the number of bits set in the corresponding byte. The parity logic is designed for odd parity. If a byte contains an even number of set bits, the parity bit is also set.

Parity data is generated during write cycles. Some delay is incurred, but because writes are posted, parity data is available in time to be latched by CS# signals. These signals drive parity DRAMs and data memory.

A separate RAS# signal is generated for the parity DRAMs. This signal must be activated for accesses to all four banks of memory. The Control-A PLD generates this signal during any cycle to the banks it controls.

Parity is checked during read cycles. The parity error signal is activated if a zero is detected on any of the parity outputs. Figure 4-13 is a block diagram of parity logic. It shows connections of the control and bus signals.

4.5.1 Dedicated Signals

The following is a summary of the signals that control parity logic.

- **RASP0#** drives parity DRAM RAS# input.
- **PARERR** is activated when a parity error is detected. It is held in a register once activated.
- **PERLTCH** activates the register which samples PARERR during read cycles.
- **WPL** activates the write parity data register.

4.5.2 Parity Function

Parity data is latched during zero-wait-state writes. It is latched in the clock after write data has been latched. The write data register is clocked by WDL in the same clock in which NRDY# is activated. WPL clocks the parity write data register as shown in Figure 4-13. WPL is activated from the rising edge of CLK. Since CSX# is activated from DCLK, parity data can propagate to the DRAMs before CSX# is active.

Parity data is generated by 74AS280 devices. The I inputs of these devices are not driven during write cycles but are pulled high to generate correct parity output. During read cycles, read parity data is driven to I inputs. Appendix C provides complete DRAM system schematics.

Read parity data is accessed in the same way that read data is accessed. The parity data register is clocked by RDL, and parity data is valid at the parity generators about the same time that read data is valid at the transceivers. The DRAMDIR signal enables output of this register. It disables the register outputs during write cycles.
The PARERR signal indicates that a parity error has occurred. It is generated by a 74AS80 nand gate. The inputs of this gate are connected to the OD outputs of the parity generator. If any of these outputs is high during a read cycle, PARERR is activated.

Once it occurs, the PARERR signal is latched. The PERLATCH signal activates the PARERR register. It is active every read cycle. The PARERR register can be cleared in the read cycle following the cycle that caused the parity error.
I/O Interfacing
CHAPTER 5
I/O INTERFACING

5.1 OVERVIEW

I/O devices can be mapped anywhere within the i860™ microprocessor’s four gigabyte physical address space and can be 64-, 32-, 16- or 8-bits wide. For accesses to 8-, 16- or 32-bit devices, byte-swap logic or 8-, 16- or 32-bit load/store instructions must be used. Although address pipelining can be used for all i860 microprocessor accesses, I/O recovery time and infrequent back-to-back I/O accesses greatly reduce pipelining benefits. The i860 microprocessor does not include I/O instructions, and there is no separate address map. All I/O devices must be mapped into the memory address space. The system distinguishes between memory and I/O accesses by decoding processor addresses. Although all reads can be cached, the I/O reads must not be cacheable and should deassert KEN# to indicate a non-cacheable access.

I/O buffers accessed by DMA devices must be marked non-cacheable or flushed from the data cache before an I/O operation. Two different approaches to I/O are described later.

5.1.1 i860™ Microprocessor I/O Subsystem

Figure 5-1 illustrates the I/O subsystem of an i860 microprocessor based system. The memory address and data are latched to allow pipelined operation. I/O addresses and data can be buffered if drive requirements exceed i860 microprocessor specifications. Processor addresses are decoded to determine whether access is to I/O or to a memory device. Decoder outputs notify control logic of cycles and indicate whether the cycles are to memory or to I/O. Read, write and chip select are generated by control logic. This logic also generates wait-states in i860 microprocessor cycles and generates READY# when control logic is ready to terminate the processor cycle. Interface to all slave devices except DRAMs is very similar and less performance sensitive. Nonvolatile memories such as ROMs and EPROMs are accessed through I/O control logic and share address and data paths with other slave I/O devices. The i860 microprocessor provides a CS/8 bootstrap mode for byte-wide ROMs used only during power-up. The mode is disabled once the system boots.

I/O devices may be mapped anywhere within the processor’s four gigabyte physical address space. To minimize the decode logic, devices are placed within one contiguous block of the i860 microprocessor’s address map. This minimizes the number of address lines needed to generate I/O select signals. If 8-, 16- or 32-bit I/O devices are used without byte swap logic, I/O ports must be placed at 8-byte increments (i.e. addresses must correspond to the data pins the device is hardwired to). This causes the processor to read or write data on the proper bytes of the data bus. For example, an 8-bit access may get data on D7–0; a 16-bit access may get data on D15–0; a 32-bit access may get data on D31–0. Here, addresses are on 64-bit boundaries.

This limitation does not apply if external byte swap logic is used. 8-bit devices can be accessed on continuous byte boundaries; 16-bit devices can be accessed on continuous even byte boundaries; and 32-bit devices can be accessed on continuous, multiple-of-four
Figure 5-1. i860™ Microprocessor System
byte boundaries. If a 16-bit access is attempted on an odd byte boundary, or if a 32-bit access is attempted on an address boundary that is not a multiple of four, the i860 microprocessor generates an exception.

5.2 GENERATING I/O CONTROL SIGNALS

Control and chip select signals for slave I/O devices are generated by I/O control logic. The i860 microprocessor does not provide a separate I/O space, and the distinction between accesses to memory and to I/O devices is made by decoding addresses. The address decoder typically provides an IOSEL (I/O select) or MEMSEL (memory select) output to indicate the access type. When IOSEL is asserted, I/O control logic generates appropriate control signals, drives KEN# inactive to indicate a non-cacheable access, inserts any needed wait-states and returns READY# to the processor to terminate the cycle. I/O control logic must enable appropriate address and data buffers, and it must eliminate bus contention caused by data float delays. Most I/O devices require a recovery period between back-to-back accesses. This can be enforced either through hardware or software.

5.2.1 I/O Control Logic

Figure 5-2 illustrates i860 microprocessor control logic. I/O select signals are generated from the decode of unlatched addresses. The decoder also generates device select signals which are latched to provide chip selects for specific devices being accessed. ADS# does not ensure a valid address when asserted. LADS#, a latched version of ADS#, is valid in state T_{11} and can be used to generate an address latch enable (ALE) signal. Address is guaranteed to be valid on the clock (rising) edge during which LADS# is active.

When LADS# is active and I/O select is asserted, I/O control logic generates read or write signals based on the state of W/R# input. During pipelined operation, I/O control logic must know of any outstanding DRAM cycles to avoid misordered cycles. BSY indicates that there are outstanding DRAM cycles and that I/O control logic should not start a cycle until BSY is deasserted. I/O control logic inserts needed I/O cycle wait-states according to the number of wait-states required by the device being accessed. In a simple design, I/O accesses can be assigned the number of wait-states required by the slowest device.

Following read cycles, I/O devices may need time to turn off the data bus. During this period, a write or read from another device causes bus contention. To ensure proper operation, I/O devices also require recovery time between consecutive accesses. It may or may not be practical to enforce this recovery with a hardware counter. The advantage of using a hardware enforced recovery mechanism is transparency and reliability. Also, future upgrades to higher CPU clock speeds will not require any changes in the I/O device drivers. If the recovery period is too long to be enforced with hardware, then software timing loops or a timer chip may be used to ensure proper operation. The hardware I/O recovery time is enforced by the control logic. Recovery logic generates a recovery delay (RECOVER) signal which prevents I/O control logic from asserting read or write until the recovery delay signal is deasserted.
5.2.2 Address Decode and Chip Select

Chip selects for various I/O subsystem devices are generated through address decoding. Decodes may be done with a PLD, such as the Intel 85C508, or with a simple one-of-four decoder such as the 74F138. Decodes can be done from latched or unlatched addresses. Unlatched address decodes allow device selects to be generated one propagation delay after an address becomes valid. Some I/O devices need a long chip-select hold time from read or write. This requires that chip selects be latched. Decodes can be done alternately from latched addresses, but this delays the chip select generation and may result in slower cycles. Device selects are also used by I/O control logic to determine the number of wait-states for a device being accessed. If device selects are generated from unlatched addresses, I/O control logic must sample the device select with LADS# to ensure a valid sample of the device select. Latched device selects must be deasserted (unlatched) once READY# is returned to the processor. They are unlatched by activating the latch enable signal which causes the latch to become transparent. The device select is deasserted when the address changes at the beginning of a new cycle.
I/O device addresses should simplify decode logic and reduce the number of address pins required to generate device selects and the I/O select signal. The I/O select signal indicates I/O device accesses to control logic. I/O devices are typically mapped with a contiguous single address space. Address space size is determined by the address lines used in decode.

5.2.3 IORD#/IOWR#

IORD# and IOWR# command signals are generated some time after the processor W/R# signal chip select. I/O devices have command-active width requirements, and wait-states are inserted in I/O cycles to accommodate these. The number of wait-states is determined by the I/O control logic of selected devices. Minimum command-active times range from 100ns to 400ns across various slave devices. I/O devices also have chip select set-up and hold time requirements. Chip select requirements are met by inserting delays in activating commands. If one delay is used across all devices, it must be the worst case delay. In the examples given here, the delay is two clocks. If necessary, hold time is met by latching chip selects and addresses.

It may be necessary to delay driving write data onto the bus for several clocks. This is necessary if the previous cycle was a read to an I/O device with a long data float delay. It may also be necessary to delay the assertion of READY# from the rising edge of IOWR# to ensure sufficient write data hold time.

5.2.4 READY#

Wait-states are introduced into the i860 CPU cycle by not asserting READY#. Since most I/O devices are much slower than the i860 processor bus, all I/O cycles require additional wait states. Since READY# can be driven by multiple PLDs, it must be tri-stated by the PLDs not in use. If several devices reside in the I/O subsystem, the wait-state logic can be simplified by inserting the same number of wait-states in all I/O accesses. This slows accesses to some of the faster I/O devices, but these accesses are infrequent, and impact on system performance is minimal. If I/O speed is critical, a small ROM or PLD can be used to insert wait-states and enforce recovery on a per device basis.

5.2.5 Recovery and Bus Contention

Most I/O devices require a recovery period between back-to-back accesses. At higher i860 CPU clock frequencies, bus contention poses an additional concern. This is because long float delays of the I/O devices can conflict with data driven out in the next cycle by another device or by write data from the CPU. All slave devices stop driving data on the bus on the rising edge of IORD#. Some delay after the rising edge of IORD# the data will float. If, however, another device drives data onto the bus before the data from the previous access floats, bus contention will occur. The i860 CPU has extremely small cycle times (30ns @33 MHz, 25ns @40 MHz), and the possibility of bus contention must be considered. The I/O control logic implements recovery to eliminate bus contention. It asserts the signal RECOVER, which inhibits new I/O cycles from starting until the data from the previous read has floated. I/O control logic can also accomplish I/O recovery using a similar mechanism. The only time hardware enforced I/O recovery may not be
possible is when recovery times are too great for the hardware counter. In this case, recovery time can be enforced with software using NOPs and delay loops or with a programmable timer.

5.3 I/O CYCLES

The I/O read and write cycle timings depend upon the implementation of the I/O control logic. Figures 5-3 and 5-4 illustrate the timings of the I/O read and write cycle for a typical implementation.

5.3.1 Read Cycle Timing

A new i860 microprocessor read cycle is initiated when ADS# is asserted in T₁. The address and status signals become valid in T₁₁, LADS# becomes valid in T₁₁, and the address is latched on the next rising edge of clock (second T₁₁). The I/O select signal is
generated from a combinatorial decode of the addresses. I/O select, when active with LADS#, indicates an I/O cycle to the control logic. The chip select is either asserted at the same time as I/O select or is latched in the clock following I/O select. The IORD# signal is asserted in the second $T_{11}$ if RECOVER is inactive. RECOVER, if active, indicates that the new cycle must be delayed in order to meet the recovery time of the I/O device or to prevent data bus contention. If RECOVER is active, then the I/O read (IORD#) signal is not asserted until RECOVER is deasserted. I/O data becomes valid on the bus one read delay after IORD# is asserted. The bus control logic will need to keep IORD# active to meet the minimum active time requirements.

The worst-case timings values are calculated by assuming the maximum delay in the address latches and decode logic and the maximum delay through the data transceivers. These formulas will yield the fastest possible cycle. Wait-states must be added to meet the access times of the particular I/O device.
The critical timings for I/O Read cycles are the following:

1. Chip Select/Address Setup to IORD#:
   \[ \geq 3Tcy + \text{PLDclk-to-output(min)} - \text{Decode delay max} - \text{Address Valid delay max} - \text{Latch Prop. Delay max} \]

   Chip select goes active later than I/O address and is latched, so chip select timing is used for the equation.

   Tcy : CLK PERIOD OF i860 Microprocessor. Address Valid Delay : Processor address valid delay in T_{11}.

2. IORD# Minimum Active Width:
   \[ \geq Tcy + \text{PLDclk-to-output} - \text{PLDclk-to-output} + nTcy \]
   \[ \geq Tcy + nTcy \]

   \( n \) is the number of wait states inserted into the I/O read cycle. \( nTcy \) represents the additional time due to wait states.

3. Chip Select/Address Hold after IORD#:
   \[ \geq Tcy - \text{PLDclk-to-output} + \text{Address Valid min} + \text{Latch Prop. Delay min} \]

   Assumes I/O address changes earliest, so hold time calculations are done using the I/O address timings.

4. IORD# Active to Data valid:
   \[ \geq Tcy - \text{PLDclk-to-output} - \text{Data Setup Time} - \text{Data buffer Prop. delay.} \]

   Data Setup Time: Processor setup time requirement for data to rising edge of clock.

   This assumes the minimum IORD# active pulse width of 1 clock. For wait states add the appropriate number of clocks to this value.

Other timings which need consideration for read cycles are the read inactive to data float delay and address/chip select hold time after read. The address and chip select hold times from read are not a problem, since both chip select and address are latched. The data float delay in slave devices can be very long (40ns). This can cause data bus contention and is taken care of with the I/O recovery logic. (Section 5.2.5 provides further information.)

5.3.2 Write Cycle Timing

An I/O write cycle starts similarly to an I/O read cycle. The address and status timings are similar to the I/O read. The processor outputs data in T_{11}. I/O write (IOWR#) may be asserted one or two clocks after the chip select (the exact delay between chip select and IOWR# depends upon the chip select/address setup to write requirements of the I/O devices). The use of latching data buffers can improve CPU write performance. Once the data and address of the cycle are latched, READY# can be returned to the processor, and the CPU operation and the write cycle to the device can continue. IOWR# is deasserted only after the data set-up to write specification is met. Data is written into the I/O device on the rising edge of IOWR#, and the processor stops driving data once READY# is sampled active.
The critical timings for the I/O write cycle are the following:

1. IOWR# Active Pulse Width:

\[ \geq T_{cy} - \text{PLDclk-to-output} - \text{PLDclk-to-output} - \text{PLDclk-to-output} + nT_{cy} \]

\[ \geq T_{cy} + nT_{cy} \]

\( n \) is the number of wait states inserted into the I/O write cycle. NTCY represents the additional time due to wait-states.

2. Address/Chip Select Set-up to Write:

\[ \geq 3T_{cy} + \text{PLDclk-to-output(min)} - \text{Decode delay max} - \text{Address Valid delay max} - \text{Latch Prop. Delay max} \]

Chip select goes active later than I/O address and is latched, so chip select timing is used for the equation.

\( T_{cy} \): CLK PERIOD OF i860 Processor.

\( \text{Address Valid Delay} \): Processor address valid delay in \( T_{11} \).

3. Data Setup To Write Inactive:

\[ \geq 4T_{cy} - 860 \text{ Data Valid Delay max} - \text{Data Buffer Prop. Delay max} + \text{PLDclk-to-output Delay} \]

Formula 3 assumes a zero-wait-state write cycle. Write is not posted. PLDclk-to-output delay is the delay in deasserting IOWR# from the rising edge of the clock.

Other timings in the I/O write cycle are the address/chip select hold time from write (IOWR#) high and data hold time after write (IOWR#) high. Since the address and chip select are latched, hold time is not a problem. However, if data is buffered through a 74F245 type device, the data could float as early as 7ns from READY#. Typical hold times for I/O devices vary from 0ns to 20ns. If data hold time requirements of the slave device are greater than 7ns, the write data must be latched; otherwise, READY# is returned some delay after the rising edge of IOWR#.

## 5.4 DESIGN EXAMPLES

This section discusses the i860 microprocessor interface to specific slave devices. It describes the basic interface, critical timings and equations.

### 5.4.1 82510 Interface

82510 is a UART with an asynchronous CPU interface. The basic interface is illustrated in Figure 5-5. The eight I/O ports of the 82510 are mapped to memory locations 0x07000000 through 0x07FF0000. The ports are located on 64-bit boundaries to allow data to be read on D7-D0 without the use of external byte swap logic. The chip select signal is generated from the decode of A20 and A19. By decoding more of the address lines, a smaller memory block can be used to map the device. For example, a full decode could allow the 82510 to be mapped to 0x07000000-0x0700000C. The address decoder generates the IOSSEL as well as the 82510SEL signal. The address and chip select are
latched to meet the minimum chip select active width and hold time. The RD# and WR# signals of the 82510 are generated by the IOCTL1 PLD. Critical timings of the 82510 are the following:

**Read Cycle**

1. Address Valid to Read Active (Tavrl):
   
   \[ T_{avrl} \leq 3T_{cy} + \text{PLDclk-to-output(min)} - \text{Decode Delay}_{max} - \text{Address Valid delay}_{max} - \text{Latch Prop. Delay}_{max} \]
   
   \[ 7\text{ns} \leq 3T_{cy} - 25 \leq 50\text{ns (@40MHz)} \]

2. Command Access Time To Data Valid (Trldv):
   
   \[ T_{rldv} \leq T_{cy} - \text{PLDclk-to-output} - \text{Data Setup Time} - \text{Data buffer Propagation delay} + N T_{cy} \]
   
   \[ N: \# \text{ of Wait States.} \]
   
   \[ 281 \leq T_{cy} - 10 - 12 - 6 + NT_{cy} \leq (N + 1)T_{cy} - 28 \]
   
   \[ N = 9 \]

3. Command Active Width (Tlrlh):
   
   \[ T_{lrlh} \leq (N + 1)T_{cy} \]
   
   \[ N = 9 \]

4. Read Inactive to Active (Tciad):
   
   \[ T_{ciad} \leq 123\text{ns} \]
   
   \[ \leq 4T_{cy} - \text{PLDclk-to-output}_{max} + \text{PLDclk-to-out}_{min} \]
   
   \[ \leq 4T_{cy} - 10 + 2 \]
   
   \[ \leq 4*31 - 8 \]
   
   \[ \leq 116\text{ns} \]

The control logic can assert IORD# no earlier than 116ns after the previous read. This violates the recovery requirements of the 82510. Control logic can enforce this requirement by delaying the assertion of IORD# by one clock, or software can implement this by using NOPs.

**Write Cycle**

Address valid to write low and write active width timings are similar to 1 and 2 for read cycles.

3. Data Valid to Write Inactive (Tdwh):
   
   \[ T_{dwh} \leq 3T_{cy} - \text{Data Valid Delay} - \text{Buffer Delay} + \text{PLDclk-to-output}_{min} \]
   
   \[ \leq 3T_{cy} - 50 - 6 + 2 \]
   
   \[ 90 \leq 3*31 - 50 - 6 + 2 = 39 \]
The IOWR# pulse must be extended by two clocks.

4. Data Hold Time after Write Inactive (Twhdx):

\[
Twhdx \leq \text{Data Float Delaymin} + \text{Buffer Delaymin} \\
12 \leq 3.5 + 2.5 = 6.0
\]

The minimum data float and buffer delays do not meet the 82510 hold time requirements. In this case, READY# can be delayed from IOWR# by one clock, or data can be latched and READY# can be returned early. The second option reduces wait-states but requires latches on the data bus.

5.4.2 Eprom Interface

The i860 CPU supports a special CS8 mode for bootup from 8-bit I/O devices. This allows the processor to bootup from an 8-bit ROM. Once the system boots, the ROM can be copied into memory or can be disabled and replaced by DRAM. In this mode, code fetches (that are misses in the code cache) are eight bits wide. To facilitate 8-bit reads, the BE2#–BE0# pins behave as A2–A0. Once the i860 microprocessor boots, the CS8 mode can be disabled by setting the CS8 bit of the dirbase register. If EPROM contents are to be copied into DRAM after the bootup, the EPROM locations need to be remapped to 64-bit boundaries. This allows the i860 microprocessor to access the EPROM bytes on D7–D0 without using byte swap logic. It does, however, require the address multiplexor to use either the processor address bus or the byte enables for A2–A0 inputs of the EPROM. A method which does not require an address mux is the “double copy load” method. This method is more dependent upon software, however. It uses the BE2#–BE0# signals which are connected directly to the EPROM A2–A0 addresses to select the bytes.

Figure 5-6 shows the i860 processor interface to a byte-wide EPROM (27010) and uses an address multiplexor to select between BE2#–BE0# and processor addresses. The chip enable of the EPROM is generated decoding the memory address and the MAP bit. The MAP bit, when set, maps the EPROM addresses into the power-up bootstrap locations. When MAP is low, the EPROM is mapped to another address and DRAM is mapped to the bootstrap locations. The OE# signal is connected to the IORD# signal generated by the I/O Control Logic. The upper 14 address bits of the EPROM are connected to i860 CPU addresses. The lower three addresses A2–A0 are multiplexed between BE2#–BE0# or three address bits of the processor. The multiplexor is controlled by the MAP input which is generated by a bit in an I/O register. The EPROM data bus D7–D0 is connected to the lower byte (D7–D0) of the processor data bus. This requires that the EPROM, when mapped as data, be on 64-bit boundaries.

Use the circuit in Figure 5-7 for i860 microprocessor systems that boot from EPROM and copy their code to DRAM for normal execution. Normally, an EPROM used in CS8 mode cannot be read as data. By adding a triple 2-to-1 multiplexor and using special addressing, the EPROM contents can be read out as data. A special routine is needed to access the EPROM. A portion of the physical address space must be reserved for addressing the EPROM as data. Figure 5-6 illustrates the connection from i860 CPU to EPROM.
The EPROM(s) are connected to the low order 8 data bits of the i860 microprocessor data bus. The top 32 Mbytes of the physical address space is reserved to address up to 2 Mbytes of EPROM code/data. A physical address space sixteen times the size of the EPROM must be used. The EPROM address space must be at the top of physical memory.

Accesses with A24 HIGH are code accesses and should activate KEN# to enable caching of instructions from EPROM. Accesses to EPROM with A24 LOW must not allow caching of the EPROM data. The signals on data bus D63–D8 are undefined and must be ignored by the program. The data multiplexor must select the B inputs when A24 is high.
The EPROM data is copied from EPROM to DRAM via a subroutine. The subroutine is called: `eprom_copy(eprom_base, addr_inc, start, count, destination)`. The subroutine copies count bytes from start offset in the EPROM to destination.

The parameter `eprom_base` points at the beginning of the \(16\times\) EPROM space. For the hardware above, `eprom_base` is 0xFE000000. The `eprom_base` is not the same address as would be loaded into the instruction pointer to execute the first four bytes as instructions. The program address of the first instruction at offset 0 in the EPROM in the example above would be 0xFFE00000. In the example above, the EPROM instructions could also be executed at addresses 0xFFF00000, 0xFFF00000, 0xFFF80000,
0xFFF600000, 0xFFF400000, 0xFFF200000, and 0xFFF000000. An attempt to read these address ranges would not allow access to all bytes of the EPROM since not all combinations of BE2#–BE0# can be generated via reads.

The parameter **addr_inc** is the address pin used to access individual bytes in the EPROM, for the hardware above this is 21. Consecutive bytes in EPROM are spaced $2^{21}$ bytes apart. At every eight bytes the low order address pins change. The table below illustrates:

<table>
<thead>
<tr>
<th>EPROM byte offset</th>
<th>Address used by <code>eprom_copy</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xFE000000</td>
</tr>
<tr>
<td>1</td>
<td>0xFE200000</td>
</tr>
<tr>
<td>2</td>
<td>0xFE400000</td>
</tr>
<tr>
<td>3</td>
<td>0xFE600000</td>
</tr>
<tr>
<td>7</td>
<td>0xFE800000</td>
</tr>
<tr>
<td>8</td>
<td>0xFE000008</td>
</tr>
<tr>
<td>9</td>
<td>0xFE200008</td>
</tr>
</tbody>
</table>

The parameter **start** is the byte offset in the EPROM to begin the copy. This offset assumes the EPROM appears as a sequence of bytes. The **start** can have any alignment. The **count** is the number of bytes to copy. The **destination** is where to copy the destination data, which can have any alignment.
This subroutine can be called at any time so long as the program can address the lower half of the 16× EPROM address space.

```c
struct {
    int R16;  // base address of EPROM block
    int R17;  // address base used to access individual bytes
    int R18;  // starting offset of data in EPROM
    int R19;  // count of bytes to copy
    int R20;  // destination address
}

eprom_copy:
    or     1, r0, r21
    adds   -1, r19, r19
    shl    r17, r21, r21
    bnc    exit
    or     7, r0, r22
    shl    r17, r22, r22
    addu   r16, r22, r25
    and    7, r18, r23
    shl    r17, r23, r23
    andnot 7, r18, r18
    addu   r23, r18, r18
    adds   -1, r0, r24
    bla    r24, r19, cloop
    addu   r16, r18, r18
}
cloop:
    ld.b 0(r18), r26
    subu r18, r25, r0
    bnc.t in_block
    addu r21, r18, r21
    andnot r22, r18, r18
    adds 8, r18, r18
    in_block:
        st.b r26, 0(r20)
        bla r24, r19, cloop
        adds 1, r20, r20
    exit:
        bri r1
        nop
```

5.4.2.1 DOUBLE COPY LOAD

An alternative to the address multiplexor is the "double copy load" method. This method requires considerably more software effort than the address mux method and also uses more EPROM space (since two copies of the code are required). In this case the EPROM A2–A0 are directly connected to the BE2#–BE0#. The 8-bit data bus of
the EPROM is connected to D7–D0 of the processor. Unlike the address mux method of copying, this method uses the on-chip data cache; and like the address mux method, the EPROM will need to be remapped from the bootstrap locations into another portion of the memory space of the i860 microprocessor. However, in this case, two copies of the code are needed in the EPROM: one copy with code bytes located on 8-bit boundaries and one copy with code bytes spaced for copying. As Table 5-1 illustrates, in the non-CS8 mode there are only six valid BE2#–BE0# combinations for the eight possible EPROM addresses. Therefore the second copy of the code needs to be spaced so that the first byte is at location 0 and the second byte is at location 3.

Table 5-1. Valid Addresses for SPACED copying

<table>
<thead>
<tr>
<th>Bus BE# Address</th>
<th>Valid Address 210 T or F Address</th>
<th>EPROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 000</td>
<td>T (Load 32-bit value)</td>
<td>0</td>
</tr>
<tr>
<td>1 001</td>
<td>F (16-bit access on odd boundary)</td>
<td>Invalid</td>
</tr>
<tr>
<td>2 010</td>
<td>F (two noncontiguous bytes)</td>
<td>Invalid</td>
</tr>
<tr>
<td>3 011</td>
<td>T (Load 1 byte)</td>
<td>3</td>
</tr>
<tr>
<td>4 100</td>
<td>T (Load 16-bit value)</td>
<td>4</td>
</tr>
<tr>
<td>5 101</td>
<td>T (Load 1 byte)</td>
<td>5</td>
</tr>
<tr>
<td>6 110</td>
<td>T (Load 1 byte)</td>
<td>6</td>
</tr>
<tr>
<td>7 111</td>
<td>T (Load 1 byte)</td>
<td>7</td>
</tr>
</tbody>
</table>

The data is first copied into the data cache by using the ld.1 or ld.b instructions to assert the proper BE2#–BE0# values. Once the data is in the cache it is read into a register and saved in the appropriate DRAM location. An untested example of the possible code for the “double copy load” method is illustrated in the following pages.

5.4.2.2 EPROM TIMINGS

The timing analysis assumes that the address multiplexor scheme is being used for the EPROM interface. The OE# input of the EPROM is connected to IORD# output of the I/O Control Logic. The following are the critical timings:

1. Address/CE# to Output (Tacc):

   Assumes that the multiplexed address bits are the last to become valid.

   \[ T\text{acc} \leq 3T\text{cy} \quad \text{Latch Propmax} \quad \text{Addr. Muxmax} \quad \text{i860 CPU data Setup} \]
   \[ \quad \text{Buffer Delaymax} \]

   \[ T\text{acc} \leq 3T\text{cy} \quad 35 \]

   For Tacc of 200ns @33 MHz Two Wait States

   Addr. Mux.: Address Multiplex Delay
2. OE# to Output (Toe):

Assumes zero-wait-state cycle.

Toe ≤ Tcy – PLDelk-to-output – Buffer Delay – i860 Microprocessor Setup

Toe ≤ Tcy – 28

For Toe of 85ns requires 3 wait states.

3. OE# High to Output Float (Tdf):

If READY# is asserted to the processor on the clock edge prior to IORD# going inactive, the EPROM data outputs can float as late as 60ns after the rising edge of IORD#. This means that EPROM data may be valid until as late as 10ns into the second T11 of the next cycle. The i860 CPU for writes can output data as early as 3.5ns from clock edge of the second T11. If this were allowed to happen it would result in data bus contention and would cause problems in the system. The IOTIME PLD eliminates bus contention by not allowing the next cycle to begin until all data outputs of the slave device float.

5.5 DMA INTERFACE RECOMMENDATIONS

Performing DMA (Direct Memory Access) transfers in an i860 microprocessor system requires deciding between two general approaches. One uses DMA in the conventional mode, performing data transfers directly to DRAM. The other technique uses an intermediate memory area. Each has important issues regarding performance, software complexity, and part count.

Any DMA design must take into account the on-chip caches. The i860 microprocessor caches use a write-back mechanism to minimize external bus traffic and increase performance. The on-chip caches are logical-address caches. External bus snooping is not provided since such cycles would conflict with internal accesses, reducing overall performance.

If DMA is permitted directly to main memory, then the software must insure that no data values associated with that memory area are in the caches. Either the I/O buffer is always marked non-cacheable or the data cache must be flushed before the DMA may read from the memory area. Both the code and instruction caches must be flushed after the DMA has written to memory if the i860 microprocessor might have an old copy of the data.

If DMA transfers directly to main memory and this I/O buffer is used directly by the application, it must also take into account the paging structure used in most systems. Paging breaks up the contiguous logical address space of the application into discontiguous memory addresses. The DMA device needs to transfer a stream of data to discontiguous addresses if that data stream crosses a memory page boundary.

Another approach to direct DMA is to allocate track buffers in memory which are always non-cacheable and occupy consecutive physical memory pages. The operating system can manage these buffers to speed up disk accesses since an entire track can be read in one revolution of the disk and kept for later disk accesses which usually want a
sector in the same track. The OS can copy data between this track buffer and the application. The copy will update all caches and handle discontiguous memory pages. DRAM-to-DRAM copy can be as fast as 56/67 Mbytes/sec.

Based on these issues, two general approaches are possible: low-cost, lower performance design using DMA directly to i860 microprocessor memory and a higher cost, higher performance approach using an intermediate memory area.

An 82380 is used in this example. It provides eight DMA channels, 17 interrupt inputs, and five 16-bit timers. The 82380 has a bus interface identical to that of the 386™ microprocessor. It uses a 2× clock. A 16 MHz 82380 requires the i860 microprocessor to run at 32 MHz. A 20 MHz 82380 can work with a 40 MHz i860 microprocessor.

The first approach is conventional. An 82380 DMA device with four extra data transceivers could perform DMA to memory. It can run directly off the i860 processor clock with special PLDs that convert the 82380 bus signals into DRAM commands. The i860 processor HOLD/HLDA signals are used. Each transfer requires at least four clocks on the i860 processor bus. More will usually be needed to allow access to real memory or I/O devices. For DRAM accesses by the 82380, the DRAM controller must perform them in pairs of clocks to match the 82380 bus timing. If the I/O device is not 32-bits wide, then the 82380 must either perform two-cycle transfers or else extra data transceivers are needed to route the I/O data to the correct part of the 64-bit data bus.

The 82380 can support paged memory systems via a set of second address and count registers per channel. After completing the transfer of one block, the DMA channel automatically switches to the next set of registers. The registers must be reprogrammed during the time of a page transfer. If the operating system cannot guarantee quick enough interrupt response, DMA transfers cannot cross a page boundary.

The second approach is to add a special I/O buffer memory between the I/O device and the i860 microprocessor. SRAMs make this easier. The 82380 performs DMA transfers into the SRAMs on an isolated bus. The SRAM area is large enough for all the simultaneous DMA transfers in progress at once. The i860 processor copies data between the SRAM space and DRAM. The copy operation updates the cache and handles page boundary crossing. This approach offers higher performance, because the DMA channels do not tie up the CPU bus when accessing the I/O device. The SRAM space should be 64-bits wide to maximize the copy speed.

The cost of using the I/O buffer memory is the SRAM devices and two or three address latches. Four data transceivers are still needed to buffer the data bus. Reads to the SRAM area must disable KEN#.

When copying data between the application and I/O buffer, use pfld instructions to prevent the data from flushing the data cache. Normal writes can be used, because a write miss does not cause a cache load. The copy function shown here achieves 1Mbyte/MHz data rate, assuming 4 wait states on first DRAM access and 1 wait state per SRAM access. At this rate, the time spent in copying data is negligible.
Note: This example has not been tested

///
/// Copy a block of data from the source to the destination.
/// Use pipelined floating-point loads to prevent data accesses
/// from flushing out data cache contents.
///
/// bcopy(source_ptr,dest_ptr,count8)
///
/// r16 on entry has pointer to the source, must be 8-byte aligned
/// r17 on entry has pointer to the destination, must be 16-byte aligned
/// r18 on entry has count of 8-byte words to transfer.
///
/// The transfers are performed on source and destination in series of
/// back-to-back operations to allow page mode accesses to DRAMs.
///
/// Assuming 3 wait states on the first access, the transfer speed is
/// 38 clocks per 64 bytes transferred for big blocks.
/// 56.1 Mbytes/sec at 33.33 Mhz
/// 67.4 Mbytes/sec at 40 Mhz
/// 0 bytes 8 clocks
/// 8-128 bytes 5+10n/8 clocks
///
/// bcopy:
///
/// addu   -16,r18,r20 // See if at least 16 transfers
/// shr    3,r20,r21 // Setup loop counter for 8 per loop
/// bc.t   do_long // Jump if long transfer
/// pfld.l (r16),f0 // Start reads on long transfer
///
/// Copy small block of data 8 bytes at a time
///
/// bte   r0,r18,exit // Jump if 0 count
/// adds  -1,r0,r19 // Setup BLA counter
///
/// dlast:
/// adds  -1,r18,r18 // Setup for BLA
/// subs  r17,r16,r17 // Form distance from source to destination
/// bla   r19,r18,sloop // Set LCC for next loop
/// adds  -8,r16,r16 // Setup for autoincrement addressing
///
/// sloop:
/// fld.l  8(r16)++,f16 // Get source and bump address
/// bla   r19,r18,sloop // Loop if more to do
/// fst.l  f16,r17(r16) // Write destination
///
/// exit:
/// bri   r1 // Return to caller
/// nop   // Nothing to do here
///
/// Start the pipelined floating-point reads
///
do_long:
  adds  -1,r0,r19       // Setup BLA counter
  pfld. 8(r16) +++,f0   // Read the next value
  bla   r19,r21,bloop    // Set LCC for next loop
  pfld.l 8(r16) +++,f0

bloop:
  pfld.l 8(r16) +++,f16  // Get value mod 0
  pfld.l 8(r16) +++,f18  // Get value mod 1
  pfld.l 8(r16) +++,f20  // Get value mod 2
  pfld.l 8(r16) +++,f22  // Get value mod 3
  pfld.l 8(r16) +++,f24  // Get value mod 4
  pfld.l 8(r16) +++,f26  // Get value mod 5
  pfld.l 8(r16) +++,f28  // Get value mod 6
  pfld.l 8(r16) +++,f30  // Get value mod 7
  fst.q  f16,(r17)       // Store 0-1 pair
  fst.q  f20,16(r17)     // Store 2-3 pair
  adds  64,r17,r17      // Update destination pointer
  fst.q  f24,-32(r17)    // Store 4-5 pair
  bla   r19,r21,bloop    // Loop if more to do
  fst.q  f28,-16(r17)    // Store 6-7 pair

  // Finish off last transfers to mod 8
  //
  pfld.l 8(r16) +++,f16  // Start last 5 reads
  pfld.l 8(r16) +++,f18  // Get value mod 1
  pfld.l 8(r16) +++,f20  // Get value mod 2
  pfld.l 8(r16) +++,f22  // Get value mod 3
  pfld.l 8(r16) +++,f24  // Get value mod 4
  pfld.l (r16),f26       // Get last three data words
  adds  8,r16,r16        // Update source pointer
  pfld.l -8(r16),f28     // Reread last source
  pfld.l -8(r16),f30     // Get value mod 7
  fst.q  f16,(r17)       // Put 0-1 pair into memory
  fst.q  f20,16(r17)     // Put 2-3 pair into memory
  fst.q  f24,32(r17)     // Put 4-5 pair into memory
  and   7,r20,r18        // Get remaining count
  fst.q  f20,48(r17)     // Put 6-7 pair into memory
  bnc.t dlast           // Jump if still work
  adds  64,r17,r17       // Update destination pointer
  bri   r1               // Else all done
  nop                          // Nothing to do here
Double copy load method:

// This program copies ROM code to DRAM. It assumes that BE2#:BE0# are
// wired to the ROM address bits 2:0 (least significant bits).

dramstart = 0x7eff0000  // non-cacheable alias for DRAM, where ROM
                     // copied code will go.
epromstart = 0xffff0000
epromend = 0xffffffff0

.copyrom:         // start copying EPROM contents to DRAM
    or l%epromstart,r0,r9
    orh h%epromstart,r9,r9
    or l%epromend,r0,r7
    orh h%epromend,r7,r7
    call flush_cache  // make sure we get a miss
    nop
    or l%dramstart,r0,r8
    orh h%dramstart,r8,r8

.loadbytes:
    ld.l 0(r9),r10   // cache data for bytes 0 & 8
    ld.b 0(r9),r10   // load byte 0 from cache
    call flush_cache // insure cache miss next time
    ld.b 8(r9),r18   // load byte 8 from cache
    ld.b 2(r9),r13   // cache data for bytes 3 & 11
    ld.b 0(r9),r13   // load byte 3 from cache
    call flush_cache // insure cache miss next time
    ld.b 8(r9),r19   // load byte 11 from cache
    ld.s 0(r9),r14   // cache data for bytes 4 & 12
    ld.b 0(r9),r14   // etc.
    call flush_cache
    ld.b 8(r9),r20
    ld.b 1(r9),r15   // cache data for bytes 5 & 13
    ld.b 0(r9),r15
    call flush_cache
    ld.b 8(r9),r21
    ld.b 0(r9),r16   // cache data for bytes 6 & 14
    ld.b 0(r9),r16
    call flush_cache
    ld.b 8(r9),r22
    ld.b 3(r9),r17   // cache data for bytes 7 & 15
    ld.b 0(r9),r17
    call flush_cache
I/O INTERFACING

```assembly
ld.b 8(r9),r23

store::
    st.b r10,0(r8) // store byte 0 to DRAM
    st.b r13,1(r8) // ' ' ' 3 ' ' ' 4 ' ' ' 5 ' ' ' 6 ' ' ' 7 ' ' ' 8 ' ' ' 9 ' ' ' 0
    st.b r17,5(r8)
    st.b r18,6(r8)
    st.b r19,7(r8)
    st.b r20,8(r8)
    st.b r21,9(r8)
    st.b r22,0xa(r8)
    st.b r23,0xb(r8)
    xor r7,r9,r0 // check for end of ROM
    bc load_reset // branch if end of ROM
    addu $0x10, r9, r9 // increment ROM counter
    br loadbyts // copy another 12 bytes
    addu $0xc, r8, r8 // increment DRAM counter

load_reset:: // Load the reset code to DRAM

w1=$0xa0000000 // This hex code is for the reset branch
w2=$0xa0000000 // branch from $0xffffffff to $0xffff0008
w3=$0x6bfffc03f // (Two nop's, then branch)
w4=$0xa0000000 // Sequence = nop; nop; br; nop

resxc_start=$0x7effffff // non-cacheable alias of feffff00
dramc_start=$0xffff0000 // cacheable DRAM that will be mapped
                      // over EPROM space after boot
byte_bucket=$0x7ffff0010 // ROM location

or 1%resxc_start,r0,r4 // bottom of reset branch code
orh 1%resxc_start,r4,r4
or 1%w1,r0,r5
orh 1%w1,r5,r5
st.l r5,0(r4) // store word 1
addu 4, r4, r4 // increment counter
or 1%w2,r0,r5
orh 1%w2,r5,r5
st.l r5,0(r4) // store word two
addu 4, r4, r4
or 1%w3,r0,r5
orh 1%w3,r5,r5
st.l r5,0(r4) // store word three
addu 4, r4, r4
or 1%w4,r0,r5
```

5-23
orh h%w4,r5,r5
st.l r5,0(r4) // store word four

// The next line must be hand patched after every reassembly

or 0xf174,r0,r1 // fffff174 is mask_cs8
orh 0xffff,r1,r1

// We go through the warp drive section twice to insure that
// it gets completely cached. We must be executing from
// cache when we zero CS8 and the boot bit.

or l%byte_bucket,r0,r5
orh h%byte_bucket,r5,r5
ld.c dirbase,r4 // get dirbase contents

warp_drive::
st.b r0,0(r5) // does nothing first time
st.c r4,dirbase // does nothing first time
bri r1 // go to mask_CS8 the first time
nop // go to fff00000 second time

mask_cs8:::
or 0xff7f,r0,r3 // kill cs8 bit
orh 0xffff,r3,r3
and r3,r4,r4
or 0x20,r4,r4 // invalidate code cache bit
xorh h%byte_bucket,r5,r5 // zero r5's high 16 bits
orh 0x0100,r5,r5
or l%dramc_start,r0,r1 // r1 now has DRAM start address
orh h%dramc_start,r1,r1
br warp_drive // r5 has the boot port address
nop // r4 has CS8 bit reset

// The following flush procedure is from the i860™ Programmer's Reference
flush_cache:::
FLUSH_P = 0x7f000000-32
//rw=r24, rx=r25, ry=r26, rz=r27
mov r1,r2
ld.c dirbase,r27
or 0x800,r27,r27
adds -1,r0,r25
call D_FLUSH
st.c  r27,dirbase
or   0x900,r27,r27
call  D_FLUSH
st.c  r27,dirbase
xor  0x900,r27,r27
mov  r2,r1
bri  r1
st.c  r27,dirbase

D_FLUSH::
or   1%FLUSH_P,r0,r24
orh  h%FLUSH_P,r24,r24
or   127,r0,r26
bla  r25,r26,D_FLUSH_LOOP
ld.l  32(r24),r0

D_FLUSH_LOOP::
bla  r25,r26,D_FLUSH_LOOP
flush 32(r24)++
bri  r1
ld.l  -512(r24),r0
.end
Graphics Subsystem
Example
CHAPTER 6
GRAPHICS SUBSYSTEM EXAMPLE

6.1 INTRODUCTION

Computer graphics technology has developed rapidly in recent years and has transformed the computing environment. Graphics provides visualization — the ability of computers to create revealing, life-like images from hard-to-interpret numerical data. Graphics offers user-friendliness to ease the man-machine interface, and it opens new applications in science, engineering and the arts. Advances in graphics hardware technology are catalysts for innovation. Raster display technology, low-cost, high-speed display memory and powerful, intelligent graphics processors are key elements to recent changes. A decade ago, high resolution graphics was reserved to centralized computing facilities. Today, home and office computers provide even better graphics at a fraction of the cost.

The underlying trend in computer graphics development is toward better display and pixel resolution. Display requirements for solids modeling and visualization are stringent because of the need to accurately represent smooth shaded objects. Graphics workstations are a new class of machines for graphic-intensive applications. They combine medium- to high-resolution color graphics with high-speed computational capability. Applications may call for real-time manipulation of complicated images composed of close to a hundred thousand polygons. These 3-D graphics applications demand high MFLOPS (millions of floating-point operations per second) performance. Object modeling, transformation and rendering are common applications that usually require supercomputer power.

6.2 GRAPHICS AND THE i860™ MICROPROCESSOR

The i860™ microprocessor’s architectural features provide high performance graphics capability. Floating-point power, parallel execution units, high integration and dedicated 3-D graphics hardware combine to provide supercomputing graphics performance and capability.

The processor provides an integer operation and up to two floating-point operations per clock cycle. Pipelined floating-point multiplication and addition units operate simultaneously using special dual operation instructions. This speeds matrix arithmetic and vector computation to provide a peak performance of 80 single-precision or 60-double precision MFLOPS at 40 MHz.

The i860 microprocessor supports 3-D graphics operations such as hidden surface elimination and Gouraud shading. It operates on 8, 16, or 32-bit pixels. Its high speed, 64-bit data bus delivers a peak 160 megabytes per second with zero-wait-state accesses. Dual instruction mode allows floating-point or graphics operations to execute in parallel with pixel loading and storing. Scoreboarding can provide continuous execution during cache-miss processing of data reads.
6.2.1 Processor Bus Bandwidth

In real-time applications, frames must be refreshed at least 10 times per second. Bandwidth requirements vary according to display and pixel resolution. Nearly 40 Mbytes/sec bandwidth must be dedicated to memory refresh to provide real-time graphics on a 1,280 \( \times \) 1,024 pixel monitor with 24-bit resolution. Additional data transfers for modeling, transformation, hidden surface elimination and shading increase the bandwidth requirement.

At 40 MHz, the i860 microprocessor provides 160 megabytes per second bus bandwidth with zero-wait-state cycles. This allows for real-time manipulation of high-resolution 3-D images.

6.3 3-D GRAPHICS EXAMPLE

This example outlines a graphics frame buffer daughter card for an i860 microprocessor evaluation vehicle. The example does not represent an ideal design but is instead employed to demonstrate the processor's 3-D graphics capabilities. The concepts employed here can be extended to complete systems where more board space and dedicated hardware/software support permit greater sophistication.

This evaluation vehicle is designed as a frame buffer extension to an i860 microprocessor based system. The processor connects to the frame buffer board through the expansion bus. The frame buffer is mapped into the expansion space as allocated by the CPU core.

6.3.1 Features

The following features are included in the frame buffer example:

- 1,024 \( \times \) 768 RGB Display with 16-Bit Pixel Resolution
- Double-Buffering at Two Megabytes Per Buffer
- 33 MHz with 40 MHz Upgrade Path
- PLD-Based VRAM Controller
- VRAM Control Options for 100ns and 80ns VRAMs
- PLD-Based Noninterlaced CRT Control

Dedicated to the three colors, red, green and blue, are 6, 6, and 4 bits per pixel respectively. Sixteen-bit pixel resolution allows a double-buffered scheme requiring only four megabytes of total display memory for a 1,024 \( \times \) 768 pixel display. It also allows four pixels per 64-bit load/store operation and four pixels per computation. Transformation and refresh rates are higher than those obtained with a 24-bit or 32-bit pixel resolution.

6.3.2 Testing

This example has been tested for the operational mode that uses 100ns VRAM. Refer to Section 6.5.1.1. PLD codes are included in Appendix A.
6.4 SYSTEM OVERVIEW

The frame buffer board is composed of three major sections: VRAM control, CRT control and expansion bus interface. The VRAM controller controls four megabytes of video RAMs which are divided into two display buffers. While the i860 processor accesses one buffer, the CRT may display the other buffer. Refer to the block diagram in Figure 6-1.

6.4.1 Expansion Bus Interface

Frame buffer logic resides on the expansion bus. The key signals are listed below:

- 64-bit buffered data bus
- Bits A23-A3 of the i860 microprocessor address bus
- i860 microprocessor control signals such as ADS#, READY# and NA#
- Decoded signals from the baseboard logic such as EXPSEL#
- Power supply and ground signals

Refer to the schematics in Appendix A for the complete list of signals.
6.4.2 Data Transceiver/Latch Control

Data is transferred to and from the i860 microprocessor through data transceivers/latches on the i860 microprocessor board. Expansion data control signals are generated to control the logic.

6.4.3 Address Transceiver/Latch

Address bits A20-A3 are used to generate row and column addresses during data access cycles. Bits A20-A12 are buffered and then latched as row address by VRAMs. Bits A11-A3 are latched externally as column address when a VRAM cycle is detected. Including bit 21 (which selects display buffer #0 or #1), four megabytes of address space are provided.

6.4.4 VRAM Control

VRAM control provides VRAM control signals for read/write cycles, refresh cycles, and RAM-to-SAM transfer cycles. VRAM read/write cycles are synchronized with pipelined cycles to the DRAM subsystem on the i860 microprocessor board.

6.4.5 Serial Row/Column Address Generation

RAM-to-SAM transfer for the serial port occurs during horizontal blank periods, and row addresses increment accordingly. Each row transfer provides two screen lines, and row addresses increment every other horizontal blank time. A column address latched during SAM transfer indicates the origin of data within the SAM register. All but the most significant bit in the column address are zero. The most significant bit alternates between one and zero. If the bit is a one, the serial shift from the serial port originated from the middle of the SAM register. This occurs during a RAM-to-SAM transfer of display data for the second screen line stored in the second half of the same VRAM row. The row address is not incremented in this case.

6.4.6 Double Buffering

Double buffering reduces flickering and partial image update. Here, each buffer consists of two megabytes of video RAM.

6.4.7 Expansion Interrupt/Buffer Switch

Upon reset, buffer #0 is enabled. When ready to display data in a second buffer, the processor may read or write to a location where A22 is high. Actual buffer switching occurs in the subsequent vertical trace. A low value on A3 indicates that display data comes from buffer #0; a high value indicates that data comes from buffer #1. The processor is interrupted when vertical retrace occurs.
6.4.8 CRT Timing Generation
Blank and sync signals are generated with PLD/TTL logic, and clocking is derived from the 64 MHz pixel clock. The 16 MHz serial clock clocks data out of the VRAM serial ports.

6.4.9 Pixel Serializer
Pixel resolution is 16 bits/pixel, and four pixels are clocked out with each rising edge of the serial clock. Pixels are loaded into shift registers at serial clock rate but shifted out of the registers at pixel clock rate.

6.4.10 Video DACs
Video DACs convert digital video data into analog video data at the pixel clock rate. The sync-on-green mode of the color monitor provides synchronization.

6.5 OPERATION

6.5.1 VRAM Control
VRAM control logic provides all control signals for VRAM operation, including serial port accesses.

6.5.1.1 SPEED MODE
Two operation modes are designed to use 100ns and 80ns VRAMs. They are designated as 1-wait-state-write and 0-wait-state-write modes respectively. The 100ns VRAMs are used and tested in initial prototyping. (PLD code for the 0-wait-state mode is not included in the appendix.) They require one wait-state for writes and two wait-states for reads in page mode. (PLD codes are included in Appendix A.) 80ns VRAMs are needed to support zero-wait-state writes and 1-wait-state reads. Refer to the timing diagrams in Figure 6-2 and 6-3 for one-wait-state-write mode operation.

6.5.1.2 PROCESSOR-INITIATED CYCLES
The frame buffer board shares processor and latched data control buses with the i860 microprocessor board. VRAM cycles may be started by the VRAM controller when the VRAM space is selected and DRAM busy signal (DRMBSY#) is deasserted (that is, pipelined cycles are completed). The expansion busy signal (EXPBSY#) is also asserted to prevent other cycles from starting while VRAM cycles are pending.

Once the first VRAM cycle is started, VRAMs remain in page mode during near (REN# asserted) cycles. Reads are pipelined with NA#. A new cycle can start when NA# is activated in pipelined mode or when READY# is activated in nonpipelined mode. Idle or far cycles put VRAMs in precharge mode. Row addresses are
latched by VRAMs on the falling edge of RAS#, and column addresses are latched on the falling edge of CAS#. Subsequent near cycles to the VRAM space need to supply column addresses only. Near cycles are indicated by the i860 microprocessor NENE# pin.

In read cycles, data is latched by the data latch on the i860 microprocessor board before being read by the processor. This provides sufficient setup and hold times. In write cycles, data is also latched before being written into VRAMs. The data latch is controlled by expansion bus data control lines. Latch latency requires that write data be latched to maintain one-wait-state operation in successive VRAM write cycles (one-wait-state-write mode operation). Latch signals require at least one clock period to recover.
When a write cycle is immediately preceded by a read, data is not driven by the processor until one clock after READY# is returned. This is illustrated in Figure 6-2 and 6-3.

6.5.1.3 REFRESH/RAM-TO-SAM TRANSFER CYCLES

The refresh and serial register (SAM) transfer VRAM cycles are not initiated by the processor. The cycles occur simultaneously in both VRAM banks and have priority over processor-initiated cycles. Outstanding processor-initiated cycles resume following refresh/SAM cycles. Refresh cycles have priority over RAM-to-SAM cycles.

Figure 6-4 illustrates a far cycle (NENE# not active) followed by a RAM-to-SAM transfer request and refresh request (REF). The refresh cycle begins as soon as RAS# precharge time has been met. The RAM-to-SAM request (TRQ) arrives one clock before the refresh request. Request lines are not sampled until the end of the precharge period where both lines are active. These cycles are equally important in displaying the correct
images continuously. RAM-to-SAM requests occur during video blanking time and minor delay is tolerable. The RAM-to-SAM cycle begins as soon as RAS# precharge time for the refresh cycle is satisfied. CAS-before-RAS refresh VRAM mode is used to generate a refresh address internally.

A RAM-to-SAM read transfer is requested when DT/OE# is asserted on the falling edge of RAS#. In real-time read transfers without RAM-to-SAM transfers, the transfer must remain active until CAS# has been deactivated. Although not indicated in the diagram, WE# must be deasserted during refresh/RAM-to-SAM transfer cycles. While external address generation is not required in refresh cycles, a scan line address is required for RAM-to-SAM transfers. The address setup and hold mechanisms are similar to those in a regular cycle.

Figure 6-4. Refresh/RAM-to-SAM Transfers
6.5.2 Expansion Bus Interface

The expansion bus allows the i860 microprocessor to access the frame buffer. The schematics provide a complete list of bus signals. Expansion address space spans 8 megabytes. Mapping is illustrated in Figure 6-5.

6.5.2.1 EXPANSION SELECT

The expansion select signal (EXPSEL#) and signals A21 and A22 are needed to decode access to the expansion address space in two megabyte increments (refer to Figure 6-5). Low levels on EXPSEL# and A22 signal VRAM accesses. A21 distinguishes between display buffers #0 and #1 when the random port (processor side) of the VRAMs is accessed. When A22 is high for an access to the expansion space, A3 enables the serial port (CRT side) of buffer #0 or buffer #1. Buffer switching occurs during vertical retrace. Although available, A23 is not used in decode logic; VRAMs and unpopulated space are assigned to two eight megabyte regions.

6.5.2.2 DATA TRANSCEIVER/LATCH SHARING

Four signals control the data transceivers/latches used in VRAM data accesses. The ERDL# and EWDL# signals latch read and write data. ERDE# is used for transceiver enable, and EWDE# is used for transfer direction control. ERDL# and EWDL# must meet the setup time requirement relative to a delay clock (delayed from CLK) on the i860 microprocessor board. Data is latched in the latter part of CLK. EWDL# activates the write data latch signal.

![Figure 6-5. Expansion Space](image-url)
### 6.5.3 CRT Timing Logic

The 64 MHz pixel clock (PCLK) keys all timing signals. A divide-by-16 clock is generated to clock the horizontal blank signal, HBLANK. As shown in Figure 6-6, each horizontal active period supports the 1,024 pixel horizontal display. Display active times are set at 16.5μs while inactive times are set at 4.5μs. The horizontal sync signal, HSYNC# has a front porch of 0.5μs, an active time of 1.5μs and a back porch of 2.5μs. The horizontal frequency is approximately 48 KHz. These timings are for the Hitachi Super-scan (model CM2085MV); they must be adjusted for use with a different monitor.

The vertical blank signal, VBLANK# is active low and remains active for 45 HBLANK periods. To provide a 768 line display (by remaining inactive for 768 HBLANK clock periods), the vertical retrace rate becomes approximately 60 times/sec. The front porch, active time and back porch for the vertical sync signal (VSYNC#) are 6, 6 and 33 HBLANK periods respectively (refer to Figure 6-6). Zeros are shifted into the shift registers during blanking periods. When blanking ends, display data is shifted out of the VRAMs and loaded into the shift register by the load/shift signal (LDSR#). Undefined data is not displayed. Figure 6-7 provides serial data clocking timing. The serial clock (SC) is held low after all required data has been moved from the shift registers at the end of the display period.

The pixel clock is divided by 16 to generate the CDIV16 clock used in most CRT timing logic. Final timings are synchronized to the pixel clock to ensure correct blanking by the video DACs where digital data, composite sync and blank signals combine to generate RGB signals.

---

**Figure 6-6. Blank and Sync Signals**

Notes
1) H = horizontal retrace period
2) VSYNC# is not generated but combined with HSYNC# to generated SYNC#

240330-48
CRT timing relationships are illustrated in Figure 6-8. The horizontal blank (HBLANK) is generated by counting COUNT, and the vertical blank (VBLANK#) is generated by counting HBLANKs. The counters are reset by HCLR# at the end of horizontal display periods and by the VCLR# (not shown) at the end of each display periods.

6.5.4 Schematics Description

Refer to Appendix A, page 24, for the schematics.

Sheet #1

This sheet contains address transceivers and latches.
Sheet #2

This sheet contains half of the VRAMs in Buffer #0.

Sheet #3

This sheet contains half of the VRAMs in Buffer #0.

Sheet #4

This sheet contains half of the VRAMs in Buffer #1.

Sheet #5

This sheet contains half of the VRAMs in Buffer #1.

Sheet #6

This sheet contains VRAM control logic. RAS1# and RAS0# control Buffers #1 and #0 respectively. RAS1# is activated when A21 is high; RAS0# is activated when A21 is low. RAS1# and RAS0# are combined to control the remaining logic.

CAS2#, CAS1# and CAS0# are activated together; they are generated to share loading.
PRECH# controls RASx# precharge time: three CLKs for far cycles; four clocks for refresh/SAM to support CAS#/DTOE# before RAS#. REFC# controls RAS# active time (five CLKs) during refresh/SAM. TRFQ# is activated when refresh (REF) or SAM request (TRQ) is active. RASDEL# is to delay RAS# active edge by one clock when TRFQ# is found active.

ROWE#/COLE# and SRAE#/SCAE# are address enabled for regular VRAM and serial port row/column addresses.

CPEND# is activated when VRAM cycles are detected. They are deactivated when CAS# is deactivated and no new VRAM cycle is detected. WDEL# delays logic activation when a write follows a read. NCLK# is added to minimize the number of inputs (related to WDEL#) to the RDY# (processor ready signal) PLD.

ERDL# is activated in the clock that VRAM data is valid. It causes RDL activation in the core logic in the same CLK.

DTOEx# enables VRAM output for read cycles and signals SAM transfer on the RASx# falling edge. CLRTRFQ# clears the refresh request (REF) while CLRTRQ# clears the SAM request (TRQ). ERDE# and EWDE# control direction (DRMDIR#) and enabling (DRMEN#) of data bus transceivers on the i860 microprocessor board. Refer to timing diagrams in the text for details.

Sheet #7

This sheet contains additional VRAM control and miscellaneous logic. DREF, PREF and TREF are decode signals to the product terms and determine the number of inputs required for DTOEx#, CASx# and RASx#, respectively.

EWDL# activates WDL in the core logic in the same CLK to capture processor write data. In the case of zero-wait-state writes, it is activated whenever a VRAM cycle is detected regardless of the read/write or near/far state.

SE1# and SE0# are activated when an access to the upper portion of the expansion space is detected. A low value on A3 activates SE0#; a high value activates SE1#. Upon reset, SE0# is activated. SE1# and SE0# cannot be active at the same time.

RDY# is activated for VRAM cycles and for buffer selection accesses to the upper portion of the expansion space. RDY# is always in the high state when floated. VSELX1 and VSELX0 are state variables for these access cycles.

EXPBSY# is activated when VRAM cycles are in progress. CBUSY# combines the various busy signals.

WEx# are write enable signals for VRAMs. They must be inactive to allow CAS-before-RAS refresh and SAM transfer on the RASx# falling edge. REFC# deactivates WEx#.

Sheet #8

LDSR# loads serial data into the shift registers. Data to be clocked out by PCLK. SCLR# clears register contents once all needed data has been shifted.
Sheet #9

The VOG signal carries composite sync information to support sync-on-green. When BLANK# is active, remaining shift register data is not displayed.

Sheet #10

PCLK is divided by 16 to generate CDIV16 for composite blank and sync clocking. SBLANK# is the composite blank signal synchronized to PCLK. The final blank signal BLANK# is used to clear the screen after remaining pixels are cleared from the shift register. The composite sync (SYNC#) and BLANK# are used by the video DACs.

Sheet #11

REF and TRQ generation. TRQ is activated by HBLANK; REF by REFREQ from the expansion bus.

Sheet #12

Terminations for VRAM control and for the expansion bus CLK signals (CLKC and CLKG).
MULTIBUS® II and the
i860™ Microprocessor
CHAPTER 7
MULTIBUS® II AND i860™ MICROPROCESSOR

The i860™ microprocessor provides supercomputing performance and capability that can be provided on a standard bus platform. This allows systems integrators to take advantage of rapidly advancing CPU technology while preserving their investment in existing hardware. Boards based on the MULTIBUS® II system bus and on the i860 microprocessor can be added to existing systems or can be the basis for a new system design.

This chapter outlines an example of such a design. Schematics for the design are to be found in Appendix B. This example has not been tested.

7.1 i860™ MICROPROCESSOR CPU BOARD

Figure 7-1 shows a block diagram of a board designed around the i860 microprocessor and MULTIBUS II system bus. The main features include the following:

- MULTIBUS II System Bus Interface
- i860 Microprocessor

![Figure 7-1. System Block Diagram of i860™ Microprocessor Based MULTIBUS® II Board](image-url)
MULTIBUS® II AND THE i860™ MICROPROCESSOR

- DRAM Main Memory
- Local I/O Devices
  - 82380 Integrated Systems Peripheral
  - 82510 UART
  - SBX Connector
  - Single 8-bit Boot EPROM
- DMA Control and SRAM Message Area
- Memory and Graphics Expansion Connector

7.2 MULTIBUS® II SYSTEM BUS STANDARD

The MULTIBUS II system bus standard is a processor-independent bus architecture with full distributed multiprocessor support. The standard defines a 32-bit parallel system bus with a maximum throughput of 40 Megabytes per second. The parallel system bus is isolated from the CPU local bus to allow the i860 microprocessor to access memory on its 64-bit data bus. The parallel system bus handles interprocessor communications and accesses to I/O devices not dedicated to a single CPU board.

The MULTIBUS II system bus architecture also includes the system expansion (iSBX™) I/O bus. Refer to the IEEE 959 specification for a full description of this bus.

7.2.1 Parallel System Bus (PSB)

The parallel system bus (PSB) is optimized for standardized interprocessor data transfer and signalling. Its burst transfer capability provides a maximum sustained bandwidth of 40 megabytes per second for high-performance data transfers. A hardware recognized data type, called a packet, is used to ensure consistent and reliable transfers between different system boards.

The PSB supports four address spaces for each bus agent (board that encompasses a functional subsystem). Conventional I/O and memory address spaces are included, as are two address spaces that support system functions:

- A 255-address message space supports message passing. Microprocessors typically perform interprocessor communication inefficiently. Message passing allows two bus agents to exchange blocks of data at full bus bandwidth without microprocessor supervision. An intelligent bus interface capable of message passing shifts the burden of interprocessor communication away from the processor and into hardware dedicated to this task, thus enhancing overall system performance.

- An interconnect space allows geographic addressing — the identification of any bus agent (board) by slot number. Every system based on the MULTIBUS II system bus contains a central services module (CSM) that provides system services for all bus agents residing on the PSB bus. These services include uniform initialization and bus timeout detection. The CSM may use the interconnect space registers of each bus agent to configure the agent dynamically. Stake pin jumpers, DIP switches and other hardware configuration devices can thus be eliminated.
Three types of bus cycles define activity on the PSB bus:

- **Arbitration Cycle** — Determines the next owner of the bus. This cycle consists of a resolution phase in which competing bus agents determine priority for bus control. It also includes an acquisition phase in which the bus agent with the highest priority initiates a transfer cycle. This cycle overlaps other cycles allowing agents to transfer bus control on back-to-back cycles.

- **Transfer Cycle** — Performs a data transfer between the bus owner and another bus agent. In the request phase, address control signals are driven. In the reply phase, two agents perform a handshake to synchronize the data transfer. The reply phase is repeated and data transfers continue until the bus owner ends the transfer cycle.

- **Exception Cycle** — Indicates that an exception (error) has occurred during a transfer cycle. In the signal phase, an exception signal from one bus agent causes all other bus agents to terminate any arbitration and transfer cycles in progress. In the recovery phase, the exception signals go inactive. A new arbitration cycle can begin on the clock cycle following the recovery phase.

Figure 7-2 illustrates how the timing of these cycles overlap.

### 7.2.2 Message Passing Coprocessor

The interface from the processor’s local bus to the PSB can be simplified with the Intel 82389 message passing coprocessor (MPC). The MPC has been designed for message passing protocols of the MULTIBUS II system bus architecture. It participates in the entire PSB bus protocol and performs bus arbitration, transfer control, error detection and reporting, and parity generation and checking. These functions occur independently of the host CPU.

The MPC decouples local bus activities from interprocessor communications over the PSB bus. The decoupled bus approach has two advantages:

- Resources that would be held in wait-states while dedicated bus access arbitration is underway are instead free. This parallelism increases system performance.

- The bandwidth of one bus does not limit the transfer rate of another. Each bus can perform full-speed, synchronous transfers.

As shown in Figure 7-3, the MPC signals can be divided into three functional groups:

- **MPC Interface to PSB**
- **Local bus interface**
- **DMA interface**

These signal groups are discussed below.

#### 7.2.2.1 MPC INTERFACE TO PSB

The primary functions of PSB interface signals are arbitration and system control. Five bidirectional arbitration signals (ARB5-ARB0) are used during reset to read card-slot ID and arbitration ID from the CSM. During arbitration, these signals output the arbitration ID for priority resolution. Bus request (BREQ#) is a bidirectional signal.
(It should not be confused with the BREQ# i860 microprocessor signal.) Each bus agent asserts BREQ# to request control of the bus and samples BREQ# to determine if other agents are also contending for bus control.

Bus error (BUSERR#) is a bidirectional signal that a bus agent sends to all other bus agents when it detects a transfer cycle parity error. The CSM sends the bus timeout signal (TIMOUT#) to all bus agents when a bus cycle fails to end within a prescribed time period.

Ten system control signals (SC9#–SC0#) coordinate transfer cycles. The MULTIBUS II Architectural Specification defines each signal. Directional enables (SCOEH and
SCOEL) are provided to let transceivers buffer these bidirectional signals. The MPC checks byte parity lines (PAR3–PAR0) are for incoming operations and set the parity lines for outgoing operations.

Other PSB signals are reset (RST#), reset-not-complete (RSTNC#) and ID latch (LACHn#, n = slot number). These signals are used only during system initialization.

The MPC coordinates interrupt handling for a bus agent on the PSB bus. Interrupts are implemented as virtual interrupts in the message space. To send an interrupt message, the processor writes to the MPC to indicate the source destination and message type. The MPC coordinates the interrupt message transfer.

The PSB interface consists of the multiplexed address/data bus (AD31#–AD0#). The MPC controls external buffers used to drive the PSB. As a requesting agent, the MPC drives addresses and data at appropriate times. As a receiving agent, the MPC monitors the address/data bus for its address. When it recognizes one of its own addresses, the MPC performs the required handshake and reads the message into the message queue.
The MPC then, if necessary, interrupts the i860 microprocessor to indicate that the message is pending in the queue. The processor reads the message and services the interrupt accordingly.

7.2.2.2 MPC LOCAL BUS INTERFACE

The local bus interface of the MPC is like that of any other simple I/O device consisting of select lines, address signals and read and write control lines. To support message passing, MPC registers have various functions. They are accessed by asserting REGSEL# and the appropriate register address while performing a read or write cycle. Among other functions, the registers are used to program data message transfers, to receive and send control transfers and handle errors.

7.2.2.3 MPC DMA INTERFACE

The DMA interface of the MPC has two channels that use the standard DREQ (DMA Request) DACK (DMA Acknowledge) hardware transfer protocol. The two channels are dedicated to the MPC one as the input channel and one as the output channel. Each has its own control lines and operates independently.

MULTIBUS II uses a message passing protocol for data transfer. Control transfers (also called unsolicited messages) are up to 32 bytes long, and data transfers (also called solicited messages) are up to 16 Mbytes long and are split into 32 byte packets by the sending and receiving MPCs. The sending and receiving DMA controllers do not know that the data is being packetized on the system bus. This bus bandwidth preserving feature does not affect local data transfers. The MPC has 32-byte internal FIFOs for packaging data before sending it on the MULTIBUS backplane.

The board includes an SRAM message area to isolate back plane data rates from the processor's local bus. For a solicited outgoing message, the i860 microprocessor transfers messages into the SRAM at a high rate. The DMA channel then transfers messages from the SRAM to the MPC's outgoing message FIFO. For a solicited incoming message, the DMA channel transfers the message into the SRAM buffer and signals the processor when it has completed. The processor can then very quickly transfer from the SRAM area into main memory. As discussed later, using the SRAM as a dedicated DMA buffer area has several advantages over DMA directly out of main memory.

7.3 i860™ MICROPROCESSOR BUS INTERFACE

The i860 microprocessor has a synchronous interface with nonmultiplexed address and data buses. The data bus is 64 bits wide and the address bus provides 32-bit addressing. Addressing consists of 29 address lines and separate byte enable for each of eight data bytes. The bidirectional data bus can accept or drive new data on every other clock, yielding a bandwidth of 160 megabytes per second at 40 MHz. The bus allows two levels of pipelining that may be selected on a cycle by cycle basis. In pipelined mode, a new cycle is started before earlier cycles have completed. Chapter 3 provides a complete description of the i860 microprocessor bus interface.
7.4 DRAM SYSTEM

The memory system consists of static column mode, noninterleaved, parity checked DRAMs. The processor's pipelined bus and address and data latches combine to hide DRAM latency for most accesses. Using 80 nanosecond static-column DRAMs, the memory system can supply the maximum data bandwidth required by the processor.

The board allows up to four megabytes of DRAM on the base board. DRAMS are static column, 80 nanosecond, 256K × 4 chips in ZIP sockets. Parity DRAMS are 1M × 1. An additional four megabytes of DRAM can be added to the expansion connector without additional parity DRAMS. Chapter 4 provides more information about the DRAM interface.

7.5 LOCAL I/O SYSTEM

The I/O system consists of an integrated systems peripheral (82380), a serial port (82510), one 8-bit boot EPROM and an iSBX bus connector. The 82380 integrated systems peripheral includes timers (8254), two interrupt controllers (8259 master and slave) and eight DMA channels. The I/O system is memory mapped.

The serial port can be used in a polled or interrupt mode. It also suitable as a console monitor. Timers are clocked from a reduced version of the serial controller's oscillator module. A control port enables and disables the timers. Timer outputs are connected as 8259 interrupts. The master and slave 8259 chips accommodate 15 interrupt sources.

Four of the DMA channels are used. Two are for the MPC interface, and two are for the iSBX bus connector. The DMA channels can transfer into or out of the SRAM message area.

7.5.1 82380 Integrated Systems Peripheral

The 82380 integrated systems peripheral serves both as a slave I/O device and as a bus master DMA controller. The 82380 contains four 16-bit programmable timers and has connections for 15 external interrupts. In addition, there are five internal interrupts that can be used with the DMA channels and the timers.

The 82380 uses a double frequency clock. This allows a 40 MHz CPU to operate synchronously with a 20 MHz 82380. At reset, a phase clock is generated and used by the control logic for 82380 accesses. The port addressing and data bus requirements of interfacing the 64-bit i860 microprocessor to the 32-bit 82380 are discussed in Chapter 5.

The DMA channels are programmed with the 82380 in slave mode. Once programmed, the DMA channels become bus masters to complete transfers. To gain control of the bus, the 82380 asserts HOLD. When HOLDA is returned to the 82380, it initiates the DMA transfer. Transfers continue until completion or until HOLDA is deasserted.
7.5.2 iSBX™ Bus Connector

An iSBX bus connector is also included. The DMA control interface can be configured as defined in the iSBX bus specification. Several jumpers are provided for connection configuring. To access 8- and 16-bit iSBX bus devices, the processor must address them on eight byte boundaries.

DMA transfers with the iSBX bus are supported. Transfers are with the SRAM message area. The DMA controller supports byte assembly and unassembly from 8 or 16 bit to 32 bits.

7.6 DMA CONTROL AND THE SRAM MESSAGE SYSTEM

DMA controllers benefit systems that have peripherals requiring block memory transfers. In this system, DMA channels are used for incoming and outgoing MULTIBUS II system bus messages and for I/O devices on the iSBX bus connector. Once a DMA channel is programmed, the processor remains uninterrupted while block transfers complete.

DMA performance depends on the DMA controller transfer rate and the effect of the DMA controller on processor memory bandwidth. The relative influence of these factors varies according to the task at hand. If the processor waits for DMA data, then transfer rate is most important. In a multitasking environment, the DMA effect on bandwidth is more critical. A multitasking system puts a task waiting for the DMA to sleep and continues to execute other tasks. Overlapping tasks in this way improves overall system performance. Any DMA method must also consider the write-back caching protocols of the CPU and that the physical memory is arranged as pages.

I/O system devices are slow relative to the processor's local bus. The DMA controller is separated from this bus by address and data buffers. This allows DMA memory accesses to occur independently of processor memory accesses. The memory portion of DMA transfers accesses the SRAM message area. Direct transfers to main memory would have to place the processor in a hold state, could not cross page boundaries and would require the operating system to maintain memory consistency between the caches and the DMA areas of memory. Transfers into the SRAM area do not initiate a hold, but they do require the processor to perform a main memory data transfer once the DMA transfer completes. The processor retains full memory bandwidth during DMA transfers to the SRAM area and may continue to execute tasks. Transfers between SRAM and DRAM area execute very quickly because SRAM and DRAM accesses occur at full bus bandwidth. DMA transfers into the SRAM can be programmed to transfer blocks larger than the 4 Kilobyte page size.

7.6.1 DMA Channels

The 82380 is the system DMA controller and contains eight DMA channels. DMA interface devices use the standard DREQ/DACK protocol. Bus control logic of the 82380 is identical to the 386™ microprocessor and similar to the i860 microprocessor. The 82380 can perform data assembly and disassembly for 8- and 16-bit DMA devices. The maximum data width for the device is 32-bits, and it uses address line A2 for accessing within 64-bit processor words.
Fly-by mode may be used for transfers between the I/O and SRAM systems. In this mode, SRAM reads and MPC writes occur together to produce the highest possible DMA transfer rate. Fly-by mode can only be used with 32-bit DMA devices such as the MPC. Figure 7-4 shows two fly-by transfers between the SRAM and the output data channel of the MPC. In the second cycle, the MPC deasserts ODREQ, indicating to the DMA channel that its FIFO is full and that it cannot accommodate another transfer. The DMA channel resumes transfers when ODREQ is again asserted.

The 82380 can assemble and disassemble 8-bit I/O accesses to 32-bit memory accesses during accesses to both main memory and the SRAM message area. Address line A2 of the 82380 determines which half of 64-bit memory data is accessed.

### 7.6.2 SRAM Message Area

The SRAM message area is ideally suited for isolating DMA traffic from the i860 microprocessor's local bus. DMA transfers may occur in parallel with CPU main memory accesses. Both the DMA controller and the CPU can act as bus masters to access the SRAM and the I/O bus.

The SRAM message area can be designed for various levels of price/performance. Two implementations are shown in Figures 7-5 and 7-6. The first figure illustrates an approach that requires fewer parts and uses four 8-bit SRAMs. In this example, the SRAM

---

**Figure 7-4. Fly-by Transfer with SRAM Read and I/O Write**
data bus is directly connected to the I/O data bus. Four data buffers are needed on the processor side. Only the lower 32-bit processor data lines are used. This reduces the processor to memory transfer rate by one-half. The processor may continue to access the SRAM with zero wait-states, but it may only access the lower half of the data bus. The auto-increment addressing mode of the processor is used, and full processor bus bandwidth is maintained.

The second example supports a full 64-bit processor interface. Four additional data buffers are needed on the processor side, the I/O side of the bus requires four additional SRAMS and data buffers. The I/O-side buffers are used to multiplex the halves of the 64-bit wide SRAM to one 32-bit I/O data bus. Eight 256 Kbit SRAMs produce a 256 Kbyte message space. 64 Kbit SRAMs can be used instead if less message space is required.

Arbitration between the two bus masters is performed by a PLD. The PLD can grant control of the DMA message system to the 82380 by asserting HOLD. If the i860 microprocessor begins a bus cycle that requires the DMA message system, the arbitration PLD forces the 82380 from the bus.

Figure 7-7 shows arbitration and DMA control logic blocks. The 82380 asserts HOLD and the arbitration PLD returns HOLDA. The 82380 takes control of the bus and performs DMA cycles until HOLDA is deasserted or until the transfer is completed. If HOLDA is deasserted, the 82380 relinquishes control of the bus. The arbitration PLD
deasserts HOLDA to the 82380 and waits for it to deassert HOLD. Although HOLD becomes active again in the next clock, HOLDA is not returned to the 82380 until the processor is finished with the bus. The 82380 is programmed in demand mode to allow this type of arbitration to work. When the processor finishes its I/O cycle, it returns HOLDA to the 82380.

7.7 EXPANSION CONNECTOR

The expansion connector can support memory expansion ranging from simple extensions to complex extensions such as those found in graphics systems. DRAM control lines and processor bus signals are both available on the connector.

7.7.1 Memory Expansion

Additional memory can be added to the expansion board. In this design, the processor address bus must be buffered on the memory board. The address buffers perform address multiplexing and provide DRAM drive capabilities. Control signals are available on the connector, and DRAM and address buffer control signals can be generated using simple decode logic. The data bus uses data buffers from the main board.
7.7.2 Intelligent Expansion

The connector provides the control signals needed to build an intelligent expansion board. One example of an intelligent board is a frame buffer board. It requires its own VRAM controller and control logic for returning READY# and NA# to the processor. Motherboard data buffer control signals are also available. The controller tracks ADS# and EXPSEL# signals to know when a cycle is intended for the expansion board controller. The controller uses the EXPBSY# signal to indicate that it will be asserting the READY# and NA# control lines. The base controller must disable control of these lines when this occurs.
Physical Design and Debugging
CHAPTER 8
PHYSICAL DESIGN AND DEBUGGING

This chapter outlines the basic design issues, ranging from power and ground issues to achieving proper thermal environment for i860™ CPU.

8.1 GENERAL DESIGN GUIDELINES

The performance and proper operation of any high-speed system greatly depends upon proper physical layout. This section gives a brief overview of general issues and design guidelines for layout which are significant to both high- and low-frequency systems.

The ever-increasing improvement of integrated circuit technology has led to an enormous increase in the number of functions that are being implemented on a single chip. Improved technology also allows these functions to be implemented to provide higher performance. The i860 microprocessor, with operating frequencies of 33 MHz/40 MHz and corresponding high edge rates, presents a challenge to the conventional interconnection technologies which to date have been adequate only for interconnecting less sophisticated VLSI devices. This challenge applies especially to system designers who are responsible for providing suitable interconnections at the systems level.

At higher frequencies, the interconnections in a circuit behave like transmission lines which degrade a system's overall speed and distort its output waveforms. In laying out a conventional printed circuit board, there is freedom in defining the length, shape and sequence of interconnections. But with high-speed devices like the i860 microprocessor, this task is carried out with careful planning, evaluation and testing of the wiring patterns. It is also critical to understand the physical properties of transmission lines because interconnection at high edge rates is analogous to a broadcasting transmission line.

8.2 POWER DISSIPATION AND DISTRIBUTION

The i860 microprocessor uses fast one-micron CHMOS* IV process technology. The main difference between the previous HMOS microprocessors and the new ones is that, in the latter, power dissipation is primarily capacitive and there is almost no DC power dissipation. As power dissipation is directly proportional to frequency, accommodating high-speed signals on printed circuit boards and through the interconnections is very critical. The power dissipation of the VLSI device in operation is expressed by the sum of the power dissipation of the circuit elements, which include internal logic gates, I/O buffers and cache RAMs. It is also a function of the operating conditions.

The worst-case power dissipation of any VLSI device is estimated in the following manner:

1. Let the following symbols stand for estimates of typical power dissipation for each circuit element:

   \[ P_G \] = Typical power dissipation for internal logic gates (mW)
   \[ P_{i/O} \] = Typical power dissipation for I/O buffers (mW)
   \[ P_{CRAM} \] = Typical power dissipation for instruction/data cache RAMs (mW)
2. To estimate total typical power dissipation:

$$P_T = P_G + P_{I/O} + P_{CRAM} \text{ (mW)}$$

where $P_T$ is the total typical power dissipation in milliwatts.

3. To estimate the worst case power dissipation:

$$P_d = P_T \times C_v \text{ (mW)}$$

where $P_d$ is the worst case power dissipation (mW) and $C_v$ is a multiplier that is dependent upon power supply voltage.

Internal logic power dissipation varies with operating frequency and to some extent with wait-states and software. It is directly proportional to the supply voltage. Process variations in manufacturing also affect the internal logic power dissipation, although to a lesser extent than with the NMOS processes.

The I/O buffer power dissipation, which accounts for roughly 10 to 25 percent of the overall power dissipation, varies with the frequency and the supply voltage. It is also affected by the capacitive bus load. The capacitive bus loadings for all output pins is specified in the i860™ 64-Bit Microprocessor Data Sheet. The i860 microprocessor’s output valid delays will increase if these loadings are exceeded. The addressing pattern of the software can affect I/O buffer power dissipation by changing the effective frequency at the address pins. The frequency variations at the data pins tend to be smaller; thus, a varying data pattern should not cause a significant change in the total power dissipation.

To calculate the total power dissipated by the board, the following formulas can be used.

8.2.1 Power and Ground Planes

Today’s high-speed CMOS logic devices are susceptible to ground noise and the problems that this noise creates in digital system design. This noise is a direct result of the fast switching speed and high drive capability of these devices, which are requisites in high-performance systems. Logic designers can use techniques designed to minimize this problem. One technique is to reduce capacitance loading on signal lines and provide optimum power and ground planes.

Power and ground lines have inherent inductance and capacitance, which affect the total impedance of the entire system. Higher impedances reduce current and therefore offer reduced power consumption, while low impedance (e.g. ground planes) minimize problems like noise and cross talk. Hence, it is very important for a designer to have a controlled impedance design where high speed signals are involved. The formula for impedance $Z$, given the inductance $L$ and the capacitance $C$, is as follows:

$$Z = (L/C)^{1/2}.$$
The total characteristic impedance for the power supply can be reduced by adding more lines. For multilayer boards, power and ground planes must be used in i860 microprocessor designs.

The effect of adding more lines to reduce impedance is illustrated in Figure 8-1, which shows that two lines in parallel have half the impedance of a single line.

To reduce impedance even further, more lines should be added. To lower the impedance a greater number of lines or a plane should be used. Planes also provide the best distribution of power and ground.

The i860 microprocessor has 24 power (Vcc) and 24 ground (Vss) pins. All power and ground pins must be connected to their respective planes. Ideally, the i860 microprocessor should be placed at the center of the board to take full advantage of these planes. Although the i860 CPU generally demands less power than conventional devices, the possibility of power surges is increased due to the processor's higher operating frequency and wide address and data buses. Peak-to-peak noise on Vcc relative to Vss can be no more than 400 mV, and preferably is no more than 200 mV.

![Figure 8-1. Reduction in Impedance](image-url)
Although power and ground planes are preferable to power and ground traces, double-layer boards present a need for routing of power and ground traces.

The inductive effect of a printed-circuit board (PCB) trace can be reduced by bypassing and careful layout procedures to minimize inductances. Figure 8-2 shows a method for reducing the inductive effects of PCB traces. The power and ground trace layout has a low series inductance, because the loop area between the integrated circuits (ICs) and the decoupling capacitors is small and the power and ground traces are closer. This results in lower characteristic impedance, which in turn reduces the line voltage drop.

Another placement technique is called orthogonal arrangement, which requires more area than the previous technique but produces similar results. This arrangement is shown in Figure 8-3. These techniques reduce the electromagnetic interference (EMI), which is discussed in section 8.4.3.1.

### 8.3 DECOUPLING CAPACITORS

The advanced, high-speed CMOS logic families available today have much higher edge rates than do the older, slower logic technologies. The switching speeds and drive capability needed to provide high performance to the systems are also associated with increased noise levels. Some noise levels are inconsequential because they fall within the switching times of the other devices. The switching activity of one device can propagate to other devices through the power supply. For example, in the TTL NAND gate shown in Figure 8-4, both the Q3 and the Q4 transistors are ON for a short time while output is switching. This increased loading causes a negative spike on \( V_{cc} \) and a positive spike on \( V_{ss} \).

In synchronous systems where several gates switch simultaneously, the result is a significant amount of noise on the power and ground lines. This noise can be removed by decoupling the power supply. First, it is necessary to match the power supply's impedance to that of the individual components. Any power supply presents a low source impedance to other circuits, whether they are individual components on the same board or on other boards in a multiboard system. It is necessary to match the supply's impedance to that of the components in order to lessen the potential for voltage drops that can be caused by IC edge rates, ground- or signal-level shifting, noise induced currents or voltage reflections.

This mismatch can be minimized by using a suitable high-frequency capacitor for bulk decoupling of major circuitry sections, or for decoupling entire PC boards in multiboard systems. This capacitor is typically placed at the supply's entry point to the board. It should be an aluminum or tantalum-electrolytic type capacitor having a low equivalent series capacitance and low equivalent series inductance. This capacitor's value is typically 10 to 47 \( \mu \)F. Additional 0.1 \( \mu \)F capacitors may be needed if supply noise is still a problem.

A second type of decoupling is used for the rest of the ICs on the board. Additional decoupling capacitors can be used across the devices between \( V_{cc} \) and \( V_{ss} \) lines. The voltage spikes that occur due to switching of gates are reduced as the extra current
required during switching is supplied by the decoupling capacitors. These capacitors should be placed close to their devices as the inductance of lengthier connection traces reduce their effectiveness.

Most popular logic families require that a capacitor of 0.01 µF to 0.1 µF RF grade be placed between every one to five packages, depending on the exact application. For high-speed CMOS logic, a good rule of thumb is to place one of these bypass capacitors
between every two to three ICs, depending on the supply voltage, the operating-speed and EMI requirements. The capacitors should be evenly distributed throughout the board to be most effective. In addition, the board should be decoupled from the external supply line with a 10 to 47 pF capacitor. In some cases, moreover, it might be helpful to add a 1 \( \mu \)F tantalum at major supply trace branches, particularly on large PCBs.

Surface mount (chip capacitors) are preferable for decoupling the i860 microprocessor because they exhibit lower inductance and require less total board space than leaded capacitors. They should be connected as shown in the Figure 8-5. These capacitors reduce the inductance, which keeps the voltage spikes to a minimum. They should be used to keep the leads as short as possible.

Inductance is also reduced by the parallel inductor relationships of multiple pins. Six leaded capacitors are required to match the effectiveness of one chip capacitor, but
because only a limited number can fit around i860 CPU, the configuration shown in Figure 8-6 is recommended.

8.4 HIGH FREQUENCY DESIGN CONSIDERATIONS

The overwhelming concern in dealing with high speed technologies is the management of transmission lines. As the edge rates of signals increase, the physical interconnections between devices behave more like transmission lines. Although transmission line theory is straightforward, the difference between ordinary interconnections and transmission line interconnections is fairly complex. Transmission lines have distributed elements which are hard to define, and designers tend to overcompensate for the effects of these elements.

Efficient i860 CPU design requires the identification of transmission lines over back plane wiring, printed circuit board traces, etc. Once this task is accomplished, the designer’s next concern should be to deal with three major problems which are associated with electromagnetic propagation: impedance control, propagation delay and coupling.

The following sections discuss the negative effects of a transmission line that occur when operating at higher frequencies. As in higher frequency design, the reflection and crosstalk effects are inevitable. It is impossible to design optimum systems without accounting for these effects. Later sections include a discussion of techniques that can minimize these effects.
8.4.1 Transmission Line Effects

As a general rule, any interconnection is considered a transmission line when the time required for the signal to travel the length of the interconnection is greater than one-eighth of the signal rise time. (True K.M., "Reflection: Computations and Waveforms, The Interface Handbook", Fairchild Corp, Mountain View, CA, 1975, Ch. 3). The rise time can be either rise time or fall time, whichever is smaller, and it corresponds to the linear ramp amplitude from 0% to 100%. Normally the rise times are specified between 10% to 90% or 20% to 80% amplitude points. These figures are multiplied by 1.25 and 1.67 to obtain the linear-ramp duration from 0% to 100% amplitude.

For example in a PCB using G-10 and polymide (the two main dielectric systems available for printed circuit boards) signals travel at approximately 5 to 6 inches per nanosecond.

If \( t_r/l \times v \leq 8 \) then the signal path is not a transmission line but a lumped element,

where

\[
\begin{align*}
    t_r & = \text{rise time } 0\% - 100\% \\
    v & = \text{speed of propagation (5 to 6 inches/sec)} \\
    l & = \text{length of interconnection (one-way only)}
\end{align*}
\]
The calculation is given by:
\[ t_r \times l \leq 8 \]
so
\[ l \geq (t_r \times 6)/8 \geq (1.25 \times 4 \times 6)/8 \geq 3.75 \text{ inches}. \]

This calculation is based on the fact that the maximum rise time of the signals for the i860 microprocessor is 4ns. For \( l \geq 3.75 \) inches interconnections act as transmission lines.

Every conductor that carries an AC signal and acts as a transmission line has a distributed resistance, an inductance and a capacitance which combine to produce the characteristic impedance (\( Z_0 \)). The value of \( Z_0 \) depends upon physical attributes such as cross-sectional area, the distance between the conductors and other ground or signal conductors, and the dielectric constant of the material between them. Because the characteristic impedance is reactive, its effect increases with frequency.

### 8.4.1.1 TRANSMISSION LINE TYPES

Although many different types of transmission lines (conductors) exist, those most commonly used on the printed circuit boards are microstrip lines, strip lines, printed circuit traces, side-by-side conductors and flat conductors.
8.4.1.1.1 Micro Strip Lines

The micro strip trace consists of a signal plane that is separated from a ground plane by a dielectric as shown in Figure 8-7. G-10 fiber-glass epoxy, which is most common, has an $e_r = 5$ where $e_r$ is the dielectric constant of the insulation. Let:

- $w$ = the width of signal line (inches)
- $t$ = the thickness of copper ( .0015 inches for 1 oz Cu/
  .003 inches for 2 oz Cu)
- $h$ = the height of dielectric for controlled impedance (inches)

The characteristic impedance $Z_0$, is a function of dielectric constant and the geometry of the board. This is theoretically given by the following formula:

$$Z_0 = \frac{87}{\sqrt{(e_r + 1.41)}} \ln \left( \frac{5.98h}{0.8w + t} \right) \text{ ohms}$$

where $e_r$ is the relative dielectric constant of the board material and $h$, $w$ and $t$ are the dimensions of the strip. Knowing the line width, the thickness of Cu and the height of dielectric, the characteristic impedance can be easily calculated.

The propagation delay ($t_{pd}$) associated with the trace is a function of the dielectric only. This is calculated as follows:

$$t_{pd} = 1.017 \sqrt{(0.475e_r + 0.67)} \text{ ns/ft}$$

For G-10 fiber-glass epoxy boards $(e_r = 5.0)$, the propagation delay of microstrip is calculated to be 1.77ns/ft.

8.4.1.1.2 Strip Lines

A strip line is a strip conductor centered in a dielectric medium between two voltage planes. The characteristic impedance is given theoretically by the equation below:

$$Z_0 = \frac{60}{\sqrt{e_r} \ln \left( \frac{5.98b}{\pi (0.8 W + t)} \right)} \text{ ohms}$$

Figure 8-7. Micro Strip Lines
where \( b \) = distance between the planes for controlled impedance as shown in Figure 8-8.

The propagation delay is given by the following formula:

\[
\text{\( t_{pd} = 1.017 \sqrt{\varepsilon_r} \text{ ns/ft} \)}
\]

For G-10 fiberglass epoxy boards \((\varepsilon_r = 5.0)\), the propagation delay of the strip lines is 2.26 ns/ft.

Typical values of the characteristic impedance and propagation delay of these types of lines are as follows:

- \( Z_0 = 50 \) ohms
- \( t_{pd} = 2 \)ns/ft (or 6 in/nsec)

The three major effects of transmission line phenomenon are impedance mismatch, coupling and skew.

The following section will discuss them briefly and provide solutions to minimize their effects.

### 8.4.2 Impedance Mismatch

As mentioned earlier the impedance of a transmission line is a function of the geometry of the line, its distance from the ground plane, and the loads along the line. Any discontinuity in the impedance will cause reflections.

Impedance mismatch occurs between the transmission line characteristic impedance and the input or output impedances of the devices that are connected to the line. The result is that the signals are reflected back and forth on the line. These reflections can attenuate or reinforce the signal depending upon the phase relationships. The results of these reflections include overshoot, undershoot, ringing and other undesirable effects.

At lower edge rates, the effects of these reflections are not severe. However at higher rates, the rise time of the signal is short with respect to the propagation delay. Thus it can cause problems as shown in Figure 8-9.

---

**Figure 8-8. Strip Lines**

240330-82
Overshoot occurs when the voltage level exceeds the maximum (upper) limit of the output voltage, while undershoot occurs when the level passes below the minimum (lower) limit. These conditions can cause excess current on the input gates which results in permanent damage to the device.

The amount of reflection voltage can be easily calculated. Figure 8-10 shows a system exhibiting reflections.
The magnitude of a reflection is usually represented in terms of a reflection coefficient. This is illustrated in the following equations:

\[ T = \frac{v_r}{v_i} = \frac{\text{Reflected voltage}}{\text{Incident voltage}} \]

\[ T_{\text{load}} = \frac{(Z_{\text{load}} - Z_0)}{(Z_{\text{load}} + Z_0)} \]

\[ T_{\text{source}} = \frac{(Z_{\text{source}} - Z_0)}{(Z_{\text{source}} + Z_0)} \]

Reflection voltage \( v_r \) is given by \( v_i \), the voltage incident at the point of the reflection, and the reflection coefficient.

The model transmission line can now be completed. In Figure 8-10, the voltage seen at point A is given by the following equation:

\[ V_a = V_s \times Z_0 / (Z_0 + Z_0) \]

This voltage \( V_a \) enters the transmission line at “A” and appears at “B” delayed by \( t_{pd} \).

\[ V_b = V_a \left( t - \frac{x}{v} \right) H(t - \frac{x}{v}) \]

where \( x = \text{distance along the transmission line from point “A”} \) and \( H(t) \) is the unit step function. The waveform encounters the load \( Z_L \), and this may cause reflection. The reflected wave enters the transmission line at “B” and appears at point “A” after time delay (\( t_{pd} \)).

\[ V_{rl} = T_{\text{load}} \times V_b \]

This phenomenon continues infinitely, but it is negligible after 3 or 4 reflections. Hence:

\[ V_{r2} = T_{\text{source}} \times V_{rl} \]

Each reflected waveform is treated as a separate source that is independent of the reflection coefficient at that point and the incident waveform. Thus the waveform from any point and on the transmission line and at any given time is as follows:

\[ V(x,t) = \left( \frac{Z_0}{Z_0 + Z_s} \right) \left\{ V_s \left( t - \frac{x}{v} \right) H\left( t - \frac{x}{v} \right) \right. \]

\[ + T_1 \left[ V_s \left( t - \left( \frac{2L-x}{v} \right) \right) H\left( t - \left( \frac{2L-x}{v} \right) \right) \right] \]

\[ + T_1 T_s \left[ V_s \left( t - \left( \frac{2L+x}{v} \right) \right) H\left( t - \left( \frac{2L+x}{v} \right) \right) \right] \]

\[ + T_1^2 T_s \left[ V_s \left( t - \left( \frac{2L-x}{v} \right) \right) H\left( t - \left( \frac{2L-x}{v} \right) \right) \right] \]

\[ + T_1^3 T_s^2 \left[ V_s \left( t - \left( \frac{2L+x}{v} \right) \right) H\left( t - \left( \frac{2L+x}{v} \right) \right) \right] \]

\[ \vdots \]
Each reflection is added to the total voltage through the unit step function \( H(t) \). The above equation can be rewritten as follows:

\[
V(x,t) = \left( \frac{Z_0}{Z_0 + Z_s} \right) \left[ V_s(t-t_{pd}) H(t-t_{pd}) \right. \\
+ T_1 [V_s(t-t_{pd}(2L-x)) H(t-t_{pd}(2L-x))] \\
+ \left[ T_1 T_s [V_s(t-t_{pd}(2L+x)) H(t-t_{pd}(2L+x))] \\
+ \ldots \right]\]

Impedance discontinuity problems are managed by imposing limits and control during the routing phase of the design. Design rules must be observed to control trace geometry, including specification of the trace width and spacing for each layer. This is very important because it ensures the traces are smooth and constant without sharp turns.

There are several techniques which can be employed to further minimize the effects caused by an impedance mismatch during the layout process:

1. Impedance matching
2. Daisy chaining
3. Avoidance of 90° corners.
4. Minimization of the number of vias.

**8.4.2.1 IMPEDANCE MATCHING**

Impedance matching is the process of matching the impedance of the source or load with that of the trace, and it is accomplished with a technique called termination. The reflection, overshoot and undershoot of signals are reduced by terminating the remote end of the transmission line from the source. The terminating impedance combines with the destination input circuitry to produce a load that closely matches the characteristic impedance of the line. (Board traces have characteristic impedances in the range of 30 ohms to 200 ohms.)

The calculation of characteristic impedance was already discussed. Impedance of the printed circuit board backplane connectors have the impedance in the same range as the traces, i.e. 30-200 ohms.

Depending upon the length of the conductors or when using twisted pairs of coaxial cable in place of PC traces, the characteristic impedance of a backplane may change. Backplane impedance is also affected by the number of boards plugged.

**8.4.2.1.1 Need for Termination**

The transmission line should be terminated when the \( t_{pd} \) exceeds one-third of \( t \). If \( t_{pd} < 1/3 \ t \), the line can be left unterminated, provided the capacitive coupling between the traces does not cause crosstalk.
Termination thus eliminates impedance mismatches, increases noise immunity, suppresses RFI/EMI and helps to ensure that signals reach their destination with minimum distortion. There are five methods for terminating traces on the board:

1. Series
2. Parallel
3. Thevenin
4. AC
5. Active

Termination requires additional components and power. In case of passive terminations, extra drivers are needed to deliver more current to the line. In case of active terminations extra power is needed which increases the power dissipation of the system.

8.4.2.1.2 Series Termination

One way of controlling ringing on longer lines is with the series termination technique also known as damping. This is accomplished by placing a resistor in series with the transmission line at the driving device end. The receiver has no termination. The value of the impedance looking into the driving device \((R_{\text{driver}} + R_L = Z_0)\) should approximate the impedance of the line as closely as possible. In this circuit the ringing dampens out when the reflection coefficient goes to zero. Figure 8-11 illustrates the series termination.

One main advantage of series termination is that only logic power dissipation results so that lower overall power is required than other termination techniques. There is one penalty, however, in that the distributed loading along the transmission line cannot be used because only half of the voltage waveform is travelling down the line. The drop in voltage across series terminating resistor limits loading to a maximum of 10 loads.

8.4.2.1.3 Parallel Terminated Lines

Parallel termination is achieved by placing a resistor of an appropriate value between the input of the loading device and the ground as shown in Figure 8-12.

Since the input impedance of the device is high as compared to the characteristic line impedance, the resistor and the line function as a single impedance with a magnitude that is defined by the value of the resistor.
When the resistor matches the line impedance, the reflection coefficient at the load approaches zero, and no reflection will occur. One useful approach is to place the termination as close to the loading device as possible.

Parallel terminated lines are used to achieve optimum circuit performance and to drive distributed loads—an important benefit of using parallel terminations.

There are two significant advantages of using the parallel termination. First, it provides an undistributed waveform along the entire line. Second, when a long line is loaded in parallel termination, it does not affect the rise and fall time or the propagation delay of the driving device. Note that parallel termination can also be used with wire wrap and backplane wiring where the characteristic impedance is not exactly defined. If the designer approximates the characteristic impedance, the reflection coefficient will be very small. This results in minimum overshoot and ringing. Parallel termination is not recommended for characteristic impedances of less than 100 ohms because of large DC current requirements.

8.4.2.1.4 Thevenin's Equivalent Termination

Thevenin's equivalent termination is an extension of parallel termination technique. It consists of connecting one resistor from the line to the ground and another from the line to $V_{cc}$. Each resistor has a value of twice the characteristic impedance of the line, so the equivalent resistance matches the line impedance. This scheme is shown in Figure 8-13.

If there were no logic devices present, the line would be placed halfway between the $V_{cc}$ and the $V_{ss}$. When a logic device is driving the line, a portion of the required current is provided by the resistors, so the drivers can supply less current than needed in parallel termination. The resistor value can be adjusted to bias the line towards $V_{cc}$ or ground $V_{ss}$. Ordinarily it is adjusted such that the two are equal, providing balanced performance. Thevenin's circuit provides good overshoot suppression and noise immunity.

Due to power dissipation, this technique is best suited for bipolar and mixed MOS/CMOS devices and is not suitable for pure CMOS implementations. There are two reasons for not having Thevenin's equivalent for the pure CMOS system design. First
CMOS circuits have very high impedance to ground and $V_{cc}$, and their switching threshold is 50% of the supply voltage. Second, besides dissipating more power, multiple input crossing may occur which creates output oscillations.

The main problem with Thevenin termination is high power dissipation in the termination resistors in relationship to the total power consumption of all of the CMOS devices on the board. For this reason, most designers prefer series terminations for CMOS to CMOS connections, as this does not introduce any additional impedance from the signal to the ground. The main advantage of the series termination technique, apart from its reduced power consumption, is its flexibility. The received signal amplitude can be adjusted to match the switching threshold of the receiver simply by changing the value of the terminating resistor. Series termination is a very useful technique for interconnecting the logic devices with long lines.

8.4.2.1.5 AC Termination

Another technique for designs that cannot tolerate high power dissipation of parallel termination and delays created by series termination is AC termination. It consists of a resistor and a capacitor connected in series from the line to the ground. It is similar to the parallel termination technique in functionality except that the capacitor blocks the DC component of the signal, and thus reduces power dissipation. This is shown in Figure 8-14.

The main disadvantage of this technique is that it requires two components. Further the optimum value of the RC time constant of the termination network is not easy to calculate. It usually begins as a resistive value which is slightly larger than the characteristic...
line impedance. It is critical to determine the capacitor value. If the value of RC time constant is small, the RC circuit will act as an edge generator and will create overshoots and undershoots. Increasing the capacitor value reduces the overshoot and undershoot, but it increases power consumption. As a rule of thumb, the RC time constant should be greater than twice the line delay. The power dissipation of the AC termination is a function of the frequency.

8.4.2.1.6 Active Termination

An active termination consists of a resistor that is connected between the input and output of a buffer driver as shown in Figure 8-15.

The main advantage of this technique is that it can tolerate large impedance variations. This tolerance is valuable when tri-state drivers are connected to backplane busses. However, the terminations are costly, and the signals that are produced are not as clean as other terminations. A common solution is to place active terminations at both ends of the bus. This helps maintain the uniform drive levels along the entire length of the bus, and it reduces crosstalk and ringing.
Table 8-1 shows the comparisons of different termination techniques.

### Table 8-1. Comparison of Various Termination Techniques

<table>
<thead>
<tr>
<th>Termination</th>
<th># of Extra Components</th>
<th>$R_L$, Power Consumption</th>
<th>Prop Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Series</td>
<td>1</td>
<td>$Z_0 - Z_{OUT}$</td>
<td>Low</td>
</tr>
<tr>
<td>Parallel</td>
<td>1</td>
<td>$Z_0$</td>
<td>High</td>
</tr>
<tr>
<td>Thevenin</td>
<td>2</td>
<td>$2Z_0$</td>
<td>High</td>
</tr>
<tr>
<td>AC*</td>
<td>2</td>
<td>$2Z_0$</td>
<td>Medium</td>
</tr>
<tr>
<td>Active</td>
<td>1</td>
<td>$2Z_0$</td>
<td>Medium</td>
</tr>
</tbody>
</table>

* A.C. also uses a capacitor of 200 pf to 500 pf.

Beyond matching impedances, there are other techniques that can help avoid reflections. These are discussed in the following sections.

#### 8.4.2.1.7 Impedance Matching Example

Previous sections discuss the techniques for calculating characteristic impedances (using transmission line theory) and the termination procedures used to match impedances. This section describes an impedance matching example that utilizes these techniques. Figure 8-16 shows a simple interconnection which acts like a transmission line as shown by the calculations.

![Figure 8-15. Active Termination](image-url)
In this example the different values are given as follows:

\[
\begin{align*}
Z_s &= 10 \text{ ohms}, \\
\text{tr}_s &= 3\text{nsec (normalized to 0\% to 100\%)} \\
Z_l &= 10 \text{ Kohms} \\
\text{tr}_l &= 3\text{nsec (normalized to 0\% to 100\%)} \\
I &= 9 \text{ in.} \\
\varepsilon_r &= 5.0 \\
h &= 0.008 \text{ in.} \\
w &= 0.01 \text{ in.} \\
t &= 0.0015 \text{ in. (1 oz. Cu)} \\
v &= 6 \text{ in./nsec}
\end{align*}
\]

The interconnection will act as a transmission line if (as was shown in Section 8.4.1)

\[
I \geq (\text{tr} \times v) / 8 \geq (3 \times 6) / 8 \geq 3 \text{ in.}
\]

The value of $I = 9$, thus the interconnection acts like a transmission line. The impedance of the transmission line is calculated as follows:

\[
Z_0 = \frac{87}{\sqrt{(\varepsilon_r + 1.41) \times \ln (5.98h/(.8w + t))}} = 34.39 \ln 5.05 = 55.6 \text{ ohms}
\]

Since $Z_s = 10$ ohms, hence the termination techniques described previously will be needed to match the difference of 45.6 ohms. One method is to use a series terminating resistor of 45.6 ohms or use AC termination where $R = 55.6$ ohms and $C = .3\mu F$. The circuit of Figure 8-16 is shown with termination in Figure 8-17.
8.4.2.2 DAISY CHAINING

In laying out PCB boards, a stub or T-connection is another source of signal reflection. These types of connections act as inductive loads in the signal path. In daisy chaining, a single trace is run from the source, and the loads are distributed along this trace. This is shown in Figure 8-18.

An alternative to this technique is to run multiple traces from the source to each load. Each trace will have unique reflections, which are then transmitted down other traces when they return to the source. In such cases, a separate termination is required for each branch. To eliminate these multiple terminators from T-connections, high-frequency designs are routed as daisy chains.

Because each gate provides its own impedance load along the chain, it is necessary to distribute these loads evenly along the length of the chain. Hence, the impedance along the chain will change in a series of steps and is easier to match. The overall speed of this line is faster and predictable. Also all loads should be placed at equal distances (regular intervals).

8.4.2.3 90 DEGREE ANGLES

Another major cause of reflections are 90 degree angles in the signal paths, which cause an abrupt change in the signal direction and promote signal reflection. For high-frequency layout of designs, avoid 90 degree angles and use 45 or 135 degree angles, as shown in Figure 8-19.

8.4.2.4 VIAS (FEED-THROUGH CONNECTIONS)

Another impedance source that degrades high-frequency circuit performance is the via. Expert layout techniques can reduce vias to avoid reflection sites on PCBs.

8.4.3 Interference

Previous sections discuss reflections in high-frequency design, their causes, and techniques to minimize them. The following sections discuss additional issues related to high-frequency design, including interference. In general, interference occurs when electrical activity in one conductor causes a transient voltage to appear in another conductor.
Two main factors increase interference in any circuit:

1. Variation of current and voltage in the lines causes frequency interference. This interference increases with increases in frequency.

2. Coupling occurs when conductors are in close proximity.

Two types of interference are observed in high frequency circuits:

1. Electromagnetic Interference (EMI)

2. Electrostatic Interference (ESI)

8.4.3.1 ELECTROMAGNETIC INTERFERENCE (CROSSTALK)

Crosstalk is a problem at high operating frequencies because, as operating frequencies increase, the signal wavelengths become comparable to the length of some of the interconnections on the PC board. Crosstalk is a phenomenon of a signal in one trace inducing another similar signal in an adjacent trace. There are two types of couplings between parallel traces which determine the amount of crosstalk in a circuit: inductive coupling and radiative coupling.

Inductive coupling occurs when a current in one trace produces a current in a parallel trace. This current reduces with the distance between the two traces. Hence, closely spaced wires or traces will incur the greatest degree of inductive coupling. Each of the traces will induce a current in the other.

Radiative coupling occurs when two parallel traces act as a dipole antenna which radiates signals that parallel wires can pick up. This results in the corruption of signal that is already present in the trace. The intensity of this type of coupling is directly proportional to the current present in the trace. However, it is inversely proportional to the square of the distance between the radiator and the receiver.
8.4.3.2 MINIMIZING CROSSTALK

When laying out a board for an i860 microprocessor-based system, several guidelines should be followed to minimize crosstalk.

One source of crosstalk is the presence of a common impedance path. Figure 8-20 shows a typical layout in which earth ground and signal ground are different.

To reduce crosstalk, it is necessary to minimize the common impedance paths, which are shown as the ground impedances $Z_2$, $Z_3$ and $Z_4$. During current switching, the ground line voltage drops causing noise emission. By enlarging the ground conductor (which reduces its effective impedance), this noise can be minimized. This technique also provides a secondary advantage in that it forms a shield which reduces the emissions of other circuit traces, particularly in multilayer circuit boards.

The impedances $Z_2$ through $Z_4$ depend upon the thickness of the copper PC-board foil, the circuit switching speeds and the effective lengths of the traces. The current flowing through these common impedance paths radiates more noise as the current increases. The amount of voltage that is generated by these switching currents and multiplied by the impedance is difficult to predict.

![Figure 8-20. Typical Layout](image-url)
An effective way of reducing EMI is to decouple the power supply by adding bypass capacitors between $V_{cc}$ and ground. This technique is similar to the general technique discussed earlier. (The goal of the previous technique was to maintain correct logic levels.)

The design of effective coupling and bypass schemes centers on maximizing the charge stored in the circuit bypass loops while minimizing the inductances in these loops. Some other precautions that can minimize EMI are as follows:

- Running a ground line between two adjacent lines. The lines should be grounded at both ends.
- Separation of the address and data busses by a ground line. This technique may however be expensive due to large number of address and data lines.
- Removing closed loop signal paths which create inductive noise as shown in Figure 8-21.

Minimizing crosstalk involves first examining the circuit’s interconnection with its nearest neighbors since parallel and adjacent lines can interact and cause EMI. It is necessary to maximize the distance between adjacent parallel wires.

![Figure 8-21. Closed Loop Signal Paths are Undesirable](image-url)
8.4.3.3 ELECTROSTATIC INTERFERENCE

We have discussed two types of coupling, inductive and radiative coupling, which are responsible for creating electromagnetic interference. A third, known as capacitive coupling, occurs when two equipotential parallel traces are separated by a dielectric and act as a capacitor. According to the standard capacitor equation, the electric field between the two capacitor surfaces varies with the permittivity of the dielectric and with the area of the parallel conductors.

Electrostatic interference (ESI) is caused by this type of coupling. The charge built on one plate of the capacitor induces opposite charge on the other. To minimize the ESI, the following steps should be taken.

- Separate the signal lines so that the effect of capacitive coupling is negated.
- Run a ground line between the two lines to cancel the electrostatic fields.

For high-frequency designs, a rule of thumb is to include ground planes under each signal layer. Ground planes limit the crosstalk caused by a capacitive coupling between small sections of adjacent layers that are at equipotentials. Additionally, when the width and thickness of signal lines and their distance from the ground is constant, the effect of capacitive coupling upon impedance remains uniform within ± 5 percent across the board. Using fixed impedance does not reduce capacitive coupling, but it does simplify the modeling of propagation delays and coupling effects. In addition, capacitive coupling can cause interference between layers, so the wires should be routed orthogonally on neighboring board layers.

8.4.4 Propagation Delay

The propagation delay of a circuit is a function of the loads on the line, the impedance, and the line segments. The term propagation delay means the propagation delay in the entire circuit, including the delay in the transmission line (which is a function of the dielectric constant).

Also, the printed circuit interconnections add to the propagation delay of every signal on the wire. These interconnections not only decrease the operating speed of the circuits, but also cause reflection, which produces undershoot and overshoot.

When the propagation delays in the circuit are significant, the design must compensate for the signal skew. Signal skew occurs when the wire lengths (and thus the propagation delays) between each source and each corresponding load are unequal.

Another negative aspect of propagation delay is that it can cause a race condition. This condition occurs when two signals must reach the same destination within one clock pulse of one another. To avoid race conditions, it is necessary to have the signals travel through the same length traces. But if one route is shorter, then the signals will arrive at different timings, causing race conditions.

One way to minimize this is by decreasing the length of the interconnections. Overall route lengths are shorter in multilayer printed circuit boards than in a double layer boards because ground and power traces are not present.
8.5 LATCH-UP

Latch-up is a common condition in the use of CMOS devices which occurs when $V_{cc}$ becomes shorted to $V_{ss}$. Latch-up is triggered when the voltage limits on the I/O pins are exceeded, causing the internal PN junction to become forward-biased. The following steps ensure the prevention of latch-up.

- Observe the maximum input voltage rating of I/O pins.
- Never apply power to an i860 microprocessor pin or to any device connected to it before applying power to the i860 microprocessor.
- Use good termination techniques to prevent overshoot and undershoot.
- Use a proper layout to minimize reflections and to reduce noise on the signals.

8.6 CLOCK CONSIDERATIONS

8.6.1 Requirements

The i860 microprocessor facilitates an easy to implement $1 \times$ clock interface. An external, 33/40 MHz clock synchronizes both the internal functional blocks of the microprocessor and the external signals. Most of the i860 microprocessor's board logic circuitry also uses this clock. A typical i860 microprocessor clock circuit, shown in Figure 8-22, is comprised of a 33/40 MHz oscillator and a buffer.

The clock input requirements for i860 microprocessor systems are more stringent than those for many commonly used TTL devices. A CMOS buffer will meet the clock input requirements as will a TTL buffer with a pullup resistor.

The minimum high and low times are specified as 7ns at 33 MHz and 5ns at 40 MHz. The clock timings are shown in Figure 8-23.

![Figure 8-22. Typical Clock Circuit](image_url)
8.6.2 Routing

Achieving the proper clock routing around a 33/40 MHz printed circuit board is delicate because a myriad of problems, some of them subtle, can arise if certain design guidelines are not followed. For example, fast clock edges cause reflections from high impedance terminations. These reflections can cause significant signal degradation in systems operating at 33/40 MHz clock rates. This section covers some design guidelines which should be observed to properly lay out the clock lines for efficient i860 microprocessor operation.

Since the rise/fall time of the clock signal is typically in the range of 2-4ns, the reflections at this speed could result in undesirable noise and unacceptable signal degradation. The degree of reflection depends on the impedance of the traces of the clock connections. These reflections can be optimized by using proper terminations and by keeping the length of the traces as short as possible. The preferred method is to connect all of the loads via a single trace as shown in Figure 8-24, thus avoiding the extra stubs associated with each load. The loads should be as close to one another as possible. Multiple clock sources should be used for distributed loads.

A less desirable method is the star connection layout in which the clock traces branch to the load as closely as possible (Figure 8-25). In this layout, the stubs should be kept as short as possible. The maximum allowable length of the traces depends upon the frequency and the total fanout, but the length of all of the traces in the star connection should be equal. Lengths of less then one inch are recommended.
8.7 Thermal Characteristics

There are thermal and electrical limitations associated with all operating electronic devices. In an i860 microprocessor-based system, these limitations must be accommodated, due to power dissipation concerns, to achieve proper system performance.

Generally, thermal and electrical characteristics are interrelated, and the actual constraints depend upon the application of a particular device.

Most of the general information on case temperature ($T_c$), maximum current and voltage ratings, maximum thermal resistance ($\theta_{ja}$) at various airflow and package thermal specifications are given in the i860™ 64-Bit Microprocessor Data Sheet. Despite the wealth of information presented in the data sheet, it is impossible to provide graphs and reference tables to cover all applications. The designer must accurately calculate several factors such as junction temperature ($T_j$) and total power dissipation ($P_d$) in particular applications. This section explains how to perform these calculations.
The thermal specifications for the i860 microprocessor are designed to ensure a tolerable temperature at the surface of the chip. This temperature, called the junction temperature ($T_j$), can be determined from external measurements using the known thermal characteristics of the package.

The following two equations facilitate the calculation of the junction temperature ($T_j$):

$$
T_j = T_a + (\theta_{ja} \cdot P_d) \quad \text{and} \\
T_j = T_c + (\theta_{jc} \cdot P_d)
$$

where

- $T_j$ = junction temperature
- $T_a$ = ambient temperature
- $T_c$ = case temperature
- $\theta_{ja}$ = junction to ambient temperature coefficient
- $\theta_{jc}$ = junction to case temperature coefficient
- $P_d$ = power dissipation (worst case $P_d = I_{cc} \cdot V_{cc}$)

Given a heat sink with a thermal resistance of $\theta_{sa}$ (sink to ambient), and given the thermal resistance from the junction to the case $\theta_{jc}$, then the equation for calculating $T_j$ is as follows:

$$
T_j = P_d(\theta_{jc} + \theta_{cs} + \theta_{sa}) + T_a \\
T_j = P_d(\theta_{jc} + \theta_{cs} + \theta_{sa}) + T_c
$$

Case temperature calculations offer many advantages over ambient temperature calculations:

- Case temperature is more easily measured compared to ambient temperature because the measurement is localized to a single point (the center of the package).
- The worst case junction temperature ($T_j$) is lower when calculated with case temperature for two reasons. First, the junction-to-case thermal coefficient ($\theta_{jc}$) is lower than the junction-to-ambient thermal coefficient ($\theta_{ja}$). Therefore, the calculated junction temperature varies less with power dissipation ($P_d$). Second, the junction-to-case coefficient ($\theta_{jc}$) is not affected by the airflow in the system, while the junction-to-ambient coefficient ($\theta_{ja}$) does vary.

Given the case temperature specification, a designer can either set the ambient temperature or use fans to control the case temperature. Finned heatsinks or conductive cooling may also be used in an environment which prohibits the use of fans.

A designer has considerable freedom in designing the heatsink, and faces only practical and economic limits. Multiple parallel devices may be helpful in reducing $\theta_{sa}$, because, if the heat input to the heat sink is dispersed rather than concentrated, the effective thermal impedance will be lower.

To approximate the case temperature for varying environments, the two equations discussed earlier should be combined by making the junction temperature the same for both, resulting in the following equation:

$$
T_a = T_c - (\theta_{ja} - \theta_{jc})P_d
$$
The i860™ 64-Bit Microprocessor Data Sheet should be consulted to determine the values of $\theta_{ja}$ (per the system’s airflow requirement) and the ambient temperature that will yield the desired case temperature. Whatever those conditions are, the proper calculations are very important in achieving an efficient and reliable i860 microprocessor system.

The i860 microprocessor is available in a 168-pin ceramic PGA. The recommended heat sinks for the device are offered in the pin fin design that utilizes air cooling. The heat sink is mounted on the PGA package with a frame and spring. A typical heat sink is shown in Figure 8-26.

### 8.8 DERATING CURVE AND ITS EFFECTS

A derating curve is a graph that plots the output buffer delay against the capacitive load. The curve is used to analyze a signal delay without necessitating a simulation every time the processor’s loading changes. This graph assumes the lumped-sum capacitance model to calculate the total capacitance. The delay in the graph should be added to the specified AC timing value for the device that is driving the load. The derating curve is different for different devices because each device has different output buffers.

![Figure 8-26. Typical Heat Sinks](image)
A derating curve is generated by tying the chip's output buffers to a range of capacitors. The voltage and resistance values chosen for the output buffers are at the highest specified temperature and are rising (worst case) values. The value of the capacitors centers around the AC timing values for the chip. For 33 MHz and above, this is 50 pF. Since the AC timing specifications are measured for a signal reaching 1.5 V, the output buffer delay is the time that it takes for a signal to rise from 0 to 1.5 V. A curve is then drawn from the range of time and capacitance values, with 50 pF representing the average and with nominal or zero derating. These curves are valid only for a 25-150 pF load range. Beyond this range the output buffers are not characterized. The derating curves for the i860 microprocessor are shown in Figure 8-27. These curves use the lumped capacitance model for circuit capacitance measurements and must be modified slightly when doing worst-case calculations that involve transmission line effects.

8.9 BUILDING AND DEBUGGING THE i860™ MICROPROCESSOR-BASED SYSTEM

While an i860 microprocessor based-system designer should plan the entire system, it is necessary to begin building different elements of the core and begin testing them before building the final system. If a printed circuit board layout has to be done, the whole system may be simulated before generating the net list for the layout vendor. It is advisable to work with a preliminary layout to avoid the problems associated with wire wrap boards that operate at high frequencies. A typical i860 microprocessor-based system is shown in Figure 8-28.
The following steps are usually carried out in designing with the i860 microprocessor.

1. Clock circuitry should use an oscillator and fast buffer. The CLK signal should be clean, without any overshoots or undershoots.

2. The reset circuitry should be designed as shown in Chapter 3. This circuitry is used to generate the RESET # signal for the i860 microprocessor. The system should be checked during reset for all of the timings. The clock continues to run during these tests.

3. The INT and HOLD pin should be tied to LOW (negated from the active state). The READY# pin is pulled HIGH so as to add additional delays (wait-states) to the first cycle. At this instance, the i860 microprocessor is reset, and the signals emitted from it are checked for the validity of the state. The i860 microprocessor will generate the physical address 0XFFFFFF00. The address latch is connected at this time, and the address is verified.

4. The PLD implementing the address decoder should be connected to the i860 microprocessor. After reset, the i860 microprocessor is checked to find out whether it accesses the EPROM for retrieving the initial code. The i860 microprocessor supports a special CS8 mode for boot-up from eight-bit I/O devices. This allows the processor to boot-up from the eight-bit EPROM. Once the system boots, the ROM can be copied into memory or can be disabled and replaced by DRAMs.
To interface the i860 microprocessor to an eight-bit EPROM, an address multiplexer is used to select between BE2#...0# in CS8 mode and processor addresses in data mode. This mode is discussed in detail in the I/O interfacing chapter.

8.9.1 Debugging Features of the i860™ Microprocessor

The i860 microprocessor supports debugging by providing data and instruction breakpoints. The various debugging features are discussed below.

- A data break-point register (db) allows the specification of the address which can be monitored by the i860 microprocessor.

- The processor status register has break read and break write bits which enable traps. For a detailed discussion, refer to the i860™ 64-Bit Microprocessor Data Sheet and the i860™ 64-Bit Microprocessor Programmer’s Reference Manual.

- A data access trap (DAT) bit in the processor status register helps the trap handler to determine when the data break point cause a trap.

- A special trap instruction sets a breakpoint in the code. Additionally, an instruction trap (IT) bit allows the trap handler to determine when a trap instruction will cause a trap.

Combined with a general understanding of debugging issues, these features are sufficient for debugging an entire i860 microprocessor-based system. These combined with certain hindsight into general debugging issues is sufficient to debug the entire i860 microprocessor based system.

If the initial run of a diagnostic program is not successful, then a logic analyzer can be used to determine the source of a problem.

After an initial debugging the i860 microprocessor should generate a code fetch cycle to the EPROM.

The i860 microprocessor can stop issuing new cycles for the following reasons:

- The READY# signal is never asserted to terminate to the bus cycles.

- There is an infinite loop executing out of the cache. Address lines will toggle but no bus cycles will be present.

8.9.2 Certain Gotchas when Debugging with i860™ Microprocessor

When designing with the i860 microprocessor, there are certain issues of which a designer should be aware:

- After reset, the instruction and data cache may contain data so the cache flush procedure should be used to reset the instruction and data cache.

- In CS8 mode, when i860 microprocessor does a cache block fill, it fetches addresses in decrementing order and executes in incrementing order.
8.9.3 Debugging

Once the i860 microprocessor-based system is designed and the printed circuit board is fabricated and stuffed, the next step is to debug the hardware in increments. The following sections provide valuable debugging concepts and techniques for writing diagnostic software. The i860™ 64-Bit Microprocessor Data Sheet, i860™ 64-Bit Microprocessor Programmer's Reference Manual and this manual provide a good start.

8.9.4 Simple Diagnostic Programs

To begin debugging an i860 microprocessor-based system, the designer should utilize a set of EPROMs with simple programs that send a message from the microprocessor signaling that the initial communication channel is working. A code example is shown in Figure 8-29.

```
DATA_PORT_ADDR = 0x1000008

ld$start::
   mov   DATA_PORT_ADDR, r4
loop::
   ld.1 message_ptr, r6
   ld.1 message_length, r7
loop1::
   ld.s (r6), r5
   st.s r5, (r4)
   addu 2, r6, r6
   adds -2, r7, r7
   bten r7, r0, 100p1
   br loop
   nop
.data
message_ptr::
   .long message
message_length::
   .long 58 // Message length must be even!
message::
   .byte 'Hello! (This message has been sent by the i860 CPU)', 13, 10
```

Figure 8-29. Simple Diagnostic Program

The loop inside this program can be utilized for various debugging purposes, such as verifying various bus cycles and checking the noise level on the clock and on other critical control signals. The rest of the system can then be exercised and tested.
Once the data transceivers are connected, it is impossible to check the data path from the i860 microprocessor for reading and writing to the external memory subsystem. A special diagnostic program (shown by the example in Figure 8-30) is written to check the read/write function to the external memory. The program should have built in loops so that the i860 microprocessor's behavior can be observed on a logic analyzer.

```
ld$start: 
  mov  DATA_PORT_ADDR, r4
  mov  0,    r5
loop:
  ld.s 0(r4), r10
  ld.s 0(r4), r11
  ld.s 0(r4), r12
  ld.s 0(r4), r13
  ld.s 0(r4), r14
  st.s r10, 0(r4)
  st.s r11, 0(r4)
  st.s r12, 0(r4)
  st.s r13, 0(r4)
  st.s r14, 0(r4)
  br  loop
  nop
```

Figure 8-30. Read/Write Diagnostic Program

When the clock generator, i860 microprocessor, address decoder, address latch, data transceivers, READY# generation logic and RESET logic are all functioning, the i860 microprocessor is capable of running the software in the EPROMs.

Once the EPROMs are installed, the READY# line should be added to the bus cycles following reset. During this state, the system is checked using the digital oscilloscope.

Another check should verify that the address latches have latched the first address, and that the address decoder is providing a chip select signal for the EPROMs. The EPROMs should supply the requested data through the data transceivers to the i860 microprocessor's data pins.

Next the READY# input should be connected to the PLD that is generating the ready signal to test the i860 microprocessor while running the simple diagnostic program. The program loops back on itself and the system runs multiple bus cycles. The logic analyzer can be used at this point to observe the dynamic behavior of the system.

**8.9.5 Other Simple Diagnostic Software**

Additional diagnostic programs can be written to test other system operations such as whether the i860 microprocessor is able to read and write from DRAMs or perform functions like cache flushing. The following are samples of diagnostic programs.
//This program writes to the i860 CPU

DATA_PORT_ADDR = 0x1000008

ld$start::
    mov DATA_PORT_ADDR, r4
    mov r0, r5
loop:: st.s r5, 0(r4)
    addu i, r5, r5
    br loop
    nop

//This program copies ROM code to DRAM on the i860 uP add-in board.

dramstart= 0x7eff0000 //noncacheable alias
text

epromstart= 0xffff0000
epromend= 0xffffffff

// initialize control regs

// leave all ints disabled

ld$start::
    st.c r0,psr
    st.c r0,fsr
    ld.c fir,r0
    st.c r0,epsr

// clean pipeline

pfadd.ss f0,f0,f0
pfadd.ss f0,f0,f0
pfadd.ss f0,f0,f0
pfmul.ss f0,f0,f0
pfmul.ss f0,f0,f0
pfmul.ss f0,f0,f0

mov epromstart,r9
mov epromend,r7

call flush_cache //make sure we get a miss
mov dramstart,r8

loadbytes:

ld.b 0(r9),r10 //cache data for bytes 0 & 8
ld.b 0(r9),r10 //load byte 0 from cache
call flush_cache //insurence cache miss next time
ld.b 8(r9),r18 //load byte 8 from cache
ld.b 2(r9),r13 //cache data for bytes 3 & 11
ld.b 0(r9),r13 //load byte 3 from cache
call flush_cache // insure cache miss next time
ld.b 8(r9),r19 // load byte 11 from cache
1d.s 0(r9),r14 // cache data for bytes 4 & 12
1d.b 0(r9),r14 // etc.
call flush_cache
1d.b 8(r9),r20
1d.b 1(r9),r15 // cache data for bytes 5 & 13
1d.b 0(r9),r15
call flush_cache
1d.b 8(r9),r21
1d.b 0(r9),r16 // cache data for bytes 6 & 14
1d.b 0(r9),r16
call flush_cache 1d.b 8(r9),r22
1d.b 3(r9),r17 // cache data for bytes 7 & 15
1d.b 0(r9),r17
call flush_cache
1d.b 8(r9),r23

store::

st.b r10,0(r8) // store byte 0 to DRAM
st.b r13,1(r8) // 3
st.b r14,2(r8) // 4
st.b r15,3(r8) // 5
st.b r16,4(r8) // 6
st.b r17,5(r8) // 7
st.b r18,6(r8) // 8
st.b r19,7(r8) // 11
st.b r20,8(r8) // 12
st.b r21,9(r8) // 13
st.b r22,0xa(r8) // 14
st.b r23,0xb(r8) // 15
xor r7,r9,r0 // check for end of ROM
bc load_reset // branch if end of ROM
addu 0x10,r9,r9 // increment ROM counter
br loadbytes // copy another 12 bytes
addu 0xc,r8,r8 // increment DRAM counter

load_reset:: // Load the reset code to DRAM

w1=0xa0000000 // This hex code is for the reset branch
w2=0xa0000000
w3=0xe4010000
w4=0xec21
w5=0x40000800
w6=0xa0000000
resxc_start=0x7effff00 // noncacheable alias of feffff00
dramc_start=0xffff0000 // cacheable DRAM that will be mapped
byte_bucket = 0x7fff0010 // ROM location

mov resxc_start, r4    // bottom of reset branch code
mov w1, r5

st.l r5, 0(r4)       // store word 1
addu 4, r4, r4       // increment counter
mov w2, r5
st.l r5, 0(r4)       // store word two
addu 4, r4, r4
mov w3, r5
st.l r5, 0(r4)       // store word three
addu 4, r4, r4
mov w4, r5
st.l r5, 0(r4)       // store word four
addu 4, r4, r4
mov w5, r5
st.l r5, 0(r4)       // store word five
addu 4, r4, r4
mov w6, r5
st.l r5, 0(r4)       // store word six

// The next line must be hand patched after every reassembly
mov 0xffffffff16c, rl  // ffffffff16c is mask_cs8

// We go through the warp drive section twice to insure that
// it gets completely cached. We must be executing from
// cache when we kill CS8 and the boot bit.

mov byte_bucket, r5
ld.c dirbase, r4     // get dirbase contents

warp_drive::
st.b r0, 0(r5)       // does nothing first time
orh 0x100, r0, r20
ld.b 0(r20), r0      // reset EVAT int
st.c r4, dirbase     // does nothing first time
bri r1                // go to mask_CS8 the first time
nop                   // go to ffffffff0000 second time

mask_cs8::
mov 0xffffffff7f, r3 // kill cs8 bit
and r3, r4, r4
or 0x20, r4, r4      // invalidate code cache bit
mov 0x10000010, r5
xorh h-byte_bucket, r5, r5 // zero r5's high 16 bits
mov dramc_start, r1   // r1 now has DRAM start address
br warp_drive         // r5 has EVAT boot port address
nop                   // r4 has CS8 bit reset
The following cache flush procedure is from the i860™ 64-Bit Microprocessor Programmer's Reference Manual. Please refer to it for additional information.

```Assembly
flush_cache:
    FLUSH_P = 0x7f000000-32
// rw=r24, rx=r25, ry=r26, rz=r27
    mov r1, r2
    ld.c dirbase, r27
    or 0x800, r27, r27
    adds -1, r0, r25
    call D_FLUSH
    st.c r27, dirbase
    or 0x900, r27, r27
    call D_FLUSH
    st.c r27, dirbase
    xor 0x900, r27, r27
    mov r2, r1
    bri r1
    st.c r27, dirbase
D_FLUSH::
    mov FLUSH_P, r24
    or 127, r0, r26
    bra r25, r26, D_FLUSH_LOOP
    ld.l 32(r24), r0
D_FLUSH_LOOP::
    bra r25, r26, D_FLUSH_LOOP
    flush 32(r24)++
    bri r1
    ld.l -512(r24), r0
.end
```

The diagnostic software verifies the ability of the system to perform the bus cycles. The i860 microprocessor fetches code from the EPROMs, and this implies that the EPROM can read functions correctly. Instructions in the program generate bus cycles to write and read the DRAM. The data value read back is checked for accuracy. The program has built-in loops which allow the designer to observe processor states on a logic analyzer and to monitor the signal level on an oscilloscope.
Testability
CHAPTER 9
TESTABILITY

9.1 INTRODUCTION

Testability is a major issue in digital system design. Testing involves two processes, test generation and test verification. Test generation is the process of determining the test sequence for a circuit that will verify its proper operation and developing test vectors. Test verification is the process of proving that the circuit works with the test vectors. Test vectors are generated and introduced into a system, and, by observing the response from the system and comparing it to expected data, error conditions can be detected. Therefore testing the system requires control over the test vectors and a means of observing the response.

Many testing methods are practiced in the industry, including Level-Sensitive Scan Design (LSSD), Scan Path, Scan/Set, and Random Access. These techniques require additional hardware, particularly shift registers, in order to input the test vectors and to observe the critical points in the system.

Various designs for testability have evolved. These designs share the same objective: to control and observe the critical points in a system. As the system becomes more complex it becomes more difficult to control and observe the signal path, and it is therefore essential to plan for testability in the design phase.

The two methods that are most often used to load in test vectors are parallel loading and serial scanning. Parallel loading of the input and output data translates to wide buses and higher costs. Further, it may not be effective in storing and analyzing the results. Serial scanning requires more clock cycles to load in test vectors through a serial channel and to run the diagnostics. The response is then read through a serial channel, whereupon a signature analysis can be performed.

The i860™ microprocessor has a boundary scan mode that involves a simple serial interface that allows the testing of all signal traces with only seven probe connections. These probes allow forcing of all the outputs and sampling of all inputs. This can be used in component testing or board-level testing for the i860 CPU's interface. This chapter discusses the testability features of the i860 CPU and the interface timings in performing board-level testing.

9.2 BOUNDARY SCAN MODE

The boundary scan mode is an elegant testability method that provides serial scan diagnostics. Only seven probes need to be connected: the CLK, BSCN, SCAN, SHI, BREQ, RESET, and HOLD signals. With this configuration the user can apply test vectors to the system via a serial channel and sample the response.

The CLK input determines the execution rate and timings of the i860 microprocessor; the timings of the other signals are specified relative to the rising edge of this signal. In addition, the clock signal determines the shift-in rate of the test vectors and the shift-out rate of the response.
The testability pins are the Boundary Scan Shift Input (SHI), the Boundary Scan Enable (BSCN), the Shift Scan Path (SCAN), and the boundary scan shift output, which is shared with the Bus Request (BREQ) output. The BREQ pin has two functions. In normal processor operations, the BREQ line is asserted when the i860 microprocessor has a pending memory request, even when the HLDA line is asserted. In the boundary scan mode it is the serial shift out pin. The Bus Hold (HOLD) line has a specific function which is described in great detail, along with the above-mentioned signals, later in this chapter. The RESET line causes the processor initialization. More details on RESET are provided in Section 3.2 of the data sheet.

When the Boundary Scan Enable (BSCN) signal is asserted, the i860 microprocessor enters the boundary scan mode on the next rising edge of CLK. When BSCN is deasserted while in boundary scan mode, the i860 microprocessor leaves the boundary scan mode on the next rising edge of CLK. After leaving the boundary scan mode the internal state is undefined and therefore RESET must be asserted. The timings to enter and exit the boundary scan mode are shown in Figure 9-1.

The BSCN, signal configures the i860 microprocessor for the test mode. When in test mode, the processor can operate in normal mode or shift mode, and the Shift Scan Path (SCAN) input causes the i860 microprocessor to assume one of the two. The test mode operations are defined in Table 9-1. The normal mode is entered on the rising clock edge when the SCAN line is deasserted; the shift mode is entered on the rising edge of the clock when the SCAN line is asserted. In normal mode, the output signals are driven, and the inputs are sampled simultaneously. In shift mode, the test vectors are shifted in at one end of the chain, and the response is shifted out simultaneously at the other end. In testing operations the user switches from shift mode to normal mode and vice versa until all of the vectors are exercised. Then the user exits from the boundary scan mode.

![Figure 9-1. Entering and Exiting the Boundary Scan Mode](240330-103)
Table 9-1. Test Mode Selection

<table>
<thead>
<tr>
<th>BSCN</th>
<th>SCAN</th>
<th>Testability Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO</td>
<td>LO</td>
<td>No testability</td>
</tr>
<tr>
<td>LO</td>
<td>HI</td>
<td>Intel Reserved</td>
</tr>
<tr>
<td>HI</td>
<td>LO</td>
<td>Boundary Scan Mode, Normal</td>
</tr>
<tr>
<td>HI</td>
<td>HI</td>
<td>Boundary Scan Mode, Shift</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHI as input, BREQ as output</td>
</tr>
</tbody>
</table>

9.2.1 Shift Mode Operation

Transferring vectors to the system and reading the responses from the system requires a serial path to the i860 microprocessor. This is provided by the shift mode operation. In effect, the shift mode creates a long serial register to and from the i860 microprocessor. Data shifted into the CPU corresponds to new test vectors that will be output to the system. A response from the system can be shifted out while in shift mode. When switching to and from normal mode, the i860 microprocessor performs either a serial-to-parallel operation or a parallel-to-serial operation. This is a simple, yet effective way to test the system.

In shift mode, the pins shown in Table 9-2 are organized as a boundary scan chain. This chain can be thought of as a long shift register that is shifted on the rising edge of CLK. The order of the boundary scan chain is shown in Figure 9-2. The SHI pin receives the input on one end of the boundary scan chain. The other end (the most significant bit) of the boundary scan chain is the BREQ pin, shown here as the 127th position. The bits that are shifted in through the SHI pin in the serial mode are in the sequence shown in Figure 9-2. For shift mode operations, BREQ is shifted in followed in order by BE0#, BE1#, ..., BSCN, SHI. A total of 127 cycles are needed. While these values are being shifted in, serial bits are coming out of the BREQ pin. These bits are in the order BREQ, BE0#, BE1#, ..., BSCN, and SHI, and they correspond to the response of a previously loaded vector.

To avoid glitches that could occur while the values are being shifted out along the chain, the RESET and HOLD pins must be asserted. In this way, all of the tristateable outputs will be disabled. The timings related to serially shifting the data into and out of the processor is shown in Figure 9-3.

The shift mode is used to shift in a new test vector, as well as to read the response to the previous test vector. These operations are performed concurrently.
Figure 9-2. Order of Boundary Scan Chain

Table 9-2. Test Mode Latches

<table>
<thead>
<tr>
<th>Input Latch</th>
<th>Output Latch</th>
<th>Associated Control Latch</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHI</td>
<td>D0–D63</td>
<td>DATAt</td>
</tr>
<tr>
<td>BSCN</td>
<td>A31–A3</td>
<td>ADDRt</td>
</tr>
<tr>
<td>SCAN</td>
<td>NENE#</td>
<td>NENEt</td>
</tr>
<tr>
<td>RESET</td>
<td>PTB#</td>
<td>PTBt</td>
</tr>
<tr>
<td>D0–D63</td>
<td>W/R#</td>
<td>W/Rt</td>
</tr>
<tr>
<td>CC1–CC0</td>
<td>ADS#</td>
<td>ADSt</td>
</tr>
<tr>
<td></td>
<td>HLDA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOCK#</td>
<td>LOCKt</td>
</tr>
<tr>
<td>READY#</td>
<td>BE7#–BE0#</td>
<td>BEt</td>
</tr>
<tr>
<td>KEN#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NA#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT/CS8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOLD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Normal Mode

During normal mode, two operations can occur simultaneously. The first is the driving of the output lines with the test vector, while the second is the sampling the response on the input lines. The outputs and inputs are shown in Table 9-2.

The three-state output pins A31–A3, BE7#–BE0#, W/R#, NENE#, ADS#, LOCK#, and PTB are enabled by storing a high value in the corresponding control latches ADDRt, BEt,W/Rt, NENEt, ADSt, LOCKt, and PTBt, respectively. If the corresponding output control latch contains a low value, then the pins are tristated.

The data pins D63–D0 are I/O pins and are enabled by the control latch, DATAt, which is similar to the other control latches. In addition when the DATAt contains a low value, then the data pins are configured as inputs and the values on the pins are sampled.

For each of the input pins RESET, HOLD, INT/CS8, NA#, READY#, KEN#, SHI, BSCN, SCAN, and CCl–0, the corresponding latch is loaded with the value that is being driven into the pin.

Normal mode is selected when the SCAN is deasserted while BSCN is asserted. All of the signals shown in Table 9-2 are loaded serially in the shift mode (i.e. while BSCN is high and SCAN is high.) The timings for the normal mode operation are shown in Figure 9-4.
9.2.3 A Test Sequence Example

The following relates to a typical testing session and addresses controllability and observability issues.

1. Enter boundary scan mode.

2. Establish a serial link via the shift mode to shift in the test vectors and to sample the response. Data can be serially shifted into and out of the i860 CPU. Values are assigned to the latches that correspond with the pins, as shown in Table 9-1. The first test vector is then transferred to the i860 microprocessor.

3. Enter normal mode. The processor drives the output and also samples the inputs. This portion of the sequence sends a test vector to the system and also reads the response synchronously to the rising edge of CLK.

4. Reenter shift mode and read the response from the output of the serial chain, while loading new values for the next test vector at the input of the serial chain. It is possible to loop between steps 3 and 4 until finished.

5. Exit boundary scan mode and reset the processor.

While running the test sequence, the user must be aware that the response being received during the normal mode may correspond to test vector presented in the previous cycle. This is true if a scan path also exists in the peripheral circuits.
9.3 USING THE TESTABILITY FEATURES IN A SYSTEM

The boundary scan mode can help in component-level testing and in board-level testing. These are discussed here, with the emphasis on board level testing.

9.3.1 Component-level Testing

For incoming testing, the serial scan device uses a serial channel to provide a means for causing the output pins to toggle. Values can be assigned to the output latches in the shift mode. During normal mode operation, the processor will drive the output pins. The values of the input pins can also be latched in normal mode and then read out in the shift mode. This provides a mechanism for monitoring the inputs and toggling the outputs via a serial channel, and may reduce test program development time.

9.3.2 System-level Testing

The benefits of the boundary scan mode are even more apparent when one is debugging or testing a system, as on a bed of nails in Automatic Test Equipment (ATE). A typical i860 microprocessor system may consist of peripherals and memory elements, as shown in Figure 9-5. If the system is being debugged and brought up, then the serial path can set up test vectors to various memory elements or peripherals. The response from the peripherals can be accessed following the propagation delay while in normal mode. Then, the response is serially shifted out in the shift mode. The received response is compared to the expected response. After the user knows the peripherals are functioning properly, code from an EPROM can be used for more extensive testing.

Figure 9-5. A Typical System with Diagnostics Capabilities
The peripherals or the memory elements must observe the set-up and hold times of the i860 microprocessor during normal mode operation. This can be accomplished by giving multiple clocks during normal mode operation to meet the delay timing of the longest path. A typical timing diagram for a memory access is shown in Figure 9-6. (If the longest delay path is four clock cycles, for example, then by asserting the normal mode for five cycles, the address and the other controls are presented to the memory element, after which the data can be read or written.) The latching of the signals in the i860 CPU can be carried out reliably on the last clock edge, before the shift mode operations are performed.

RAM and other circuits external to the i860 microprocessor peripherals may consist of state machines. The user must ensure that the state machines are enabled or disabled at the appropriate time during boundary scan mode. One way to ensure this is to have separate clocks for the on-board state machines.

If the peripheral or memory elements have a serial scanning scheme as well, then the clock can be asserted for as briefly as one cycle during normal mode. This implies that the system will be responding to the test vectors of the previous cycle. The timings would be similar to those shown in Figure 9-6, with a smaller number of clock pulses for timeslots 2-3 and 4-5. Again, this assumes that the peripherals have a serial scan chain as well. This method reduces the number of clock pulses for the normal mode; however, the overhead in the number of external components is larger than in the first method.
TIMESLOT 1–2 LOAD TEST VECTOR SERIALLY—SHIFT.
2–3 OUTPUT THE TEST VECTOR, AND MONITOR THE INPUTS. THE INPUTS ARE LATCHED ON EVERY CLOCK CYCLE FROM CYCLE 128 TO CYCLE 133. THE EXTERNAL DEVICES MUST HAVE THE DATA SET UP AROUND THE RISING EDGE OF CLOCK NO. 133. THE RESPONSE TO TEST VECTOR 1 IS LATCHED FROM THE INPUT PINS.
3–4 LOAD IN TEST VECTOR 2 SERIALLY, AND SERIALLY SHIFT OUT THE RESPONSE FROM TEST VECTOR 1.
4–5 OUTPUT TEST VECTOR 2, AND MONITOR THE INPUTS. THE NUMBER OF CYCLES THAT SCAN IS IN THE LOW STATE IS SHORTER THAN THAT OF TIMESLOT 2–3 AND THIS IS BECAUSE THE EXTERNAL DEVICE HAS A FASTER ACCESS TIME. THE RESPONSE IS LATCHED ON CLOCK CYCLE 261 TO 265.
NOTE 1 INPUTS ARE SAMPLED IN NORMAL SCAN MODE ON EVERY RISING EDGE OF CLK.

Figure 9-6. A Typical Timing for Serial Scan Mode
Graphics Frame Buffer
Schematics and
PLD Code*

*This design has been tested.
# APPENDIX A

## GRAPHICS FRAME BUFFER

### SCHEMATICS AND PLD CODE

<table>
<thead>
<tr>
<th>SIGNAL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>RASx#</td>
<td>RAS# for buffer #x</td>
</tr>
<tr>
<td>RAS#</td>
<td>RAS1# ANDed with RAS0#</td>
</tr>
<tr>
<td>LRAS#</td>
<td>Delayed RAS#</td>
</tr>
<tr>
<td>CASx#</td>
<td>CAS# for buffer #x</td>
</tr>
<tr>
<td>WEx#</td>
<td>Write control for VRAMs</td>
</tr>
<tr>
<td>DTOEx#</td>
<td>Output control and SAM transfer request</td>
</tr>
<tr>
<td>CPEND#</td>
<td>Cycle pending</td>
</tr>
<tr>
<td>ROWE#</td>
<td>Row address enable</td>
</tr>
<tr>
<td>COLE#</td>
<td>Column address enable</td>
</tr>
<tr>
<td>WDEL#</td>
<td>Write delay</td>
</tr>
<tr>
<td>NCLK#</td>
<td>Timing control for WDEL#</td>
</tr>
<tr>
<td>REF</td>
<td>Refresh request</td>
</tr>
<tr>
<td>TRQ</td>
<td>SAM transfer request</td>
</tr>
<tr>
<td>TRFQ#</td>
<td>Conditioned refresh/SAM transfer request</td>
</tr>
<tr>
<td>RASDEL#</td>
<td>RAS# delay for CAS-/DTOE-before RAS</td>
</tr>
<tr>
<td>PRECH#</td>
<td>Controlling RAS# precharge time</td>
</tr>
<tr>
<td>REFC#</td>
<td>Controlling RAS# active time for TRFQ#</td>
</tr>
<tr>
<td>ERDL#</td>
<td>Expansion read data latch</td>
</tr>
<tr>
<td>LERDL#</td>
<td>Delayed ERDL#</td>
</tr>
<tr>
<td>EWLD#</td>
<td>Expansion write data latch</td>
</tr>
<tr>
<td>ERDE#</td>
<td>Expansion read data enable</td>
</tr>
<tr>
<td>EWDE#</td>
<td>Expansion write data enable</td>
</tr>
<tr>
<td>CLRTRQ#</td>
<td>Clearing TRQ</td>
</tr>
<tr>
<td>CLRREF#</td>
<td>Clearing REFRESH</td>
</tr>
<tr>
<td>SRAE#</td>
<td>Row address enable for SAM transfer</td>
</tr>
<tr>
<td>SCAE#</td>
<td>Column address enable for SAM transfer</td>
</tr>
<tr>
<td>HBLANK</td>
<td>Horizontal blank</td>
</tr>
<tr>
<td>HCOUNT</td>
<td>HBLANK count for generating vertical signal</td>
</tr>
<tr>
<td>HCLR#</td>
<td>Clear counter used to generate HBLANK</td>
</tr>
<tr>
<td>VCLR#</td>
<td>Clear counter used to generate VBLANK#</td>
</tr>
<tr>
<td>VBLANK#</td>
<td>Vertical blank</td>
</tr>
<tr>
<td>SYNC#</td>
<td>Composite sync for video DACs</td>
</tr>
<tr>
<td>CBLANK#</td>
<td>Composite blank</td>
</tr>
<tr>
<td>SBLANK#</td>
<td>Composite blank synchronized to PCLK</td>
</tr>
<tr>
<td>VD Bus</td>
<td>Video data bus</td>
</tr>
<tr>
<td>LDSR#</td>
<td>Loading 4 pixels into shift registers</td>
</tr>
<tr>
<td>SC</td>
<td>Serial clock for shifting VRAM data out, 16MHz</td>
</tr>
<tr>
<td>BLANK#</td>
<td>Blank signal for video DACs</td>
</tr>
<tr>
<td>VOG/VOR/VOB</td>
<td>RGB signals</td>
</tr>
<tr>
<td>PCLK</td>
<td>Pixel clock, 64 MHz</td>
</tr>
</tbody>
</table>

A-1
module FB_VRM flag '-r3'
title 'Buffer Select'

U1 device 'P16R6';

VCC,GND,OCn pin 20,10,11;
CLKC,VBF0n,RESET,VSINC,CLFRX0 pin 1,2,4,5,6;
NC1,NC2,NC3,NC4,NC5,NSS1,NSS2 pin 3,7,8,9,12,13,18;
SE1n,SE0n,VSINC,CLFRQn,VSINCn pin 14,15,16,17,19;

Equations
CLRFRQn := CLFRX0;
VSINC := VSINC;
VSINCn := !VSINC;

state_diagram [SE1n,SE0n]

state [1,1]: if(RESET # VSINC) then [1,1]
else if(!VSINC & VBF0n) then [0,1]
else if(!VSINC & !VBF0n) then [1,0]
else [1,1];

state [0,1]: if(RESET # VSINC) then [1,1]
else [0,1];

state [1,0]: if(RESET # VSINC) then [1,1]
else [1,0];

state [0,0]: goto [1,1];

"Description:
Buffer switch happens at the time of vertical sync.
"

end FB_VRM;
module FB_VRM flag '-r3'
title 'Buffer Switch'

U2 device 'P16R4';

VCC,GND,OCh pin 20,10,11;
CLKC,RESET,LADSn,EXPSELn,A22,A3 pin 1,2,3,4,5,6;
DRMBSYn,BSYn,EXPBSYn,LADSA22n pin 7,8,9,12;
CBUSYn,VBF0n,LRDYn,VSELX0,VSELX1 pin 13,14,15,16,17;
NC1,RDYN pin 18,19;

Equations
RDYN = !(VSELX1 & VSELX0 & DRMBSYn & BSYn & EXPBSYn);
enable RDYN = !(VSELX1 & DRMBSYn & BSYn & EXPBSYn);
CBUSYN = BSYn & !(DRMBSYn & !EXPBSYn);
LADS22n = LADSn # A22;
LRDYn := RDYN;

state_diagram [VSELX1, VSELX0]
state [1,1]: if(RESET) then [1,1]
else if(!EXPSELn & A22 & !LADSn) then [0,1]
else [1,1];
state [0,1]: if(RESET & !(RDYN & DRMBSYn & BSYn & EXPBSYn)) then [0,0]
else [0,1];
state [0,0]: if(!RESET & !EXPSELn & A22 & !LADSn) then [0,1]
else [1,1];
state [1,0]: goto [1,1];

state_diagram [VBF0n]
state [1]: if(RESET & !LADSn & !EXPSELn & A22 & !A3)
then [0]
else [1];
state [0]: if(RESET & !LADSn & !EXPSELn & A22 & A3)
then [1]
else [0];

"Description:
" READYn is asserted when accesses direct to
" the expansion space when A22 = 1. It is not
" enabled until no busy signals are active, i.e.,
" no pipelined cycles are pending.
end FB_VRM;
module FB_VRM flag '-r3'
title 'Write Enable'
U3 device 'P20L8';

VCC,GND pin 24,12;
LWRn,LBE0n,LBE1n,LBE2n,LBE3n,LBE4n pin 1,2,3,4,5,6;
LBE5n,LBE6n,LBE7n,PRECHn,CAS0n pin 7,8,9,10,11;
RASn,WE0Xn,WE0n,WE1n,WE2n,WE3n pin 13,14,15,16,17,18;
WE4n,WE5n,WE6n,WE7n,WE7Xn pin 19,20,21,22,23;

Equations
WE0n = (LBE0n # !LWRn) & CAS0n # !PRECHn # RASn # WE0Xn & !CAS0n;
WE1n = (LBE1n # !LWRn) & CAS0n # !PRECHn # RASn # WE1n & !CAS0n;
WE2n = (LBE2n # !LWRn) & CAS0n # !PRECHn # RASn # WE2n & !CAS0n;
WE3n = (LBE3n # !LWRn) & CAS0n # !PRECHn # RASn # WE3n & !CAS0n;
WE4n = (LBE4n # !LWRn) & CAS0n # !PRECHn # RASn # WE4n & !CAS0n;
WE5n = (LBE5n # !LWRn) & CAS0n # !PRECHn # RASn # WE5n & !CAS0n;
WE6n = (LBE6n # !LWRn) & CAS0n # !PRECHn # RASn # WE6n & !CAS0n;
WE7n = (LBE7n # !LWRn) & CAS0n # !PRECHn # RASn # WE7Xn & !CAS0n;

"Description:
" The write enables are not asserted when RASn goes active
" during CAS-Before-RAS refresh and write cycles
" starting up (to disable write-per-bit).

end FB_VRM;
module FB_VRM flag '-r3'

title 'CRT Clocks'
U7 device 'P16R8';

VCC,GND,OCn   pin  20,10,11;
PCLK,VCLRn,HCLRn,RESET   pin  1,2,3,4;
NC1,NC2,NC3,NC4,NC5   pin  5,6,7,8,9;
HQ0,VQ0,CDIV16,CDIV8,CDIV4,CDIV2   pin  12,13,14,15,16,17;
HCLRXn,VCLRXn   pin  18,19;

CDIV = [CDIV16,CDIV8,CDIV4,CDIV2];

equations

  CDIV := (CDIV + 1) & !RESET;

state_diagram [HCLRXn,HQ0]

state [1,1]: if(RESET) then [1,1]  
           else if(!HCLRn) then [0,1]
state [0,1]: goto [1,0];
state [1,0]: if(RESET # HCLRn) then [1,1]  
           else [1,0];
state [0,0]: goto [1,1];

state_diagram [VCLRXn,VQ0]

state [1,1]: if(RESET) then [1,1]  
           else if(!VCLRn) then [0,1]
state [0,1]: goto [1,0];
state [1,0]: if(RESET # VCLRn) then [1,1]  
           else [1,0];
state [0,0]: goto [1,1];

"Description:
"   HCLRXn and VCLRXn clear the counters for horizontal and
"   vertical timings.

end FB_VRM;
module FB_VRM flag '-r3'
title 'Ready Logic'

UB device 'P20R4';

VCC,GND,OCn pin 24,12,13;
CLK,RASn,COEn,LADSn,WRn,TRFQn pin 1,2,3,4,5,6;
CPENDn,WDELX0,WDELn,NC1,EXPBSYn pin 7,8,9,10,11;
LERDLn,NC2,WAIT,RASn,LWRn,NAX0 pin 14,15,16,17,18,19;
RDYX0,RESET,RDYN,NENEn pin 20,21,22,23;

RIDLE = 1;
RACT = Ø;

equations

LRASn := RASn;
RDYN = !((WAIT & !RDYX0 & TRFQn & !COEn & WDELn & !CPENDn & LWRn) # (!LERDLn));

enable RDYN = !EXPBSYn;

state_diagram [RDYX0]

state RIDLE: if (RASn) then RIDLE
else if(!RASn & TRFQn) then RACT
else RIDLE;

state RACT : if (RASn # !RDYN & !RASn # !LADSn & NENEn) then RIDLE
else RACT;

state_diagram [NAX0]

state [1]: if (RASn) then [1]
else if(WAIT & !CPENDn & !LWRn & !COEn & !RASn) then [Ø]
else [1];

state [Ø] : goto [1];

state_diagram [LWRn]

state [1]: if(!LADSn & !WRn) then [Ø]
else [1];

state [Ø]: if(!LADSn & WRn) then [1]
else [Ø];

End FB_VRM;
module FB_VRM flag '-r3'
title 'Output Enables'

U9 device 'P20R8';

VCC,GND,OCn  pin 24,12,13;
CLK,DREF,EXPSELn,CPENDn,LADSA22n,TRFQn  pin 1,2,3,4,5,6;
WRn,LWRn,RASn,NC1,NC2,REF,NC3  pin 7,8,9,10,11,14,15;
CLRRFX0,DTOE0n,DTOE1n,EWDEn,ERDEX0  pin 16,17,18,19,20;
CLRTRQn,NC4,PRECHn  pin 21,22,23;

state_diagram [DTOE1n]
state [1]:  if (!CPENDn & !LWRn & !RASn
            # RASn & !TRFQn & PRECHn & !REF) then [0]
            else [1];
state [0]:  if (DREF # !LADSA22n & WRn & !EXPSELn) then [1]
            else [0];

state_diagram [DTOE0n]
state [1]:  if (!CPENDn & !LWRn & !RASn
            # RASn & !TRFQn & PRECHn & !REF) then [0]
            else [1];
state [0]:  if (DREF # !LADSA22n & WRn & !EXPSELn) then [1]
            else [0];

state_diagram [CLRRFX0]
state [1]:  if (RASn & REF & PRECHn & !TRFQn)
            then [0]
            else [1];
state [0]:  goto [1];

state_diagram [CLRTRQn]
state [1]:  if (!DTOE1n & PRECHn & !RASn)
            then [0]
            else [1];
state [0]:  goto [1];

state_diagram [ERDEX0]
state [1]:  if (RASn) then [1]
            else if (!RASn & !LWRn & TRFQn
            # !RASn & !LADSA22n & !WRn & !EXPSELn) then [0]
            else [1];
state [0]:  if (ILADSA22n & WRn & !EXPSELn # RASn) then [1]
            else [0];

state_diagram [EWDEn]
state [1]:  if (RASn) then [1]
            else if (!RASn & LWRn) then [0]
            else [1];
state [0]:  if (ILADSA22n & !WRn & !EXPSELn # RASn) then [1]
            else [0];
"Description:
"   ERDEX0 is an intermediate signal to generate ERDEn.
"   CLRRFX0 is an intermediate signal to generate the
"       CLear_Refresh_ReQuest (CLRRFQ)
"
end FB_VRM;
module FB_VRM flag '-r3'
title 'Random Logic'

U10 device 'P20L8';

VCC,GND pin 24,12;
RAS1n,RAS0n,SCAEn,TRFQn,PRECHn pin 1,2,3,4,5;
REF,CAS1n,CASX1,ERDEX0,WAIT,REFCn pin 6,7,8,9,10,11;
LWRn,WRn,DREF,ERDEn,RASDELn,TREF pin 13,14,15,16,17,18;
NC1,LADSBn,RASn,PREF,LADSn pin 19,20,21,22,23;

Equations
RASn = RAS1n & RAS0n;
TREF = !TRFQn & REFCn & CAS1n
# !SCAEn & REFCn
# LADSn & REFCn & !CAS1n & !CASX1 & LWRn & WAIT & TRFQn
# LADSn & REFCn & !CAS1n & !CASX1 & !LWRn & TRFQn;
PREF = (!RAS1n # !RAS0n) & !SCAEn & RASDELn
# !TRFQn & REF & RAS1n & RAS0n & PRECHn & RASDELn;
DREF = RAS1n & RAS0n & REFCn
# !SCAEn & REFCn;
ERDEn = ERDEX0 & (LADSn # WRn # RAS1n & RAS0n);
LADSBn = LADSn;

"Description:
" TREF feeds into the RAS state machine. The !TRFQn term
" is for CAS-Before-RAS refresh; the !SCAEn term
" for serial register load inside the VRAMs.
" PREF feeds into the CAS state machine. The !TRFQn term is
" for VRAM refresh request. The !SCAEn term is for
" serial register load inside the VRAMs.
" DREF feeds into the Output Enable logic.
" LADSBn is for non-critical logic.
end FB_VRM;
module FB_VRM flag '-r3'
title 'Composite Sync and Blank'

u11 device 'P22V10';

VCC,GND pin 24,12;
HBLANK,NC1,HSYNCn, pin 1,2,3;
HBO,HB1,HB2,HB3,RESET pin 4,5,6,7,8;
NC2,NC3,NC4,NC5 pin 9,10,11,13;
VCLRn,HCNT2,HCNT0,HCNT4,HCNT5 pin 14,15,16,17,18;
VBLANKn,VSYNC,HCNT1,HCNT3,CSYNCn pin 19,20,21,22,23;

HB = [HB3,HB2,HB1,HBO];
HCNT = [HCNT5,HCNT4,HCNT3,HCNT2,HCNT1,HCNT0];
equations

WHEN ((HB = = 15) & !(HCNT = = 50) & !RESET)
THEN HCNT := (HCNT + 1);
ELSE HCNT := HCNT & !(HCNT = = 50) & (HB = = 12) & !RESET;

CSYNCn = !VSYNC & HSYNCn;

state_diagram [VBLANKn]
state [Ø]: if((HCNT = = 2) & (HB = = 12) & !RESET) then [1]
    else [Ø]; "Blank for 45 horizontal lines"
state [1]: if((HCNT = = 50) & (HB = = 12) # RESET) then [Ø]
    else [1]; "Active time = 768 horizontal lines"

state_diagram [VSYNC]
state [Ø]: if((HCNT = = 0) & (HB = = 5) & !RESET) then [1]
    else [Ø]; "Front Porch = 6 H lines"
state [1]: if((HCNT = = 0) & (HB = = 11) # RESET) then [Ø]
    else [1]; "Active for 6 H lines"

state_diagram [VCLRn]
state [1]: if((HCNT = = 50) & (HB = = 12) # RESET) then [Ø]
    else [1];
state [Ø]: if(!RESET) then [1]
    else [Ø];

end FB_VRM;
module FB_VRM flag '-r3'
title 'Horizontal Sync and Blank'

u14 device 'P16R6';

VCC,GND,OCn pin 20,10,11;
CDIV16,CC6,CC5,CC4,CC3,CC2,CC1,CC0 pin 1,2,3,4,5,6,7,8;
VBLANKn,NC1,HCLRn,CBLANKn,HSYNChn pin 9,12,13,14,15;
EXPINTn,EINTX0,HBLANKn,RESET pin 16,17,18,19;

CC = [CC6,CC5,CC4,CC3,CC2,CC1,CC0];

state_diagram [EXPINTn,EINTX0]
state [1,1]: if(RESET) then [1,1]
else if(!VBLANKn) then [0,1]
else [1,1];
state [0,1]: goto [1,0];
state [1,0]: if(RESET # VBLANKn) then [1,1]
else [1,0];
state [0,0]: goto [1,1];

state_diagram [HBLANK]
"Horizontal Blank"
state [1]: if(RESET) then [1]
else if(CC = 17) then [0]
else [1];
state [0]: if((CC = 81) # RESET) then [1]
else [0];

state_diagram [HCLRn]
state [1]: if((CC = 81) # RESET) then [0]
else [1];
state [0]: if(!RESET) then [1]
else [0];

state_diagram [HSYNCh]
"Horizontal Sync"
state [1]: if (RESET) then [1]
else if(CC = 1) then [0]
else [1];
state [0]: if ((CC = 7) # RESET) then [1];
else [0];

state_diagram [CBLANKn]
"Composite Blank"
state [1]: if(HBLANK !VBLANKn # RESET) then [0]
else [1];
state [0]: if(!HBLANK & VBLANKn & !RESET) then [1]
else [0];

"Description:

CDIV16 is a 4MHz clock; therefore, horizontal scan period
is about 20.5 usec and the active display time is 16.0
usec (each CC count is 0.25 usec.).

end FB_VRM;
module FB_VRM flag '-r3'
title 'Write Data Latch'

U25 device 'P16R4';

VCC, GND, OCn pin 20, 10, 11;
CLK, RESET, TRFQn, COLEn, NAX0, CPENDn pin 1, 2, 3, 4, 5, 6;
WDELn, LWRn, RASn, ERDLn, WAIT, LERDLn pin 7, 8, 9, 12, 13, 14;
EWDLX0, EWDLn, EXPBSyn, NC1, NAn pin 15, 16, 17, 18, 19;

Equations
NAn = NAX0 # !TRFQn;
enable NAn = !EXPBSyn;
LERDLn := ERDLn;

state_diagram [EXPBSyn]
state [1]: if (RESET) then [1]
else if (IRASn & TRFQn) then [0]
else [1];
state [0]: if(RESET # CPENDn & RASn & COLEn & TRFQn) then [1]
else [0];

state_diagram [EWDLn, EWDLX0]
state [1,1]: if (RESET) then [1,1]
else if (IRASn & LWRn & TRFQn) then [0,1]
else [1,1];
state [0,1]: if(RESET # RASn # !WAIT) then [1,1]
else [1,0];
state [1,0]: if (WDELn) then [1,1]
else [0,1];
state [0,0]: goto [1,1];

"Description:
" NAn is driven only when it is a VRAM cycle.
" EWDLn is asserted whenever, it is a write cycle.

End FB_VRM;
module FB_VRM flag
'-r3' title 'Cycle Pending'

U26 device 'P16R6';

VCC,GND,OCn pin 20,10,11;
CLK,LADS22n,RESET,TRFQn,RDYn,CAS1n,RASn pin 1,2,3,4,5,6,7;
EXPSELn,WRn,NCi,WDELx0,WDELn,NCLKx0
pin 8,9,12,13,14,15;
NCLKn,NC2,CPENDn,WAIT pin 16,17,18,19;

state_diagram [CPENDn]

state [1]: if (RESET) then [1]
else if(!LADS22n & !EXPSELn) then [0]
else [1];

state [0]: if(LADS22n & !CAS1n & TRFQn
# EXPSELn & !CAS1n & TRFQn # RESET) then [1]
else [0];

state_diagram [WDELn, WDELx0]

state [1,1]: if (RESET) then [1,1] "IDLE
else if(!LADS22n & !WRn & !EXPSELn
# !CPENDn & !WRn & !EXPSELn & !RASn)
then [1,0] "INACT
else [1,1];

state [1,0]: if (RESET # RASn) then [1,1]
else if(!LADS22n & WRn & !EXPSELn) then [0,1] "ACTIVE1
else [1,0];

state [0,1]: if (RASn) then [1,1]
else if(!NCLKx0) then [0,0]
else [0,1];

state [0,0]: goto [1,1];

state_diagram [NCLKn, NCLKx0]

state [1,1]: if(RESET) then [1,1] "IDLE
else if(!RDYN) then [0,1] "INACT
else [1,1];

state [0,1]: goto [0,0];"ACTIVE

state [0,0]: if(RESET # RDYN) then [1,1]
else if(!RDYN) then [0,1]
else [0,0];

state [1,0]: goto [1,1];
"Description:
" CPENDn (cycle pending) is deasserted only when CASxn has
" been asserted for the cycle.
" WDELn is for freezing the state machines when a write
" cycle is pipelined right after a read cycle. The
" i860 processor does not drive the data bus until
" one clock after READYn is asserted for the
" previous read cycle.

end FB_VRM;
module FB_VRM flag '-r3'
title 'Refresh Requests'

U27 device 'P16R8';

VCC,GND,OCn   pin 20,10,11;
CLKG,PREChn,REF,TRQ,RESET,RASn   pin 1,2,3,4,5,6;
CLRTRQn,RASDeln,RYn,ROWen,SRAEn   pin 7,8,9,12,13;
TRFQx0,TRFQn,REFCX0,REFCn,SCEn,COLEn   pin 14,15,16,17,18,19;

TI = 3;
TAl = 1;
TA2 = 0;
TI1 = 2;

RIDLE = 3;
RINACT1 = 1;
RINACT2 = 0;
RINACT3 = 2;

state_diagram [REFCn, REFCX0]
state RIDLE:   if(RESET) then RIDLE
  else if (RASn & !TRFQn) then RINACT1
  else RIDLE;
state RINACT1:   if(RESET) then RIDLE
  else if (!RASn & RASDeln) then RINACT2
  else RINACT1;
state RINACT2: goto RIDLE;
state RINACT3: goto RIDLE;

state_diagram [TRFQn,TRFQX0]
state TI:   if(RESET) then TI
  else if (REF & RDYn # TRQ & RDYn) then TA1
  else TI;
state TA1:   if (RESET) then TI
  else if (!RASn & !REFc) then TA2
  else TA1;
state TA2:   if (RESET # REFc) then TI
  else TA2;
state TI1: goto TI;

state_diagram [SRAEn]
state [1]:   if(RESET) then [1]
  else if(RASn & !TRFQn & !REF & PREChn) then [0]
  else [1];
state [0]:   if(!RASn # RESET) then [1]
  else [0];
state_diagram [SCAEn]
state [1]: if(RESET) then [1] else if(!CLRTRQn) then [0]
else [1];
state [0]: if(REFCn # RESET) then [1]
else [0];

state_diagram [ROWEn]
state [0]: if(RESET) then [0]
else if(!RASn # !TRFQn) then [1]
else [0];
state [1]: if (RESET # TRFQn & COLEn & RASn) then [0]
else [1];

state_diagram [COLEn]
state [1]: if (RESET) then [1]
else if(ROWEn & !RASn & TRFQn) then [0]
else [1];
state [0]: if (RASn) then [1]
else [0];

"Description:
* ROWEn enables row address while COLEn enables column
  address for i860 processor cycles. SRAEn and
* SCAEn enable the serial row and column addresses
  for serial register load cycles.
* TRFQn combines the VRAM refresh and serial register load requests.

end FB_VRM;
module "FB_VRM" flag '-r3"
title 'CAS Logic'

U28 device 'P16R6';

VCC,GND,OCn pin 20,10,11;
CLK,CPENDn,RASn,WAIT,LWRn,WDELn pin 1,2,3,4,5,6;
COLEn,TRFqn,PREF,RESET,ERDELn pin 7,8,9,12,13;
CASX0,CASX1,CAS0n,CAS1n,CAS2n,NC1 pin 14,15,16,17,18,19;

CIDLE = ~b11111;
CACTIVE1 = ~b00011;
CACTIVE2 = ~b00010;
CACTIVE3 = ~b00001;
CNS1 = 0;
CNS2 = 4;
CNS3 = 5;
CNS4 = 6;
CNS5 = 7;
CNS6 = 8;
CNS7 = 9;
CNS8 = 10;
CNS9 = 11;
CNS10 = 12;
CNS11 = 13;
CNS12 = 14;
CNS13 = 15;
CNS14 = 16;
CNS15 = 17;
CNS16 = 18;
CNS17 = 19;
CNS18 = 20;
CNS19 = 21;
CNS20 = 22;
CNS21 = 23;
CNS22 = 24;
CNS23 = 25;
CNS24 = 26;
CNS25 = 27;
CNS26 = 28;
CNS27 = 29;
CNS28 = 30;

state_diagram [CAS2n, CAS1n, CAS0n, CASX1, CASX0]

state CIDLE: if (RESET) then CIDLE
    else if(!RASn & !COLEn & !CPENDn & TRFqn & WDELn
              # PREF) then CACTIVE1
    else CIDLE;
state CACTIVE1 : if (WAIT & LWRn & !RASn & !COLEn
    # !WAIT & !LWRn & !RASn & !COLEn
    # !TRFOn & COLEn)
    then CACTIVE3
    else if(!RASn & WAIT & !LWRn & !COLEn) then CACTIVE2
    else if(!RASn & !WAIT & LWRn & !COLEn # RESET) then

CIDLE
    else CACTIVE1;

state CACTIVE2: goto CACTIVE3;
state CACTIVE3: goto CIDLE;
state CNS1: goto CIDLE;
state CNS2: goto CIDLE;
state CNS3: goto CIDLE;
state CNS4: goto CIDLE;
state CNS5: goto CIDLE;
state CNS6: goto CIDLE;
state CNS7: goto CIDLE;
state CNS8: goto CIDLE;
state CNS9: goto CIDLE;
state CNS10: goto CIDLE;
state CNS11: goto CIDLE;
state CNS12: goto CIDLE;
state CNS13: goto CIDLE;
state CNS14: goto CIDLE;
state CNS15: goto CIDLE;
state CNS16: goto CIDLE;
state CNS17: goto CIDLE;
state CNS18: goto CIDLE;
state CNS19: goto CIDLE;
state CNS20: goto CIDLE;
state CNS21: goto CIDLE;
state CNS22: goto CIDLE;
state CNS23: goto CIDLE;
state CNS24: goto CIDLE;
state CNS25: goto CIDLE;
state CNS26: goto CIDLE;
state CNS27: goto CIDLE;
state CNS28: goto CIDLE;

state_diagram [ERDLn]

state [1]: if(RESET) then [1]
    else if(!WAIT & !COLEn & !LWRn & !CAS1n & CASX1 & CASX0
    # WAIT & !COLEn & !LWRn & !CAS1n & CASX1
    & !CASX0) then [0]
    else [1];
state [0]: goto [1];
"Description:
" The SCAEn term in the CAS state machine is for serial
" register load cycle while the REF term for
" CAS-Before-RAS refresh request.
end FB_VRM;
module FB_VRM flag '-r3'
title 'Serial Clocks'

U29 device 'P16R8';

VCC,GND,OCn  pin  20,10,11;
PCLK,SBLANKn,SC0,SC1,LDSRn,SCX0  pin  1,2,12,13,14,15;
NC1,NC2,NC3,NC4,NC5,NC6,NC7  pin  3,4,5,6,7,8,9;
BLANKn,BLANKX1,BLANKX0,NSS1  pin  16,17,18,19;

SI = ~b0011;
SA1 = ~b1111;
SA2 = ~b1110;
SI1 = ~b0000;
SNS1 = 5;
SNS2 = 4;
SNS3 = 2;
SNS4 = 1;
SNS5 = 6;
SNS6 = 7;
SNS7 = 8;
SNS8 = 9;
SNS9 = 10;
SNS10 = 11;
SNS11 = 12;
SNS12 = 13;
BI = ~b011;
B11 = ~b111;
B12 = ~b110;
B13 = ~b101;
B14 = ~b100;
BNS1 = 2;
BNS2 = 1;
BNS3 = 0;

variables

equations

state_diagram [SC1,SC0,LDSR,SCX0]
state SI:  if(SBLANKn) then SA1
else SI;
state SA1:  goto SA2;
state SA2:  goto SI1;
state SI1:  goto SI;
state SNS1:  goto SI;
state SNS2:  goto SI;
state SNS3:  goto SI;
state SNS4:  goto SI;
state SNS5:  goto SI;
state SNS6:  goto SI;

A-20
state SNS7: goto SI;
geoto SI;
state SNS9: goto SI;
state SNS10: goto SI;
state SNS11: goto SI;
state SNS12: goto SI;

state_diagram [BLANKn,BLANKX1,BLANKXØ]
state BI: if(SBLANKn) then BI1 else BI;
state BI1: if(!SBLANKn) then BI2 else BI1;
state BI2: goto BI3;
state BI3: goto BI4;
state BI4: goto BI;
state BNS1: goto BI;
state BNS2: goto BI;
state BNS3: goto BI;

"Description:
" SCØ and SC1 control the serial register loading inside
" the VRAMs.
" BLANKn feeds into the video DAC to blank out the monitor
" screen.

end FB_VRM;
module FB_VRM flag '-r3'
title 'RAS Logic'

U34 device 'P20R8';

VCC,GND,OCh
CLK,LADSn,NENEn,A22,A21,CBUSYn,TRFQn
TREF,RESET,EXPSELn,CPENDn,NC1,NC2
PRCHX0,PRECHn,NC3,RASDELn,RAS1,RAS0n
NC4,NC5

pin 24,12,13;

pin 1,2,3,4,5,6,7;

pin 8,9,10,11,14,15;

pin 16,17,18,19,20,21;

pin 22,23;

IDLE = 1;
ACTIVE = 0;

PIDLE = 3;
PACT1 = 1;
PACT2 = 0;
PNS1 = 2;

state_diagram [RAS0n]

state IDLE: if(!LADSn & !EXPSELn & !A22 & A21 & CBUSYn
& TRFQn & PRECHn
# !CPENDn & !EXPSELn & !A22 & A21 & PRECHn
# !TRFQn & PRECHn & !RASDELn) then ACTIVE
else IDLE;

state ACTIVE :if ( TREF
# RESET
# !LADSn & EXPSELn & TRFQn
# !LADSn & A22 & TRFQn
# !LADSn & A21 & TRFQn
# !LADSn & !EXPSELn & !A22 & A21
# NENEn & TRFQn)
then IDLE
else ACTIVE;

state_diagram [RAS1n]

state IDLE: if(!LADSn & !EXPSELn & !A22 & A21 & CBUSYn
& TRFQn & PRECHn
# !CPENDn & !EXPSELn & !A22 & A21 & PRECHn
# !TRFQn & PRECHn & !RASDELn) then ACTIVE
else IDLE;
state ACTIVE : if ( TREF
    # RESET
    # !LADSn & EXPSELn & TRFQn
    # !LADSn & A22 & TRFQn
    # !LADSn & !A21 & TRFQn
    # !LADSn & !EXPSELn & !A22 & A21
    & NENEn & TRFQn)
    then IDLE
    else ACTIVE;
    
state_diagram [RASDEln]
state [1]: if (RESET) then [1]
    else if (!TRFQn & PRECHn) then [0]
    else [1];
state [0]: if(RESET # !PRECHn) then [1]
    else [0];
    
state_diagram [PRECHn, PRCHX0]
state PIDLE: if (RESET) THEN PIDLE
    else if (!RAS0n # !RAS1n) then PACT1
    else PIDLE;
state PACT1: if (RESET) THEN PIDLE
    else if (RAS1n & RAS0n) then PACT2
    else PACT1;
state PACT2: goto PIDLE;
state PNS1: goto PIDLE;

"Description:
"  RASxn is asserted only for VRAM cycles and only when no
"  other non-VRAM cycles are pending.
"  RASDEln delays RASxn activation when refresh and serial
"  register load requests are detected.
"  PRECHn guarantees minimum RASxn precharge time.

end NI0_VRM;
THESE R-PACKS ARE EQUAL TO
200 OHM THEVENIN
CONNECT 330 OHM TO VDD
680 OHM TO GROUND
MULTIBUS® II Schematics*  

*This design has not been tested.
Memory Interface
Schematics and
PLD Code*

*This is a tested example.
APPENDIX C
MEMORY INTERFACE
SCHEMATICS AND PLD CODE
"TITLE DRAM Control A.
"PATTERN DCTLA0.ABL

" U98
" Reference Sheet 4 of the Schematics.
" This PLD must have a propagation delay of 10ns or faster

module dctla0
U98 device 'E0320';

A31, A23, A22, A21, !S0
!S1, !S2, !S3, !S4, GND
NC0, HOLD, !HREQ, !RAS0, !RAS3
!RAS2, !RAS1, !RAS0, !KEN, VCC

EQUATIONS

RAS0 = A23 & A22 & A21 & S4 & S3 & !S2 & !S1 & !S0
# A23 & A22 & A21 & !S4 & !S3 & S2 & !S1 & !S0
# !S4 & S3 & !S2 & !S1 & S0
# RAS0 & S4
# RAS0 & S3
# RAS0 & S2;

"start write
"start read
"start refresh
"hold
"hold

RAS1 = A23 & A22 & !A21 & S4 & S3 & !S2 & !S1 & !S0
# A23 & A22 & !A21 & !S4 & !S3 & S2 & !S1 & !S0
# !S4 & S3 & !S2 & !S1 & S0
# RAS1 & S4
# RAS1 & S3
# RAS1 & S2;

"start write
"start read
"start refresh
"hold
"hold

RAS2 = A23 & !A22 & A21 & S4 & S3 & !S2 & !S1 & !S0
# A23 & !A22 & A21 & !S4 & !S3 & S2 & !S1 & !S0
# !S4 & S3 & !S2 & !S1 & S0
# RAS2 & S4
# RAS2 & S3
# RAS2 & S2;

"start write
"start read
"start refresh
"hold
"hold

RAS3 = A23 & !A22 & !A21 & S4 & S3 & !S2 & !S1 & !S0
# A23 & !A22 & !A21 & !S4 & !S3 & S2 & !S1 & !S0
# !S4 & S3 & !S2 & !S1 & S0
# RAS3 & S4
# RAS3 & S3
# RAS3 & S2;

"start write
"start read
"start refresh
"hold
"hold
RA5P0 = A23 & S4 & S3 & !S2 & !S1 & !S0
# A23 & !S4 & !S3 & S2 & !S1 & !S0
# !S4 & S3 & !S2 & !S1 & S0
# RA5P0 & S4
# RA5P0 & S3
# RA5P0 & S2;

KEN = A31;
IHOLD = !HREQ;
end dctla0

"Description:
"This PLD decodes and drives the RA5 lines for the DRAMs from
"the main state machine and the N10 addresses. It drives the RAS
"lines for the 8 Mbytes on the EV-AT board.
"
" The addresses (A21-A23) are decoded and activated when the
"DRAM state machine starts a read or a write cycle. They are held
"active until the state machine enters the Precharge sequence
"(State 31). All RA5 lines are active during a refresh cycle."

"start write
"start read
"start refresh
"hold
"hold
"hold

C-11
"TITLE DRAM Control B.
"PATTERN DCTLB1.ABL

" U99
" This PLD decodes and drives numerous DRAM control signals.
" This PLD must have an external clock frequency of 50MHz or faster.

module dctlb1
U99 device 'E0320';

@ALTERNATE
DCLK, /EWDL, /ERDL, /S0, /S1 pin 1,2,3,4,5,;
/S2, /S3, /S4, /RTDS, GND pin 6,7,8,9,10;
/OE, /RTAB, /WDL, /RDL, /WEL pin 11,12,13,14,15;
/CSX, /CAE, /CAL, /RTBA, VCC pin 16,17,18,19,20;

EQUATIONS
RTAB  = RTDS;
RTBA  = RTDS;
RDL   := /ERDL * /S0               "/RDL IS HIGH DURING
  + /ERDL * /S1
  + /ERDL * /S2
  + /ERDL * S3 * S4;
  "STATE 8,16,24
WEL   := S4 * S3 * S2 * S1 * /S0   "state 1
  + S4 * S3 * S2 * /S1 * /S0
  + S4 * S3 * /S2 * S1 * /S0
  + S4 * S3 * /S2 * /S1 * /S0;
  "state 3,5,7
WDL   := /EWDL * /S0               "/WDL IS HIGH DURING
  + /EWDL * /S3
  + /EWDL * /S4;
  "STATE 0,2,4,6
CSX   := S4 * S3 * S2 * /S0        "state 1,3
  + S4 * S3 * /S2 * S1 * /S0
  + S4 * /S3 * S2 * S1
  + S4 * /S3 * /S2 * S1
  + /S4 * S3 * S2 * S1
  + /S4 * S3 * /S2 * /S1
  + /S4 * /S3 * S2 * S1
  + /S4 * /S3 * S2 * S0;
  "state 5,8,9,12,13,16,17,22,23,24,25,18,26
CAL   := /S0;                       " all odd states.
CAE  :=  S4 * S3 * S2  "states 0,1,2,3
+  S4 * S3 */S2 * S1  "states 4,5
+  S4 * S3 */S2 */S1 * S0  "state 6
+  S4 */S3  * S1  "states 8,9,12,13
+ */S4 * S3 * S2 * S1  "states 16,17
+ */S4 */S3 * S2 * S1  "states 24,25
+ */S4  * S2 */S1 * S0;  "state 18,26

end dctlbl

" Description: Refer to the DRAM State Machine document for an explanation
of when these signals are to be active.
"

" CAE Column Address Enable. Turns on the Column address drivers.
"

" CAL Column Address Latch. Latches the N10 addresses for the DRAM
column address.
"

" CSX Chip Select for the DRAM array.
"

" WEL Write Enable Latch. Latches the BEN bits to drive the active
DRAM WEs.
"

" WDL Write Data Latch. For DRAM accesses only.
"

" RDL Read Data Latch. For DRAM accesses only.
"

" CHANGE HISTORY:
" This PLD adds the state 18 to eliminate the tCAC timing violation
for reads immediately following writes.
"TITLE DRAM Control C.
"PATTERN DCTLC3.ABL

"U100
"This PLD decodes and drives numerous DRAM control signals.
"This PLD must have an external count frequency of 58MHz or faster

module dctlc3
U100 device 'E0320';

@ALTERNATE

CLKA, WR, /S0, /S1, /S2
/S3, /S4, /EWDE, /ERDE, GND
/OE, /PERLTCH, /DRMEN, /DRMDIR, /CLRFRQ
/WPL, /OEX, /RAE, /CLRCYC, VCC

EQUATIONS

PERLTCH = S4 * /S3 * S2 * S1 * /S0
+ S4 * /S3 * /S2 * S1 * /S0; "state 9
+ S4 * /S3 * S2 * S1 * /S0; "state 13

CLRCYC = S4 * S3 * S2 * S1 * /S0
+ S4 * S3 * /S2 * S1 * /S0 "state 5
+ S4 * /S3 * S2 * S1 * /S0 "state 9
+ /S4 * S3 * S2 * /S1 * S0 "state 18
+ /S4 * /S3 * S2 * S1 * /S0; "state 25

OEX := S4 * S3 * S2 * S1 * S0 * /WR "state Ø if near read cycle.
+ S4 * S3 * /S2 * S1 * S0 * /WR "state 4 if near read cycle.
+ S4 * /S3 * S2 * S1 "state 8,9
+ /S4 * S3 * S2 * S1 "state 16,17
+ /S4 * /S3 * S2 * /S1 * S0 "state 18
+ /S4 * /S3 * S2; "state 24,25,26,27

RAE := /S4 * /S3 * /S2 * S1 "state 28,29
+ /S4 * /S3 * /S2 * /S1 * S0; "state 30

CLRFRQ := S4 "ALL EXCEPT state 23
+ /S4 * S3 * S2
+ /S4 * S3 * /S2 * S1
+ /S4 * S3 * /S2 * /S1 * S0
+ /S4 * /S3;

WPL := S4 * S3 * S2 * S1 * /S0 "state 1
+ S4 * S3 * S2 * /S1 * /S0 "state 3
+ S4 * S3 * /S2 * S1 * /S0 "state 5
+ S4 * S3 * /S2 * /S1 * /S0; "state 7
MEMORY INTERFACE SCHEMATICS AND PLD CODE

DRMEN := S4 * S3 * /S1  "state 2,3,6,7
+ S4 * S3 * S1 * /S0  "state 1,5
+ S4 * /S3 * S2 * S1  "state 8,9
+ /S4 * S2 * S1  "state 16,17,24,25
+ S4 * S3 * S1 * WR  "state 0,4 if write cycle.
+ EWDE
+ ERDE;

DRMDIR := S4 * /S3 * S2 * S1  "state 8,9
+ S4 * /S3 * /S2 * S1 * /S0  "state 13
+ /S4 * S2 * /S1 * S0  "state 18,26
+ /S4 * S2 * S1  "state 16,17,24,25
+ ERDE;

end dctlc3

" Description: Refer to the DRAM State Machine document for an explanation of when these signals are to be active.

" PERLTPCH  Parity Error Latch. Latches parity errors (if any) during DRAM read cycles.
" CLRFRQ  Clear Refresh Request. Restarts the refresh counter.
" WPL  Write Parity Latch. Latches the parity bits during a DRAM write.
" OEX  Output Enable. For the DRAMS.
" RAE  Row Address Enable. Turns on the row addresses to the DRAM.
" CLRRCYC  Clear Cycle. Clears queued cycles.

" CHANGE HISTORY
" REV 1  change to 16R6D to correct for speed path problem, and fix possible buffer contention.
" REV 2  add state 18 to fix tCAC violation.
" REV 3  Added state 1 to CLRRCYC
"TITLE=N10 EV-AT DRAM Control State Machine.
"PATTERN=DSAT1.PDS

"U104
"This PLD contains the state machine for the DRAM controller.
"This PLD must have an external clock frequency of 58MHz or faster.

module dstat1
U104 device 'E0320';

@ALTERNATE
CLKA, WR, /NENE, /LADS, /CPEN
/EXPBSY, ATMBSY, /DSEL, REFREQ, GND
/OE, /NA, /S4, /S3, /S2
/S1, /S0, /DRAMBSY, /RDY, VCC

EQUATIONS
S0 := S4 * S3 * S1 * S0 * LADS * /WR * DSEL * NENE * /REFREQ
+ S4 * S3 * S2 * S1 * S0 * LADS * /WR * DSEL * NENE * /REFREQ
+ S4 * S3 * S2 * S1 * S0 * CPEN * /WR * DSEL * NENE * /REFREQ
+ /S4 * S3 * S2 * S1 * S0
+ /S4 * S3 * S2 * S1 * S0 * /LADS * /CPEN * /REFREQ
+ /S4 * S3 * S2 * S1 * S0 * /DSEL * /REFREQ
+ /S0;

S1 := S4 * S3 * S1 * S0 * LADS * WR * DSEL * NENE * /REFREQ
+ S4 * S3 * S2 * S1 * S0
+ /S4 * S3 * S2 * S1 * S0
+ /S4 * S3 * S2 * S1 * S0 * /LADS * /CPEN * /REFREQ
+ /S4 * S3 * S2 * S1 * S0 * /DSEL * /REFREQ
+ /S1 * S0
+ S1 * /S0;

S2 := S4 * S1 * S0 * LADS * /WR * DSEL * NENE * /REFREQ
+ S4 * S3 * S2 * S1 * S0 * LADS * DSEL * NENE * /REFREQ
+ S4 * S3 * S2 * S1 * S0 * CPEN * DSEL * NENE * /REFREQ
+ /S4 * S2 * S1 * S0 * LADS * /WR * DSEL * NENE * /REFREQ
+ /S4 * S3 * S2 * S1 * S0 * LADS * /WR * DSEL * /REFREQ
* /EXPBSY * /ATMBSY
+ /S4 * S3 * S2 * S1 * S0 * CPEN * /WR * DSEL * /REFREQ
* /EXPBSY * /ATMBSY
+ S2 * /S1
+ S2 * /S0;
S3 := S4 * S3 * S1 * S0 * LADS * DSEL * NENE * /REFREQ
+ S4 * /S3 * /S2 * S1 * S0 * LADS * WR * DSEL * NENE * /REFREQ
+ S4 * /S3 * /S2 * S1 * S0 * CPEN * WR * DSEL * NENE * /REFREQ
+ /S4 * /S3 * /S2 * S1 * S0 * LADS * WR * DSEL * /REFREQ
* /EXPBSY * /ATMBSY
+ /S4 * /S3 * /S2 * S1 * S0 * CPEN * WR * DSEL * /REFREQ
* /EXPBSY * /ATMBSY
+ /S4 * /S3 * /S2 * S1 * S0 * REFREQ
+ S3 * /S1
+ S3 * /S0;

S4 := S4 * S1 * S0 * LADS * WR * DSEL * NENE * /REFREQ
+ S4 * /S3 * S2 * S1 * S0
+ S4 * /S3 * S2 * S1 * S0 * CPEN * WR * DSEL * NENE * /REFREQ
+ /S4 * S2 * S1 * S0
+ /S4 * /S3 * S2 * S1 * S0 * LADS * WR * DSEL * /REFREQ
* /EXPBSY * /ATMBSY
+ /S4 * /S3 * S2 * S1 * S0 * CPEN * WR * DSEL * /REFREQ
* /EXPBSY * /ATMBSY
+ S4 * /S1
+ S4 * /S0;

DRAMBSY := S4 * S3
+ S4 * /S3
+ /S4 * S3 * S2
+ /S4 * /S3 * S2
+ CPEN
+ EXPBSY
+ ATMBSY;

NA = S4 * S3 * S1 * S0 * LADS * /WR * NENE * /REFREQ
+ S4 * /S3 * S2 * S1 * S0 * LADS * /WR * NENE * /REFREQ
+ /S4 * S2 * S1 * S0 * LADS * /WR * NENE * /REFREQ
+ /S4 * /S3 * S2 * /S1 * S0;

ENABLE NA = /EXPBSY * /ATMBSY * DRAMBSY;

RDY = S4 * S3 * S1 * S0 * LADS * WR * NENE * /REFREQ
+ S4 * S3 * /S1 * S0
+ S4 * /S3 * S1 * /S0;

ENABLE RDY = /EXPBSY * /ATMBSY * DRAMBSY;
end dstat1

"Description: This PLD is the state machine for the DRAM interface to the N10.
"There are 32 states some of which are redundant to minimize the number of
"product terms. This design supports a non-interleaved memory design using
"Static-column 256K x4 DRAMs for the main memory, and 1M x1 DRAMs for the
"parity bits."
There are six types of cycles possible: far read, far write, near read, near write, refresh, and idle. Near read and near write cycles imply memory accesses with the same row address as the preceding cycle. Refresh request has the highest priority and will interrupt near memory cycles.

Read cycles will use full pipelining, write cycles are never pipelined. Near read cycles will be zero wait-state when following read cycles, and near write cycles will be zero wait-state when following write cycles.

DRAMBSY prevents other units from starting a cycle when the DRAM controller is busy.
TITLE DRAM Control State Machine.
PATTERN DSTAT1.PDS

; U104
; This PLD contains the state machine for the DRAM controller.

CHIP DSTAT1 PAL16R6

CLKA WR /NENE /LADS /CPEN /EXPBSY ATMBSY /DSEL REFREQ GND
/OE /NNA /S4 /S3 /S2 /S1 /SO /DRAMBSY /NRDY VCC

EQUATIONS

S0  :=  S4 * S3  * S1 * SO * LADS * /WR * DSEL * NENE * /REFREQ
     + S4 * /S3 * /S2 * S1 * SO * LADS * /WR * DSEL * NENE * /REFREQ
     + S4 * /S3 * /S2 * S1 * SO * CPEN * /WR * DSEL * NENE * /REFREQ
     + /S4 * S3 * /S2 * S1 * SO
     + /S4 * /S3 * /S2 * S1 * SO * /LADS * /CPEN * /REFREQ
     + /S4 * /S3 * /S2 * S1 * SO * /DSEL * /REFREQ
     + /S0

S1  :=  S4 * S3  * S1 * SO * LADS * WR * DSEL * NENE * /REFREQ
     + S4 * /S3 * S2 * S1 * SO
     + /S4  * S2 * S1 * SO
     + /S4 * /S3 * /S2 * S1 * SO * /LADS * /CPEN * /REFREQ
     + /S4 * /S3 * /S2 * S1 * SO * /DSEL * /REFREQ
     + /S1 * SO
     + S1 * /SO

S2  :=  S4  * S1 * SO * LADS * /WR * DSEL * NENE * /REFREQ
     + S4 * /S3 * /S2 * S1 * SO * LADS * DSEL * NENE * /REFREQ
     + S4 * /S3 * /S2 * S1 * SO * CPEN * DSEL * NENE * /REFREQ
     + /S4 * S2 * S1 * SO * LADS * /WR * DSEL * NENE * /REFREQ
     + /S4 * /S3 * /S2 * S1 * SO * LADS * /WR * DSEL * /REFREQ
     * /EXPBSY * /ATMBSY
     + /S4 * /S3 * /S2 * S1 * SO * CPEN * /WR * DSEL * /REFREQ
     * /EXPBSY * /ATMBSY
     + S2 * /S1
     + S2 * /SO
S3 := S4 * S3 * S1 * SO * LADS * DSEL * NENE * /REFREQ
+ S4 * /S3 * /S2 * S1 * SO * LADS * WR * DSEL * NENE * /REFREQ
+ S4 * /S3 * /S2 * S1 * SO * CPEN * WR * DSEL * NENE * /REFREQ
+ /S4 * /S3 * /S2 * S1 * SO * LADS * WR * DSEL * /REFREQ
* /EXPBSY * /ATMBSY
+ /S4 * /S3 * /S2 * S1 * SO * CPEN * WR * DSEL * /REFREQ
* /EXPBSY */ATMBSY
+ /S4 * /S3 * /S2 * S1 * SO * REFREQ
+ S3 * /S1
+ S3 * /SO

S4 := S4 * S1 * SO * LADS * WR * DSEL * NENE * /REFREQ
+ S4 * /S3 * S2 * S1 * SO
+ S4 * /S3 * /S2 * S1 * SO * CPEN * WR * DSEL * NENE * /REFREQ
+ /S4 * S2 * S1 * SO
+ /S4 */S3 */S2 * S1 * SO * LADS * WR * DSEL * /REFREQ
* /EXPBSY * /ATMBSY
+ /S4 */S3 */S2 * S1 * SO * CPEN * WR * DSEL * /REFREQ
* /EXPBSY */ATMBSY
+ S4 * /S1
+ S4 * /SO

DRAMSY := S4 * S3
+ S4 * /S3
+ /S4 * S3 * S2
+ /S4 * /S3 * S2
+ CPEN
+ EXPBSY
+ ATMBSY

NNA = S4 * S3 * S1 * SO * LADS * /WR * NENE * /REFREQ
+ S4 * /S3 * S2 * S1 * SO * LADS * /WR * NENE * /REFREQ
+ /S4 * S2 * S1 * SO * LADS * /WR * NENE * /REFREQ
+ /S4 */S3 */S2 */S1 */SO

NNA.TRST = /EXPBSY * /ATMBSY * DRAMSY

NRDY = S4 * S3 * S1 * SO * LADS * WR * NENE * /REFREQ
+ S4 * S3 * /S1 */SO
+ S4 * /S3 * S1 * /SO

NRDY.TRST = /EXPBSY * /ATMBSY * DRAMSY

; Description: This PLD is the state machine for the i860 DRAM interface.
; There are 32 states some of which are redundant to minimize the number of
; product terms. This design supports a non-interleaved memory design using
; Static-column 256K x4 DRAMs for the main memory, and 1M x1 DRAMs for the
; parity bits.
There are six types of cycles possible: far read, far write, near read, near write, refresh, and idle. Near read and near write cycles imply memory accesses with the same row address as the preceding cycle. Refresh request has the highest priority and will interrupt near memory cycles.

Read cycles will use full pipelining, write cycles are never pipelined. Near read cycles will be zero wait-state when following read cycles, and near write cycles will be zero wait-state when following write cycles.

DRAMBSY prevents other units from starting a cycle when the DRAM controller is busy.

Title: DRAM Control A.
Pattern: DCTLA0.PDS

;U98
; Reference Sheet 4 of the Schematics.

CHIP DCTLA0 PAL16L8

A31 A23 A22 A21 /S0 /S1 /S2 /S3 /S4 GND
NCO HOLD /HREQ /RASPO /RAS3 /RAS2 /RAS1 /RAS0 /KEN VCC

Equations:

```
RAS0 = A23 * A22 * A21 * S4 * S3 * /S2 * /S1 * /S0 ;start write
+ A23 * A22 * A21 * /S4 * /S3 * S2 * /S1 * /S0 ;start read
+ /S4 * S3 * /S2 * /S1 * S0 ;start refresh
+ RAS0 * S4 ;hold
+ RAS0 * S3 ;hold
+ RAS0 * S2 ;hold

RAS1 = A23 * A22 * /A21 * S4 * S3 * /S2 * /S1 * /S0 ;start write
+ A23 * A22 * /A21 * /S4 * /S3 * S2 * /S1 * /S0 ;start read
+ /S4 * S3 * /S2 * /S1 * S0 ;start refresh
+ RAS1 * S4 ;hold
+ RAS1 * S3 ;hold
+ RAS1 * S2 ;hold

RAS2 = A23 * /A22 * A21 * S4 * S3 * /S2 * /S1 * /S0 ;start write
+ A23 * /A22 * A21 * /S4 * /S3 * S2 * /S1 * /S0 ;start read
+ /S4 * S3 * /S2 * /S1 * S0 ;start refresh
+ RAS2 * S4 ;hold
+ RAS2 * S3 ;hold
+ RAS2 * S2 ;hold
```
RAS3 = A23 * /A22 * /A21 * S4 * S3 * S2 * /S1 * /S0 ; start write
  + A23 * /A22 * /A21 * /S4 * S3 * S2 * /S1 * /S0 ; start read
  + /S4 * S3 * /S2 * /S1 * S0 ; start refresh
  + RAS3 * S4
  + RAS3 * S3
  + RAS3 * S2

RASPO = A23 * S4 * S3 * /S2 * /S1 * /S0 ; start write
  + A23 * /S4 * /S3 * S2 * /S1 * /S0 ; start read
  + /S4 * S3 * /S2 * /S1 * S0 ; start refresh
  + RASP0 * S4
defs; ; hold
  + RASP0 * S3
defs; ; hold
  + RASP0 * S2
defs; ; hold

KEN = A31
/HOLD = /HREQ

; Description:
; This PLD decodes and drives the RAS lines for the DRAMs from
; the main state machine and the i860 addresses. It drives the RAS
; lines for the 8 Mbytes on the board.
; The addresses (A21-A23) are decoded and activated when the
; DRAM state machine starts a read or a write cycle. They are held
; active until the state machine enters the Precharge sequence
; (State 31). All RAS lines are active during a refresh cycle.
; This PLD must be "D" speed.

TITLE DRAM Control B.
PATTERN DCTLB1.PDS

; U99
; This PLD decodes and drives numerous DRAM control signals.
; Must be "D" speed.
CHIP DCTLB1 PAL16R6

DCLK /EWDL /ERDL /SO /S1 /S2 /S3 /S4 /RTDS GND
/OE /RTAB /WDL /RDL /WEL /CSX /CAE /CAL /RTBA VCC

EQUATIONS
RTAB = RTDS
RTBA = RTDS
RDL := /ERDL * /SO ;/RDL IS HIGH DURING
+ /ERDL * /S1 ;STATE 8,16,24
+ /ERDL * /S2
+ /ERDL * S3 * S4

WEL := S4 * S3 * S2 * S1 * /SO ;state 1
+ S4 * S3 * S2 * /S1 * /SO ;state 3
+ S4 * S3 * /S2 * S1 * /SO ;state 5
+ S4 * S3 * /S2 * /S1 * /SO ;state 7

WDL := /EWDL * /SO ;/WDL IS HIGH DURING
+ /EWDL * /S3 ;STATE 0,2,4,6
+ /EWDL * /S4

CSX := S4 * S3 * S2 * /SO ;state 1, 3
+ S4 * S3 * /S2 * S1 * /SO ;state 5
+ S4 * /S3 * S2 * S1 ;state 8, 9
+ S4 * /S3 * /S2 * S1 ;state 12,13
+ /S4 * S3 * S2 * S1 ;state 16,17
+ /S4 * S3 * /S2 * /S1 ;state 22,23
+ /S4 * /S3 * S2 * S1 ;state 24,25
+ /S4 * S2 * /S1 * /SO ;state 18,26

CAL := /SO ;all odd states.

CAE := S4 * S3 * S2 ;states 0,1,2,3
+ S4 * S3 * /S2 * S1 ;states 4,5
+ S4 * S3 * /S2 * /S1 * /SO ;state 6
+ S4 * /S3 * S1 ;states 8,9,12,13
+ /S4 * S3 * S2 * S1 ;states 16,17
+ /S4 * /S3 * S2 * S1 ;states 24,25
+ /S4 * S2 * /S1 * /SO ;state 18,26

; Description:

CAE Column Address Enable. Turns on the Column address drivers.

CAL Column Address Latch. Latches the N10 addresses for the DRAM column address.

CSX Chip Select for the DRAM array.

WEL Write Enable Latch. Latches the BEN bits to drive the active DRAM WEs.

WDL Write Data Latch. For DRAM accesses only.

RDL Read Data Latch. For DRAM accesses only.
MEMORY INTERFACE SCHEMATICS AND PLD CODE

TITLE          DRAM Control C.
PATTERN        DCTLC2.PDS

; U100
; This PLD decodes and drives numerous DRAM control signals.
; Must be "D" speed.

CHIP DCTLC2 PAL16R6

CLKA WR /S0 /S1 /S2 /S3 /S4 /EWDE /ERDE GND
/OE /PERLTC /DRMEN /DRMDIR /CLRFRQ /WPL /OEX /RAE /CLRCYC VCC

EQUATIONS

PERLTC = S4 * /S3 * S2 * S1 * /S0 ; state 9
        + S4 * /S3 * /S2 * S1 * /S0 ; state 13

CLRCYC = S4 * S3 * S2 * S1 * /S0 ; state 1
        + S4 * S3 * /S2 * S1 * /S0 ; state 5
        + S4 * /S3 * S2 * S1 * /S0 ; state 9
        + /S4 * S3 * S2 * /S1 * S0 ; state 18
        + /S4 * /S3 * S2 * S1 * /S0 ; state 25

OEX := S4 * S3 * S2 * S1, S0 * /WR ; state 0 if near read cycle.
        + S4 * S3 * /S2 * S1 * S0 * /WR ; state 4 if near read cycle.
        + S4 * /S3 * S2 * S1 ; state 8,9
        + /S4 * S3 * S2 * S1 ; state 16,17
        + /S4 * S3 * S2 * /S1 * S0 ; state 18
        + /S4 * /S3 * S2 ; state 24,25,26,27

RAE := /S4 * /S3 * /S2 * S1 ; state 28,29
        + /S4 * /S3 * /S2 * /S1 * S0 ; state 30

CLRFRQ := S4 ; ALL EXCEPT state 23
        + /S4 * S3 * S2
        + /S4 * S3 * /S2 * S1
        + /S4 * S3 * /S2 * /S1 * S0
        + /S4 * /S3

WPL := S4 * S3 * S2 * S1 * /S0 ; state 1
        + S4 * S3 * S2 * /S1 * /S0 ; state 3
        + S4 * S3 * /S2 * S1 * /S0 ; state 5
        + S4 * S3 * /S2 * /S1 * S0 ; state 7

DRMEN := S4 * S3 */S1 ; state 2,3,6,7
        + S4 * S3 * /S1 * /S0 ; state 1,5
        + S4 * /S3 * S2 * S1 ; state 8,9
        + /S4 * S2 * S1 ; state 16,17,24,25
        + S4 * S3 * /S1 * S0 * WR ; state 0,4 if write cycle.
        + EWDE
        + ERDE

DRMDIR := S4 * /S3 * S2 * S1 ; state 8,9
        + S4 * /S3 * /S2 * S1 * /S0 ; state 13
        + /S4 * S2 * /S1 * S0 ; state 18,26
+ /S4 * S2 * S1 ;state 16,17,24,25
+ ERDE

; Description:

; PERLTCH Parity Error Latch. Latches parity errors (if any) during DRAM read cycles.

; CLRFRQ Clear Refresh Request. Restarts the refresh counter.

; WPL Write Parity Latch. Latches the parity bits during a DRAM write.

; OEX Output Enable. For the DRAMS.

; RAE Row Address Enable. Turns on the row addresses to the DRAM.

; CLRCYC Clear Cycle. Clears queued cycles.

TITLE DRAM Control D.
PATTERN DCTLD0.PDS

;This PLD decodes and drives the DRAM Write Enable signals.
;Can be "B" speed.

CHIP DCTLD0 PAL20R8

/WEL /CPEN /DSEL /WR /BE0 /BE1 /BE2 /BE3 /BE4 /BE5 /BE6 GND
/OE /BE7 /WE7 /WE6 /WE5 /WE4 /WE3 /WE2 /WE1 /WE0 /LADS VCC

EQUATIONS

WE7 := LADS * DSEL * /WR * BE7
     + CPEN * DSEL * /WR * BE7

WE6 := LADS * DSEL * /WR * BE6
     + CPEN * DSEL * /WR * BE6

WE5 := LADS * DSEL * /WR * BE5
     + CPEN * DSEL * /WR * BE5

WE4 := LADS * DSEL * /WR * BE4
     + CPEN * DSEL * /WR * BE4

WE3 := LADS * DSEL * /WR * BE3
     + CPEN * DSEL * /WR * BE3

WE2 := LADS * DSEL * /WR * BE2
     + CPEN * DSEL * /WR * BE2

WE1 := LADS * DSEL * /WR * BE1
Description: This PLD samples the BEN lines during write cycles and
drives the DRAM Write Enables directly.
# DOMESTIC SALES OFFICES

**ALABAMA**
Intel Corp., 5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010
FAX: (205) 837-2640

**ARIZONA**
Intel Corp., 11325 N. 28th Dr. Suite D-214
Phoenix 85249
Tel: (602) 869-4983
FAX: (602) 869-4294

**CALIFORNIA**
Intel Corp., 1161 N. El Dorado Place
Suite 301
Tucson 85715
Tel: (520) 299-8815
FAX: (520) 296-6234

**CONNECTICUT**
Intel Corp., 301 Lee Farm Corporate Park
80 W. Knights Rd.
Danbury 86810
Tel: (203) 748-3130
FAX: (203) 794-0339

**FLORIDA**
Intel Corp., 6366 N.W. 66th Way
Suite 600
Fl. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407
FAX: (305) 772-8193

**GEORGIA**
Intel Corp., 20 Technology Parkway, N.W.
Suite 150
Norcross 30092
Tel: (770) 443-0541
FAX: (404) 605-9762

**ILLINOIS**
Intel Corp. *300 N. Martingale Road
Suite 140
 Schaumburg 60173
Tel: (847) 815-6231
FAX: (815) 796-0762

**INDIANA**
Intel Corp., 8777 Purdue Road
Suite 125
Indianapolis 46268
Tel: (317) 875-0023
FAX: (317) 675-8938

**IOWA**
Intel Corp., 1830 St. Andrews Drive N.E.
2nd Floor
Cedar Rapids 52402
Tel: (319) 393-1294

**KANSAS**
Intel Corp., 10886 Cody St.
Suite 140, Bldg. D
Overland Park 66020
Tel: (913) 345-2727
FAX: (913) 345-2076

**MARYLAND**
Intel Corp., 10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2960
FAX: (301) 206-3678

**MASSACHUSETTS**
Intel Corp. *Westford Corp. Center
2 Carlisle Road
2nd Floor
Westford 01886
Tel: (978) 692-3022
TWX: 710-343-6333
FAX: (978) 692-7687

**MICHIGAN**
Intel Corp., 7571 Orchard Lake Road
Suite 130
West Bloomfield 48322
Tel: (248) 851-8098
FAX: (248) 851-8770

**MINNESOTA**
Intel Corp., 3500 W. 30th St.
Bloomington 55431
Tel: (952) 835-6722
TWX: 910-576-2867
FAX: (612) 831-6497

**MISSOURI**
Intel Corp., 4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990
FAX: (314) 291-4341

**NEW JERSEY**
Intel Corp. *Parkway 109 Office Center
328 HoHo Springs Road
Red Bank 07701
Tel: (201) 747-2233
FAX: (201) 747-2893

**NEW YORK**
Intel Corp., 260 Corporate Center
75 Livingston Avenue
First Floor
Rochester 14609
Tel: (585) 706-0768
FAX: (585) 706-0111
FAX: (585) 706-0266

**OHIO**
Intel Corp. *3401 Park Center Drive Suite 220
Dayton 45414
Tel: (937) 890-5350
TWX: 910-452-2528
FAX: (937) 690-8658

**OMAHA**
Intel Corp., 25700 Science Park Drive
Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9288
FAX: (216) 242-0793

**OKLAHOMA**
Intel Corp., 6801 N. Broadway
Suite 115
Oklahoma City 73162
Tel: (405) 648-8086
FAX: (405) 643-9199

**OREGON**
Intel Corp., 15254 N.W. Greenbrier Parkway
Building B
Beaverton 97070
Tel: (503) 645-8051
TWX: 910-467-9741
FAX: (503) 645-8181

**PENNSYLVANIA**
Intel Corp. *455 Pennsylvania Avenue
Suite 230
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077
FAX: (215) 641-0785

**PUERTO RICO**
Intel Corp., 400 Penn Center Blvd.
Suite 610
Pittsburgh 15235
Tel: (412) 853-4970
FAX: (412) 829-7578

**QUEBEC**
Intel Corp., 1504 Santa Rosa Road
Suite 108
Richmond 23288
Tel: (804) 292-5668
FAX: (216) 464-2270

**WASHINGSTON**
Intel Corp., 125 10th Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002
FAX: (206) 451-9556

**WISCONSIN**
Intel Corp., 330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087
FAX: (414) 796-2115

**BRITISH COLUMBIA**
Intel Semiconductors of Canada, Ltd.
555 Canada Way
Burnaby V5G 4L6
Tel: (604) 598-4378
FAX: (604) 258-8234

**ONTARIO**
Intel Semiconductors of Canada, Ltd.
2650 Queenview Drive
Suite 250
Ottawa 0457
Tel: (613) 623-9714
FAX: (416) 520-9296

**QUEBEC**
Intel Semiconductors of Canada, Ltd.
620 St. Jean Boulevard
Pointe Claire H9R 3K2
Tel: (514) 594-0130
FAX: (514) 594-0034

---

*Sales and Service Office
*Field Application Location
DOMESTIC DISTRIBUTORS (Contd.)

NEW JERSEY
1 Atlantic Electronics, Inc. 6 Century Drive Four East South Road Unit 11 Tele 590.596-8000 710.897-9239
Marion 08053 (Tel: (609) 424-0110 710.940-0262

HamlinTe/Avnet Electronics 36 Century Drive Parsippany 07054 (Tel: (201) 538-0900

HamlinTe/Avnet Electronics 10 Industrial Fairfield 07006 (Tel: (201) 575-5030 710.734-4388

1Mt Systems Sales 37 Kuki Fairfield 07006 (Tel: (201) 575-5552

1tPioneer Electronics 45 Round Pinebrook Tpke (Tel: (201) 757-7608 710.734-4382

NEW MEXICO
Alliance Electronics Inc. 10305 Cochele S.E. Albuquerque 87123 (Tel: (505) 292-3860 710.969-9115

HamlinTe/Avnet Electronics 2554 Bay Drive S.E. Albuquerque 87106 (Tel: (505) 765-1500 710.969-9614

NEW YORK
1Atlantic Electronics, Inc. 3375 Brighten Henrietta Townline Road Rochester 14623 (Tel: (716) 275-2000 710.525-4766

1HamlinTe/Avnet Electronics 20 022 Avenue Hauppauge 11788 (Tel: (516) 231-1000 710.527-8623

1HamlinTe/Avnet Electronics 6533 Motor Park Hauppauge 11788 (Tel: (516) 231-8600 710.524-6616

1tPioneer Electronics 333 Metro Park Rochester 14623 (Tel: (716) 475-9100 710.253-5470

1HamlinTe/Avnet Electronics 103 2nd Ave Laurel Station 13208 (Tel: (315) 477-0000 710.547-1560

1Mt Systems Sales 38 Harbor Park Drive Port Washington 11050 (Tel: (516) 621-6200

1tComputer System Technology Distribution Center
<table>
<thead>
<tr>
<th>Country</th>
<th>Address</th>
<th>Phone</th>
<th>Fax</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUSTRIA</td>
<td>Bachelor Electronics G.m.b.H.</td>
<td>(43) 35 85 47</td>
<td>45094080</td>
<td>Munich</td>
</tr>
<tr>
<td></td>
<td>Rothenmauhlgasse 26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1120 Wien</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tel: (31) 02 45 16 22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fax: 83 56 46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31532</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BELGIUM</td>
<td>Inetco Belgium S.A.</td>
<td>(02) 24 40 73</td>
<td>02/244012</td>
<td>Brussels</td>
</tr>
<tr>
<td></td>
<td>Av. des Croix de Guerre 94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1120 Brussels</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tel: (02) 216 01 60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fax: 64 4754 or 2290</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DENMARK</td>
<td>ITT-Multicomponent Naverland 29</td>
<td>(46) 10 58 24</td>
<td>460461</td>
<td>Aarhus</td>
</tr>
<tr>
<td></td>
<td>2600 Glostrup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tel: (3) 2 45 66 45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fax: 33 35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FINLAND</td>
<td>OY Fintronico Ab</td>
<td>(46) 40 40 12</td>
<td>46044006</td>
<td>Helsinki</td>
</tr>
<tr>
<td></td>
<td>Melkontula 24A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>02010 Helsinki</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tel: (358) (0) 8296222</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fax: 124222</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FRANCE</td>
<td>Arnex</td>
<td>(1) 46 04 40</td>
<td>46044006</td>
<td>Paris</td>
</tr>
<tr>
<td></td>
<td>Zone industrielle d’Antony 48, rue de l’Alibépine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BP 102</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>92164 Antony cedex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tel: (33) (1) 46 66 21 12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fax: (33) (1) 49 78 49 78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2615683</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Metrologie</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tour d’Aireieres 4, av. Laurent-Cely</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>92068 Aireieres Cedex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tel: (33) (1) 47 90 62 40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fax: 611448</td>
<td></td>
<td></td>
<td></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, Tsukuba-shi
Ibaraki, 300-26

FRANCE
Intel Corporation S.A.R.L.
1, Rue Edison, BP 303
78054 Saint-Quentin-en-Yvelines Cedex

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen

HONG KONG
Intel Semiconductor Ltd.
10/F East Tower
Bond Center
Queensway, Central

CANADA
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

Order Number: 240330-002

Printed in U.S.A. /1289/20K/RRD JM
Microprocessors

ISBN 1-55512-106-3