400029

National Semiconductor

IBM® Data Communications Handbook

IBM DATA COMMUNICATIONS HANDBOOK

1992 Edition

IBM Data Communications

Application Notes

Physical Dimensions

1

2

3

TRADEMARKS

Following is the most current list of National Semiconductor Corporation's trademarks and registered trademarks.

ABiC™
Abuseable™
Anadig™
ANS-R-TRAN™
APPS™
ASPECT™
Auto-Chem Deflasher™
BCP™
BI-FET™
BI-FET II™
BI-LINE™
BIPLAN™
BLC™
BLX™
BMAC™
Brite-Lite™
BSI™
CDD™
CheckTrack™
CIM™
CIMBUS™
CLASIC™
Clock✓Chek™
COMBO®
COMBO I™
COMBO II™
COPS™ microcontrollers
CRD™
DA4™
Datachecker®
DENSPAK™
DIB™
DISCERN™
DISTILL™
DNR®
DPVM™
E²CMOS™
ELSTAR™
Embedded System Processor™
EP™
E-Z-LINK™
FACT™
FACT Quiet Series™
FAIRCAD™
Fairtech™
FAST®
FASTr™
5-Star Service™
Flash™
GENIX™
GNX™
GTO™
HAMR™
HandiScan™
HEX 3000™
HPC™
I³L®
ICM™
INFOCHEX™
Integral ISE™
Intelisplay™
ISE™
ISE/06™
ISE/08™
ISE/16™
ISE32™
ISOPLANAR™
ISOPLANAR-Z™
KeyScan™
LERIC™
LMCMOS™
M²CMOS™
Macrobus™
Macrocomponent™
MAPL™
MAXI-ROM®
Meat✓Chek™
MenuMaster™
Microbus™ data bus
MICRO-DAC™
μtalker™
Microtalker™
MICROWIRE™
MICROWIRE/PLUS™
MOLE™
MPA™
MST™
Naked-8™
National®
National Semiconductor®
National Semiconductor Corp.®
NAX 800™
Nitride Plus™
Nitride Plus Oxide™
NML™
NOBUS™
NSC800™
NSCISE™
NSX-16™
NS-XC-16™
NTERCOM™
NURAM™
OPAL™
OXISS™
P²CMOS™
PC Master™
Perfect Watch™
Pharma✓Chek™
PLAN™
PLANAR™
PLAYER™
Plus-2™
Polycraft™
POSilink™
POSitalker™
Power + Control™
POWERplanar™
QUAD3000™
QUIKLOOK™
RAT™
RIC™
RTX16™
SABR™
Script✓Chek™
SCX™
SERIES/800™
Series 900™
Series 3000™
Series 32000®
Shelf✓Chek™
Simple Switcher™
SofChek™
SONIC™
SPIRE™
Staggered Refresh™
STAR™
Starlink™
STARPLEX™
ST-NIC™
SuperAT™
Super-Block™
SuperChip™
SuperScript™
SYS32™
TapePak®
TDS™
TeleGate™
The National Anthem®
Time✓Chek™
TINA™
TLC™
Trapezoidal™
TRI-CODE™
TRI-POLY™
TRI-SAFE™
TRI-STATE®
TURBOTRANSCEIVER™
VIP™
VR32™
WATCHDOG™
XMOS™
XPU™
Z STAR™
883B/RETS™
883S/RETS™

abel™ is a trademark of Data I/O Corporation.
BRIEF™ and UnderWare™ are trademarks of UnderWare, Inc.
Crosstalk® and DCA® are registered trademarks of Digital Communication Associates, Inc.
Hewlett Packard™ is a trademark of Hewlett Packard Company.
IBM®, MICROCHANNEL®, PC®, and PS/2® are registered trademarks of International Business Machines Corp.
IRMA™ and SMART ALEC™ are trademarks of Digital Communication Associates, Inc.
Microsoft® and MS-DOS® are registered trademarks of Microsoft Corporation.
PAL® is a registered trademark of and used under license from Advanced Micro Devices, Inc.
RELAY Gold™ is a trademark of RELAY Communications, Inc.
SimPC Master™ is a trademark of Simware Inc.
Xeus™ is a trademark of Fischer International Systems Corporation.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation 2900 Semiconductor Drive, P.O. Box 58090, Santa Clara, California 95052-8090 1-800-272-9959
TWX (910) 339-9240

National Semiconductor

Introduction to IBM Data Communications

IBM 3270/3299 PROTOCOL

The IBM 3270/3299 serial communications protocol was developed by IBM for the cluster controller-peripheral link in 370 class mainframe systems. Fortune 1000 corporations that use these systems have large scale networking needs and often support thousands of terminals and printers. Although PC-based networks have increased in popularity, shipments of IBM 3270 peripherals have remained steady over the last few years due to the huge investments made in both hardware and software application development.

The 3299 protocol is a variation of the 3270 protocol in that an 8-bit address byte is asserted between the starting sequence and the first word for each out board transmission from the controller. This was done to allow up to eight 3270 peripherals to be multiplexed and connected to the controller via a single coax cable, thus reducing cabling costs. The multiplexing and de-multiplexing is done with a 3299 terminal multiplexer.

IBM 5250 PROTOCOL

The 5250 serial communications protocol was developed by IBM originally for the mid-range System 3x line of computers. IBM has updated the System 3x series to the AS/400. The AS/400 line can vary from small office environment processors to more powerful processors with greatly enhanced networking facilities that rival the smaller 370 class mainframes. They are typically used in hotels, bank branch offices and hospitals for a variety of tasks.

NATIONAL'S SOLUTION

With over a decade of shipments into the IBM 3270 connectivity market, National is the leading standard product semiconductor supplier. The first generation DP8340/41 protocol translation chips were used in DCA's industry standard IRMA cards which were the first 3270 terminal emulation products available for IBM PC's. Although the DP8340/41 pair solved many design issues regarding IBM 3270 protocol, bit slice microcontrollers were still required to meet the fast response times specified by IBM. To address this issue National introduced the DP8344 Biphase Communications Processor in 1987. This product features a 3270/3299/5250 transceiver tightly coupled to a high speed RISC CPU. The BCP was the first single hardware platform capable of supporting the 3270, 3299 and 5250 datastreams. This new product was well received by corporations such as Memorex Telex, IBM, DEC, Harris Adacom, Tandberg Data, IIS, Apple Computer, and many others.

With a combination of experience in IBM connectivity protocols, mixed signal design capabilities, extensive laboratory resources, and knowledge of IBM peripherals (terminals, printers, terminal emulation cards), National will continue to develop products that meet the semiconductor needs of our customers. By working in conjunction with third parties, National can offer the complete hardware and software solution to IBM Data Communications.

Table of Contents

Alpha-Numeric Index

National Semiconductor

Product Status Definitions

Definition of Terms

Data Sheet Identification	Product Status	Definition
Advance Information	Formative or In Design	This data sheet contains the design specifications for product development. Specifications may change in any manner without notice.
Preliminary	First Production	This data sheet contains preliminary data, and supplementary data will be published at a later date. National Semiconductor Corporation reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.
No Identification Noted	Full Production	This data sheet contains final specifications. National Semiconductor Corporation reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.

National Semiconductor Corporation reserves the right to make changes without further notice to any products herein to improve reliability, function or design. National does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others.

Section 1

IBM Data Communications

Section 1 Contents

National Semiconductor

DP8340 IBM 3270 Protocol Transmitter/Encoder

General Description

The DP8340 generates a complete encoding of parallel data for high speed serial transmission which conforms to the protocol as defined by the IBM 3270 information display system standard. The DP8340 converts parallel input data into a serial data stream. Although the IBM standard covers biphase serial data transmission over a coax line, the DP8340 also adapts to general high speed serial data transmission over other than coax lines, at frequencies either higher or lower than the IBM standard.

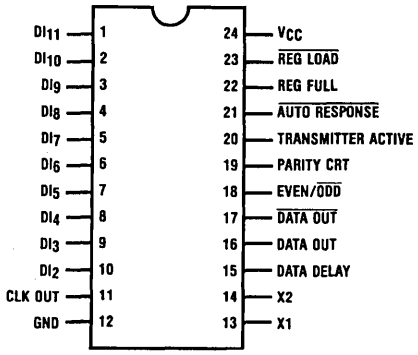
The DP8340 and its complementary chip, the DP8341 (receiver/decoder) have been designed to provide maximum flexibility in system designs. The separation of the transmitter/receiver functions provides convenient addition of more receivers at one end of a biphase line without the need of unused transmitters. This is specifically advantageous in control units where typical biphase data is multiplexed over many biphase lines and the number of receivers generally exceeds the number of transmitters.

Features

- Ten bits per data byte transmission
- Single-byte or multi-byte transmission
- Internal parity generation (even or odd)
- Internal crystal controlled oscillator used for the generation of all required chip timing frequencies
- Clock output directly drives receiver (DP8341) clock input
- Input data holding register
- Automatic clear status response feature
- Line drivers at data outputs provide easy interface to biphase coax line or general transmission lines
- < 2 ns driver output skew
- Bipolar technology provides TTL input/output compatibility
- Data outputs power up/down glitch free
- Internal power up clear and reset
- Single +5V power supply

Connection Diagram

Dual-In-Line Package



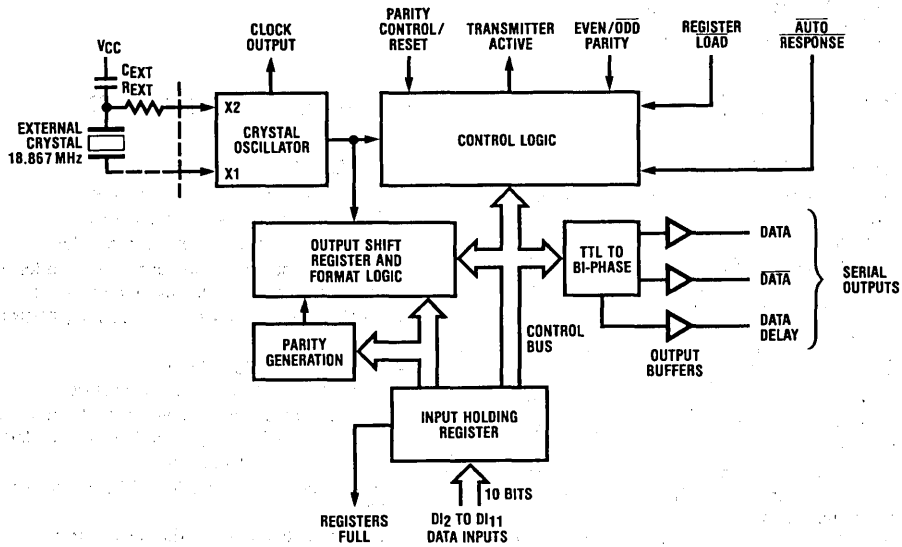
TL/F/5251-1

Top View

FIGURE 1

Order Number DP8340N
See NS Package Number N24A

Block Diagram



TL/F/5251–2

FIGURE 2. DP8340 Serial Bi-Phase Transmitter/Encoder Block Diagram

Functional Description

Figure 2 is a block diagram of the DP8340 biphase Transmitter/Encoder. The transmitter/encoder contains a crsytal oscillator whose input is a crystal with a frequency eight (8) times the data rate. A Clock Output is provided to drive the DP8341 receiver/decoder Clock Input and other system components at the oscillator frequency. Additionally, the oscillator drives the control logic and output shift register/format logic blocks.

Data is parallel loaded from the sytem data bus to the transmitter/encoder's input holding register. This data is in turn loaded by the transmitter/encoder to its output shift register if this register was empty at the time of the load. During this load, message formatting and parity are generated. The formatted message is then shifted out at the bit rate frequency to the TTL to biphase block which generates the proper data bit formatting. The three data outputs, DATA, $\overline{\text{DATA}}$, and DATA DELAY provide for flexible interface to the coax line with a minimum of external components.

The Control Logic block interfaces to all blocks to insure proper chip operation and sequencing. It controls the type of parity generation through the Even/Odd Parity input. An additional feature provided by the transmitter/encoder is generation of odd parity and placement in bit 10 position while still maintaining even or odd parity in the bit 12 position. This is the format of data word bytes and other commands in the 3270 Standard. The Parity Control input is the pin which controls when this operation is in effect.

Another feature of the transmitter/encoder is the internal TT/AR (Transmission Turnaround/Auto Response) capability. After each Write type message from the control unit in the 3270 Standard, the receiving unit must respond with clean status (bits 2 through 11). With the transmitter/encoder, this function is accomplished simply by forcing the $\overline{\text{Auto-Response}}$ input to the Logic "0" state.

Operation of the transmitter/encoder is automatic. After the first data byte is loaded, the Transmitter Active output is set and the transmitter/encoder immediately formats the input data and serially shifts it out its data outputs. If the message is a multi-byte message, the internal format logic will modify the message data format for multibyte as long as the next byte is loaded to the input holding register before the last data bit of the previous data byte is transferred out of the internal output shift register. After all data is shifted out of the transmitter/encoder the Transmitter Active output will return to the inactive state.

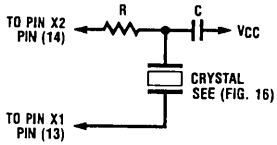
Detailed Pin/Functional Description

Crystal Inputs X1 and X2

The oscillator is controlled by an external, parallel resonant crystal connected between the X1 and X2 pins. Normally, a fundamental mode crystal is used to determine the operating frequency of the osicllator; however, overtone mode crystals may be used.

Crystal Specifications (Parallel Resonant)

Type	AT-cut crystal
Tolerance	0.005% at 25°C
Stability	0.01% from 0°C to +70°C
Resonance	Fundamental (Parallel)
Maximum Series Resistance	Dependent on Frequency (For 18.867 MHz, 50Ω)
Load Capacitance	15 pF



FREQ	R	C
10 MHz to 20 MHz	500Ω ±10%	30 pF
>20 MHz	120Ω ±10%	15 pF

TL/F/5251-3

FIGURE 3. Connection Diagram

If the DP8340 transmitter is clocked by a system (clock crystal oscillator not used), pin 13 (X1 input) should be clocked directly using a Schottky series (74S) circuit. Pin 14 (X2 input) may be left open. The clocking frequency must be set at eight times the data bit rate. Maximum input frequency is 28 MHz. For the IBM 3270 Interface, this frequency is 18.867 MHz. At this frequency, the serial bit rate will be 2.358 Mbits/sec.

Clock Output

The Clock Output is a buffered output derived directly from the crystal oscillator block and clocks at the oscillator frequency. It is designed to directly drvie the DP8341 receiver/decoder Clock Input as well as other system components.

Registers Full

This output is used as a flag by the external operating system. A logic "1" (active state) on this output indicates that both the internal output shift register and the input holding register contain active data. No additional data should be loaded until this output returns to the logic "0" state (inactive state).

Transmitter Active

This output will be in the logic "1" state while the transmitter/encoder is about to transmit or in the process of transmitting data. Otherwise, it will assume the logic "0" state indicating no data presently in either the input holding or output shift registers.

$\overline{\text{Register Load}}$

The $\overline{\text{Register Load}}$ input is used to load data from the Data Inputs to the input holding register. The loading function is edge sensitive, the data present during the logic "0" state of this input is loaded, and the input data must be valid before the logic "0" to logic "1" transition. It is after this transition that the transmitter/encoder begins formatting of data for serial transmission.

$\overline{\text{Auto Response}}$ (TT/AR)

This input provides for automatic clear data transmission (all bits in logic "0") without the need of loading all zero's. When a logic "0" is forced on this inpiut the transmitter/encoder immediately responds with transmission of "clean status". This function is necessary after the completion of each write type command and in other functions in the 3270 specification. In the logic "1" state the transmitter/encoder transmits data entered on the Data Inputs.

Even/$\overline{\text{Odd}}$ Parity

This input sets the internal logic of the DP8340 transmitter/encoder to generate either even or odd parity for the data byte in the bit 12 position. When this pin is in the logic "0" state odd parity is generated. In the logic "1" state even parity is generated. This feature is useful when the control unit is performing a loop back check and at the same time the controller wishes to verify proper data transmission with its receiver/decoder.

Parity Control/Reset

Depending on the type of message transmitted, it is at times necessary in the IBM 3270 specification to generate an additional parity bit in the bit 10 position. The bit generated is odd parity on the previous eight (8) bits of data. When the Parity Control input is in the logic "1" state the data entered at the Data Bit 10 position is placed in the transmitted word. With the Parity Control input in the logic "0" state the Data Bit 10 input is ignored and odd parity on the previous data bits is placed in the normal bit 10 position while overall word parity (bit 12) is even or odd (controlled by Even/$\overline{\text{Odd}}$ Parity input). This eliminates the need for external logic to generate the parity on the data bits.

Truth Table

Parity Control Input	Transmitted Data Bit 10
Logic "1"	Data entered on Data Input 10
Logic "0"	Odd Parity on 8-bit data byte

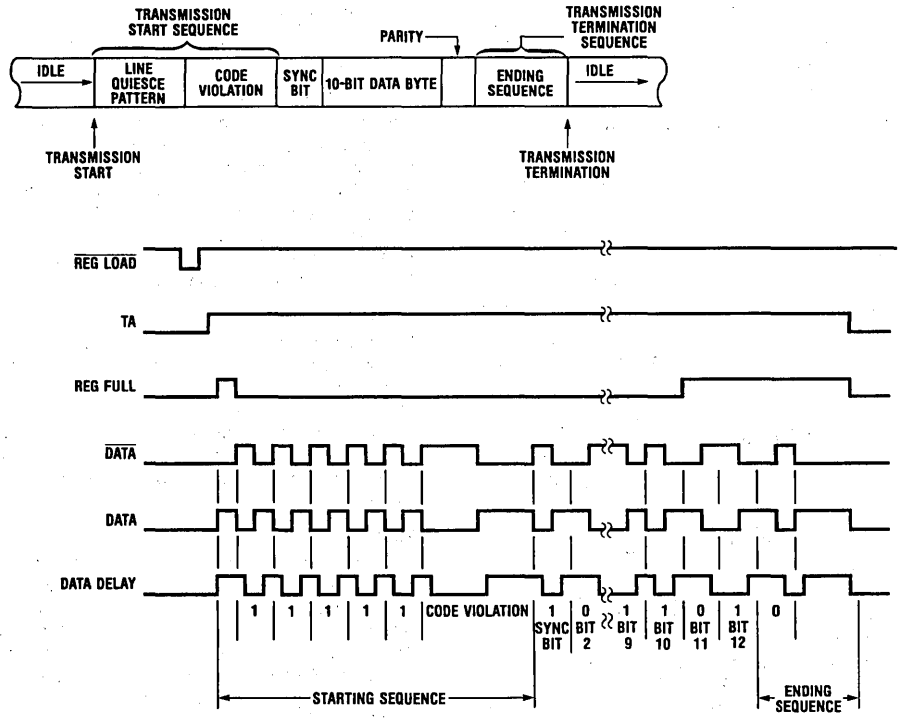
When this input is driven to a voltage that exceeds the power supply level (9V to 13V) the transmitter/encoder is reset.

Serial Outputs—DATA, $\overline{\text{DATA}}$, and DATA DELAY

These three output pins provide for convenient application of data to the biphase Coax line (see *Figure 15* for application). The Data outputs are a direct bit representation of the biphase data while the DATA DELAY output provides the necessary increment to clearly define the four (4) DC levels of the pulse. The DATA and $\overline{\text{DATA}}$ outputs add flexibility to the DP8340 transmitter/encoder for use in high speed differential line driving applications.

Functional Timing Waveforms—Message Format

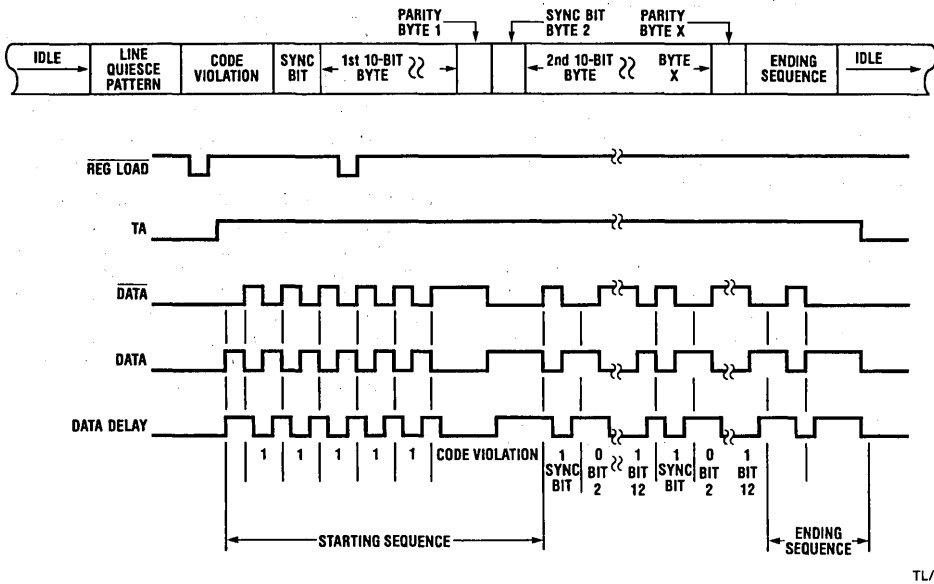
Single Byte Transmission



TL/F/5251–4

FIGURE 4. Overall Timing Waveforms for Single Byte

Multi-Byte Transmission



TL/F/5251–5

FIGURE 5. Overall Timing Waveforms for Multi-Byte

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage, V_{CC}	7V
Input Voltage	5.5V
Output Voltage	5.25V
Storage Temperature Range	−65°C to +150°C
Lead Temperature (Soldering, 10 sec.)	300°C
Maximum Power Dissipation @25°C* Dual-In-Line Package	2500 mW

*Derate dual-in-line package 20 mW/°C above 25°C.

Operating Conditions

	Min	Max	Units
Supply Voltage, (V_{CC})	4.75	5.25	V
Ambient Temperature, T_A	0	+70	°C

Electrical Characteristics (Notes 2 and 3)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Logic "1" Input Voltage (All Inputs Except X1 and X2)		2.0			V
V_{IL}	Logic "0" Input Voltage (All Inputs Except X1 and X2)				0.8	V
V_{CLAMP}	Input Clamp Voltage (All Inputs Except X1 and X2)	I_{IN} = −12 mA		−0.8	−1.2	V
I_{IH}	Logic "1" Input Current Register Load Input	V_{CC} = 5.25V, V_{IN} = 5.25V		0.3	120	μA
	All Others Except X1 and X2			0.1	40	μA
I_{IL}	Logic "0" Input Current Register Load Input	V_{CC} = 5.25V, V_{IN} = 0.5V		−15	−300	μA
	All Inputs Except X1 and X2			−5	−100	μA
V_{OH1}	Logic "1" All Outputs Except CLK OUT, DATA, $\overline{DATA}$, and DATA DELAY	I_{OH} = −100 μA	3.2	3.9		V
		I_{OH} = −1 mA	2.5	3.4		V
V_{OH2}	Logic "1" for CKL OUT, DATA, $\overline{DATA}$ and DATA DELAY Outputs	I_{OH} = −10 mA	2.6	3.0		V
V_{OL1}	Logic "0" All Outputs Except CLK OUT, DATA, $\overline{DATA}$ and DATA DELAY Outputs	I_{OL} = 5 mA		0.35	0.5	V
V_{OL2}	Logic "0" for CLK OUT, DATA, $\overline{DATA}$ and DATA DELAY Outputs	I_{OL} = 20 mA		0.4	0.6	V
I_{OS1}	Short Circuit Current for All Outputs Except CLK OUT, DATA, $\overline{DATA}$, and DATA DELAY	V_{OUT} = 0V (Note 4)	−10	−30	−100	mA
I_{OS2}	Short Circuit Current for DATA, $\overline{DATA}$, and DATA DELAY Outputs	V_{OUT} = 0V (Note 4)	−50	−140	−350	mA
I_{OS3}	Short Circuit Current for CLK OUT	(Note 4)	−30	−90	−200	mA
I_{CC}	Power Supply Current	V_{CC} = 5.25V		170	250	mA

Timing Characteristics Oscillator Frequency = 18.867 MHz (Notes 2 and 3)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
t_{pd1}	$\overline{REG\ LOAD}$ to Transmitter Active (T_A) Positive Edge	Load Circuit 1 *Figure 7*		60	90	ns
t_{pd2}	$\overline{REG\ LOAD}$ to REG Full; Positive Edge	Load Circuit 1 *Figure 7*		45	75	ns
t_{pd3}	Register Full to T_A; Negative Edge	Load Circuit 1 *Figure 7*		40	70	ns
t_{pd4}	Positive Edge of $\overline{REG\ LOAD}$ to Positive Edge of DATA	Load Circuits 1 & 2 *Figure 9*		50	80	ns

Timing Characteristics Oscillator Frequency = 18.867 MHz (Notes 2 and 3) (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
t_{pd5}	$\overline{\text{REG LOAD}}$ to $\overline{\text{DATA}}$; Positive Edge	Load Circuits 1 & 2 *Figure 9,* (Note 6)		380	475	ns
t_{pd6}	$\overline{\text{REG LOAD}}$ to DATA DELAY; Positive Edge	Load Circuits 1 & 2 *Figure 9,* (Note 6)		160	250	ns
t_{pd7}	Positive Edge of $\overline{\text{DATA}}$ to Negative Edge of DATA DELAY	Load Circuit 2 *Figure 9,* (Note 6)		100	115	ns
t_{pd8}	Positive Edge of DATA DELAY to Negative Edge of $\overline{\text{DATA}}$	Load Circuit 2 *Figure 9,* (Note 6)		110	125	ns
t_{pd9}, t_{pd10}	Skew between DATA and $\overline{\text{DATA}}$	Load Circuit 2 *Figure 9*		2	6	ns
t_{pd11}	Negative Edge of Auto Response to Positive Edge of TA	Load Circuit 1 *Figure 10*		70	110	ns
t_{pd12}	Maximum Time Delay to Load Second Byte after Positive Edge of REG FULL	Load Circuit 1 *Figure 8,* (Note 6)			$4 \times T - 50$	ns
t_{pd13}	X1 to CLK OUT; Positive Edge	Load Circuit 2 *Figure 13*		21	30	ns
t_{pd14}	X1 to CLK OUT; Negative Edge	Load Circuit 2 *Figure 13*		23	33	ns
t_{pd15}	Negative Edge of AR to Positive Edge of REG FULL	Load Circuit 1 *Figure 10*		45	75	ns
t_{pd16}	Skew between TA and REG FULL during Auto Response	Load Circuit 1 *Figure 10*		50	80	ns
t_{pd17}	$\overline{\text{REG LOAD}}$ to REG FULL; Positive Edge for Second Byte	Load Circuit 1 *Figure 14*		45	75	ns
t_{pw1}	$\overline{\text{REG LOAD}}$ Pulse Width	*Figure 12*	40			ns
t_{pw2}	First REG FULL Pulse Width (Note 5)	Load Circuit 1 *Figure 7,* (Note 6)		$8 \times T + 60$	$8 \times T + 100$	ns
t_{pw3}	REG FULL Pulse Width prior to Ending Sequence (Note 5)	Load Circuit 1, *Figure 7,* (Note 6)		$5 \times B$		ns
t_{pw4}	Pulse Width for Auto Response	*Figure 10*	40			ns
t_S	Data Setup Time prior to $\overline{\text{REG LOAD}}$ Positive Edge, Hold Time (t_H) = 0 ns	*Figure 12*		15	25	ns
t_{r1}	Rise Time for DATA, $\overline{\text{DATA}}$, and DATA DELAY Output Waveform	Load Circuit 2 *Figure 11*		7	13	ns
t_{f1}	Fall Time for DATA, $\overline{\text{DATA}}$, and DATA DELAY Output Waveform	Load Circuit 2 *Figure 11*		5	11	ns
t_{r2}	Rise Time for TA and REG FULL	Load Circuit 1 *Figure 15*		20	30	ns
t_{f2}	Fall Time for TA and REG FULL	Load Circuit 1 *Figure 15*		15	25	ns
f_{MAX}	Data Rate Frequency (Clock Input must be 8X this Frequency)	(Note 7)	DC		3.5	Mbits/s

Note 1: "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. They are not meant to imply that the device should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

Note 2: Unless otherwise specified, min./max. limits apply across the 0°C to +70°C temperature range and the 4.75V to 5.25V power supply range. All typical values are for T_A = 25°C and V_{CC} = 5.0V.

Note 3: All currents into device pins are shown as positive; all currents out of device pins are shown as negative; all voltages are referenced to ground, unless otherwise specified. All values shown as max. or min. are so classified on absolute basis.

Note 4: Only one output should be shorted at a time. Output should not be shorted for more than one second at a time.

Note 5: T = 1/(Oscillator Frequency), unit for T should be ns. B = 8T

Note 6: Oscillator Frequency Dependent.

Note 7: For the IBM 3270 Interface, the data rate frequency is 2.358 Mbits/s. 28 MHz clock frequency corresponds to 3.75% jitter when referenced to *Figure 10* of DP8341 Datasheet.

Timing Characteristics (Continued)

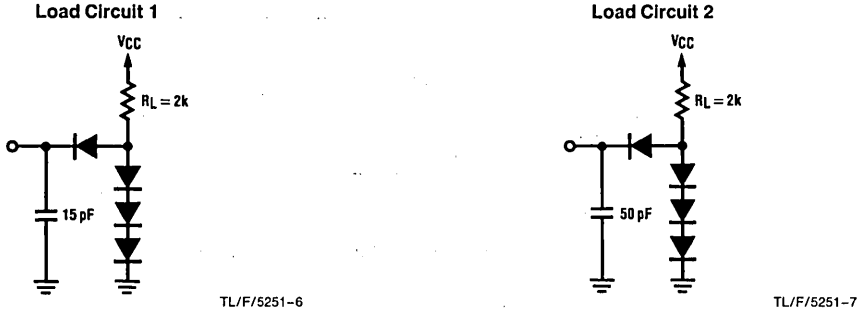


FIGURE 6. Test Load Circuits

Timing Waveforms

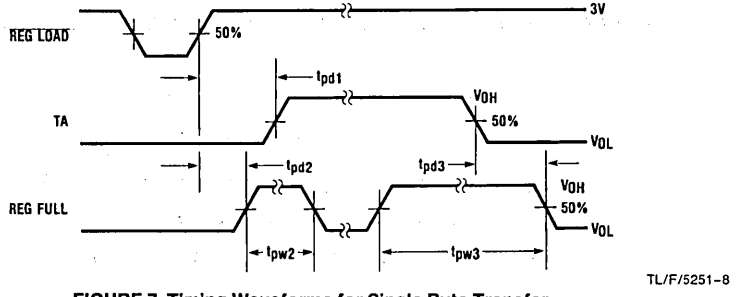


FIGURE 7. Timing Waveforms for Single Byte Transfer

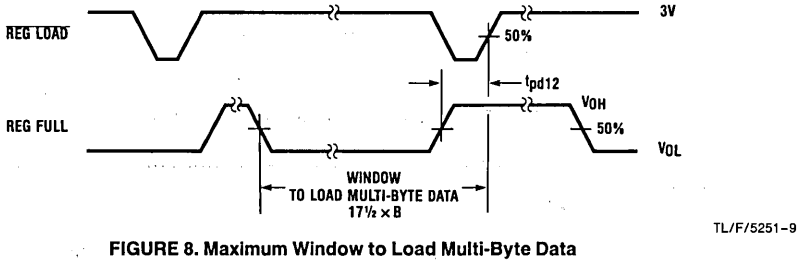


FIGURE 8. Maximum Window to Load Multi-Byte Data

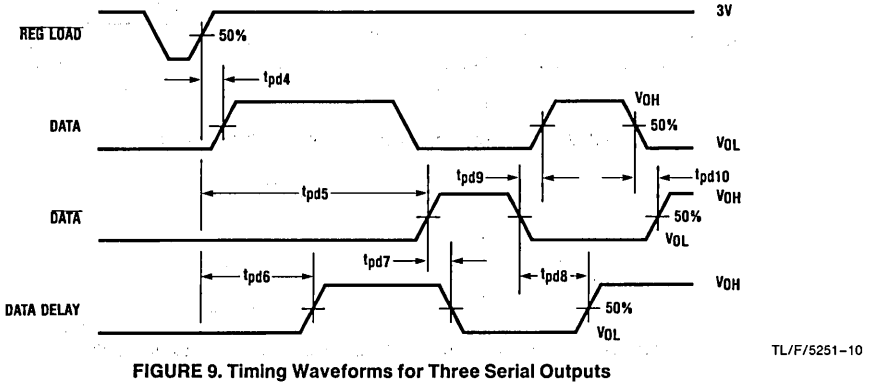
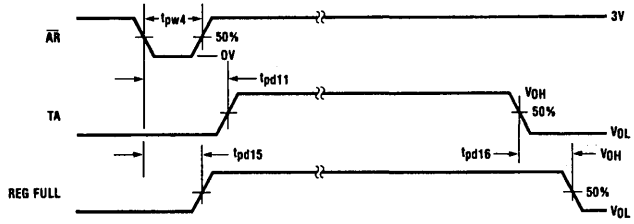


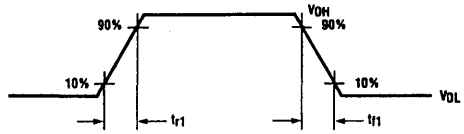
FIGURE 9. Timing Waveforms for Three Serial Outputs

Timing Waveforms (Continued)



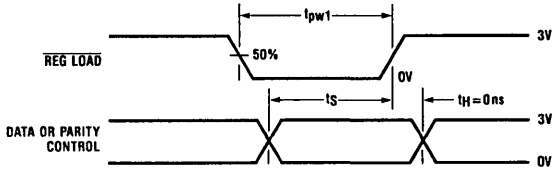
TL/F/5251–11

FIGURE 10. Timing Waveforms for $\overline{\text{Auto-Response}}$



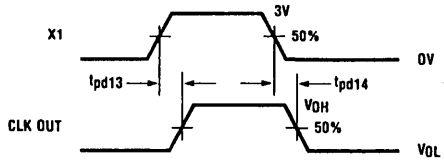
TL/F/5251–12

FIGURE 11. Output Waveform for DATA, $\overline{\text{DATA}}$, DATA DELAY (Load Circuit 2)



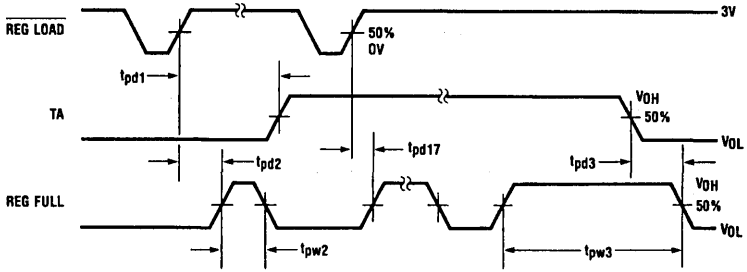
TL/F/5251–13

FIGURE 12. Register Load Waveform Requirement



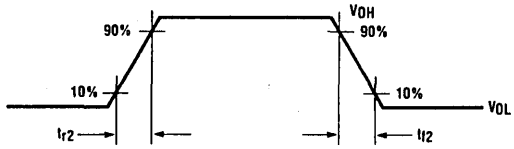
TL/F/5251–14

FIGURE 13. Timing Waveforms for Clock Pulse



TL/F/5251–15

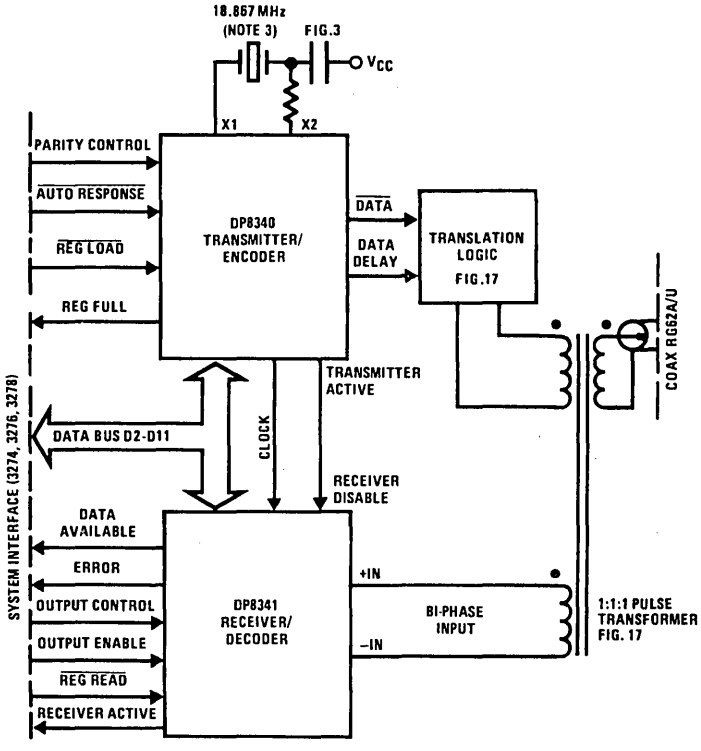
FIGURE 14. Timing Waveforms for Two Byte Transfer



TL/F/5251–16

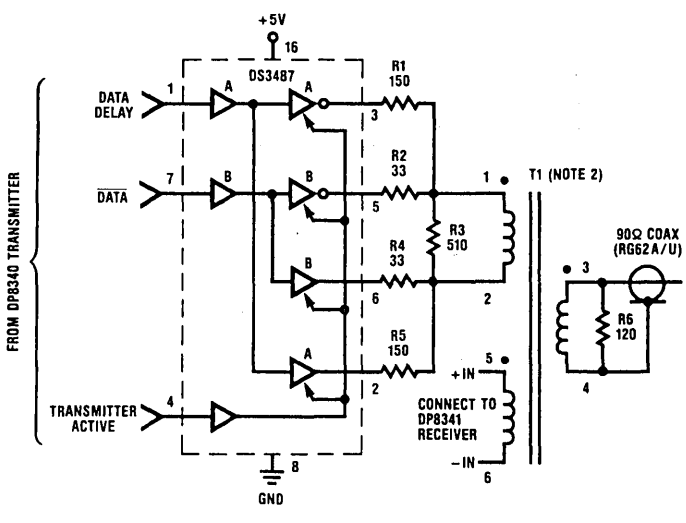
FIGURE 15. Rise and Fall Time Measurement for TA and REG Full

Typical Applications



TL/F/5251–17

FIGURE 16. Typical Applications for IBM 3270 Interface



TL/F/5251–18

Note 1: Resistance values are in Ω, ±5%, ¼ W

Note 2: T1 is a 1:1:1 pulse transformer, L_{MIN} = 500 μH for 18 MHz system clock. Pulse Engineering Part No. 5762/Surface Mount, 5762M/PE-85762. Technitrol Part No. 11LHA, Valor Electronics Part No. CT1501 or equivalent transformers.

Note 3: Crystal manufacturer's Midland Ross Corp. NEL Unit Part No. NE-18A (C2560N) @ 18.867 MHz and the Viking Group of San Jose, CA Part No. VXB46NS @ 18.867 MHz.

FIGURE 17. Translation Logic

National Semiconductor

DP8341 IBM 3270 Protocol Receiver/Decoder

General Description

The DP8341 provides complete decoding of data for high speed serial data communications. In specific, the DP8341 recognizes serial data that conforms to the IBM 3270 Information Display System Standard and converts it into ten (10) bits of parallel data. Although this standard covers biphase serial data transmission over a coax line, this device easily adapts to generalized high speed serial data transmission on other than coax lines at frequencies either higher or lower than the IBM 3270 standard.

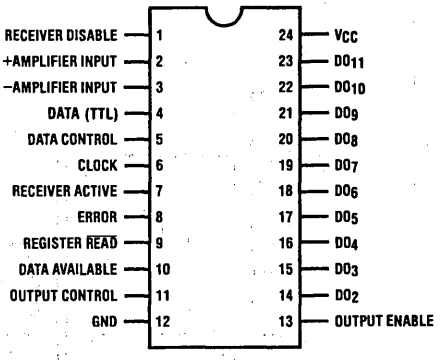
The DP8341 receiver and its complementary chip, the DP8340 transmitter, are designed to provide maximum flexibility in system designs. The separation of transmitter and receiver functions allows addition of more receivers at one end of the biphase line without the necessity of adding unused transmitters. This is advantageous specifically in control units where typically biphase data is multiplexed over many biphase lines and the number of receivers generally outnumber the number of transmitters. The separation of transmitter and receiver function provides an additional advantage in flexibility of data bus organization. The data bus outputs of the receiver are TRI-STATE®, thus enabling the bus configuration to be organized as either a common transmit/receive (bi-directional) bus or as separate transmit and receive busses for higher speed.

Features

- DP8341 receivers ten (10) bit data bytes and conforms to the IBM 3270 Interface Display System Standard
- Separate receiver and transmitter provide maximum system design flexibility
- Even parity detection
- High sensitivity input on receiver easily interfaces to coax line
- Standard TTL data input on receiver provides generalized transmission line interface and also provides hysteresis
- Data holding register
- Multi-byte or single byte transfers
- TRI-STATE receiver data outputs provide flexibility for common or separated transmit/receive data bus operation
- Data transmission error detection or receiver provides for both error detection and error type definition
- Bi-polar technology provides TTL input/output compatibility with excellent drive characteristics
- Single +5V power supply operation

Connection Diagram

Dual-In-Line Package



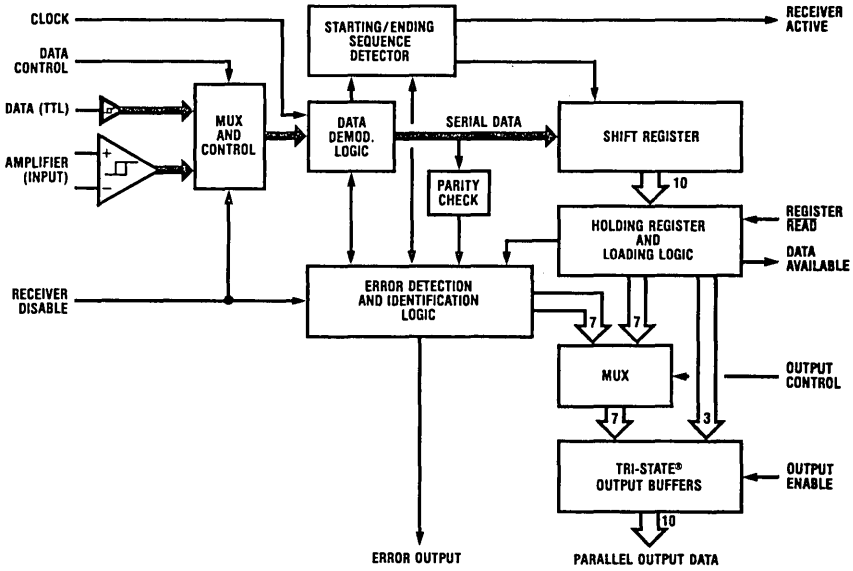
TL/F/5238-2

Top View

Order Number DP8341N
See NS Package Number N24A

FIGURE 1

Block Diagram



TL/F/5238–3

FIGURE 2. DP8341 Serial Bi-Phase Receiver/Decoder Block Diagram

Block Diagram Functional Description

Figure 2 is a block diagram of the DP8341. This chip is essentially a serial in/parallel out shift register. However, the serial input data must conform to a very specific format (see *Figures 3–5*). The message will not be recognized unless the format of the starting sequence is correct. Deviations from the format in the data, sync bit, parity or ending sequence will cause an error to be detected, terminating the message.

Data enters the receiver through the differential input amplifier or the TTL Data input. The differential amplifier is a high sensitivity input which may be used by connecting it directly to a transformer coupled coax line, or other transmission medium. The TTL Data input provides 400 mV of hysteresis and recognizes TTL logic levels. The data then enters the demodulation block.

The data demodulation block samples the data at eight (8) times the data rate and provides signals for detecting the starting sequence, ending sequence, and errors. Detection of the starting sequence sets the Receiver Active output high and enables the input shift register.

As the ten bits of data are shifted into the shift register, the receiver will verify that even parity is maintained on the data bits and the sync bit. After one complete data byte is received, the contents of the input shift register is parallel loaded to the holding register, assuming the holding register is empty, and the Data Available output is set. If the holding register is full, this load will be delayed until that register has been read. If another data byte is received when the shift register and the holding register are full a Data Overflow Error will be detected, terminating the message. Data is read from the holding register through the TRI-STATE Output Buffers. The Output Enable input is the TRI-STATE control for these outputs and the Register Read input signals the receiver that the read has been completed.

When the receiver detects an ending sequence the Receiver Active output will be reset to a logic "0" indicating the message has been terminated. A message will also terminate when an error is detected. The Receiver Active output used in conjunction with the Error output allows quick response to the transmitting unit when an error free message has been received.

The Error Detection and Identification block insures that valid data reaches the outputs of the receiver. Detection of an error sets the Error output to a logic "1" and resets the Receiver Active output to a logic "0" terminating the message. The error type may be read from the data bus outputs by setting the Output Control input to logic "0" and enabling the TRI-STATE outputs. The data bit outputs have assigned error definitions (see error code definition table). The Error output will return to a logic "0" when the next starting sequence is received, or when the error is read (Output Control to logic "0" and a Register Read performed).

The Receiver Disable input is used to disable both the amplifier and TTL Data receiver inputs. It will typically be connected directly to the Transmitter Active output of the DP8340 transmitter circuit (see *Figure 12*).

Detailed Functional Pin Description

RECEIVER DISABLE

This input is used to disable the receiver's data inputs. The Receiver Disable input will typically be connected to the Transmitter Active output of the DP8340. However, at the system controller it is necessary for both the transmitter and receiver to be active at the same time in the loop-back check condition. This variation can be accomplished with the addition of minimal external logic.

Truth Table

Receiver Disable	Data Inputs
Logic "0"	Active
Logic "1"	Disabled

AMPLIFIER INPUTS

The receiver has a differential input amplifier which may be directly connected to the transformer coupled coax line. The amplifier may also be connected to a differential type TTL line. The amplifier has 20 mV of hysteresis.

DATA INPUT

This input can be used either as an alternate data input or as a power-up check input. If the system designer prefers to use his own amplifier, instead of the one provided on the receiver, then this TTL input may be used. Using this pin as an alternate data input allows self-test of the peripheral system without disturbing the transmission line.

DATA CONTROL

This input is the control pin that selects which of the inputs are used for data entry to the receiver.

Truth Table

Data Control	Data input To
Logic "0"	Data Input
Logic "1"	Amplifier Inputs

Note: This input is also used for testing. When the input voltage is raised to 7.5V the chip resets.

CLOCK INPUT

The input is the internal clock of the receiver. It must be set at eight (8) times the line data bit rate. For the IBM 3270 Standard, this frequency is 18.87 MHz or a data bit rate of 2.358 MHz. The crystal-controlled oscillator provided in the DP8340 transmitter also operates at this frequency. The Clock Output of the transmitter is designed to directly drive the receiver's Clock Input. In addition, the receiver is designed to operate correctly to a data bit rate of 3.5 MHz.

RECEIVER ACTIVE

The purpose of this output is to inform the external system when the DP8341 is in the process of receiving a message. This output will transition to a logic "1" state after the receipt of a valid starting sequence and transition to logic "0" when a valid ending sequence is received or an error is detected. This output combined with the Error output will inform the operating system of the end of an error free data transmission.

ERROR

The Error output transitions to a logic "1" when an error is detected. Detection of an error causes the Receiver Active and the Data Available outputs to transition to a logic "0". The Error output returns to a logic "0" after the error register has been read or when the next starting sequence is detected.

REGISTER READ

The Register Read input when driven to the logic "0" state signals the receiver that data in the holding register is being read by the external operating system. The data present in the holding register will continue to remain valid until the Register Read input returns to the logic "1" condition. At this time, if an additional byte is present in the input shift register it will be transferred to the holding register, otherwise the data will remain valid in the holding register. The Data Available output will be in the logic "0" state for a short interval while a new byte is transferred to the holding register after a register read.

DATA AVAILABLE

This output indicates the existence of a data byte within the output holding register. It may also indicate the presence of a data byte in both the holding register and the input shift register. This output will transition to the logic "1" state as soon as data is available and return to the logic "0" state after each data byte has been read. However, even after the last data byte has been read and the Data Available output has assumed the logic "0" state, the last data byte read from the holding register will remain until new data has been received.

Detailed Functional Pin Description (Continued)

OUTPUT CONTROL

The Output Control input determines the type of information appearing at the data outputs. In the logic "1" state data will appear, in the logic "0" state error codes are present.

Truth Table

Output Control	Data Outputs
Logic "0"	Error Codes
Logic "1"	Data

OUTPUT ENABLE

The Output Enable input controls the state of the TRI-STATE Data outputs.

Truth Table

Output Enable	TRI-STATE Data Outputs
Logic "0"	Disabled
Logic "1"	Active

DATA OUTPUTS

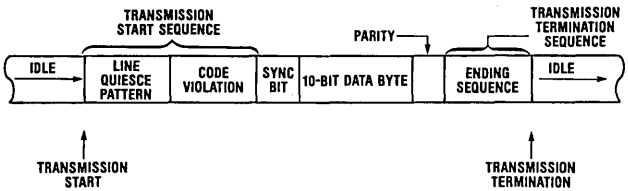
The DP8341 has a ten (10) bit TRI-STATE data bus. Seven bits are multiplexed with error bits. The error bits are defined in the table below. The Output Control input is the multiplexer control for the Data/Error bits.

Error Code Definition

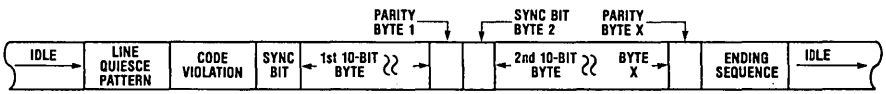
Data Bit	Error Type
DO2	Data Overflow (Byte not removed from holding register when it and the input shift register are both full and new data is received)
DO3	Parity Error (Odd parity detected)
DO4	Transmit Check conditions (existence of errors on any or all of the following data bits: DO3, DO5, and DO6
DO5	An invalid ending sequence
DO6	Loss of mid-bit transition detected at other than normal ending sequence time
DO7	New starting sequence detected before data byte in holding register has been read
DO8	Receiver disabled during receiver active mode

Message Format

Single Byte Transmission



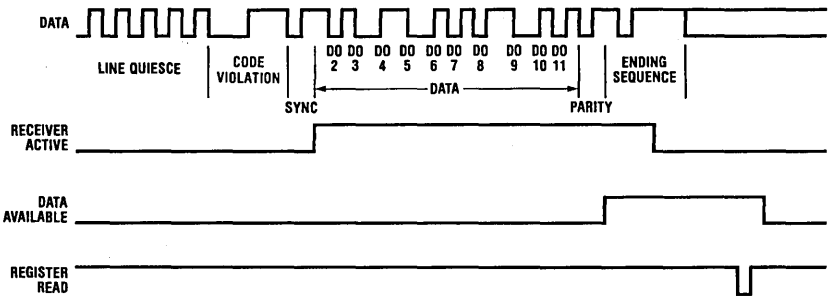
Multi-Byte Transmission



TL/F/5238–4

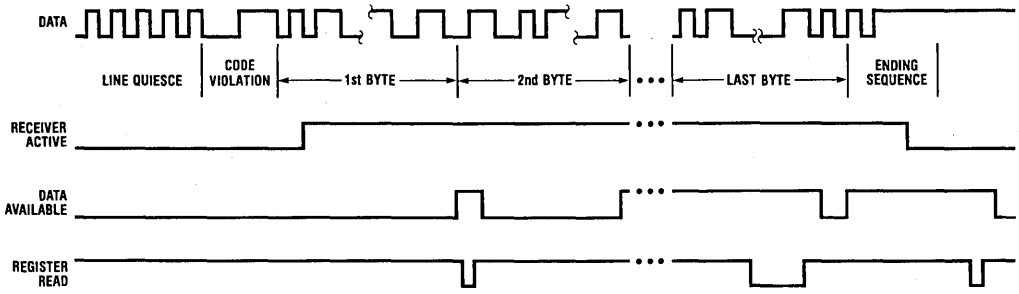
FIGURE 3. IBM 3270 Message Format

Message Format (Continued)



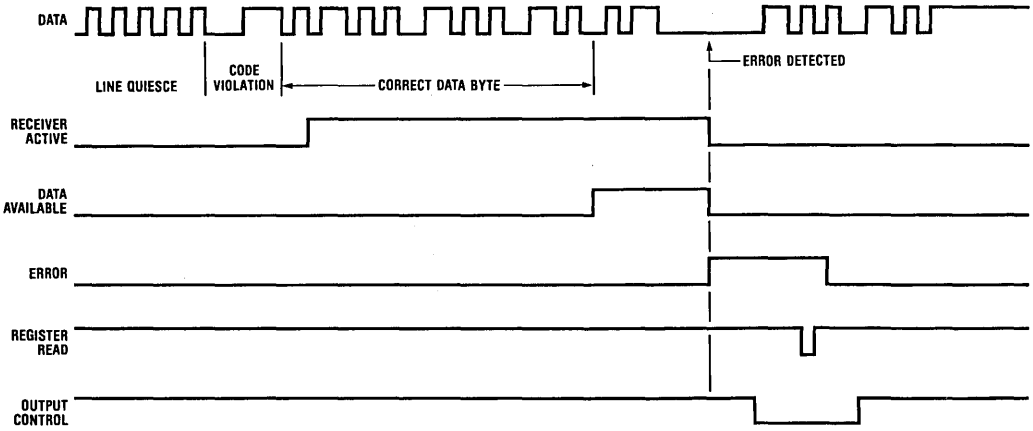
TL/F/5238-5

FIGURE 4a. Single Byte Message



TL/F/5238-6

FIGURE 4b. Multi-Byte Message



TL/F/5238-7

FIGURE 5. Message with Error

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage, V_{CC}	7V
Input Voltage	+5.5V
Output Voltage	5.25V
Storage Temperature Range	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Maximum Power Dissipation* at 25°C Dual-In-Line Package	2237 mW

*Derate Dual-In-Line package 17.9 mW/°C above 25°C.

Operating Conditions

	Min	Max	Units
Supply Voltage, (V_{CC})	4.75	5.25	V
Ambient Temperature, (T_A)	0	+70	°C

Electrical Characteristics (Notes 2, 3, and 5)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Input High Level		2.0			V
V_{IL}	Input Low Level				0.8	V
$V_{IH}-V_{IL}$	Data Input Hysteresis (TTL, Pin 4)		2.0	0.4		V
V_{CLAMP}	Input Clamp Voltage	$I_{IN} = -12$ mA		−0.8	−1.2	V
I_{IH}	Logic "1" Input Current	$V_{CC} = 5.25V$, $V_{IN} = 5.25V$		2	40	μA
I_{IL}	Logic "0" Input Current	$V_{CC} = 5.25V$, $V_{IN} = 0.5V$		−20	−250	μA
V_{OH}	Logic "1" Output Voltage	$I_{OH} = -100$ μA	3.2	3.9		V
		$I_{OH} = -1$ mA	2.5	3.2		V
V_{OL}	Logic "0" Output Voltage	$I_{OL} = 5$ mA		0.35	0.5	V
I_{OS}	Output Short Circuit Current	$V_{CC} = 5V$, $V_{OUT} = 0V$ (Note 4)	−10	−20	−100	mA
I_{OZ}	TRI-STATE Output Current	$V_{CC} = 5.25V$, $V_O = 2.5V$	−40	1	+40	μA
		$V_{CC} = 5.25V$, $V_O = 0.5V$	−40	−5	+40	μA
A_{HYS}	Amplifier Input Hysteresis		5	20	30	mV
I_{CC}	Power Supply Current	$V_{CC} = 5.25V$		160	250	mA

Timing Characteristics (Notes 2, 6, 7, and 8)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
T_{D1}	Output Data to Data Available Positive Edge		5	20	40	ns
T_{D2}	Register Read Positive Edge to Data Available Negative Edge		10	25	45	ns
T_{D3}	Error Positive Edge to Data Available Negative Edge		10	30	50	ns
T_{D4}	Error Positive Edge to Receiver Active Negative Edge		5	20	40	ns
T_{D5}	Register Read Positive Edge to Error Negative Edge		20	45	75	ns
T_{D6}	Delay from Output Control to Error Bits from Data Bits		5	20	50	ns
T_{D7}	Delay from Output Control to Data Bits from Error Bits		5	20	50	ns
T_{D8}	First Sync Bit Positive Edge to Receiver Active Positive Edge			3.5 × T +70		ns

Timing Characteristics (Notes 2, 6, 7, and 8) (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
T_{D9}	Receiver Active Positive Edge to First Data Available Positive Edge			92 × T		ns
T_{D10}	Negative Edge of Ending Sequence to Receiver Active Negative Edge			11.5 × T + 50		ns
t_{D11}	Data Control Set-Up Multiplexer Time Prior to Receiving Data through Selected Input		40	30		ns
T_{PW1}	Register Read (Data) Pulse Width		40	30		ns
T_{PW2}	Register Read (Error) Pulse Width		40	30		ns
T_{PW3}	Data Available Logic "0" State between Data Bytes		25	45		ns
T_S	Output Control Set-Up Time Prior to Register Read Negative Edge		0	−5		ns
T_H	Output Control Hold Time After the Register Read Positive Edge		0	−5		ns
T_{ZE}	Delay from Output Enable to Logic "1" or Logic "0" from High Impedance State	Load Circuit 2		25	35	ns
T_{EZ}	Delay from Output Enable to High Impedance State from Logic "1" or Logic "0"	Load Circuit 2		25	35	ns
F_{MAX}	Data Bit Frequency (Clock Input must be 8 × the Data Bit Frequency)	(Note 9)	DC		3.5	MBits/s

Note 1: "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. They are not meant to imply that the device should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

Note 2: Unless otherwise specified, min./max. limits apply across the 0°C to +70°C temperature range and the 4.75V to 5.25V power supply range. All typical values are for T_A = 25°C and V_{CC} = 5.0V.

Note 3: All currents into device pins are shown as positive; all currents out of device pins are shown as negative; all voltages are referenced to ground, unless otherwise specified. All values shown as max. or min. are so classified on absolute value basis.

Note 4: Only one output at a time should be shorted.

Note 5: Input characteristics do not apply to amplifier inputs (pins 2 and 3).

Note 6: Unless otherwise specified, all AC measurements are referenced to the 1.5V level of the input to the 1.5V level of the output and load circuit 1 is used.

Note 7: AC tests are done with input pulses supplied by generators having the following characteristics: Z_{OUT} = 50Ω and $T_r \leq$ 5 ns, $T_f \leq$ 5 ns.

Note 8: T = 1/(clock input frequency), units for "T" should be ns.

Note 9: 28 MHz clock frequency corresponds to 3.75% jitter when referenced to *Figure 10.*

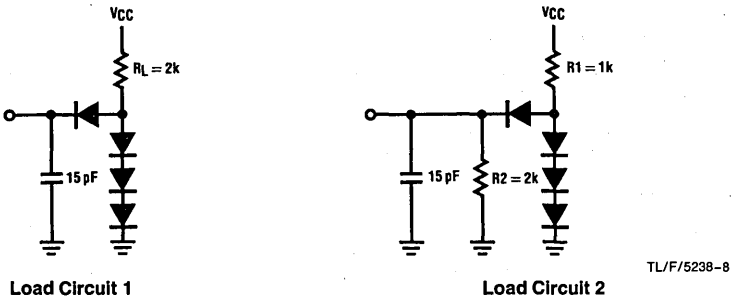
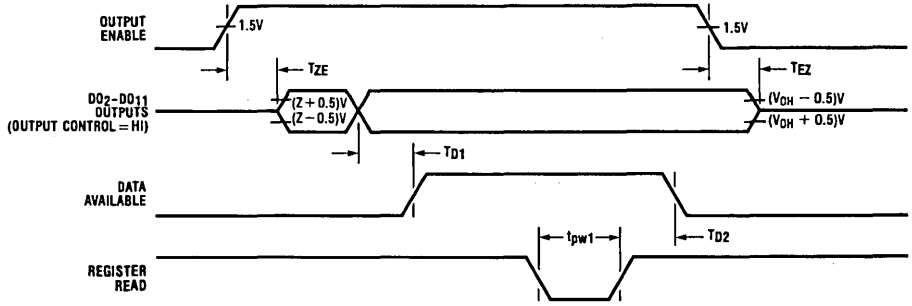


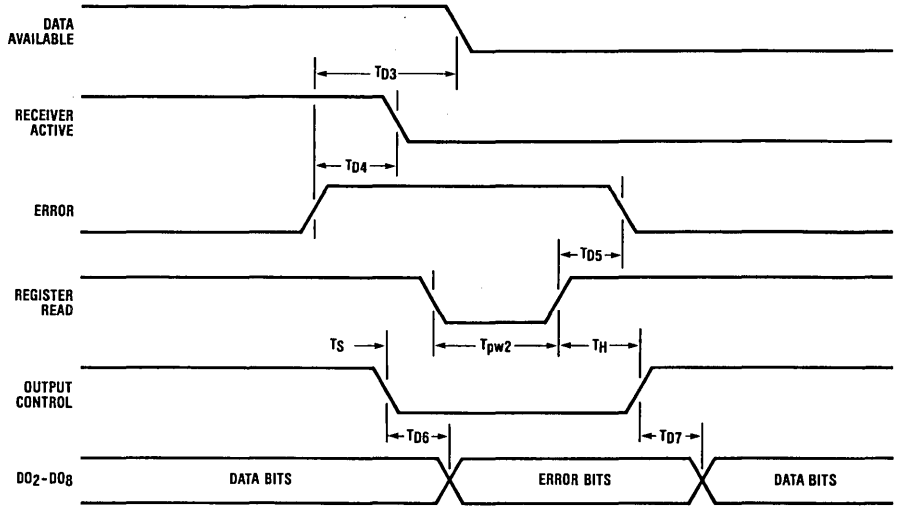
FIGURE 6. Test Load Circuits

Timing Waveforms



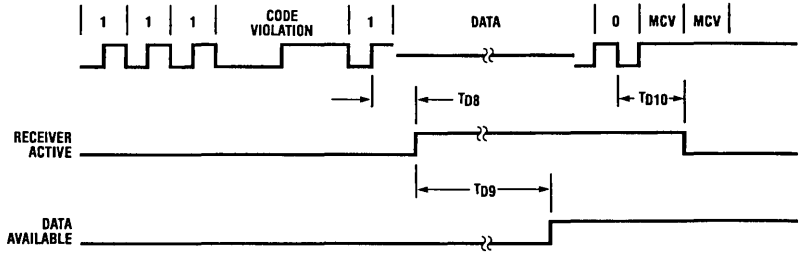
TL/F/5238-9

FIGURE 7. Data Sequence Timing



TL/F/5238-10

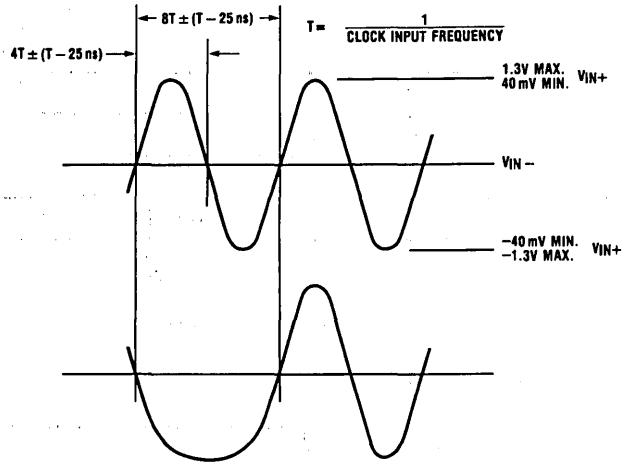
FIGURE 8. Error Sequence Timing



TL/F/5238-11

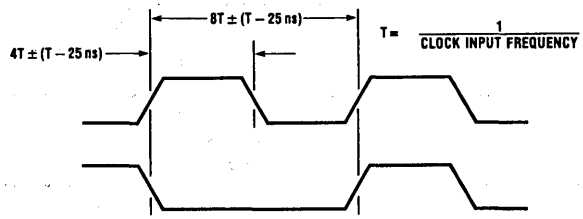
FIGURE 9. Message Timing

Timing Waveforms (Continued)



TL/F/5238-12

FIGURE 10. Data Waveform Constraints: Amplifier Inputs

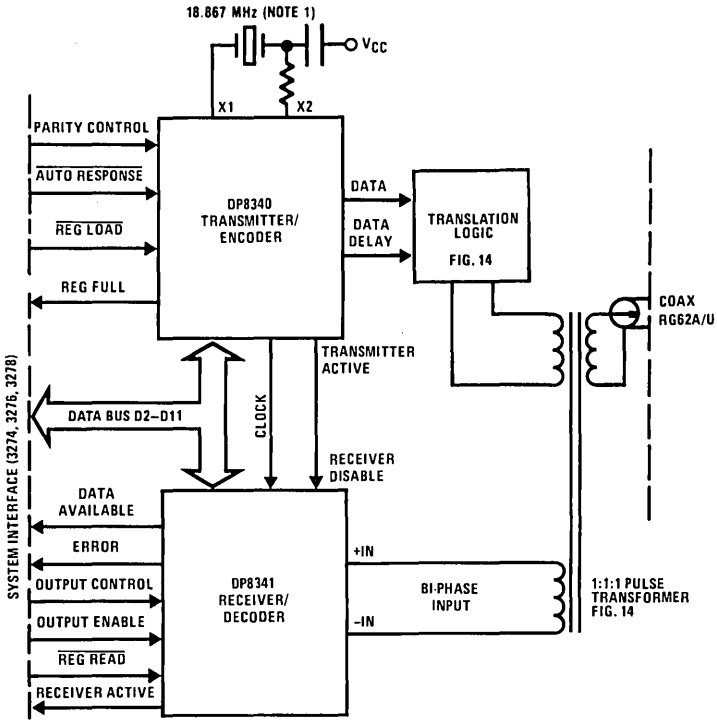


Note: $|T_r - T_f| \leq 10$ ns

TL/F/5238-13

FIGURE 11. Data Waveform Constraints: Data Input (TTL)

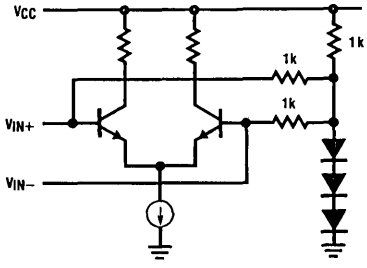
Typical Applications



TL/F/5238–14

Note 3: Crystal manufacturers: Midland Ross Corp.
NEL Unit Part No. NE18A (C2560N) @ 18.867 MHz
The Viking Group Part No. VXB-46NS @ 18,867 MHz. Located in San Jose, CA.

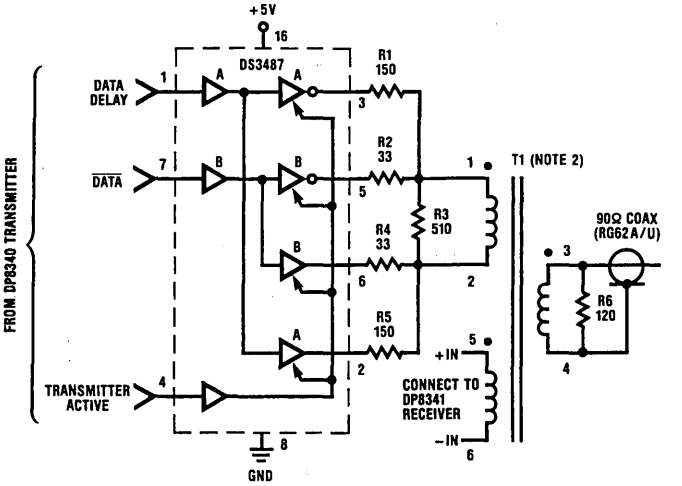
FIGURE 12. Typical Application for IBM 3270 Interface



TL/F/5238–15

FIGURE 13. Equivalent Circuit for DP8341 Input Amplifier

Typical Applications (Continued)

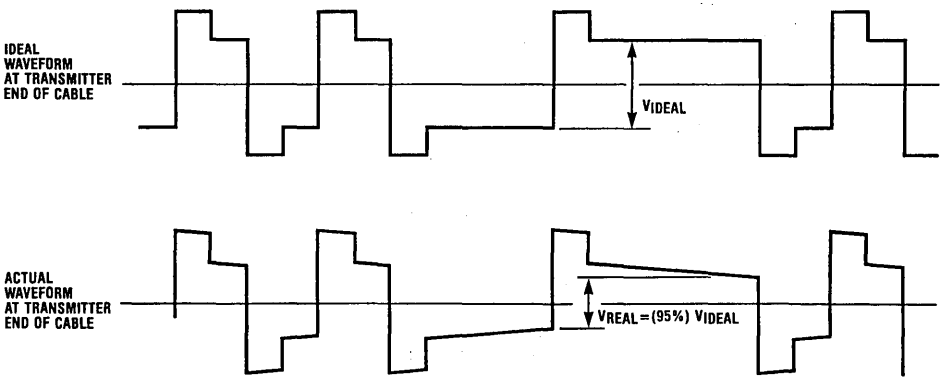


TL/F/5238–16

Note 1: Resistance values are in Ω, ±5%, ¼W

Note 2: T1 is a 1:1:1 pulse transformer, L_{MIN} = 500 μH for 18 MHz system clock
Pulse Engineering Part No. 5762/Surface Mount, 5762M/PE-85762
Valor Electronics Part No. CT1501
Technitrol Part No. 11LHA or equivalent transformers

FIGURE 14. Translation Logic



TL/F/5238–17

*To maintain loss at 95% of ideal signal, select transformer inductance such that:

$$L_{(MIN)} = \frac{10{,}000}{f_{CLK}}$$

f_{CLK} = System Clock Frequency (e.g., 18.87 MHz)

EXAMPLE:

$$L = \frac{10{,}000}{18.87 \times 10^6} \rightarrow L(MIN) = 530\mu H$$

Note 1: Less inductance will cause greater amplitude attenuation

Note 2: Greater inductance may decrease signal rise time slightly and increase ringing, but these effects are generally negligible.

FIGURE 15. Transformer Selection

National Semiconductor

DP8342 High-Speed 8-Bit Serial Transmitter/Encoder

General Description

The DP8342 generates a complete encoding of parallel data for high speed serial transmission. It generates a five bit starting sequence, three bit code violation, followed by a syn bit and eight bit per byte of data plus a parity bit. A three-bit ending code signals the termination of the transmission. The DP8342 adapts to generalized high speed serial data transmission as well as the coax lines at a maximum data rate of 3.5 MHz.

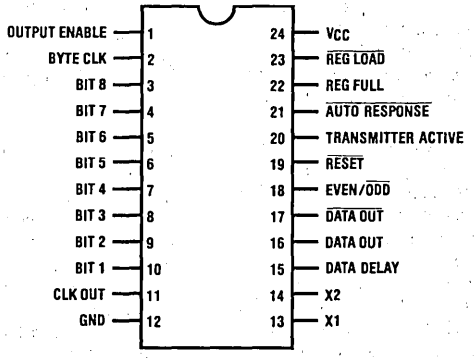
The DP8342 and its complementary chip, the DP8343 (receiver/decoder) have been designed to provide maximum flexibility in system designs. The separation of the transmitter receiver functions provides convenient addition of more receivers at one end of a biphase line without the need of unused transmitters. This is specifically advantageous in control units where typical biphase data is multiplexed over many biphase lines and the number of receivers generally exceeds the number of transmitters.

Features

- Eight bits per data byte transmission
- Single-byte or multi-byte transmission
- Internal parity generation (even or odd)
- Internal crystal controlled oscillator used for the generation of all required chip timing frequencies
- Clock output directly drives receiver (DP8343) clock input
- Input data hold register
- Automatic clear status response feature
- Line drivers at data outputs provide easy interface to bi-phase coax line or general transmission media
- <2 ns driver output skew
- Bipolar technology provides TTL input/output compatibility
- Data outputs power up/down glitch free
- Internal power up clear and reset
- Single +5V power supply

Connection Diagram

Dual-In-Line Package

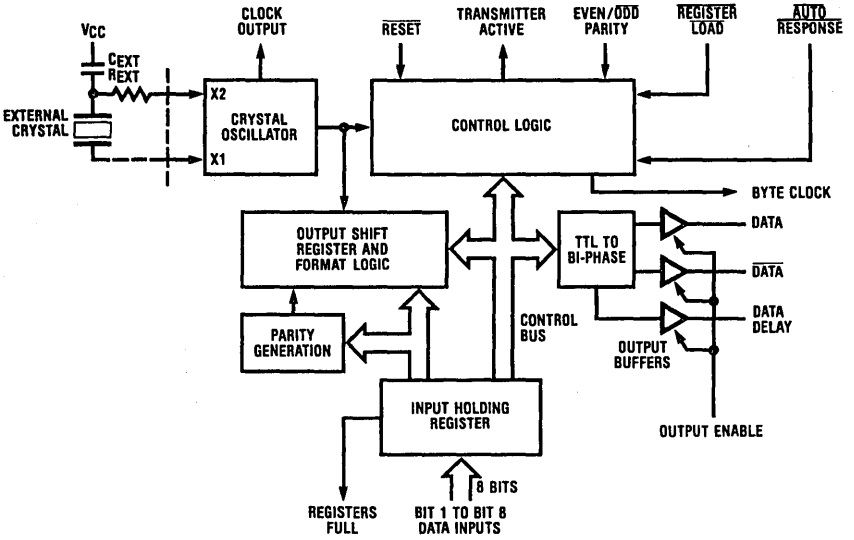


TL/F/5236–1

FIGURE 1

Order Number DP8342N
See NS Package Number N24A

Block Diagram



TL/F/5236–2

FIGURE 2

Functional Description

Figure 2 is a block diagram of the DP8342 Biphase Transmitter/Encoder. The transmitter/encoder contains a crystal oscillator whose input is a crystal with a frequency eight (8) times the data rate. A Clock Output is provided to drive the DP8342 receiver/decoder Clock Input and other system components at the oscillator frequency. Additionally, the oscillator drives the control logic and output shift register/format logic blocks.

Data is parallel loaded from the system data bus to the transmitter/encoder's input holding register. This data is in turn loaded by the transmitter/encoder to its output shift register if this register was empty at the time of the load. During this load, message formatting and parity are generated. The formatted message is then shifted out at the bit rate frequency to the TTL to Biphase block which generates the proper data bit formatting. The data outputs, DATA, $\overline{\text{DATA}}$, and DATA DELAY provide for flexible interface to the transmission medium with little or no external components.

The control Logic block interfaces to all blocks to insure proper chip operation and sequencing. It controls the type of parity generation through the Even/$\overline{\text{Odd}}$ Parity input. An additional feature provided by the transmitter/encoder is the Reset and Output-TRI-STATE® capability. Another feature of the DP8342 is the Byte Clock output which keeps track of the number of bytes transferred.

The transmitter/encoder is also capable of internal TT/AR (Transmission Turnaround/Auto Response). When the $\overline{\text{Auto-Response}}$ ($\overline{\text{AR}}$) input is forced to the logic "0" state, the transmitter/encoder responds with clean status (all zeros on data bits).

Operation of the transmitter/encoder is automatic. After the first data byte is loaded, the Transmitter Active output is set and the transmitter/encoder immediately formats the input data and serially shifts it out its data outputs. If the message is a mutli-byte message, the internal format logic will modify the message data format for multibyte as long as the next byte is loaded to the input holding format logic will modify the message data format for multibyte as long as the next byte is loaded to the input holding register before the last data bit of the previous data byte is transferred out of the internal output shift register. After all data is shifted out of the transmitter/encoder the Transmitter Active output will return to the inactive state.

Detailed Pin/Functional Description

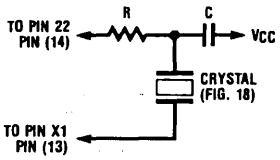
CRYSTAL INPUTS X1 AND X2

The oscillator is controlled by an external, parallel resonant crystal connected between the X1 and X2 pins. Normally, a fundamental mode crystal is used to determine the operating frequency of the oscillator; however, over-tone mode crystals may be used.

CRYSTAL SPECIFICATIONS (PARALLEL RESONANT)

Type	<20 MHz AT-cut or > 20 MHz BT-cut
Tolerance	0.005% at 25°C
Stability	0.01% from 0°C to +70°C
Resonance	Fundamental (Parallel)
Maximum Series Resistance	Dependent on Frequency (For 20 MHz, 50Ω)
Load Capacitance	15 pF

Connection Diagram



TL/F/5236-3

Freq	R	C
10 MHz–20 MHz	500Ω	30 pF
>20 MHz	120Ω	15 pF

If the DP8342 transmitter is clocked by a system clock (crystal oscillator not used), pin 13 (X1 input) should be clock directly using a Schottky series (74S) circuit. Pin 14 (X2 input) may be left open. The clocking frequency must be set at eight times the data bit rate. Maximum input frequency is 28 MHz.

CLOCK OUTPUT

The Clock Output is a buffered output derived directly from the crystal oscillator block and clocks at the oscillator frequency. It is designed to directly drive the DP8343 receiver/decoder Clock Input as well as other system components.

REGISTERS FULL

This output is used as a flag by the external operating system. A logic "1" (active state) on this output indicates that both the internal output shift register and the input holding register contain active data. No additional data should be loaded until this output returns to the logic "0" state (inactive state).

TRANSMITTER ACTIVE

This output will be in the logic "1" state while the transmitter/encoder is about to transmit or is in the process of transmitting data. Otherwise, it will assume the logic "0" state indicating no data presently in either the input holding or output shift registers.

$\overline{\text{REGISTER LOAD}}$

The Register Load input is used to load data from the Data Inputs to the input holding register. The loading function is level sensitive, the data present during the logic "0" state of this input is loaded, and the input data must be valid before the logic "0" to logic "1" transition. It is after this transition that the transmitter/encoder begins formatting of data for serial transmission.

$\overline{\text{AUTO RESPONSE}}$ (TT/AR)

This input provides for automatic clear data transmission (all bits in logic "0") without the need of loading all zero's. When a logic "0" is forced on this input the transmitter/encoder immediately responds with transmission of "clean status". When this input is in the logic "1" state the transmitter/encoder transmits data entered on the Data Inputs.

EVEN/$\overline{\text{ODD}}$ PARITY

This input sets the internal logic of the DP8342 transmitter/encoder to generate either even or odd parity for the data byte in the bit 10 position. When this pin is in the logic "0" state odd parity is generated. In the logic "1" state even parity is generated. This feature is useful when the control unit is performing a loop back check and at the same time the controller wishes to verify proper data transmission with its receiver/decoder.

SERIAL OUTPUTS—DATA, $\overline{\text{DATA}}$, AND DATA DELAY

These three output pins provide for convenient application of data to the Bi-Phase transmission line. The Data outputs are a direct bit representation of the Biphase data while the Data Delay output provides the necessary increment to clearly define the four (4) DC levels of the pulse. The DATA and $\overline{\text{DATA}}$ outputs add flexibility to the DP8342 transmitter/encoder for use in high speed differential line driving applications. The typical DATA to $\overline{\text{DATA}}$ skew is 2 ns.

$\overline{\text{RESET}}$

When a logic "0" is forced on this input, all outputs except Clock Output are latched low.

OUTPUT ENABLE

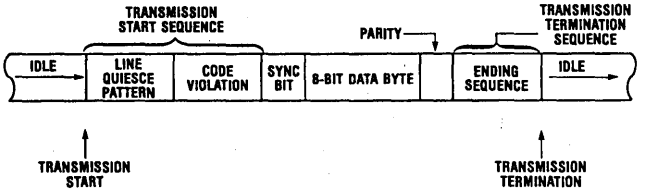
When a logic "0" is forced on this input the three serial data outputs are in the high impedence state.

BYTE CLOCK

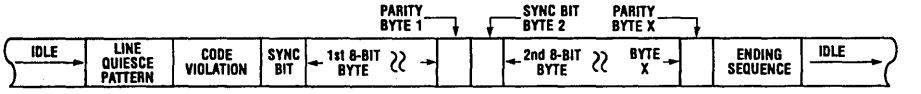
This pin registers a pulse at the end of each byte transmission. The number of pulses registered corresponds to the number of bytes transmitted.

Message Format

Single Byte Transmission



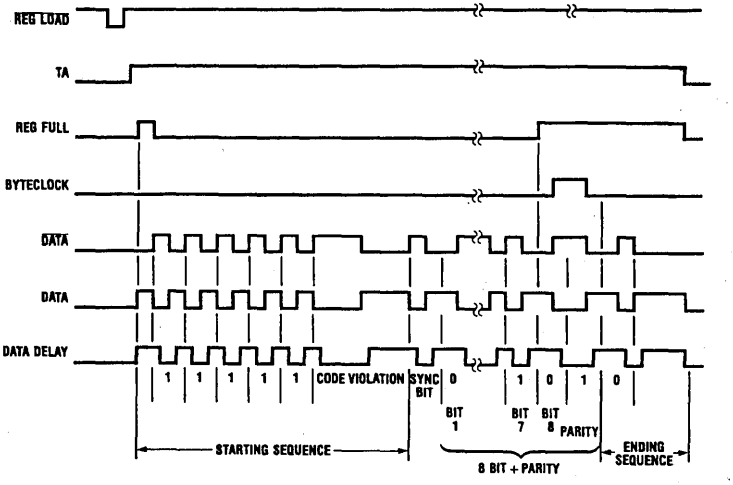
Multi-Byte Transmission



TL/F/5236-4

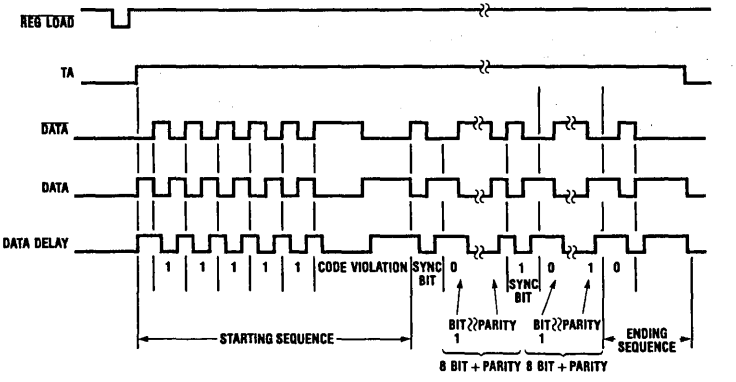
FIGURE 3

Functional Timing Waveforms



TL/F/5236-5

FIGURE 4. Overall Timing Waveforms for Single Byte



TL/F/5236-6

FIGURE 5. Overall Timing Waveforms for Multi-Byte

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage, V_{CC}	7V
Input Voltage	5.5V
Output Voltage	5.25V
Storage Temperature Range	−65°C to +150°C
Lead Temperature (Soldering, 10 sec.)	300°C

Maximum Power Dissipation* at 25°C	
Cavity Package	2237 mW
Dual-In-Line package	2500 mW

*Derate cavity package 14.9 mW/°C above 25°C; derate dual in line package 20 mW/°C above 25°C.

Operating Conditions

	Min	Max	Units
Supply Voltage, (V_{CC})	4.75	5.25	V
Ambient Temperature, T_A	0	+70	°C

Electrical Characteristics (Notes 2 and 3)

Symbol	Parameter		Conditions	Min	Typ	Max	Units
V_{IH}	Logic "1" Input Voltage (All Inputs Except X1 and X2)		V_{CC} = 5V	2.0			V
V_{IL}	Logic "0" Input Voltage (All Inputs Except X1 and X2)		V_{CC} = 5V			0.8	V
V_{CLAMP}	Input Clamp Voltage (All Inputs Except X1 and X2)		I_{IN} = −12 mA		−0.8	−1.2	V
I_{IH}	Logic "1" Input Current	Register Load Input	V_{CC} = 5.25V, V_{IN} = 5.25V		0.3	120	μA
		All Others Except X1 and X2			0.1	40	μA
I_{IL}	Logic "0" Input Current	Register Load Input	V_{CC} = 5.25V, V_{IN} = 0.5V		−15	−300	μA
		All Inputs Except X1 and X2			−5	−100	μA
V_{OH1}	Logic "1" All Outputs Except CLK OUT, DATA, $\overline{DATA}$, and DATA DELAY		I_{OH} = −100 μA, V_{CC} = 4.75V	3.2	3.9		V
			I_{OH} = −1 mA	2.5	3.4		V
V_{OH2}	Logic "1" for CLK OUT, DATA, $\overline{DATA}$, and DATA DELAY Outputs		V_{CC} = 4.75V, I_{OH} = −10 mA	2.6	3.0		V
V_{OL1}	Logic "0" All Outputs Except CLK OUT, DATA, $\overline{DATA}$, and DATA DELAY		V_{CC} = 4.75V, I_{OL} = 5 mA		0.35	0.5	V
V_{OL2}	Logic "0" for CLK OUT, DATA $\overline{DATA}$, and DATA DELAY Outputs		V_{CC} = 4.75V, I_{OL} = 20 mA		0.4	0.6	V
I_{OS1}	Output Short Circuit Current for All Except CLK OUT, DATA, $\overline{DATA}$, and DATA DELAY Outputs		(Note 5) V_{OUT} = 0V	−10	−30	−100	mA
I_{OS2}	Output Short Circuit Current DATA, $\overline{DATA}$, and DATA DELAY Outputs		(Note 5) V_{OUT} = 0V	−50	−140	−350	mA
I_{OS3}	Output Short Circuit Current for CLK OUT		(Note 5) V_{OUT} = 0V	−30	−90	−200	mA
I_{CC}	Power Supply Current		V_{CC} = 5.25V		170	250	mA

Timing Characteristics V_{CC} = 5V ±5%, T_A = 0°C to 70°C, Oscillator Frequency = 28 MHz (Notes 2 and 3)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
t_{pd1}	$\overline{REG\ LOAD}$ to Transmitter Active (TA) Positive Edge	Load Circuit 1 *Figure 6*		60	90	ns
t_{pd2}	$\overline{REG\ LOAD}$ to Register Full; Positive Edge	Load Circuit 1 *Figure 6*		45	75	ns
t_{pd3}	TA to Register Full; Negative Edge	Load Circuit 1 *Figure 6*		40	70	ns
t_{pd4}	Positive Edge of $\overline{REG\ LOAD}$ to Positive Edge of DATA	Load Circuit 2 *Figure 9*		50	80	ns
t_{pd5}	$\overline{REG\ LOAD}$ to $\overline{DATA}$; Positive Edge	Load Circuit 2 *Figure 9*		280	380	ns
t_{pd6}	$\overline{REG\ LOAD}$ to DATA DELAY; Positive Edge	Load Circuit 2 *Figure 9*		150	240	ns

Timing Characteristics (Continued)

V_{CC} = 5V ±5%, T_A = 0°C to 70°C, Oscillator Frequency = 28 MHz (Notes 2 and 3)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
t_{pd7}	Positive Edge of $\overline{DATA}$ to Negative Edge of DATA DELAY	Load Circuit 2 *Figure 9*		70	85	ns
t_{pd8}	Positive Edge of DATA DELAY to Negative Edge of $\overline{DATA}$	Load Circuit 2 *Figure 9*		80	95	ns
t_{pd9}, t_{pd10}	Skew between DATA and $\overline{DATA}$	Load Circuit 2 *Figure 9*		2	6	ns
t_{pd11}	Negative Edge of $\overline{\text{Auto Response}}$ ($\overline{AR}$) to Positive Edge of TA	Load Circuit 1 *Figure 10*		70	100	ns
t_{pd12}	Maximum Time Delay to Load Second Byte after Positive Edge of REG FULL	Load Circuit 1 *Figure 8*, (Note 7)			4 × T − 50	ns
t_{pd13}	X1 to CLK OUT; Positive Edge	Load Circuit 2 *Figure 11*		21	30	ns
t_{pd14}	X1 to CLK OUT; Negative Edge	Load Circuit 2 *Figure 11*		23	33	ns
t_{pd15}	Negative Edge of $\overline{AR}$ to Positive Edge of REG FULL	Load Circuit 1 *Figure 10*		45	75	ns
t_{pd16}	Skew between TA and REG FULL during $\overline{\text{Auto Response}}$	Load Circuit 1 *Figure 10*		50	80	ns
t_{pd17}	$\overline{\text{REG LOAD}}$ to REG FULL; Positive Edge for Second Byte	Load Circuit 1 *Figure 7*		45	75	ns
t_{pd18}	REG FULL to BYTE CLK; Negative Edge	Load Circuit 1 *Figure 7*		60	90	ns
t_{pd19}	REG FULL to BYTE CLK; Positive Edge	Load Circuit 1 *Figure 7*		145	180	ns
t_{ZH}	Output Enable to DATA, $\overline{DATA}$, or DATA DELAY outputs: HiZ to High	CL = 50 pF *Figures 16, 17*		25	45	ns
t_{ZL}	Output Enable to DATA, $\overline{DATA}$, or DATA DELAY Outputs; HiZ to High	CL = 50 pF *Figures 16, 17*		15	30	ns
t_{HZ}	Output Enable to DATA, $\overline{DATA}$, or DATA DELAY Outputs; High to HiZ	CL = 15 pF *Figures 16, 17*		65	100	ns
t_{LZ}	Output Enable to DATA, $\overline{DATA}$, or DATA DELAY Outputs; Low to HiZ	CL = 15 pF *Figures 16, 17*		45	70	ns
t_{pw1}	$\overline{\text{REG LOAD}}$ Pulse Width	*Figure 12*	40			ns
t_{pw2}	First REG FULL Pulse Width (Note 6)	Load Circuit 1 *Figure 7*, (Note 7)		8 × T + 60	8 × T + 100	ns
t_{pw3}	REG FULL Pulse Width Prior to Ending Sequence (Note 6)	Load Circuit 1 *Figure 7*		5 × B		ns
t_{pw4}	Pulse Width for $\overline{\text{Auto Response}}$	*Figure 10*	40			ns
t_{pu5}	Pulse Width for BYTE CLK	Load Circuit 1 *Figure 7*, (Note 7)		8 × T + 30	8 × T + 80	ns
t_s	Data Setup Time prior to $\overline{\text{REG LOAD}}$ Positive Edge; Hold Time = 0 ns	*Figure 12*		15	23	ns
t_{r1}	Rise Time for DATA, $\overline{DATA}$, and DATA DELAY Output Waveform	Load Circuit 2 *Figure 13*		7	13	ns
t_{f1}	Fall Time for DATA, $\overline{DATA}$, and DATA DELAY Output Waveform	Load Circuit 2 *Figure 13*		5	11	ns
t_{r2}	Rise Time for TA and REG FULL	Load Circuit 1 *Figure 14*		20	30	ns
t_{f2}	Fall Time for TA and REG FULL	Load Circuit 1 *Figure 14*		15	25	ns

Timing Characteristics (Continued)

V_{CC} = 5V ±5%, T_A = 0°C to 70°C, Oscillator Frequency = 28 MHz (Notes 2 and 3)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
f_{MAX}	Data Rate Frequency (Clock Input must be 8× this Frequency)		DC		3.5	Mbits/s
C_{IN}	Input Capacitance—Any Input	(Note 4)		5	15	pF

Note 1: "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. They are not meant to imply that the device should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

Note 2: Unless otherwise specified, min/max limits apply across the 0°C to +70°C temperature range and the 4.75V to 5.25V power supply range. All typical values are for T_A = 25°C and V_{CC} = 5.0V.

Note 3: All currents into device pins are shown as positive; all currents out of device pins are shown as negative; all voltages are referenced to ground, unless otherwise specified. All values shown as max or min are so classified on absolute basis.

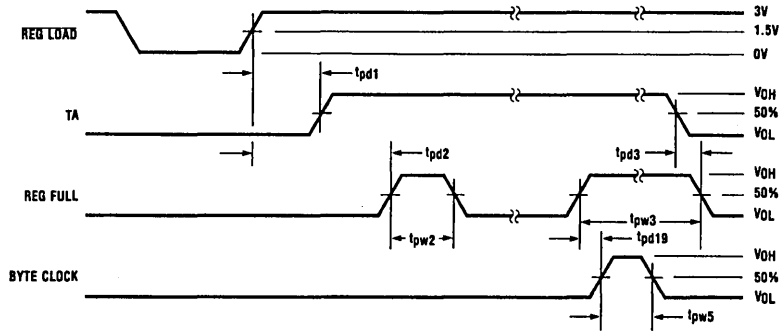
Note 4: Input capacitance is guaranteed by periodic testing. f_{TEST} = 10 kHz at 300 mV, T_A = 25°C.

Note 5: Only one output should be shorted at a time.

Note 6: T = 1/(Oscillator Frequency). Unit for T should be in ns. B = 8T.

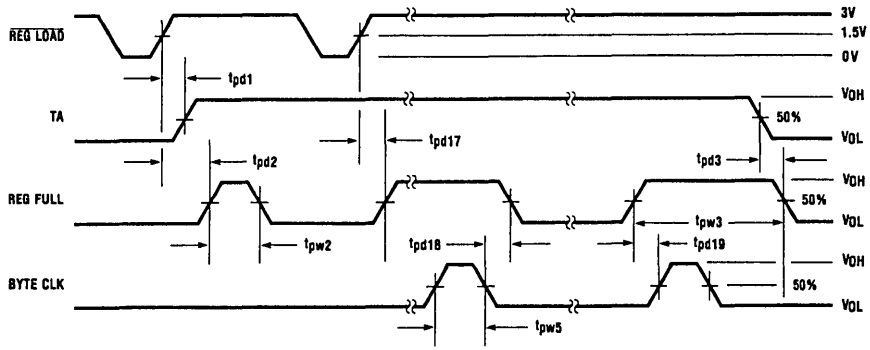
Note 7: Oscillator Frequency Dependent.

Timing Waveforms (Continued)



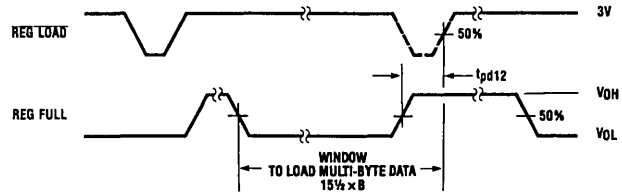
TL/F/5236–7

FIGURE 6. Single Byte Transfer



TL/F/5236–8

FIGURE 7. Two-Byte Transfer



TL/F/5236–9

FIGURE 8. Maximum Window to Load Multi-Byte Data

Functional Timing Waveforms (Continued)

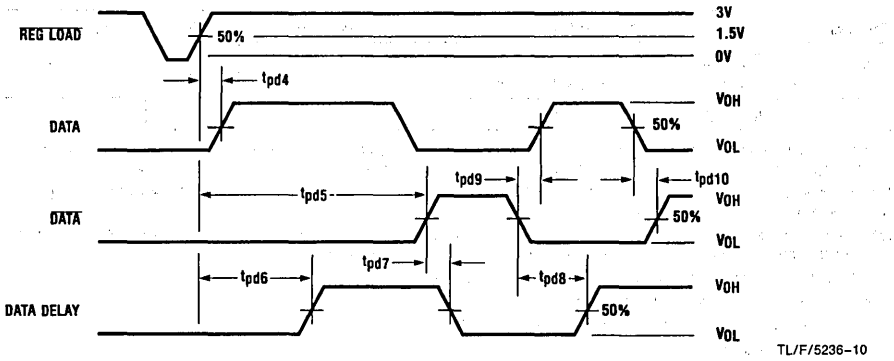


FIGURE 9. Three Serial Outputs

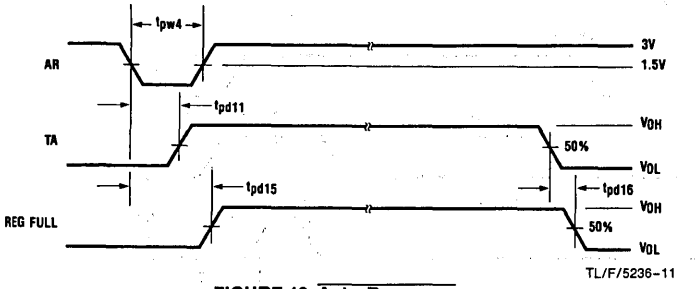


FIGURE 10. $\overline{\text{Auto-Response}}$

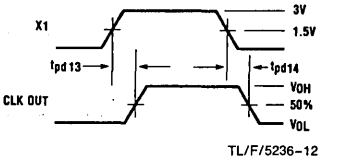


FIGURE 11. Clock Pulse

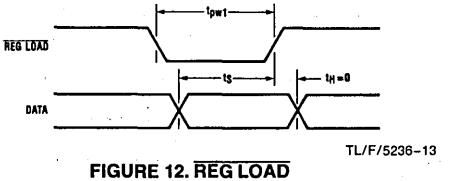


FIGURE 12. $\overline{\text{REG LOAD}}$

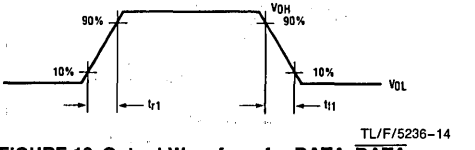


FIGURE 13. Output Waveform for DATA, $\overline{\text{DATA}}$, DATA DELAY (Load Circuit 2)

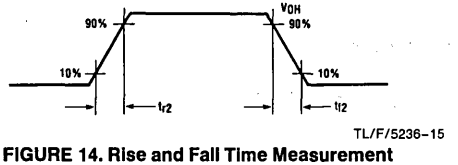
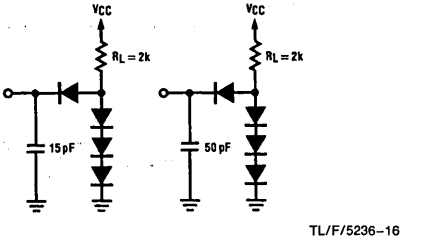


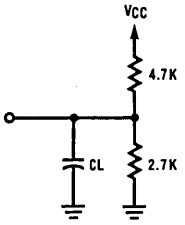
FIGURE 14. Rise and Fall Time Measurement for TA and REG FULL



Load Circuit 1 Load Circuit 2

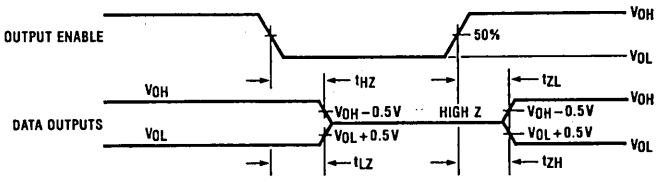
FIGURE 15. Test Load Circuits

Timing Waveforms (Continued)



TL/F/5236–17

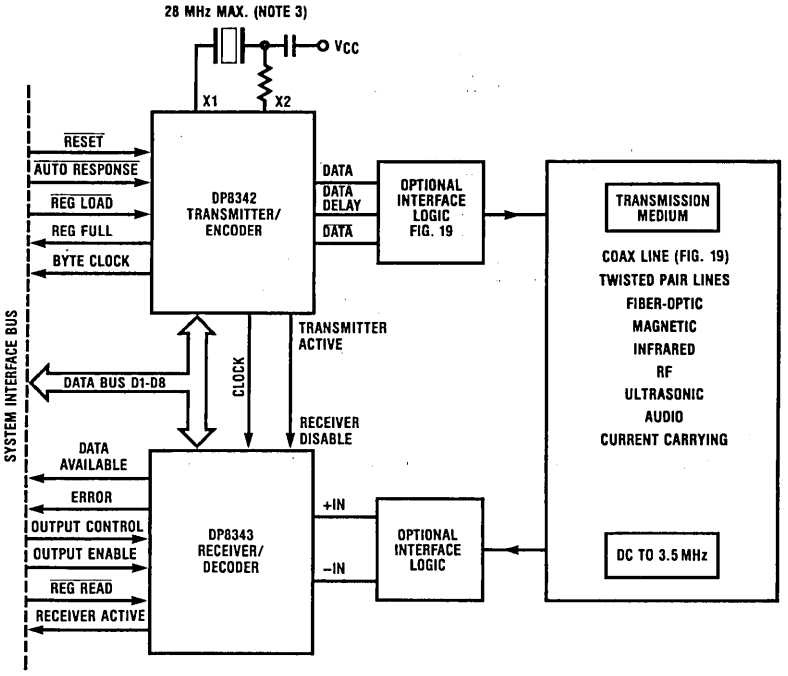
FIGURE 16. Load Circuit for Output TRI-STATE Test



TL/F/5236–18

FIGURE 17. TRI-STATE Test

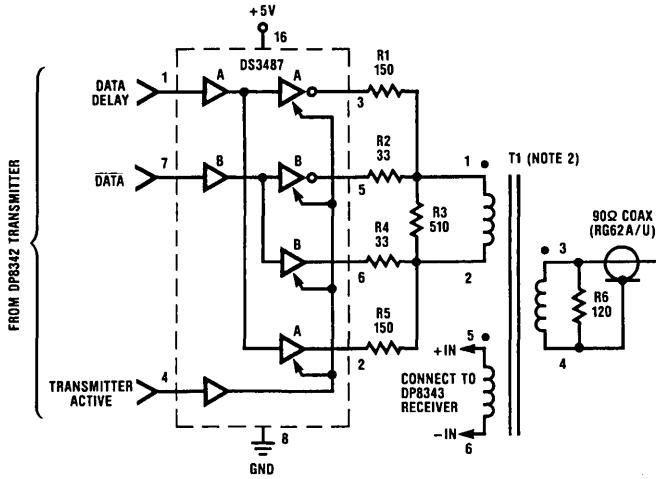
Typical Applications



TL/F/5236–19

FIGURE 18

Typical Applications (Continued)

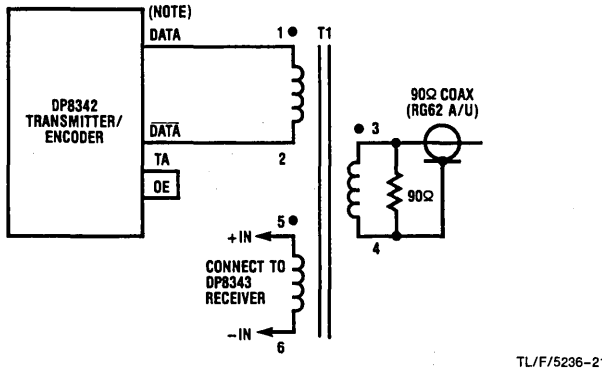


Note 1: Resistance values are in Ω, ±5%, ¼W.

Note 2: T1 is a 1:1:1 pulse transformer, L = 500 μH for 18 MHz to 28 MHz system clock. Pulse Engineering Part No. 5762; Technitrol Part No. 11LHA, Valor Electronics Part No. CT1501, or equivalent transformer.

Note 3: Crystal manufacturer Midland Ross Corp. NEL Unit Part No. NE-18A at 28 MHz.

FIGURE 19. Interface Logic for a Coax Transmission Line



Note: Data rates up to 3.5 Mbits/s at 5000′ still apply.

FIGURE 20. Direct Interface for a Coax Transmission Line (Non-IBM Voltage Levels)

National Semiconductor

DP8343 High-Speed 8-Bit Serial Receiver/Decoder

General Description

The DP8343 provides complete decoding of data for high speed serial data communications. In specific, the DP8343 receiver recognizes biphase serial data sent from its complementary chip, the DP8342 transmitter, and converts it into 8 bits of parallel data. These devices are easily adapted to generalized high speed serial data transmission systems that operate at bit rates up to 3.5 MHz.

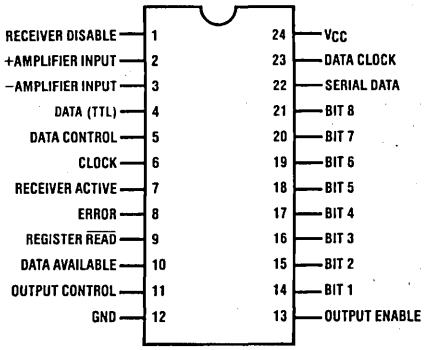
The DP8343 receiver and the DP8342 transmitter are designed to provide maximum flexibility in system designs. The separation of transmitter and receiver functions allows addition of more receivers at one end of the biphase line without the necessity of adding unused transmitters. This is advantageous in control units where the data is typically multiplexed over many lines and the number of receivers generally exceeds the number of transmitters. The separation of transmitter and receiver function provides an additional advantage in flexibility of data bus organization. The data bus outputs of the receiver are TRI-STATE®, thus enabling the bus configuration to be organized as either a common transmit/receive (bi-directional) bus or as separate transmit and receive busses for higher speed.

Features

- DP8343 receives 8-bit data bytes
- Separate receiver and transmitter provide maximum system design flexibility
- Even parity detection
- High sensitivity input on receiver easily interfaces to coax line
- Standard TTL data input on receiver provides generalized transmission line interface and also provides hysteresis
- Data holding register
- Multi-byte or single byte transfers
- TRI-STATE receiver date outputs provide flexibility for common or separated transmit/receive data bus operation
- Data transmission error detection on receiver provides for both error detection and error type definition
- Bipolar technology provides TTL input/output compatibility with excellent drive characteristics
- Single +5V power supply operation

Connection Diagram

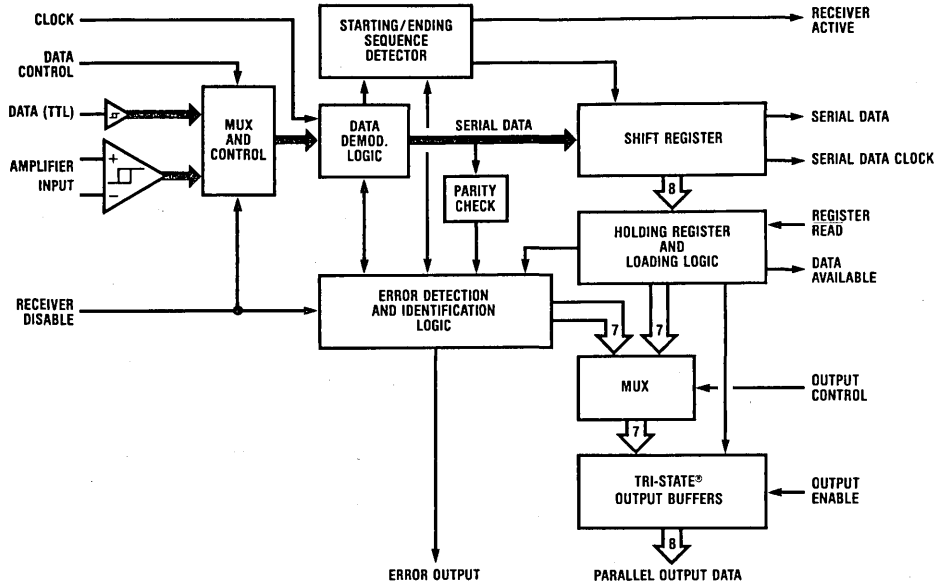
Dual-In-Line Package



TL/F/5237–1

FIGURE 1
Order Number DP8343N
See NS Package Number N24A

Block Diagram



TL/F/5237–2

FIGURE 2. DP8343 Biphase Receiver

Functional Description

Figure 2 is a block diagram of the DP8343 receiver. This chip is essentially a serial in/parallel out shift register. However, the serial input data must conform to a very specific format (see *Figures 3–6*). The message will not be recognized unless the format of the starting sequence is correct. Deviations from the format in the data, sync bit, parity or ending sequence will cause an error to be detected, terminating the message.

Data enters the receiver through the differential input amplifier or the TTL Data input. The differential amplifier is a high sensitivity input which may be used by connecting it directly to a transformer coupled coax line, or other transmission medium. The TTL Data input provides 400 mV of hysteresis and recognizes TTL logic levels. The data then enters the demodulation block.

The data demodulation block samples the data at eight (8) times the data rate and provides signals for detecting the starting sequence, ending sequence, and errors. Detection of the starting sequence sets the Receiver Active output high and enables the input shift register.

As the eight bits of data are shifted into the shift register, the receiver will verify that even parity is maintained on the data bits and the sync bit. Serial Data and Serial Data Clock, the inputs to the shift register, are provided for use with external error detecting schemes. After one complete data byte is received, the contents of the input shift register is parallel loaded to the holding register, assuming the holding register is empty, and the Data Available output is set. If the holding register is full, this load will be delayed until that register has been read or the start of another data byte is received, in which case a Data Overflow Error will be detected, terminating the message. Data is read from the holding register through the TRI-STATE Output Buffers. The Output Enable input is the TRI-STATE control for these outputs and the Register Read input signals the receiver that the read has been completed.

When the receiver detects an ending sequence the Receiver Active output will be reset to a logic "0" indicating the message has been terminated. A message will also terminate when an error is detected. The Receiver Active output used in conjunction with the Error output allows quick response to the transmitting unit when an error free message has been received.

The Error Detection and Identification block insures that valid data reaches the outputs of the receiver. Detection of an error sets the Error output to a logic "1" and resets the Receiver Active output to a logic "0" terminating the message. The error type may be read from the data bus outputs by setting the Output Control input to logic "0" and enabling the TRI-STATE outputs. The data bit outputs have assigned error definitions (see error code definition table). The Error output will return to a logic "0" when the next starting sequence is received, or when the error is read (Output Control to logic "0" and a Register Read performed).

The Receiver Disable input is used to disable both the amplifier and TTL Data receiver inputs. It will typically be connected directly to the Transmitter Active output of the DP8342 transmitter circuit.

Detailed Functional Pin Description

RECEIVER DISABLE

This input is used to disable the receiver's data inputs. The Receiver Disable input will typically be connected to the Transmitter Active output of the DP8342. However, at the system controller it may be necessary for both the transmitter and receiver to be active at the same time. This variation can be accomplished with the addition of minimal external logic.

Truth Table

Receiver Disable	Data Inputs
Logic "0"	Active
Logic "1"	Disabled

AMPLIFIER INPUTS

The receiver has a differential input amplifier which may be directly connected to the transformer coupled coax line. The amplifier may also be connected to a differential type TTL line. The amplifier has 20 mV of hysteresis.

DATA INPUT

This input can be used either as an alternate data input or as a power-up check input. If the system designer prefers to use his own amplifier, instead of the one provided on the receiver, then this TTL input may be used. Using this pin as an alternate data input allows self-test of the peripheral system without disturbing the transmission line.

DATA CONTROL

This input is the control pin that selects which of the inputs are used for data entry to the receiver.

Truth Table

Data Control	Data Input To
Logic "0"	Data Input
Logic "1"	Amplifier Inputs

Note: This input is also used for testing. When the input voltage is raised to 7.5V the chip resets.

CLOCK INPUT

This input is the internal clock of the receiver. It must be set at eight (8) times the line data bit rate. The crystal-controlled oscillator provided in the DP8342 transmitter also operates at this frequency. The Clock Output of the transmitter is designed to directly drive the receiver's Clock Input. In addition, the receiver is designed to operate correctly to a data bit rate of 3.5 MHz.

RECEIVER ACTIVE

The purpose of this output is to inform the external system when the DP8343 is in the process of receiving a message. This output will transition to a logic "1" state after a receipt of a valid starting sequence and transition to logic "0" when a valid ending sequence is received or an error is detected. This output combined with the Error output will inform the operating system of the end of an error free data transmission.

ERROR

The Error output transitions to a logic "1" when an error is detected. Detection of an error causes the Receiver Active and the Data Available outputs to transition to a logic "0". The Error output returns to a logic "0" after the error register has been read or when the next starting sequence is detected.

REGISTER READ

The Register Read input when driven to the logic "0" state signals the receiver that data in the holding register is being read by the external operating system. The data present in the holding register will continue to remain valid until the Register Read input returns to the logic "1" condition. At this time, if an additional byte is present in the input shift register it will be transferred to the holding register, otherwise the data will remain valid in the holding register. The Data Available output will be in the logic "0" state for a short interval while a new byte is transferred to the holding register after a register read.

DATA AVAILABLE

This output indicates the existence of a data byte within the output holding register. It may also indicate the presence of a data byte in both the holding register and the input shift register. This output will transition to the logic "1" state as soon as data is available and return to the logic "0" state after each data byte has been read. However, even after the last data byte has been read and the Data Available output has assumed the logic "0" state, the last data byte read from the holding register will remain until new data has been received.

OUTPUT CONTROL

The Output Control input determines the type of information appearing at the data outputs. In the logic "1" state data will appear, in the logic "0" state error codes are present.

Truth Table

Output Control	Data Outputs
Logic "0"	Error Codes
Logic "1"	Data

OUTPUT ENABLE

The Output Enable input controls the state of the TRI-STATE Data outputs.

Truth Table

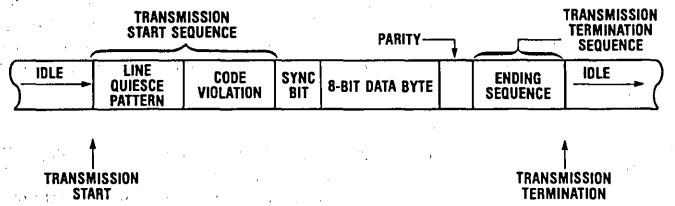
Output Enable	TRI-STATE Data Outputs
Logic "0"	Disabled
Logic "1"	Active

DATA OUTPUTS

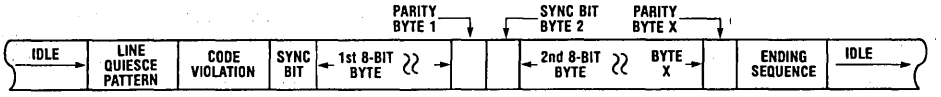
The DP8343 has an 8-bit TRI-STATE data bus. Seven bits are multiplexed with error bits. The error bits are defined in the following table. The Output Control input is the multiplexer control for the Data/Error bits.

Message Format

Single Byte Transmission

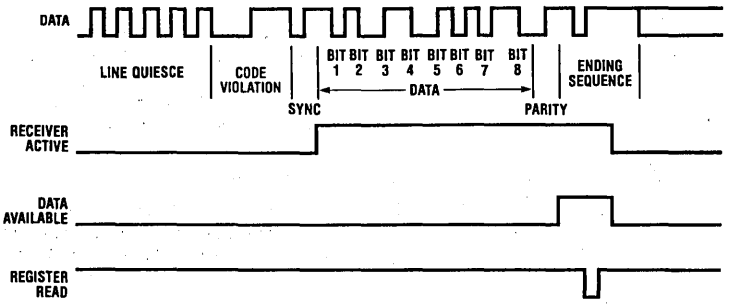


Multi-Byte Transmission



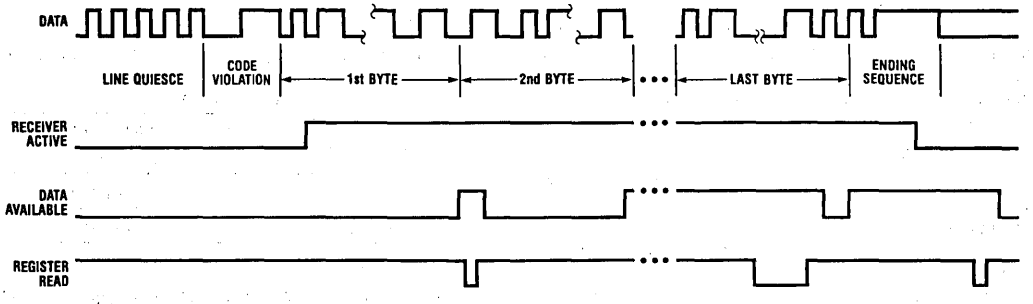
TL/F/5237-3

FIGURE 3



TL/F/5237-4

FIGURE 4a. Single Byte (8-Bit) Message



TL/F/5237-5

FIGURE 4b. Multi-Byte Message

Message Format (Continued)

Error Code Definition

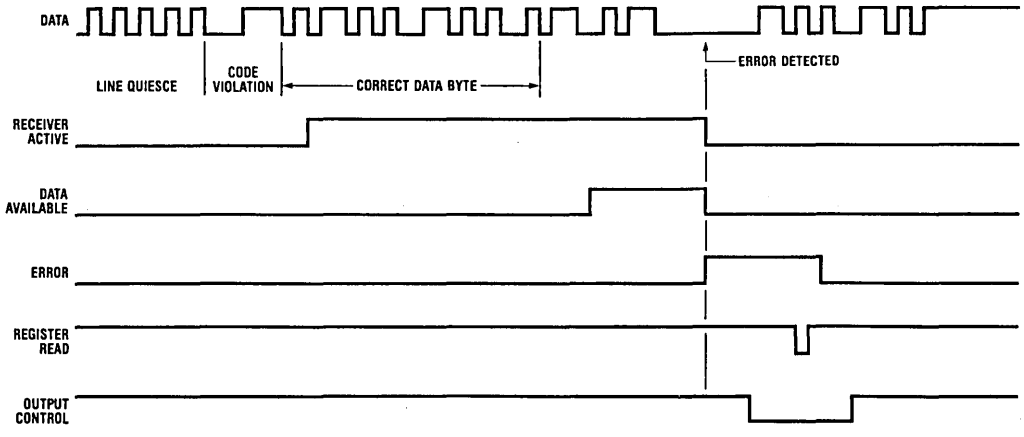
Data Bit DP8343	Error Type
Bit 1	Data Overflow (Byte not removed from holding register when it and the input shift register are both full and new data is received)
Bit 2	Parity Error (Odd parity detected)
Bit 3	Transmit Check conditions (existence of errors on any or all of the following data bits: Bit 2, Bit 4, and Bit 5)
Bit 4	An invalid ending sequence
Bit 5	Loss of mid-bit transition detected at other than normal ending sequence time
Bit 6	New starting sequence detected before data byte in holding register has been read
Bit 7	Receiver disabled during receiver active mode

SERIAL DATA

The Serial Data output is the serial data coming into the input shift register.

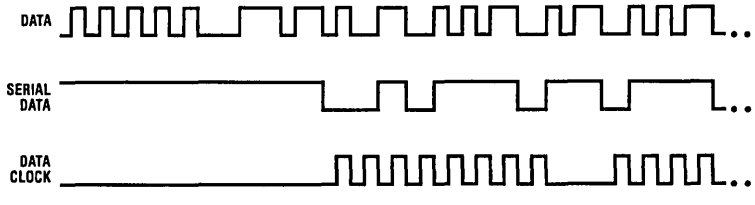
DATA CLOCK

The Data Clock output is the clock to the input shift register.



TL/F/5237–6

FIGURE 5. Message with Error



TL/F/5237–7

FIGURE 6. Data Clock and Serial Data

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage, (V_{CC})	7.0V
Input Voltage	5.5V
Output Voltage	5.25V
Storage Temperature Range	−65°C to +150°C
Lead Temperature (Soldering, 10 sec.)	300°C

Operating Conditions

	Min	Max	Units
Supply Voltage, (V_{CC})	4.75	5.25	V
Ambient Temperature, T_A	0	+70	°C

Electrical Characteristics (Notes 2, 3 and 5)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Input High Level		2.0			V
V_{IL}	Input Low Level				0.8	V
V_{IH}-V_{IL}	Data Input Hysteresis (TTL, Pin 4)		0.2	0.4		V
V_{CLAMP}	Input Clamp Voltage	I_{IN} = −12 mA		−0.8	−1.2	V
I_{IH}	Logic "1" Input Current	V_{CC} = 5.25V, V_{IN} = 5.25V		2	40	μA
I_{IL}	Logic "0" Input Current	V_{CC} = 5.25V, V_{IN} = 0.5V		−20	−250	μA
V_{OH}	Logic "1" Output Voltage	I_{OH} = −100 μA	3.2	3.9		V
		I_{OH} = −1 mA	2.5	3.2		V
V_{OL}	Logic "0" Output Voltage	I_{OL} = 5 mA		0.35	0.5	V
I_{OS}	Output Short Circuit Current	V_{CC} = 5V, V_{OUT} = 0V (Note 4)	−10	−20	−100	mA
I_{OZ}	TRI-STATE Output Current	V_{CC} = 5.25V, V_O = 2.5V	−40	1	+40	μA
		V_{CC} = 5.25V, V_O = 0.5V	−40	−5	+40	μA
A_{HYS}	Amplifier Input Hysteresis		5	20	30	mV
I_{CC}	Power Supply Current	V_{CC} = 5.25V		160	250	mA

Timing Characteristics (Notes 2, 6, 7, and 8)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
T_{D1}	Output Data to Data Available Positive Edge		5	20	40	ns
T_{D2}	Register Read Positive Edge to Data Available Negative Edge		10	25	45	ns
T_{D3}	Error Positive Edge to Data Available Negative Edge		10	30	50	ns
T_{D4}	Error Positive Edge to Receiver Active Negative Edge		5	20	40	ns
T_{D5}	Register Read Positive Edge to Error Negative Edge		20	45	75	ns
T_{D6}	Delay from Output Control to Error Bits from Data Bits		5	20	50	ns
T_{D7}	Delay from Output Control to Data Bits from Error Bits		5	20	50	ns
T_{D8}	First Sync Bit Positive Edge to Receiver Active Positive Edge			3.5 × T +70		ns
T_{D9}	Receiver Active Positive Edge to First Data Available Positive Edge			76 × T		ns
T_{D10}	Negative Edge of Ending Sequence to Receiver Active Negative Edge			11.5 × T +50		ns
T_{D11}	Data Control Set-up Multiplexer Time Prior to Receiving Data through Selected Input		40	30		ns
T_{D12}	Serial Data Set-Up Prior to Data Clock Positive Edge			3 × T		ns

Timing Characteristics (Notes 2, 6, 7, and 8) (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
T_{PW1}	Register Read (Data) Pulse Width		30	40		ns
T_{PW2}	Register Read (Error) Pulse Width		40	30		ns
T_{PW3}	Data Available Logic "0" State between Data Bytes		25	45		ns
T_S	Output Control Set-Up Time Prior to Register Read Negative Edge		0	−5		ns
T_H	Output Control Hold Time after the Register Read Positive Edge		0	−5		ns
T_{ZE}	Delay from Output Enable to Logic "1" or Logic "0" from High Impedance State	Load Circuit 2		25	35	ns
T_{EZ}	Delay from Output Enable to High Impedance State from Logic "1" or Logic "0"	Load Circuit 2		25	35	ns
F_{MAX}	Data Bit Frequency (Clock Input must be 8 × the Data Bit Frequency)		DC		3.5	MBits/s

Note 1: "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. They are not meant to imply that the device should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

Note 2: Unless otherwise specified, min./max. limits apply across the 0°C to +70°C temperature range and the 4.75V to 5.25V power supply range. All typical values are for T_A = 25°C and V_{CC} = 5.0V.

Note 3: All currents into device pins are shown as positive; all currents out of device pins are shown as negative; all voltages are referenced to ground, unless otherwise specified. All values shown as max. or min. are so classified on absolute value basis.

Note 4: Only one output at a time should be shorted.

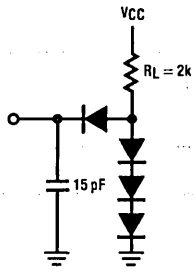
Note 5: Input characteristics do not apply to amplifier inputs (pins 2 & 3).

Note 6: Unless otherwise specified, all AC measurements are referenced to the 1.5V level of the input to the 1.5V level of the output and load circuit 1 is used.

Note 7: AC tests are done with input pulses supplied by generators having the following characteristics: Z_{OUT} = 5Ω, $T_r \leq$ 5 ns, and $T_f \leq$ 5 ns.

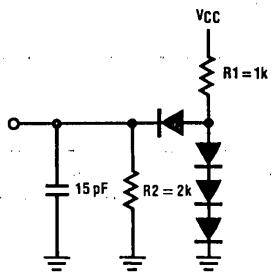
Note 8: T = 1/(clock input frequency), units for "T" should be ns.

Test Load Circuits



TL/F/5237–8

Load Circuit 1

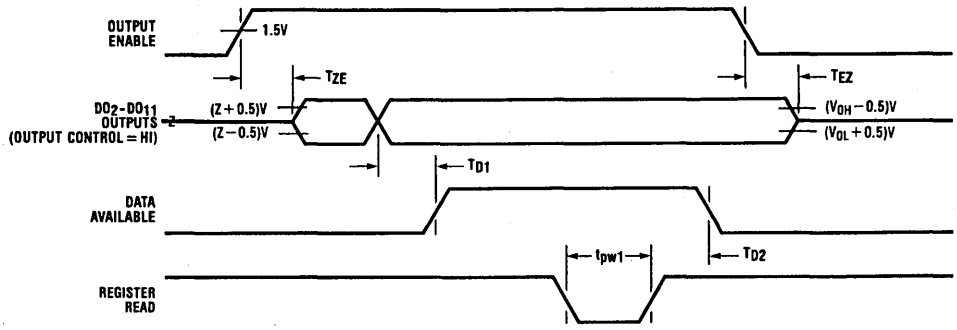


TL/F/5237–9

Load Circuit 2

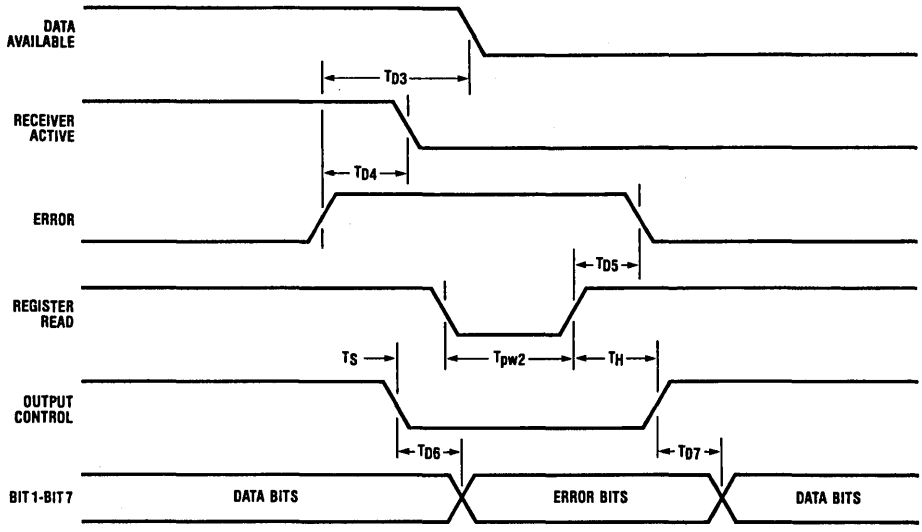
FIGURE 7

Timing Waveforms



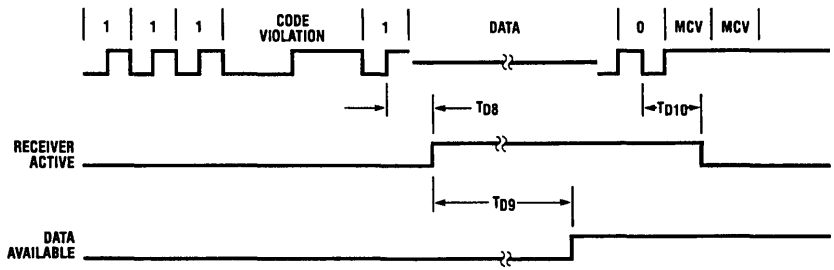
TL/F/5237-10

FIGURE 8. Data Sequence Timing



TL/F/5237-11

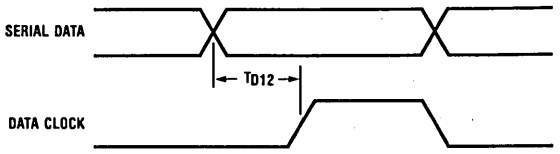
FIGURE 9. Error Sequence Timing



TL/F/5237-12

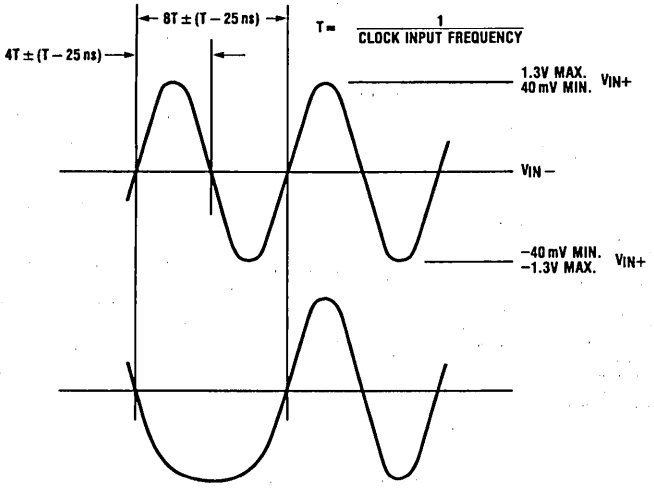
FIGURE 10. Message Timing

Timing Waveforms (Continued)



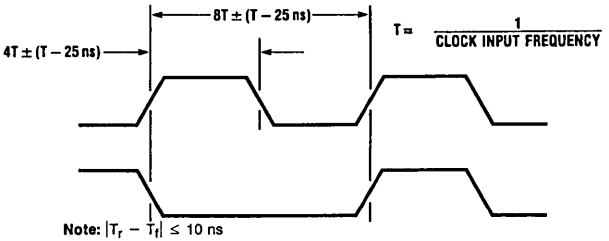
TL/F/5237–13

FIGURE 11. Data Clock and Serial Data Timing



TL/F/5237–14

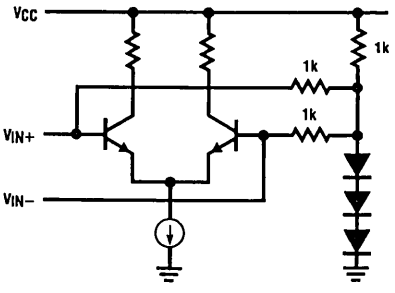
FIGURE 12. Data Waveform Constraints: Amplifier Inputs



Note: $|T_r - T_f| \leq 10$ ns

TL/F/5237–15

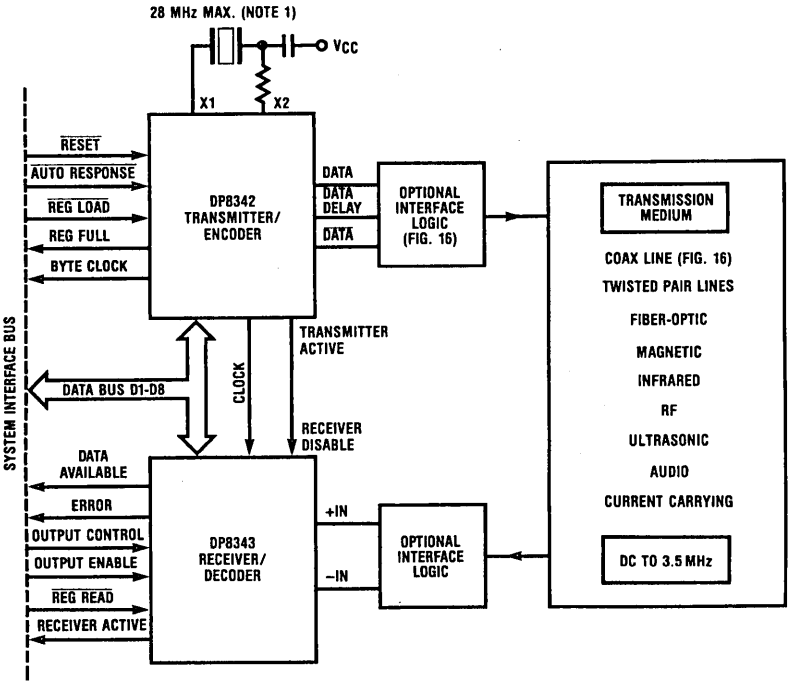
FIGURE 13. Data Waveform Constraints: Data Input (TTL)



TL/F/5237–16

FIGURE 14. Equivalent Circuit for DP8343 Input Amplifier

Typical Applications

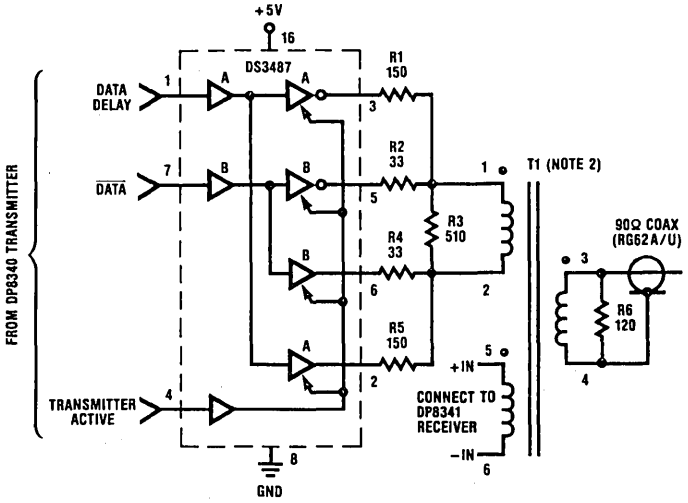


TL/F/5237–17

Note 1: Crystal manufacturer Midland Ross Corp., NEL Unit Part No. NE-18A @ 28 MHz

FIGURE 15

Typical Applications (Continued)

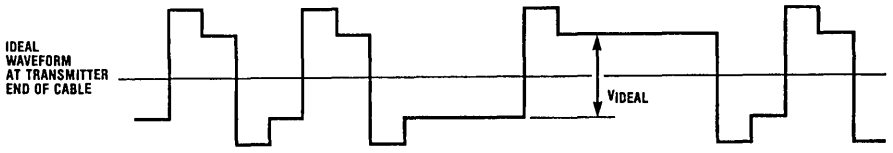


TL/F/5237-18

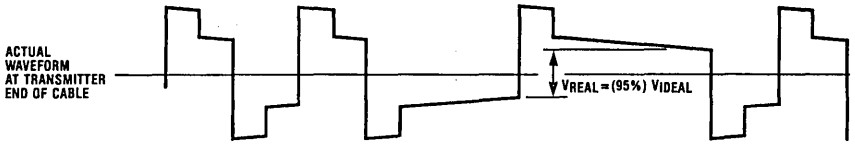
Note 1: Resistance values are in Ω, ±5%, ¼W.

Note 2: T1 is a 1:1:1 pulse transformer, L_{MIN} = 500 μH for 18 MHz system clock.
Pulse Engineering Part No. 5762,
Valor Electronics Part No. CT1501
Technitrol Part No. 11LHA or equivalent transformers.

FIGURE 16. Interface Logic for a Coax Transmission Line



TL/F/5237-19



TL/F/5237-20

*To maintain loss at 95% of ideal signal, select transformer inductance such that:

$$L_{(MIN)} = \frac{10{,}000}{f_{CLK}}$$

f_{CLK} = System Clock Frequency (e.g., 18.87 MHz)

Example:

$$L = \frac{10{,}000}{18.87 \times 10^6} \rightarrow L(MIN) = 530\ \mu H$$

Note 1: Less inductance will cause greater amplitude attenuation.

Note 2: Greater inductance may decrease signal rise time slightly and incease ringing, but these effects are generally negligible.

FIGURE 17. Transformer Selection

National Semiconductor

DP8344B Biphase Communications Processor—BCP®

General Description

The DP8344B BCP is a communications processor designed to efficiently process IBM® 3270, 3299 and 5250 communications protocols. A general purpose 8-bit protocol is also supported.

The BCP integrates a 20 MHz 8-bit Harvard architecture RISC processor, and an intelligent, software-configurable transceiver on the same low power microCMOS chip. The transceiver is capable of operating without significant processor interaction, releasing processor power for other tasks. Fast and flexible interrupt and subroutine capabilities with on-chip stacks make this power readily available.

The transceiver is mapped into the processor's register space, communicating with the processor via an asynchronous interface which enables both sections of the chip to run from different clock sources. The transmitter and receiver run at the same basic clock frequency although the receiver extracts a clock from the incoming data stream to ensure timing accuracy.

The BCP is designed to stand alone and is capable of implementing a complete communications interface, using the processor's spare power to control the complete system. Alternatively, the BCP can be interfaced to another processor with an on-chip interface controller arbitrating access to data memory. Access to program memory is also possible, providing the ability to download BCP code.

A simple line interface connects the BCP to the communications line. The receiver includes an on-chip analog comparator, suitable for use in a transformer-coupled environment, although a TTL-level serial input is also provided for applications where an external comparator is preferred.

A typical system is shown below. Both coax and twinax line interfaces are shown, as well as an example of the (optional) remote processor interface.

Features

Transceiver

- Software configurable for 3270, 3299, 5250 and general 8-bit protocols
- Fully registered status and control
- On-chip analog line receiver

Processor

- 20 MHz clock (50 ns T-states)
- Max. instruction cycle: 200 ns
- 33 instruction types (50 total opcodes)
- ALU and barrel shifter
- 64k x 8 data memory address range
- 64k x 16 program memory address range (note: typical system requires <2k program memory)
- Programmable wait states
- Soft-loadable program memory
- Interrupt and subroutine capability
- Stand alone or host operation
- Flexible bus interface with on-chip arbitration logic

General

- Low power microCMOS; typ. I_{CC} = 25 mA at 20 MHz
- 84-pin plastic leaded chip carrier (PLCC) package

Block Diagram

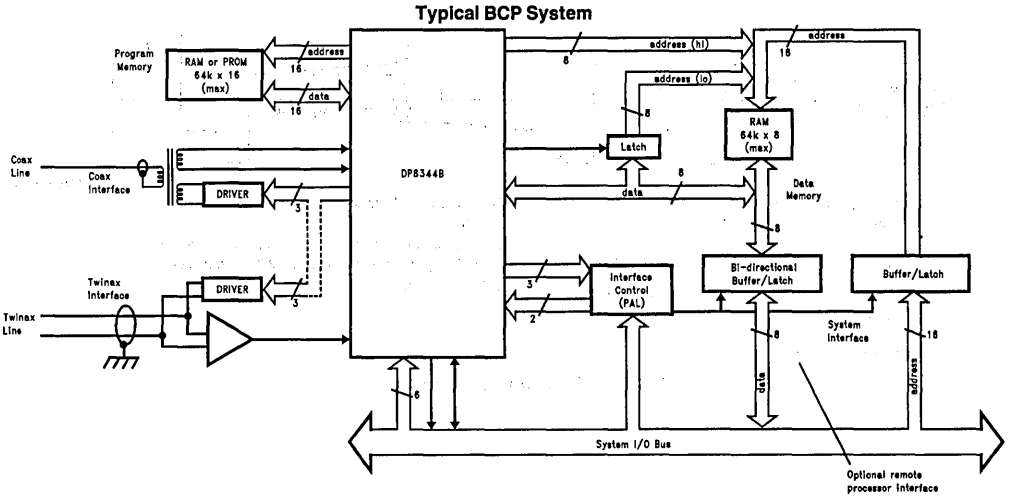


FIGURE 1

TL/F/9336-51

The DP8344B is an enhanced version of the DP8344A, exhibiting improved switching performance and additional functionality. The device has been been characterized in a number of applications and found to be a compatible replacement for the DP8344A. Differences between the DP8344A and DP8344B are noted by shading of the text on the pages of this data sheet. For more information, refer to Section 6.6.

Note: In this document [XXX] denotes a control or status bit in a register, {YYY} denotes a register.

Table of Contents

Table of Contents (Continued)

List of Illustrations

List of Illustrations (Continued)

List of Illustrations (Continued)

List of Tables

List of Tables (Continued)

Note: To match Timing table number with appropriate Timing illustration, Tables 5-1 and 5-2 are purposely omitted.

1.0 Communications Processor Introduction

The increased demand for computer connectivity has driven National Semiconductor to develop the next generation of special purpose microprocessors. The DP8344B is the first example of a "Communications Processor" for the IBM environment. It integrates a very fast, full function microprocessor with highly specialized transceiver circuitry. The combination of speed, power, and features allows the designer to easily implement a state-of-the-art communications interface. Typical applications for a communications processor are terminal emulation boards for PCs, stand-alone terminals, printer interfaces, and cluster controllers.

The transceiver is designed to simplify the handling of specific communication protocols. This feature makes it possible to quickly develop interfaces and software with little concern for the "housekeeping" details of the protocol being used.

1.1 COMMUNICATIONS PROTOCOLS

A communication protocol is a set of rules which defines the physical, electrical, and software specifications required to successfully transfer data between two systems.

The physical specification includes the network architecture, as well as the type of connecting medium, the connectors used, and the maximum distance between connections. Networks may be configured in "loops," "stars," or "daisy chains," and they often use standard coaxial or twisted-pair cable.

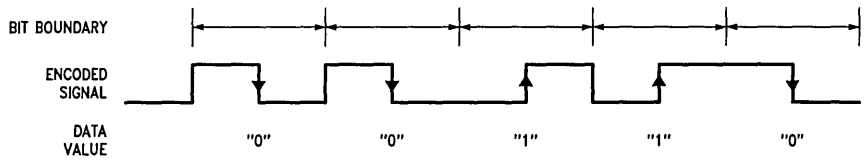
The electrical specification includes the polarity and amplitude of the signal, the frequency (bit rate), and encoding technique. One common method of encoding is called "biphase" or "Manchester II." This technique combines the clock and data information into one transmission by encoding data as a "mid-bit" transition. *Figure 1-1* shows how the data transition is related to the bit boundary in a typical transmission. The polarity of the "mid-bit" transition encodes the data value, other transitions lie on bit boundaries. Bit boundaries are not always indicated by transitions, so techniques employing start sequences and sync bits are used with bi-phase transmissions to ensure proper frame alignment and synchronization.

The software specification covers the use of start sequences and sync bits, as well as defining the message format. Parity bits may be used to ensure data integrity. The message format is the "language" that is used to exchange information across the connecting medium. It defines command and control words, response times, and expected responses.

The DP8344B Bi-phase Communications Processor supports both the IBM 3270 and 5250 communication protocols, as well as IBM 3299 and a general purpose 8-bit protocol. The specialized transceiver is combined with a microprocessor whose instruction set is optimized for use in a communications environment. This makes the DP8344 a powerful single-chip solution to a wide range of communication applications.

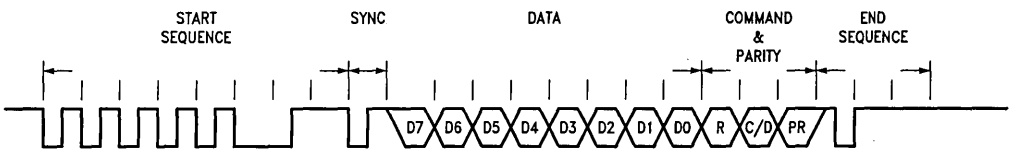
An example of an IBM 3270 message is shown in *Figure 1-2*. The transmission begins with a very specific start sequence and sync pulse for synchronization. This is followed by the data, command, and parity bits. Finally, the end sequence defines the end of the transmission.

The IBM 3270 and 5250 are two widely used protocols. The 3270 protocol was developed for the 370 class mainframe, and it employs coaxial cable in a "star" configuration. The 5250 protocol was developed for the System/3x machines, and it uses a "daisy-chain" of twin-ax cable. A good overview of both of these environments may be found in the "Multi-Protocol Adapter System User Guide" from National Semiconductor, and in the Transceiver section of this document.



TL/F/9336-B7

FIGURE 1-1. Biphase Encoding



TL/F/9336-B8

FIGURE 1-2. IBM 3270 Message Format

1.0 Communications Processor Introduction (Continued)

1.2 INTERNAL ARCHITECTURE INTRODUCTION

The DP8344B Biphase Communications Processor (BCP) is divided into three major functional blocks: the Transceiver, the Central Processing Unit (CPU), and the Remote Interface and Arbitration System, RIAS. *Figure 1-3* shows how these blocks are related to each other and to other system components.

The transceiver consists of an asynchronous transmitter and receiver which can communicate across a serial data path. The transmitter takes parallel data from the CPU and appends to it the appropriate framing information. The resulting message is shifted out and is available as a serial data stream on two output pins. The receiver shifts in serial messages, strips off the framing information, and makes the data available in parallel form to the CPU. The framing information supplied by the BCP provides the proper message format for several popular communication protocols. These include IBM 3270, 3299, and 5250, as well as a general purpose 8-bit mode.

The transceiver clock may be derived from the internal oscillator, either directly or through internal divide-down circuitry. There is also an input for an external transceiver clock, thus allowing complete flexibility in the choice of data rates.

The receiver input can come from three possible sources. There is a built-in differential amplifier which is suitable for most line interfaces, a single-ended digital input for use with an external comparator, and an internal loopback path for self testing. Refer to the Transceiver section for a detailed description of all transmitter and receiver functions, and to the application note on coax interfaces for the proper use of the differential amplifier.

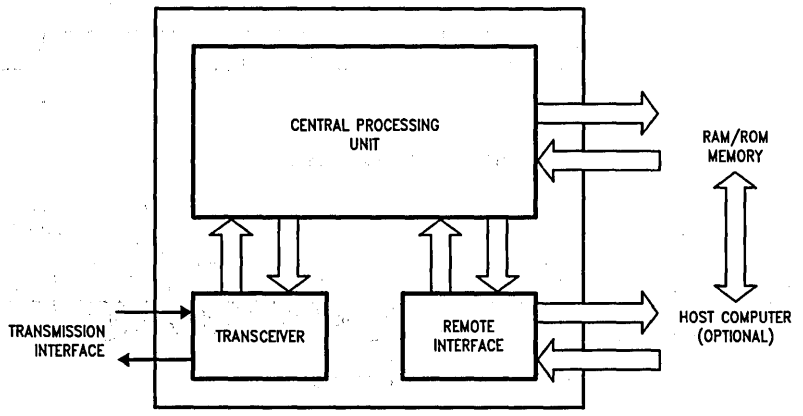
The CPU is a general purpose, 8-bit microprocessor capable of 20 MHz operation. It has a reduced instruction set which is optimized for transceiver and data handling performance. It also has a full function arithmetic/logic unit (ALU) which performs addition, subtraction, Boolean operations, rotations and shifts. Separate instruction and data memory systems are supported, each with 16-bit address buses, for a total of 64k address space in each.

There are 44 internal registers accessible to the CPU. These include special configuration and control registers for the transceiver and processor, four 16-bit indices to data memory, and 20 8-bit general purpose registers. There is also a 16-bit timer and a 16-byte deep LIFO data stack which are accessible in the register address space. For more detailed information, see the specific sections on the Register set, the Timer, and the ALU.

The BCP can operate independently or with another processor as the host system. If such a system is required, communication with the BCP is possible by sharing data memory. The Remote Interface controls bus arbitration and access to data memory, as well as program up-loading and execution. For example, it is possible for a host system to load the BCP's instruction memory and begin program execution, then pass data back and forth through data memory accesses. The section on the Remote Interface and Arbitration System provides all of the necessary timing and control information to implement an interface between a BCP and a remote system.

As shown in *Figure 1-4,* the BCP uses two entirely separate memory systems, one for program storage and the other for data storage. This type of memory arrangement is referred to as Harvard architecture. Each system has 16 address lines, for a maximum of 64k words in each, and its own set of data lines. The instruction (program) memory is two bytes (16 bits) wide, and the data memory is one byte (8 bits) wide.

In order to reduce the number of pins required for these signals, the address and data lines for data memory are multiplexed together. This requires an external latch and the Address Latch Enable signal (ALE) for de-multiplexing.



TL/F/9336-B9

FIGURE 1-3. Simplified Block Diagram

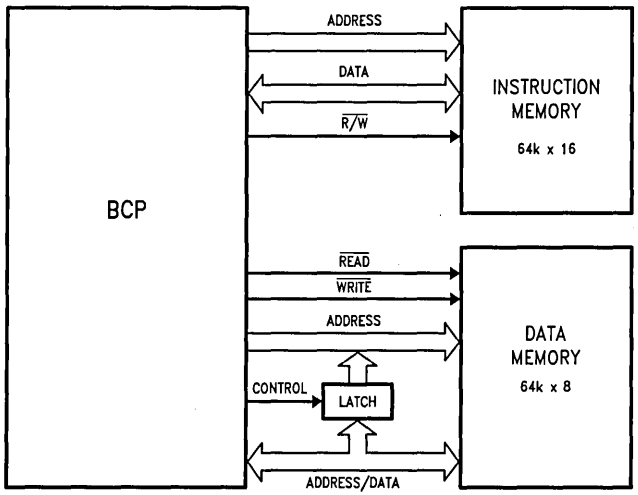
1.0 Communications Processor Introduction (Continued)

Simultaneous access to both data and program memory, and instruction pipelining greatly enhance the speed performance of the BCP, making it well suited for real-time processing. The pipeline allows the next instruction to be retrieved from program memory while the current instruction is being executed.

1.3 TIMING INTRODUCTION

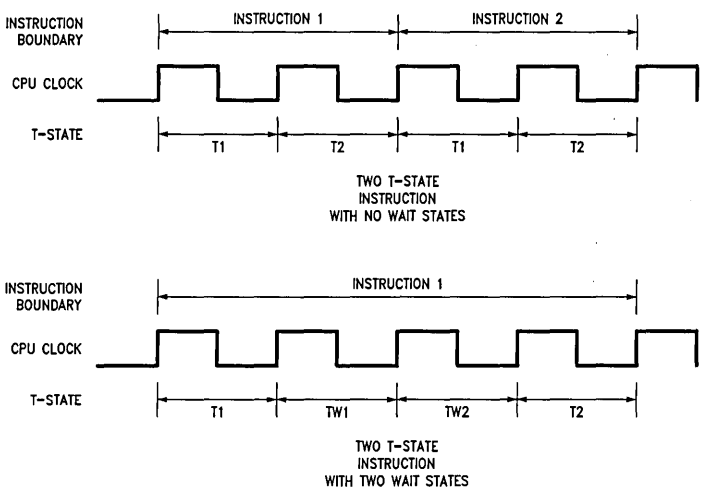
The timing of all CPU operations, instruction execution and memory access is related to the CPU clock. This clock is usually generated by a crystal and the internal oscillator, with optional divide by two circuitry. The period of the resulting CPU clock is referred to as a T-state; for example, a 20 MHz CPU clock yields a 50 ns T-state. Most CPU functions, such as arithmetic and logical operations, shifts and rotates, and register moves, require only two T-states. Branching instructions and data memory accesses require three to four T-states.

Each memory system has a separate, programmable number of wait states to allow the use of slower memory devices. Instruction memory wait states are inserted into all instructions, as shown in *Figure 1-5,* thus they affect the overall speed of program execution. Instruction memory wait states can also apply when the Remote Interface is loading a program into instruction memory. Data memory wait states are only inserted into data memory access instructions, hence there is less degradation in overall program execution. Refer to the Timing section for detailed examples of all BCP instruction and data memory timing.



TL/F/9336–C1

FIGURE 1-4. Memory Configuration



TL/F/9336–C2

FIGURE 1-5. Effect of Memory Wait States on Timing

1.0 Communications Processor Introduction (Continued)

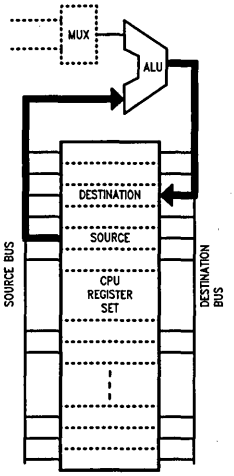
1.4 DATA FLOW

The CPU registers are all dual port, that is, they have separate input and output paths. This arrangement allows a single register to function as both a source and a destination within the same instruction.

Figures 1-6a through *1-6f* show the internal data flow path for the BCP. The CPU registers are a central element to this path. When a register functions as an output, its contents are placed on the Source bus. When a register is an input, data from the Destination bus is written into that register.

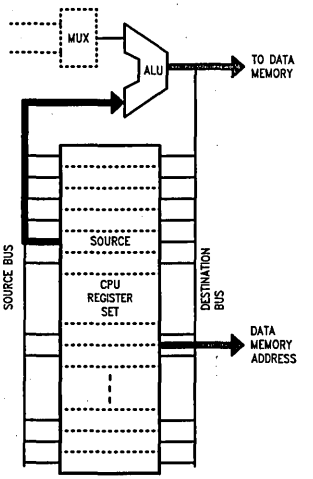
The other key element in the data path is the ALU. This unit does all of the arithmetic and data manipulation operations, but it also has bus multiplexing capabilities. Both the Data Memory bus and a portion of the Instruction Memory bus are routed to this unit and serve as alternative sources of data. Since the data flow is always through this unit, most data moves may include arithmetic manipulations with no penalty in execution time.

Figure 1-6a shows the data path for all arithmetic instructions and register to register moves. The source register contents are placed on the Source bus, routed through the



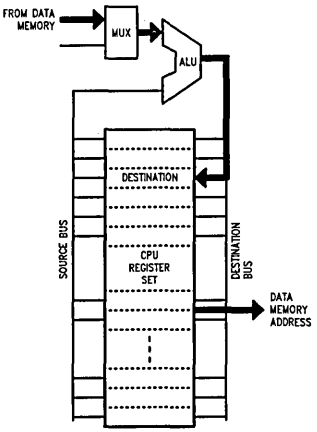
TL/F/9336–C3

FIGURE 1-6a. Register to Register



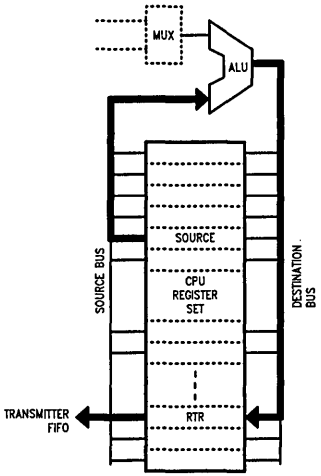
TL/F/9336–C4

FIGURE 1-6b. Data Memory WRITE



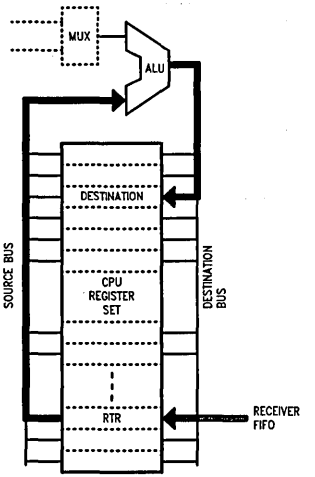
TL/F/9336–C5

FIGURE 1-6c. Data Memory READ



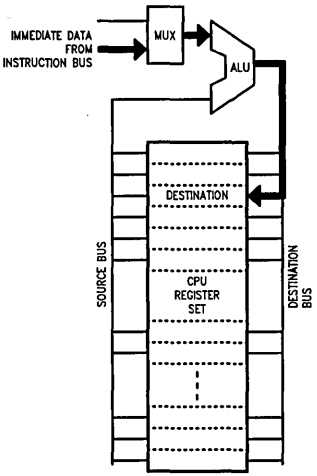
TL/F/9336–C6

FIGURE 1-6d. WRITE to Transmitter



TL/F/9336–C7

FIGURE 1-6e. READ from Receiver



TL/F/9336–C8

FIGURE 1-6f. Load Immediate Data

1.0 Communications Processor Introduction (Continued)

ALU/MUX, and then placed on the destination bus. This data is then stored into the appropriate destination register.

Figures 1-6b and *1-6c* show the data path for data memory accesses. For a WRITE operation, the source register contents follow the same path through the ALU/MUX, but the Destination bus is routed to output pins and on to data memory. For a READ operation, incoming data is routed onto the Destination bus by the ALU/MUX, and then stored in a register. The address for all data memory accesses is provided by one of four 16-bit index registers which can operate in a variety of automatic increment and decrement modes.

Transfer of the data byte between the CPU and the Transceiver is accomplished through a register location. This register, {RTR}, appears as a normal CPU register, but writing to it automatically transfers data to the transmitter FIFO, and reading from it retrieves data from the receiver FIFO. These paths are illustrated in *Figures 1-6d* and *1-6e.*

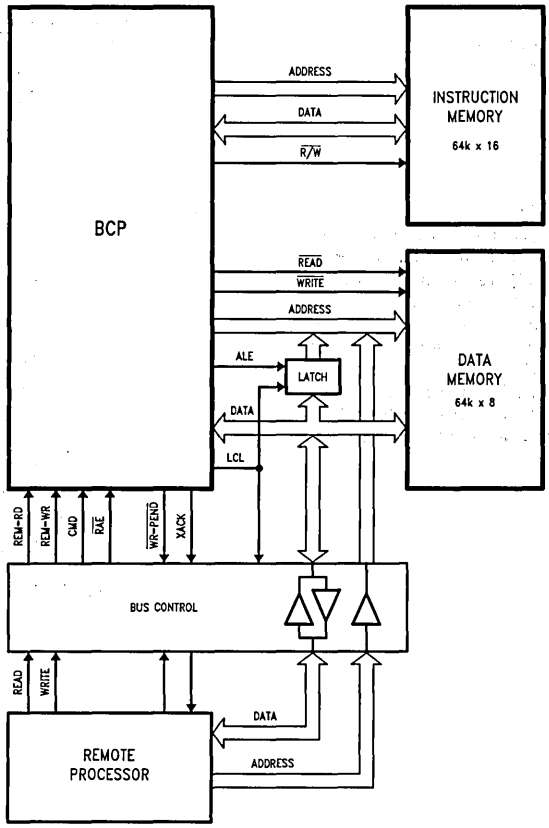
It is also possible to load immediate data into a CPU register. This data is supplied by the program and is usually a constant such as a pointer or character. As shown in *Figure 1-6f,* a portion of the Instruction bus is routed through the ALU/MUX for this purpose.

1.5 REMOTE INTERFACE AND ARBITRATION SYSTEM INTRODUCTION

The BCP is designed to serve as a complete, stand alone communications interface. Alternately, it can be interfaced with another processor by means of the Remote Interface and Arbitration System. Communication between the BCP and the remote processor is possible by sharing data memory. Harvard architecture allows the remote system to access any BCP data memory location while the BCP continues to fetch and execute instructions, thereby minimizing performance degradation.

Figure 1-7 shows a simplified remote processor interface. This includes tri-state buffers on the address and data buses of the BCP's Data Memory, and all of the control and handshaking signals required to communicate between the BCP and the host system.

There is an 8-bit control register, Remote Interface Control {RIC}, accessible only to the remote system, which is used to control a variety of features, including the types of memory accesses, interface speeds, single step program execution, CPU start/stop, instruction memory loads, and so forth. Detailed information on all interface options is provided in the section on Remote Interface and Arbitration System, and in the related Reference section.



TL/F/9336–C9

FIGURE 1-7. Basic Remote Interface

2.0 CPU Description

The CPU is a general purpose, 8-bit microprocessor capable of 20 MHz operation. It contains a large register set for standard CPU operations and control of the transceiver. The reduced instruction set is optimized for the communications environment. The following sections are an architectural and functional description of the DP8344B CPU.

2.1 CPU ARCHITECTURAL DESCRIPTION

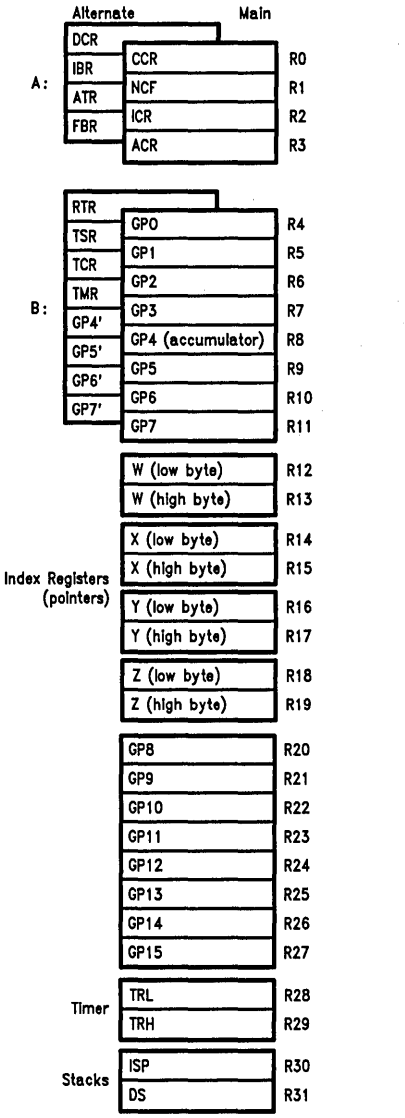
2.1.1 Register Set

This section describes the BCP's internal CPU registers. It is a general overview of the register structure and the functions mapped into the CPU register space. It is not a detailed or exhaustive description of every bit. For such a description, please refer to Section 6.2, Register Set Reference. Also, the Remote Interface Configuration register, {RIC}, is not accessible to the BCP (being accessible only by the remote system) and is described in Section 6.3, Remote Interface Reference.

The register set of the BCP provides for a compliment of both special function and general purpose registers. The special function registers provide access to on-chip peripherals (transceiver, timer, interrupt control, etc.) while the general purpose registers maximize CPU throughput by minimizing accesses to external data memory. The CPU can address a total of 44 8-bit registers, providing access to:

- 20 general purpose registers
- 8 configuration and control registers
- 4 transceiver access registers
- 2 8-bit accumulators
- 4 16-bit pointers
- 16-bit timer
- 16 byte data stack
- address and data stack pointers

The CPU addresses internal registers with a 5-bit field, addressing 32 locations generically named R0 through R31. The first twelve locations (R0–R11) are further organized by function as two groups of banked registers (A and B) as shown in *Figure 2-1.* Each group contains both a main and an alternate bank. Only one bank is active for group A and one for bank B and thus accessible during program execution. Switching between the banks is performed by the exchange instruction EXX which selects whether Main A or Alternate A occupies R0–R3 and whether Main B or Alternate B occupies R4–R11.



TL/F/9336-32

FIGURE 2-1. Register Map

2.0 CPU Description (Continued)

Registers in the R0–R11 address space are allocated in a manner that minimizes the need to switch banks:

Main A: CPU control and transceiver status

Alternate A: CPU and transceiver configuration

Main B: 8 general purpose

Alternate B: 4 transceiver access, 4 general purpose

Most of the BCP's instructions with register operand(s) can access all 32 register locations. Only instructions with an immediate operand are limited to the first sixteen register locations (R0–R15). These instructions, however, still have access to all registers required for transceiver operation, CPU status and control registers, 12 general purpose registers, and two of the index registers.

The general purpose registers are used for the majority of BCP operations. There are 8 general purpose registers in Main Bank B (R4–R11), 4 in Alternate Bank B (R8–R11), and 8 more (R20–R27) that are always accessible but are outside the limited register range. Since these registers are internal to the BCP, they can be accessed without data memory wait states, speeding up processing time. The index registers may also be used as general purpose registers if required.

For those instructions that require two operands, an accumulator (R8, one in each bank) serves as the second operand. The result of such an operation is stored back in the accumulator only if it is specified as the destination, thus allowing three operand operations such as R5 + R8 → R20. See Section 2.1.3 Instruction Set for further explanation.

Most registers have a predetermined state following a reset to the BCP. Refer to Section 6.2, Register Set Reference for a detailed summary.

2.1.1.1 Banked Registers

The CPU register set was designed to optimize CPU performance in an environment which supports multiple tasks. Generally the most important and time critical of these tasks will be maintaining the serial link (servicing the transceiver section) which often requires real time processing of commands and data. Therefore, all transceiver functions have been mapped into special function registers which the CPU can access quickly and easily. Switching between this task and other tasks has been facilitated by dedicating a register bank (Alternate B) to transceiver functions. Alternate Bank B provides access to all transceiver status, control, and data, in addition to four general purpose registers for protocol related storage. Main Bank B contains eight general purpose registers for use by other tasks. Having general purpose registers in both B banks allows for quick context switching and also helps eliminate some of the overhead of saving general purpose registers. The main objective of this banked register structure is to expedite servicing of the transceiver as a background (interrupt driven) task allowing the CPU to efficiently interleave that function with other background and foreground operations.

To facilitate using the transceiver in a polled fashion (instead of using interrupts), many of the status flags necessary to handshake with the transceiver are built into the conditional jump instructions, with others available in the Main A bank (normally active) so that Alternate Bank B does not have to be switched in to poll the transceiver. Timer and BIRQ tasks may also be run using polling techniques to Main A bank.

In general, the registers have been arranged within the banks so as to minimize the need to switch banks. The power-up state is Alternate bank A, Alternate bank B allowing access to configuration registers. Again, the banks switch by using the EXX instruction which explicitly specifies which bank is active (Main or Alternate) for each register group (A and B). The EXX instruction allows selecting any of four possible bank settings with a single two T-state instruction. This instruction also has the option of enabling or disabling the maskable interrupts.

The contents of the special function registers can be divided into several groups for general discussion—timing/control, interrupt control, the transceiver, the condition codes, the index registers, the timer, the stacks, and remote interface.

2.1.1.2 Timing/Control Registers

The BCP provides a means to configure its external timing through setting bits in the Device Control Register, {DCR}, and the Auxiliary Control Register, {ACR}. One of the first configuration registers to be initialized on power-up/reset is {DCR} which defines the hardware environment in which the BCP is functioning. Specifically, {DCR} controls the clock select logic for both the CPU and transceiver, in addition to the number of wait states to be used for instruction and data memory accesses.

The BCP allows either one clock source operation for the CPU and the transceiver from the on-chip oscillator, or an independent clock source can run the transceiver from the eXternal Transceiver CLocK input, X-TCLK. The Transceiver Clock Select bits, [TCS1,0], select the clock source for the transceiver which is either the on-chip Oscillator CLocK, OCLK, or X-TCLK. Options for selecting divisions of the on-chip oscillator frequency are also provided (see the description of {DCR} in Section 6.2, Register Set Reference. The CPU Clock Select bit, [CCS], allows the CPU to run at the OCLK frequency or at half that speed. The clock output at the pin CLK-OUT, however, is never divided and always reflects the crystal frequency OCLK. The frequency selected for the transceiver (referred to as TCLK) should always be eight times the desired serial data rate. The frequency selected for the CPU defines the length of each T-state (e.g., 20 MHz implies 50 ns T-states).

There are two independent fields for defining wait states, one for instruction memory access (n_{IW}) and one for data memory access (n_{DW}). These fields specify to the BCP how many wait states to insert to meet the access time requirements of both memory systems. The Instruction memory Wait-state select bits, [IW1,0], and the Data memory Wait-state select bits, [DW2–0], control the number of inserted wait states for instruction and data memory, respectively.

After a reset, the maximum number of wait states are set in {DCR}, n_{IW} = 3 T-states and n_{DW} = 7 T-states. Wait-states are discussed in more detail in Section 2.2.2, Timing. For a complete discussion on choosing your memory and determining the number of wait states required, please refer to the application note *Choosing Your RAM for the Biphase Communication Processor.*

2.0 CPU Description (Continued)

Another control bit in the {ACR} register is the Clock Out Disable bit, [COD]. When [COD] is asserted, the buffered clock output at pin CLK-OUT is tri-stated.

2.1.1.3 Interrupt Control Registers

The configuration bank (Alternate Bank A) includes an Interrupt Base Register, {IBR}, which defines the high byte of all interrupt and trap vector addresses. Thus, the interrupt vector table can be located in any 256 byte page of the 64k range of instruction addresses. The interrupt base is normally initialized once on reset before interrupts are enabled or any traps are executed. Since $\overline{NMI}$ is nonmaskable and may occur before {IBR} is initialized, the power-up/reset value of {IBR} (00h) should be used to accommodate $\overline{NMI}$ during initialization. In other words, if $\overline{NMI}$ is used in the system, the absolute address 001Ch (the $\overline{NMI}$ vector) should contain a jump to an $\overline{NMI}$ service routine.

The Interrupt Control Register, {ICR}, provides individual masks [IM4–0] for each of the maskable interrupts. The Global Interrupt Enable bit, [GIE], located in {ACR} works in conjunction with these individual masks to control each of the maskable interrupts.

The external pin called $\overline{BIRQ}$ is a Bidirectional Interrupt ReQuest. $\overline{BIRQ}$ is defined as an input or an output by the Bidirectional Interrupt Control bit, [BIC], in {ACR}. [IM3] functions as BIRQ's interrupt mask if $\overline{BIRQ}$ is an input as defines by [BIC]. When [BIC] defines BIRQ as an output, [IM3] controls the output state of $\overline{BIRQ}$.

Section 2.2.3, Interrupts provides a further description of these registers.

2.1.1.4 Timer Registers

The timer block interfaces with the CPU via two registers, TimeR Low byte, {TRL}, and TimeR High byte, {TRH}, which form the input/output ports to the timer. Writing to {TRL} and {TRH} stores the low and high byte, respectively, of a 16-bit time-out value into two holding registers. The word stored in the holding registers is the value that the timer will be loaded with via [TLD]. Also, the timer will automatically reload this word upon timing out. Reading {TRL} and {TRH} provides access to the count down status of the timer.

Control of timer operation is maintained via three bits in the Auxiliary Control Register {ACR}. Timer STart [TST], bit 7 in {ACR}, is the start/stop control bit. Writing a one to [TST] allows the timer to start counting down from its current value. When low, the timer stops and the timer interrupt is cleared. Timer Load [TLD], bit 6 in {ACR}, is the load control of the timer. After writing the desired values into {TRL} and {TRH}, writing a one to [TLD] will load the 16-bit word in the holding registers into the timer and initialize the timer clock to zero in preparation to start counting. Upon completing the load operation, [TLD] is automatically cleared. Timer Clock Selection [TCS], bit 5 in {ACR}, determines the clock frequency of the timer count down. When low, the timer divides the CPU clock by sixteen to form the clock for the down counter. When [TCS] is high, the timer divides the CPU clock by two. The input clock to the timer is the CPU clock and should not be confused with the oscillator clock, OCLK. The rate of the CPU clock will be either equal to OCLK or one-half of OCLK depending on the value of bit 7 in the Device Control Register, {DCR}.

When the timer reaches a count of zero, the timer interrupt is generated, the Time Out flag, [TO], (bit 7 in the Condition Code Register {CCR}), goes high, and the timer reloads the 16-bit word stored in the holding registers to recycle through a count down. The timer interrupt and [TO] can be cleared by either writing a one to [TO] in {CCR} or stopping the timer by writing a zero to [TST] in {ACR}. Refer to Section 2.1.2, Timer for more information on the timer operation.

2.1.1.5 Transceiver Registers

Two registers in the Alternate A bank initialize transceiver functions. The Auxiliary Transceiver Register, {ATR}, specifies a station address used by the address recognition logic within the transceiver when using the non-promiscuous 5250 and 8-bit protocol modes. In 5250 modes, {ATR} also defines how long the TX-ACT pin stays asserted after the end of a transmitted message. The Fill Bit Register, {FBR}, specifies the number of optional fill bits inserted between frames in a multiframe 5250 message.

{ICR} contains the Receiver Interrupt Select bits, [RIS1,0]. These bits determine the receiver interrupt source selection. The source may be either Receiver FIFO Full, Data Available, or Receiver Active.

The Receive/Transmit Register, {RTR}, is the input/output port to both the transmitter and receiver FIFO's. It appears to the BCP CPU like any other register. The {RTR} register provides the least significant eight bits of data in both received and transmitted messages.

The Transceiver Mode Register, {TMR}, contains bits used to set the configuration of the transceiver. As long as the Transceiver RESet bit, [TRES], is high, the transceiver remains in reset. Internal LOOP-back operation of the transceiver can be selected by asserting [LOOP]. The RePeat ENable bit, [RPEN], allows the receiver to be active at the same time as the transmitter. When the Receiver INvert bit, [RIN], is set, all data sent to the receiver is inverted. The Transmitter INvert bit, [TIN], is analogous to [RIN] except it is for the transmitter. The protocol that the transceiver is using is selected with the Protocol Select bits, [PS2–0].

The Transceiver Command Register, {TCR}, controls the workings of the transmitter. To generate 5.5 line quiesce pulses at the start of a transmission rather than 5, the Advance Transmitter Active bit, [ATA], must be set high. Parity is automatically generated on a transmission and the Odd Word Parity bit, [OWP], determines whether that parity is even or odd. Bits 2–0 of {TCR} make up part of the Transmitter FIFO [TF10–8] along with {RTR}. Whenever a write is made to {RTR}, [TF10–8] are automatically pushed on the FIFO with the 8 bits written to {RTR}.

Other bits in {TCR} control the operation of the on-chip receiver. The number of line quiesce bits the receiver must detect to recognize a valid message is determined by the Receive Line Quiesce bit, [RLQ]. The BCP has its own internal analog comparator, but an off-chip one may be connected to DATA-IN. The receiver source is determined by the Select Line Receiver bit, [SLR]. To view transceiver errors in the Error Code Register, {ECR}, the Select Error Codes, [SEC], bit in {TCR} must be set high. When [SEC] is high, Alternate Bank B R4 is remapped from {RTR} to {ECR} so that {ECR} can be read.

2.0 CPU Description (Continued)

Just as [TF10–8] bits get pushed onto the transmitter FIFO when a write to {RTR} occurs, the Receiver FIFO bits, [RF10–8], in the Transceiver Status Register, {TSR}, reflect the state of the top word of the receive FIFO. {TSR} also contains flags that show Transmit FIFO Full, [TFF], Transmitter Active, [TA], Receiver Error, [RE], Receiver Active, [RA], and Data AVailable, [DAV]. These flags may be polled to determine the state of the transceiver. For instance, during a Receiver Active interrupt, the BCP can query the [DAV] bit to determine whether data is ready in the receiver FIFO yet.

The Error Code Register, {ECR}, contains flags for receiver errors. As previously stated, the [SEC] bit in {TRC} must be set high to read this register. Reading {ECR} or resetting the transceiver with [TRES] will clear all the errors that are present. The receiver OVerFlow flag, [OVF], is set when the receiver attempts to add another word to the FIFO when it is full. If internally checked parity and parity transmitted with a 3270 message conflict, then the PARity error bit, [PAR], is set high. The Invalid Ending Sequence bit, [IES], is set when the ending sequence in a 3270, 3299, or 8-bit message is incorrect. When the expected mid-bit transition in the Manchester waveform does not occur, a Loss of Mid-Bit Transition occurs ([LMBT]). Finally, if the transmitter is activated while the receiver is active, the Receiver DISabled while active flag, [RDIS], will be set unless [RPEN] is asserted.

The second register in Main A bank is called the Network Command Flag register, {NCF}, and contains information about the transceiver which is useful for polling the transceiver (during other tasks for example) to see if it needs servicing. These flags include bits to indicate Transmit FIFO Empty [TFE], Receive FIFO Full [RFF], Line Active [LA], and a Line Turn Around [LTA]. [LTA] indicates that a message has been received without error and a valid ending sequence has occurred. These flags facilitate polling of the transceiver section when transceiver interrupts are not used. Also included in this register is a bit called [DEME] (Data Error/Message End). In 3270/3299 modes, this bit indicates a mismatch between received and locally generated byte parity. In 5250 modes, [DEME] decodes an end of message indicator (111 in the address field). Three other bits: Received Auto Response [RAR], Acknowledge [ACK] and Poll [POLL] are decoded from a received message (at the output of the receive FIFO) and are valid only in 3270/3299 modes where response time is critical.

Section 3.0 Transceiver provides comprehensive coverage of this on-chip peripheral.

2.1.1.6 Condition Codes/Remote Handshaking Register

The ALU condition codes are available in the Condition Code Register {CCR}. The [Z] bit is set when a zero result is generated by an arithmetic, logical, or shift instruction. Similarly, [N] indicates the Negative result of the same operations. An oVerflow condition from an arithmetic instruction sets the [V] bit in {CCR}. The Carry bit [C] indicates a carry or borrow result from an arithmetic instruction. See Section 2.2.2, ALU for more information.

The Condition Code Register, {CCR}, also contains [BIRQ], a status bit which reflects the logic level of the bidirectional interrupt input pin BIRQ. Hence, this pin can be used as a general purpose input/output port as well as a bidirectional interrupt request as defined by bits in {ACR} and {ICR}. If a remote CPU is present and shares data memory (dual port memory) with the BCP, handshaking can be accomplished by using the two status bits in {CCR} called [RR] and [RW], which indicate Remote Read and Remote Write accesses, respectively.

In {ACR}, a lock bit, [LOR], is available to lock out all host accesses. When this bit is set, all host accesses are disabled. Locking out remote accesses is often done during interrupts to ensure quick response times.

The Remote Interface Configuration register, {RIC}, is not available to the BCP internally. The Remote Interface Reference section provides further detail on {RIC} and interfacing a remote processor.

2.1.1.7 Index Registers

Four index registers called IW, IX, IY, and IZ provide 16-bit addressing for both data memory and instruction memory. Each of these index registers is actually a pair of 8-bit registers which are individually addressable just like any other CPU register. They occupy register addresses R12 through R19. Thus, the first two pointers IW and IX (comprising R12–R15) can be accessed with immediate mode instructions (which can access only R0 to R15). Refer to Section 2.1.3.2, Addressing Modes to see how the index registers are formed from R12–R19.

Accessing data memory requires the use of one of the four index registers. All such instructions allow you to specify which pointer is to be used, except the immediate-relative moves: MOVE rs,[IZ+n] and MOVE [IZ+n],rd. These instructions always use the IZ pointer. Register indirect operations have options to alter the value of the index register; the options include pre-increment, post-increment, and post-decrement. These options facilitate block moves, searches, etc. Refer to Section 2.1.3, Instruction Set for more information about data moves.

Since the BCP's ALU is 8 bits wide, all code that manipulates the index registers must act on them eight bits at a time.

The index registers can also be used in register indirect jumps (LJMP [Ir]), useful in implementing relocatable code. Any one of the index registers can be specified to provide the 16-bit instruction address for the indirect jump.

2.1.1.8 Stack Registers

The last two register addresses (R30,R31) are dedicated to provide access to the two on-chip stacks—the data stack and the address stack. The data stack is 8 bits wide and 16 words deep. It is a Last In First Out (LIFO) type and provides high speed storage for variables, pointers, etc. The address stack is 23 bits wide and 12 words deep, providing twelve levels of nesting of subroutines and interrupts. It is also a LIFO structure and stores processor status as well as return addresses from CALL instructions, TRAP instructions, and interrupts. The seven bits of processor status consist of the four ALU flags, ([C], [N], [V], and [Z]), the current bank setting (two bits), and [GIE].

Stack pointers for both the on-chip stacks are provided in R30, the Internal Stack Pointer register, {ISP}. The lower four bits are the pointer for the data stack and the upper four bits are the pointer for the address stack. Both internal stacks are circular. For example if 16 bytes are written to

2.0 CPU Description (Continued)

the data stack, the next byte pushed will overwrite the first. {ISP} can be read and written to like any other register, but after a write, the BCP must execute one instruction before reading the stack whose pointer was modified.

The Data Stack register, {DS}, is the input/output port for the data stack. This port is accessed like any other register, but a write to it will "push" a byte onto the stack and a read from it will "pop" a byte from the stack. The data stack pointer is updated when a read or write of {DS} occurs.

Information bits in the instruction address stack are not mapped into the CPU's register space and, therefore, are not directly accessible. A remote system running a monitor program can access this information by forcing the BCP to single-step through a return instruction and then reading the program counter. Since the stack pointers are writeable, the remote system can access any location (return address) in the address stack to trace program flow and then restore the stack pointer to its original position.

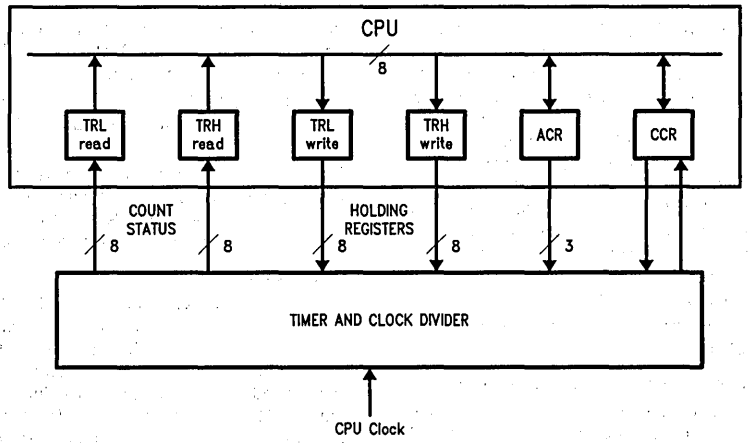
2.1.2 Timer

The BCP has an internal 16-bit timer that can be used in a variety of ways. The timer counts independently of the CPU, eliminating the waste of valuable processor bandwidth. The timer can be used in a polled or interrupt driven configuration for user software flexibility.

The timer interfaces with the CPU via two registers, TimeR Low byte, {TRL}, and TimeR High byte, {TRH}, which form the input/output ports to the timer. Writing to {TRL} and {TRH} stores the low and high byte, respectively, of a 16-bit time-out value into two holding registers. The word stored in the holding registers is the value that the timer will be loaded with via [TLD]. Also, the timer will automatically reload this word upon timing out. Reading {TRL} and {TRH} provides access to the count down status of the timer.

Control of timer operation is maintained via three bits in the Auxiliary Control Register {ACR}. Timer STart [TST], bit 7 in {ACR}, is the start/stop control bit. Writing a one to [TST] allows the timer to start counting down from its current value. When low, the timer stops and the timer interrupt is cleared. Timer Load [TLD], bit 6 in {ACR}, is the load control of the timer. After writing the desired values into {TRL} and {TRH}, writing a one to [TLD] will load the 16-bit word in the holding registers into the timer and initialize the timer clock to zero in preparation to start counting. Upon completing the load operation, [TLD] is automatically cleared. Timer Clock Selection [TCS], bit 5 in {ACR}, determines the clock frequency of the timer count down. When low, the timer divides the CPU clock by sixteen to form the clock for the down counter. When [TCS] is high, the timer divides the CPU clock by two. The input clock to the timer is the CPU clock and should not be confused with the oscillator clock, OCLK. The rate of the CPU clock will be either equal to OCLK or one-half of OCLK depending on the value of bit 7 in the Device Control Register, {DCR}.

When the timer reaches a count of zero, the timer interrupt is generated, the Time Out flag, [TO], (bit 7 in the Condition Code Register {CCR}), goes high, and the timer reloads the 16-bit word stored in the holding registers to recycle through a count down. The timer interrupt and [TO] can be cleared by either writing a one to [TO] in {CCR} or stopping the timer by writing a zero to [TST] in {ACR}. A block diagram of the timer is shown in *Figure 2-2.*



TL/F/9336-D1

FIGURE 2-2. Timer Block Diagram

2.0 CPU Description (Continued)

2.1.2.1 Timer Operation

After the desired 16-bit time-out value is written into {TRL} and {TRH}, the start, load, and clock selection can be achieved in a single write to {ACR}. A restriction exists on changing the timer clock frequency in that [TCS] should not be changed while the timer is running (i.e., [TST] is high). After a write to {ACR} to load and start the timer, the timer begins counting down at the selected frequency from the value in {TRL} and {TRH}. Upon reaching a count of zero, the timer interrupt is generated and, the timer reloads the current word from {TRL} and {TRH} to cycle through a countdown again. The timing waveforms shown in *Figure 2-3* show a write to {ACR} that loads, starts, selects the CPU clock rate/2 for the countdown rate, and asserts the Global Interrupt Enable [GIE]. Prior to the write to {ACR}, {TRL} and {TRH} were loaded with 00h and 01h respectively, the timer interrupt was unmasked in the Interrupt Control Register {ICR} by clearing bit 4, and zero instruction wait states were selected in {DCR}. Since the write to {ACR} asserted [GIE], the timer interrupt is enabled and the CPU will vector to the timer interrupt service routine address when the timer reaches a count of zero. The timer interrupt is the lowest priority interrupt and is latched and maintained until it is cleared in software. (See CPU Interrupts section). For very long time intervals, time-outs can be accumulated under software control by writing a one to [TO] in {CCR} allowing the timer to recycle its count down with no other intervention. For time-outs attainable with one count down, stopping the timer will clear the interrupt and [TO]. When the timer interrupt is enabled, the call to the interrupt service routine occurs at different instruction boundaries depending on when the timer interrupt occurs in the instruction cycle. If the timer times out prior to T2, where T2 is the last T-state of an instruction cycle, the call to the interrupt service routine will occur in the next instruction. When the time-out occurs in T2, the call to the interrupt service routine will not occur in the next instruction. It occurs in the second instruction following T2.

The count status of the timer can be monitored by reading {TRL} and/or {TRH}. When the registers are read, the output of the timer, not the value in the input holding registers, is presented to the ALU. Some applications might require monitoring the count status of the timer while it is counting down. Since the timer can time-out between reads of {TRL} and {TRH}, the software should take this fact into consideration. To read back what was written to {TRL} and {TRH}, the timer must first be loaded via [TLD] without starting the timer followed by a one instruction delay before reading {TRL} and {TRH} to allow the output registers to be updated from the load operation.

To determine the time-out delay for a given value in {TRL} and {TRH} other than 0000h, the following equation can be used:

$$TD = (\text{value in } \{TRH\}\{TRL\}) * T * k$$

where:

k = 2 when [TCS] = 1 or 16 when [TCS] = 0

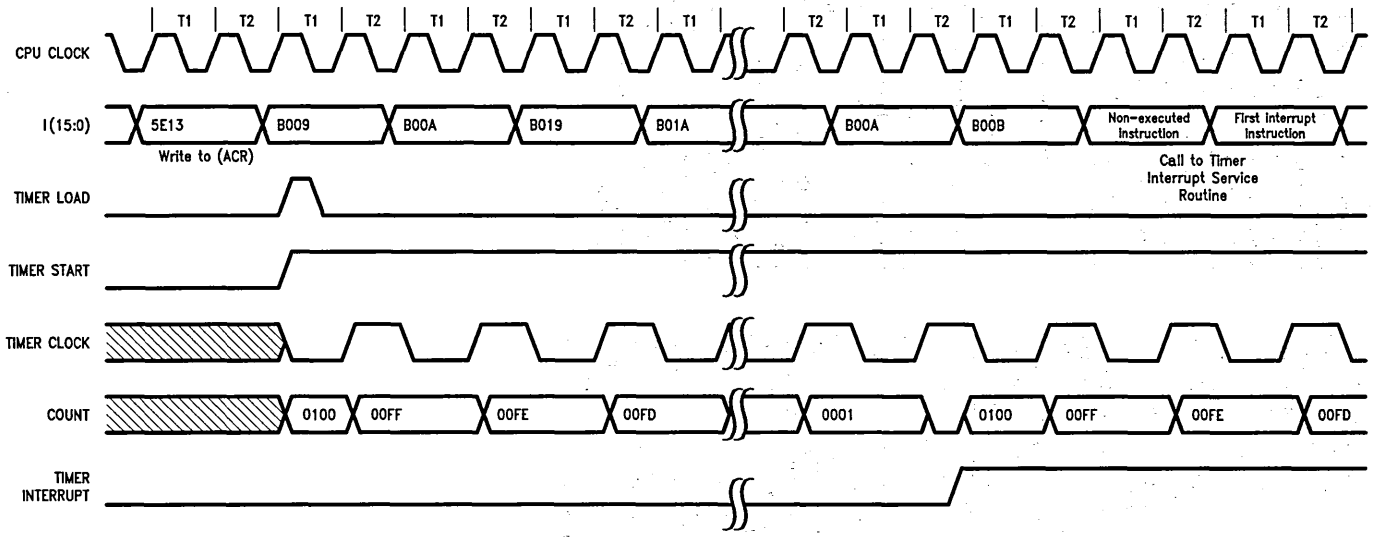
T = The period of the CPU clock

TD = The amount of time delay after the end of the instruction that asserts [TST] in {ACR}

When the value of 0000h is loaded in the timer, the maximum time-out is obtained and is calculated as follows:

$$TD = 65536 * T * k$$

With the CPU running full speed with an 18.8 MHz crystal, the maximum single loop time delay attainable would be 55.6 ms ([TCS] = 0). The minimum time delay with the same constraints is 106 ns ([TCS] = 1). For accumulating time-out intervals, the total time delay is simply the number of loops accumulated multiplied by the calculated time delay. The equations above do not account for any overhead for processing the timer interrupt. The added overhead of processing the interrupt may need to be included for precision timing.



TL/F/9336–D2

FIGURE 2-3. Timer Interrupt Diagram

2.0 CPU Description (Continued)

2.1.3 Instruction Set

The followng paragraphs introduce the BCP's architecture by discussing addressing modes and briefly discussing the Instruction Set. For detailed explanations and examples of each instruction, refer to the Instruction Set Reference Section.

2.1.3.1 Harvard Architecture Implications

The BCP utilizes a true Harvard Architecture, where the instruction and data memory are organized into two independent memory banks, each with their own address and data buses. Both the Instruction Address Bus and the Instruction Bus are 16 bits wide with the Instruction Address Bus addressing memory by words. (A word of memory is 16 bits long; i.e., 1 word = 2 bytes.) Most of the instructions are one word long. The exceptions are two words long, containing a word of instruction followed by a word of immediate data. The combination of word sized instructions and a word based instruction address bus eliminates the typical instruction alignment problems faced by many CPU's.

The Data Address Bus is 16 bits wide (with the low order 8 bits multiplexed on the Data Bus), and the Data Bus is 8 bits wide (i.e., one byte wide). The Data Address Bus addresses memory by bytes. Most of the BCP's instructions operate on byte-sized operands.

Note that although both instruction addresses and data addresses are 16 bits long, these addresses are for two different buses and, therefore, have two different numerical meanings, (i.e., byte address or word address.) Each instruction determines whether the meaning of a 16-bit address is that of an instruction word address or a data byte address. Little confusion exists though because only the program flow instructions interpret 16-bit addresses as instruction addresses.

2.1.3.2 Addressing Modes

An addressing mode is the mechanism by which an instruction accesses its operand(s). The BCP's architecture supports five basic addressing modes: register, immediate, indexed, immediate-relative, and register-relative. The first two allow instructions to execute the fastest because they require no memory access beyond instruction fetch. The remaining three addressing modes point to data or instruction memory. Typical of a RISC processor, most of the instructions only support the first three addressing modes, with one of the operands always limited to the register addressing mode.

Register Addressing Modes

There are two terminologies for the register addressing modes: Register and Limited Register. Instructions that allow Register operands can access all the registers in the CPU. Note that only 32 of the 44 CPU registers are available at any given point in time because the lower 12 register locations (R0–R11) access one of two switchable register banks each. (See Section 2.1.1.1, Banked Registers for more information on the CPU register banks.) Instructions that allow the Limited Register operands can access just the first 28 registers of the CPU. Again, note that only 16 of these 28 registers are available at any given point in time. Table 2-1 shows the notations used for the Register and Limited Register operands. Some instructions also imply the use of certain registers, for example the accumulators. This is noted in the discussions of those instructions.

Immediate Addressing Modes

The two types of the immediate addressing modes available are: Immediate numbers and Absolute numbers. Immediate numbers are 8 bits of data, (one data byte), that code directly into the instruction word. Immediate numbers may represent data, data address displacements, or relative instruction addresses. Absolute numbers are 16-bit numbers. They code into the second word of two word instructions and they represent absolute instruction addresses. Table 2-2 shows the notations used for both of these addressing modes.

TABLE 2-1. Register Addressing Mode Notations

Notation	Type of Register Operand	Registers Allowed
Rs	Source Register	R0–R31
Rd	Destination Register	R0–R31
Rsd	Register is both a Source & Destination	R0–R31
rs	Limited Source Register	R0–R15
rd	Limited Destination Register	R0–R15
rsd	Limited Register is both a Source & Destination	R0–R15

TABLE 2-2. Immediate Addressing Mode Notations

Notation	Type of Immediate Operand	Size
n	Immediate Number	8 Bits
nn	Absolute Number	16 Bits

2.0 CPU Description (Continued)

Indexed Addressing Modes

Indexed operands involve one of four possible CPU register pairs referred to as the index registers. *Figure 2-4* illustrates how the index registers map into the CPU Register Set. Note that the index registers are 16 bits wide.

Index registers allow for indirect memory addressing and usually contain data memory addresses, although, the LJMP instruction can use index registers to hold instruction memory addresses. Most of the instructions that allow memory indirect addressing, (i.e. the use of index registers), also allow pre-incrementing, post-incrementing, or post-decrementing of the index register contents during instruction execution, if desired. Table 2-3 lists the notations used for the index register modes.

The index registers are set to zero when the BCP's RESET pin is asserted.

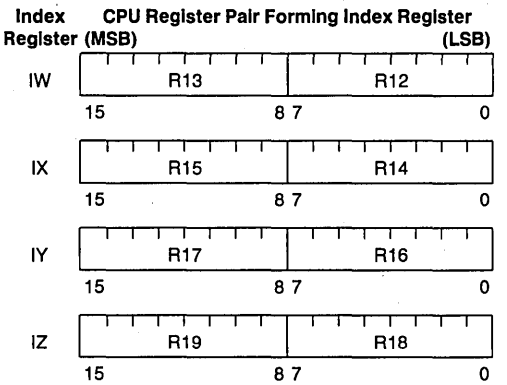


FIGURE 2-4. Index Register Map

Immediate-Relative and Register-Relative Address Modes

The Immediate-Relative mode adds an unsigned 8-bit immediate number to the index register IZ forming a data byte address. The Register-Relative mode adds the unsigned 8-bit value in the current accumulator, A, to any one of the index registers forming a data byte address. Both of these indirect memory addressing modes are available only on the MOVE instruction. Table 2-4 shows the notation used for these two addressing modes.

2.1.3.3 Instruction Set Overview

The BCP's RISC instruction set contains seven categories of instructions: Data Movement, Integer Arithmetic, Logic, Shift-Rotate, Comparison, Program Flow, and Miscellaneous.

Data Movement Instructions

The MOVE instruction is responsible for all the data transfer operations that the BCP can perform. Moving one byte at a time, five different types of transfer are allowed: register to register, data memory to register, register to data memory, instruction memory to register, and instruction memory to data memory. Table 2-5 lists all the variations of the MOVE instruction.

TABLE 2-3. Index Register Addressing Mode Notations

Notation	Meaning
[Ir]	Index Register, Contents Not Changed
[Ir−]	Index Register, Contents Post-Decremented
[Ir+]	Index Register, Contents Post-Incremented
[+Ir]	Index Register, Contents Pre-Incremented
[mIr]	General Notation Indicating that Any of the Above Modes Is Allowed

Note: [ ] denotes indirect memory addressing and is part of the instruction syntax.

TABLE 2-4. Relative Index Register Mode Notations

Notation	Type of Action Performed to Calculate a Data Memory Address
[IZ + n]	IZ + Immediate Number (unsigned) → Data Memory Address
[Ir + A]	Index Register + Current Accumulator (unsigned) → Data Memory Address

Note: [ ] denotes indirect memory addressing and is part of the instruction syntax.

TABLE 2-5. Data Movement Instructions

Syntax	Instruction Operation	Addressing Modes
MOVE Rs, Rd	register → register	Register, Register
MOVE Rs, [mIr]	register → data memory	Register, Indexed
MOVE [mIr], Rd	data memory → register	Indexed, Register
MOVE Rs, [Ir + A]	register → data memory	Register, Register-Relative
MOVE [Ir + A], Rd	data memory → register	Register-Relative, Register
MOVE rs, [IZ + n]	register → data memory	Limited Register, Immediate-Relative
MOVE [IZ + n], rd	data memory → register	Immediate-Relative, Limited Register
MOVE n, rd	instruction memory → register	Immediate, Limited Register
MOVE n, [Ir]	instruction memory → data memory	Immediate, Indexed

2.0 CPU Description (Continued)

Integer Arithmetic Instructions

The integer arithmetic instructions operate on 8-bit signed (two's complement) binary numbers. Two arithmetic functions are supported: Add and Subtract. Three versions of the Add and Subtract instructions exist: operand ± accumulator, operand ± accumulator ± carry, and immediate operand ± operand. The first two versions support both the register and indexed addressing modes for the destination operand. These two versions also allow the specification of a separate register or data address for the destination operand so that the sources may retain their integrity; (i.e., true three-operand instructions). Note that the currently active "B" register bank selects which accumulator is used in these instructions. The third version, immediate operand ± operand, only supports the register addressing mode for the destination operand with the register as both a source and the destination. Table 2-6 lists the integer arithmetic instructions along with their variations.

Logic Instructions

The logic instructions operate on 8-bit binary data. A full set of logic functions is supported by the BCP: AND, OR, eXclusive OR, and Complement. All the logic functions except complement allow either an immediate operand or the currently active accumulator as an implied operand. Complement only allows one register operand which is both the source and destination. The other logic instructions include the following addressing modes: register, indexed, and immediate. As with the integer arithmetic instructions, the integrity of the sources may be maintained by specifying a destination register which is different from the source. Table 2-7 lists all the logic instructions.

TABLE 2-6. Integer Arithmetic Instructions

Syntax		Instruction Operation	Addressing Modes
ADD	n, rsd	register + n → register	Immediate, Limited Register
ADDA	Rs, Rd	Rs + accumulator → Rd	Register, Register
ADDA	Rs, [mlr]	Rs + accumulator → data memory	Register, Indexed
ADCA	Rs, Rd	Rs + accumulator + carry → Rd	Register, Register
ADCA	Rs, [mlr]	Rs + accumulator + carry → data memory	Register, Indexed
SUB	n, rsd	register − n → register	Immediate, Limited Register
SUBA	Rs, Rd	Rs − accumulator → Rd	Register, Register
SUBA	Rs, [mlr]	Rs − accumulator → data memory	Register, Indexed
SBCA	Rs, Rd	Rs − accumulator − carry → Rd	Register, Register
SBCA	Rs, [mlr]	Rs − accumulator − carry → data memory	Register, Indexed

TABLE 2-7. Logic Instructions

Syntax		Instruction Operation	Addressing Modes
AND	n, rsd	register & n → register	Immediate, Limited Register
ANDA	Rs, Rd	Rs & accumulator → Rd	Register, Register
ANDA	Rs, [mlr]	Rs & accumulator → data memory	Register, Indexed
OR	n, rsd	register | n → register	Immediate, Limited Register
ORA	Rs, Rd	Rs | accumulator → Rd	Register, Register
ORA	Rs, [mlr]	Rs | accumulator → data memory	Register, Indexed
XOR	n, rsd	register ⊕ n → register	Immediate, Limited Register
XORA	Rs, Rd	Rs ⊕ accumulator → Rd	Register, Register
XORA	Rs, [mlr]	Rs ⊕ accumulator → data memory	Register, Indexed
CPL	Rsd	$\overline{\text{register}}$ → register	Register

Note: & = logical AND operation
| = logical OR operation
⊕ = logical exclusive OR operation
$\bar{r}$ = one's complement

2.0 CPU Description (Continued)

Shift and Rotate Instructions

The shift and rotate instructions operate on any of the 8-bit CPU registers. The BCP supports shift left, shift right, and rotate operations. Table 2-8 lists the shift and rotate instructions.

Comparison Instructions

The BCP utilizes two comparison instructions. The CMP instruction performs a two's complement subtraction between a register and immediate data. The BIT instruction tests selected bits in a register by ANDing it with immediate data. Neither instruction stores its results, only the ALU flags are affected. Table 2-9 lists both of the comparison instructions.

Program Flow Instructions

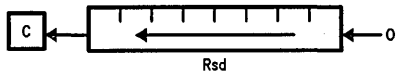
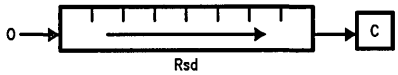
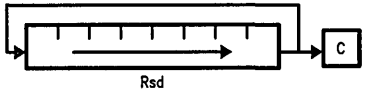
The BCP has a wide array of program flow instructions: unconditional jumps, calls and returns; conditional jumps, calls, and returns; relative or absolute instruction addressing on jumps and calls; a specialized register field decoding jump; and software interrupt capabilities. These instructions redirect program flow by changing the Program Counter.

The unconditional jump instructions support both relative instruction addressing, the (JuMP instruction), and absolute instruction addressing, (the Long JuMP instruction), using the following addressing modes: Immediate, Register, Absolute, and Indexed. Table 2-10 lists the unconditional jump instructions and their variations.

The conditional jump instructions support both relative instruction addressing and absolute instruction addressing using the Immediate and Absolute addressing modes. The conditional relative jump instruction tests flags in the Condition Code Register, {CCR}, and the Transceiver Status Register, {TSR}. Two possible syntaxes are supported for the conditional relative jump instruction; see Table 2-11.

Table 2-12 lists the various flags "f" that the conditional JMP instruction can test and Table 2-13 lists the various conditions "**cc**" that the **Jcc** instruction can test for. Keep in

TABLE 2-8. Shift and Rotate Instructions

Syntax	Instruction Operation	Addressing Mode
SHL Rsd,b		Register
SHR Rsd,b		Register
ROT Rsd,b		Register

Note: "b" = the number of bit shifts/rotates to perform.

TABLE 2-9. Comparison Instructions

Syntax	Instruction Operation	Addressing Mode
CMP rs, n	register − n	Limited Register
BIT rs, n	register & n	Limited Register

Note: & = logical AND operation

TABLE 2-10. Unconditional Jump Instructions

Syntax	Instruction Operation	Operand Range	Addressing Mode
JMP n	PC + n (sign extended) → PC	−128, +127	Immediate
JMP Rs	PC + Rs (sign extended) → PC	−128, +127	Register
LJMP nn	nn → PC	0, 64k	Absolute
LJMP [Ir]	Ir → PC	0, 64k	Indexed

Note: PC = Program Counter; contents initially points to instruction following jump.

2.0 CPU Description (Continued)

mind that the Jcc instruction is just an optional syntax for the conditional JMP instruction.

The example in *Figure 2-5* demonstrates two possible ways to code the conditional relative jump instruction when testing for a false [Z] flag in {CCR}. In the example, assume that the symbol "Z" equals "000" binary, that the symbol "NS" equals "0" binary, and that the symbol "SKIP.IT" points to the desired instruction with which to begin execution if [Z] is false.

On the other hand, the conditional absolute jump instruction, LJMP, can test any bit in any currently active CPU register. Table 2-14 shows the conditional long jump instruction syntax.

```
JMP  Z,NS,SKIP.IT   ;If [Z]=0 goto SKIP.IT
     -or-
JNZ  SKIP.IT        ;If [Z]=0 goto SKIP.IT
```

FIGURE 2-5. Coding Examples of Equivalent Conditional Jump Instructions

TABLE 2-11. Conditional Relative Jump Instruction

Syntax	Instruction Operation	Operand Range	Addressing Mode
JMP f,s,n	If the flag "f" is in the state "s" then PC + n (sign extended) → PC	−128, +127	Immediate
Jcc n	If the condition "cc" is met then PC + n (sign extended) → PC	−128, +127	Immediate

Note: PC = Program Counter; contents initially points to instruction following jump.

TABLE 2-12. "f" Flags

"f"(Binary)	Flag	Flag Name	Register Containing Flag
000	Z	Zero	{CCR}
001	C	Carry	{CCR}
010	V	Overflow	{CCR}
011	N	Negative	{CCR}
100	RA	Receiver Active	{TSR}
101	RE	Receiver Error	{TSR}
110	DAV	Data Available	{TSR}
111	TFF	Transmitter FIFO Full	{TSR}

TABLE 2-13. "cc" Conditions Tested

"cc" Field	Condition Tested for	Flag "f"'s Condition
Z	Zero	[Z] = 1
NZ	Not Zero	[Z] = 0
EQ	Equal	[Z] = 1
NEQ	Not Equal	[Z] = 0
C	Carry	[C] = 1
NC	No Carry	[C] = 0
V	Overflow	[V] = 1
NV	No Overflow	[V] = 0
N	Negative	[N] = 1
P	Positive	[N] = 0
RA	Receiver Active	[RA] = 1
NRA	Not Receiver Active	[RA] = 0
RE	Receiver Error	[RE] = 1
NRE	No Receiver Error	[RE] = 0
DA	Data Available	[DAV] = 1
NDA	No Data Available	[DAV] = 0
TFF	Transmitter FIFO FULL	[TFF] = 1
NTFF	Transmitter FIFO Not Full	[TFF] = 0

TABLE 2-14. Conditional Absolute Jump Instruction

Syntax	Instruction Operation	Operand Range	Addressing Mode
LJMP Rs,p,s,nn	If the bit of register "Rs" in position "p" is in the state "s" then nn → PC	0, 64k	Register, Absolute

Note: PC = Program Counter

2.0 CPU Description (Continued)

The BCP also has a specialized relative jump instruction called relative Jump with Rotate and Mask on source register, JRMK. This instruction facilitates the decoding of register fields often involved in communications processing. JRMK does this by rotating and masking a copy of its register operand to form a signed program counter displacement which usually points into a jump table. Table 2-15 shows the syntax and operation of the JRMK instruction.

JRMK's masking, (setting to zero), the least significant bit of the displacement allows the construction of a jump table using either one or two word instructions; for instance, a table of JMP and/or LJMP instructions, respectively. The example in *Figure 2-6* demonstrates the JRMK instruction decoding the address frame of the 3299 Terminal Multiplexer protocol which is located in the Receive/Transmit Register, {RTR[4–2]}.

The BCP has two unconditional call instructions; CALL, which supports relative instruction addressing and LCALL, (Long CALL), which supports absolute instruction addressing. These instructions push the following information onto the CPU's internal Address Stack: the address of the next instruction; the status of the Global Interrupt Enable flag, [GIE]; the status of the ALU flags [Z], [C], [N], and [V]; and the status of which register banks are currently active. Table 2-16 lists the two unconditional call instructions. Note that the Address Stack is only twelve positions deep; therefore, the BCP allows twelve levels of nested subroutine invocations, (this includes both interrupts and calls).

TABLE 2-15. JRMK Instruction

Syntax	Instruction Operation	Displacement Range	Addressing Mode
JRMK Rs, b, m	(a) Rotate a copy of register "Rs" "b" bits to the right. (b) Mask the most significant "m" bits and the least significant bit of the above result. (c) PC + resulting displacement (sign extended) → PC.	−128, +126	Register

Note: PC = Program Counter; contents initially points to instruction following jump.

Example Code

```
JRMK  RTR,1,4    ;decode terminal address
LJMP  ADDR.0     ;jump to device handler #0
LJMP  ADDR.1     ;jump to device handler #1
. . .
LJMP  ADDR.7     ;jump to device handler #7
```

Instruction Execution	JRMK Displacement Register Contents							
(a) Copy {RTR} into JRMK's displacement register:	x	x	x	A2	A1	A0	y	y
(b) Rotate displacement register 1 bit to the right:	y	x	x	x	A2	A1	A0	y
(c) AND result with "00001110" binary mask:	0	0	0	0	A2	A1	A0	0
(d) Sign extend resulting displacement and add it to the program counter, (PC). If the bits A2 A1 A0 equal "0 0 1" binary then + 2 is added to the Program Counter; (i.e., PC + 2 → PC).	0	0	0	0	0	0	1	0

(e) Execute the instruction pointed to by the PC, which in this example is:
LJMP ADDR.1

FIGURE 2-6. JRMK Instruction Example

TABLE 2-16. Unconditional Call Instructions

Syntax	Instruction Operation	Operand Range	Addressing Mode
CALL n	PC & [GIE] & ALU flags & reg. bank selection → Address Stack PC + n (sign extended) → PC	−128, +127	Immediate
LCALL nn	PC & [GIE] & ALU flags & reg. bank selection → Address Stack nn → PC	0, 64k	Absolute

Note: PC = Program Counter; contents initially points to instruction following call.
[GIE] = Global Interrupt Enable bit.
& = concatenation operator, combines operands together forming one long operand.

2.0 CPU Description (Continued)

The BCP has one conditional call instruction capable of testing any bit in any currently active CPU register. This call only supports absolute instruction addressing. Table 2-17 shows the conditional call instruction syntax and operation.

The return instruction complements the above call instructions. Two versions of the return instruction exist, the uncondtional return and the conditional return. When the unconditional return instruction is executed, it pops the last address on the CPU's Address Stack into the program counter and it can optionally affect the [GIE] bit, the ALU flags, and the register bank selection. Table 2-18 shows the syntax and operation of the unconditional return instruction.

The conditional return instruction functions the same as the unconditional return instruction if a desired condition is met. As with the conditional jump instruction, the conditional return instruction has two possible syntaxes. Table 2-19 lists the syntax for the conditional return. The "f" flags and the "**cc**" conditions for the return instruction are the same as for the conditional jump instruction, therefore refer to Table 2-12 and Table 2-13 for the listing of "**f**" and "**cc**", respectively.

TABLE 2-17. Conditional Call Instruction

Syntax	Instruction Operation	Operand Range	Addressing Mode

LCALL Rs, p, s, nn	If the bit of register "Rs" in position "p" is in the state "s" then
	PC & [GIE] & ALU flags &
	reg. bank selection → Address Stack
	nn → PC
	End if
		0, 64k	Register, Absolute

Note: PC = Program Counter; contents initially points to instruction following call.
[GIE] = Global Interrupt Enable bit
& = concatenation operator, combines operands together forming one long operand.

TABLE 2-18. Unconditional Return Instruction

Syntax	Instruction Operation

RET {g {, rf}}	Case "g" of
	0: leave [GIE] unaffected, (default)
	1: restore [GIE] from Address Stack
	2: set [GIE]
	3: clear [GIE]
	End case
	If "rf" = 1 then
	restore ALU flags from Address Stack
	restore register bank selection from Address Stack
	Else (the default)
	leave the ALU flags and register bank selections unchanged
	End if
	Address Stack → PC

Note: PC = Program Counter
[GIE] = Global Interrupt Enable bit
{} = surrounds optional operands that are not part of the instruction syntax.
Optional operands may either be specified or omitted.

TABLE 2-19. Conditional Return Instruction

Syntax	Instruction Operand
RETF f, s {, {g}, {, rf}}	If the flag "f" is in the state "s" then perform a RET {g {, rf}}
Rcc {g {, rf}}	If the condition "**cc**" is met then perform a RET {g {,rf}}

Note: See Table XVIII for an explanation of "RET {g {, rf}}"
{} = surrounds optional operands that are not part of the instruction syntax.
Optional operands may either be specified or omitted.

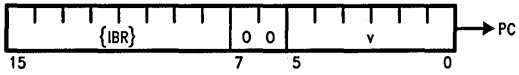
2.0 CPU Description (Continued)

In addition to the above jump, call and return program flow instructions, the BCP is capable of generating software interrupts via the TRAP instruction. This instruction generates a call to any one of 64 possible interrupt table addresses based on its vector number operand. This allows both the simulation of hardware interrupts and the construction of special software interrupts, if desired. The actual interrupt table entry address is determined by concatenating the Interrupt Base Register, {IBR}, to an 8-bit representation of the vector number operand in the TRAP instruction. This instruction may also clear the [GIE] bit, if desired. Table 2-20 shows the syntax and operation of the TRAP instruction.

Miscellaneous Instructions

As stated in the "CPU Register Set" section, the BCP has 44 registers with 24 of them arranged into four register banks: Main Bank A, Alternate Bank A, Main Bank B, and Alternate Bank B. The exchange instruction, EXX, selects which register banks are currently available to the CPU, for example either Main Bank A or Alternate Bank A. The deselected register banks retain their current values. The EXX instruction can also alter the state of [GIE], if desired. Table 2-21 shows the EXX instruction syntax and operation.

TABLE 2-20. TRAP Instruction

Syntax	Instruction Operation	Operand Range
TRAP v {, g'}	PC & [GIE] & ALU flags & reg. Bank Selection → Address Stack If "g'" = 1 then clear [GIE] Form PC address as shown below: 	0, 63

Note: PC = Program Counter; contents initially points to instruction following call.
[GIE] = Global Interrupt Enable bit
IBR = Interrupt Base Register
& = concatenation operator, combines operands together forming one long operand.
{ } = surrounds optional operands that are not part of the instruction syntax.
Optional operands may either be specified or omitted.

TABLE 2-21. EXX Instruction

Syntax	Instruction Operation
EXX ba, bb {, g}	Case "ba" of 0: activate Main Bank A 1: activate Alternate Bank A End case Case "bb" of 0: activate Main Bank B 1: activate Alternate Bank B End case Case "g" of 0: leave [GIE] unaffected, (default) 1: (reserved) 2: set [GIE] 3: clear [GIE] End case

Note: [GIE] = Global Interrupt Enable bit
{ } = surrounds optional operands that are not part of the instruction syntax.
Optional operands may either be specified or omitted.

2.0 CPU Description (Continued)

2.2 CPU FUNCTIONAL DESCRIPTION

2.2.1 ALU

The BCP provides a full function high speed 8-bit Arithmetic Logic Unit (ALU) with full carry look ahead, signed arithmetic, and overflow decision capabilities. The ALU can perform six arithmetic, nine logic, one rotate and two shift operations on binary data. Full access is provided to all CPU registers as both source and destination operands, and using the indirect addressing mode, results may be placed directly into data memory. All operations which have an internal destination (register addressing) are completed in two (2) T-states. External destination operations (indirect addressing to data memory) complete in three (3) T-states.

Arithmetic operations include addition with or without carry, and subtraction with or without borrow (represented by carry). Subtractions are performed using 2's complement addition to accommodate signed operands. The subtrahend is converted to its 2's complement equivalent by the ALU and then added to the minuend. The result is left in 2's complement form.

The remaining ALU operations include full logic, shift and rotate operations. The logic functions include Complement, AND, OR, Exclusive-OR, Compare and Bit Test. Zero through seven bit right and left shift operations are provided, along with a zero through seven bit right rotate operation. Note that the shift and rotate operations may only be performed on a register, which is both the source and destination. (See the Instruction Set Overview section for detailed descriptions of these operations.)

The BCP ALU provides the programmer with four instruction result status bits for conditional operations. These bits (known as condition code flags) indicate the status (or condition) of the destination byte produced by certain instructions. Not all instructions have an affect on every status flag. (See the Instruction Set Reference section for the specific details on what status flags a given instruction affects.) These flags are held in the Condition Code Register, {CCR}, see *Figure 2-7.*

7	6	5	4	3	2	1	0
TO	RR	RW	BIRQ	N	V	C	Z

where:

N = Negative

C = Carry

V = Overflow

Z = Zero

FIGURE 2-7. Condition Code Register ALU Flags

If an instruction is documented as affecting a given flag, then the flags are set (to 1) or cleared (to 0) under the following conditions:

[N]— The Negative flag is set if the most significant bit (MSB) of the result is one (1), otherwise it is cleared. This flag represents the sign of the result if it is interpreted as a 2's complement number.

[C] — The Carry flag is set if:

a) An addition operation generates a carry, see *Figure 2-8a.*

b) A subtract or compare operation generates a borrow, see *Figure 2-8b.*

c) The last bit shifted out during a shift operation (in either direction) is a one (1), see *Figure 2-9.*

d) The last bit rotated by the rotate operation is a one (1), see *Figure 2-10.*

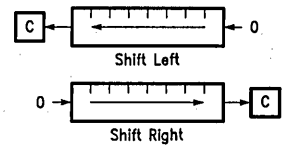
In all other conditions [C] is cleared.

[V]— Overflow is set whenever the result of an arithmetic or compare operation on signed operands is not representable by the operand size, thereby producing an incorrect result. For example, the addition of the two signed negative numbers in *Figure 2-8a* would set [V] since the correct representation of the result, both sign and magnitude, is not possible in 8 bits. On the other hand, in *Figure 2-8b* and *2-8c* [V] would be cleared because the results are correctly represented in both sign and magnitude. It is important to remember that Overflow is only meaningful in signed arithmetic and that it is the programmer's responsibility to determine if a given operation involves signed or unsigned values.

[Z]— The Zero flag is set only when an operation produces an all bits cleared result (i.e., a zero). In all other conditions [Z] is cleared.

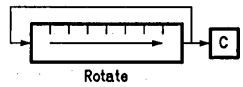
(a)	(b)	(c)
11101010	10111010	11011100
+ 10001100	− 11000100	+ 01100011
1 ← 01110110	1 → 11110110	1 ← 00111111
[C] = 1	[C] = 1	[C] = 1
[V] = 1	[V] = 0	[V] = 0

FIGURE 2.8. Carry and Overflow Calculations



TL/F/9336–D3

FIGURE 2-9. Shifts' Effect on Carry



TL/F/9336–D4

FIGURE 2-10. Rotate's Effect on Carry

2.0 CPU Description (Continued)

Several conditions apply to these flags, independent of their operation and the way they are calculated. These conditions are:

1. A flag's previous state is retained when an instruction has no affect on that flag.
2. Direct reading and writing of all ALU flags is possible via the {CCR} register.
3. Currrent flag values are saved onto the address stack during interrupt and call operations, and can be restored to their original values if a return instruction with the restore flags option is executed.
4. Flag status is calculated in parallel with the instruction result, therefore no time penalty is associated with flag operation.

When performing single byte arithmetic (i.e., the values are completely represented in one byte) the Add (ADD,ADDA) and Subtract (SUB,SUBA) instructions should be used, but when performing multi-byte arithmetic the Add with Carry (ADCA) and Subtract with Carry (SBCA) instructions should be used. This is because the carry (in an add operation) or the borrow (in a subtract operation) must be carried forward to the higher order bytes. *Figure 2-11* demonstrates an instruction sequence for a 16-bit add and an instruction sequence for a 16-bit subtract.

Assume the 16-bit variable X is represented by the register pair R4(MSB), R5(LSB), and that the 16-bit variable Y is represented by the register pair R6(MSB), R7(LSB).

To perform the assignment Y = X + Y:

```
MOVE   R7,A    ;GET LSB OF Y
ADDA   R5,R7   ;Y(LSB)=X(LSB)+Y(LSB)
MOVE   R6,A    ;GET MSB OF Y
ADCA   R4,R6   ;Y(MSB)=X(MSB)+Y(MSB)
                +CARRY
```

To perform the assignment Y = X – Y:

```
MOVE   R7,A    ;GET LSB OF Y
SUBA   R5,R7   ;Y(LSB)=X(LSB)-Y(LSB)
MOVE   R6,A    ;GET MSB OF Y
SBCA   R4,R6   ;Y(MSB)=X(MSB)-Y(MSB)
                -CARRY
```

FIGURE 2-11. Multi-Byte Arithmetic Instruction Sequences

When using the ALU to perform comparisons, the programmer has two options. If the compare is to a constant value then the CMP instruction can be used, else one of the subtract instructions must be used. When determining the results of any compare, the programmer must keep in mind whether they are comparing signed or unsigned values. Table 2-22 lists the Boolean condition that must be met for unsigned comparisons and Table 2-23 lists the Boolean condition that must be met for signed comparisons.

TABLE 2-22

Unsigned Comparison Results	
Comparison: x – y	**Boolean Condition**
$x < y$	C
$x \le y$	$C \mid Z$
$x = y$	Z
$x \ge y$	$\overline{C}$
$x > y$	$\overline{C} \& \overline{Z}$

Note: & = logical AND
| = logical OR
$\overline{z}$ = one's complement

TABLE 2-23

Signed Comparison Results	
Comparison: x – y	**Boolean Condition**
$x < y$	$(N\&\overline{V}) \mid (\overline{N}\&V)$
$x \le y$	$Z \mid (N\&\overline{V}) \mid (\overline{N}\&V)$
$x = y$	Z
$X \ge y$	$(N\&V) \mid (\overline{N}\&\overline{V})$
$x > y$	$(N\&V\&\overline{Z}) \mid (\overline{N}\&\overline{V}\&\overline{Z})$

Note: & = logical AND
| = logical OR
$\overline{z}$ = one's complement

2.2.2 Timing

Timing on the BCP is controlled by an internal oscillator and circuitry that generates the internal timing signals. This circuitry in the CPU is referred to as Timing Control. The internal timing of the CPU is synchronized to an internal clock called the CPU clock, CPU-CLK. A period of CPU-CLK is referred to as a T-state. The clock for the BCP is provided by a crystal connected between X1 and X2 or from a clock source connected to X1. This clock will be referred to as the oscillator clock, OCLK. The frequency of OCLK is divided in half when the CPU clock select bit, [CCS], in the Device Control Register, {DCR}, is set to a one. Either OCLK or OCLK/2 is used by Timing Control to generate CPU-CLK and other synchronous signals used to control the CPU timing.

After the BCP is reset, [CCS] is high and CPU-CLK is generated from OCLK/2. Since the output of the divider that creates OCLK/2 can be high or low after reset, CPU-CLK can also be in a high or low state. Therefore, the exact number of clock cycles to the start of the first instruction cannot be determined. Automatic test equipment can synchronize to the BCP by asserting $\overline{RESET}$ as shown in *Figure 2-12.* The falling edge of $\overline{RESET}$ generates a clear signal which causes CPU-CLK to fall. The next rising edge of X1 removes the clear signal from CPU-CLK. The second rising edge of X1 will cause CPU-CLK to rise and the relationship between X1 and CPU-CLK can be determined from this point.

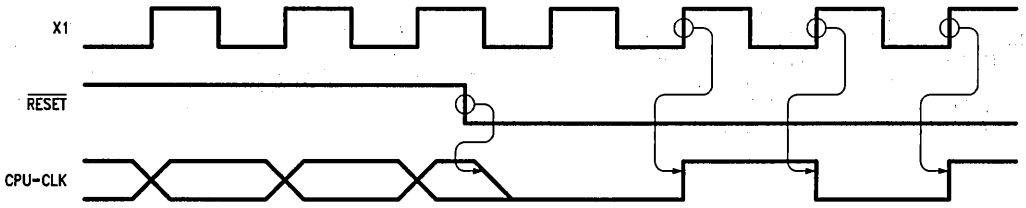
Writing a zero to [CCS] causes CPU-CLK to switch from OCLK/2 to OCLK. The transition from OCLK to OCLK/2 occurs following the end of the instruction that writes to

2.0 CPU Description (Continued)

[CCS] as shown in *Figure 2-13.* The switch occurs on the falling edge of X1 when CPU-CLK is low. CPU-CLK can be changed back to OCLK/2 by writing a one to [CCS]. The point at which CPU-CLK changes depends on whether there has been an odd or even number of T-states since [CCS] was set low. The change would require a maximum of two T-states and a minimum of one T-state following the end of the instruction that writes to [CCS].

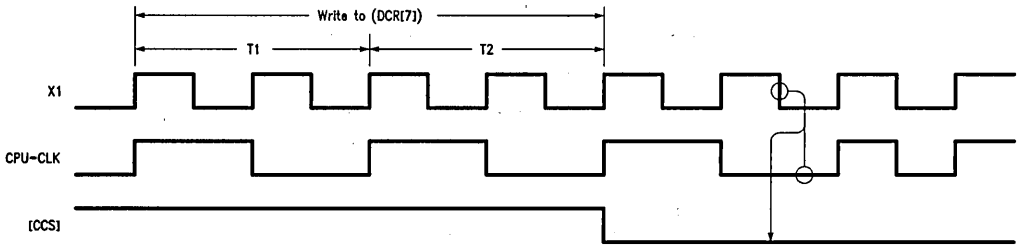
The CPU is a RISC processor with a limited number of instructions which execute in a short period of time. The maximum instruction cycle time is four T-states and the minimum is two T-states. Six types of instruction timing are used in the CPU: two T-state, three T-state program control, three T-state data memory access, four T-state read data memory access, four T-state program control, and four T-state two word program control. The first T-state of each instruction is T1 and the last T-state is T2. Intermediate T-states required to complete the instruction are referred to as TX.

The instruction clock output, ICLK, defines the instruction boundaries. ICLK rises at the beginning of each instruction and falls one-half T-state after the next address is generated on the instruction address bus, IA. Thus, ICLK indicates the start of each instruction and when the next instruction address is valid.



TL/F/9336-D5

FIGURE 2-12. CPU-CLK Synchronization with X1



TL/F/9336-D6

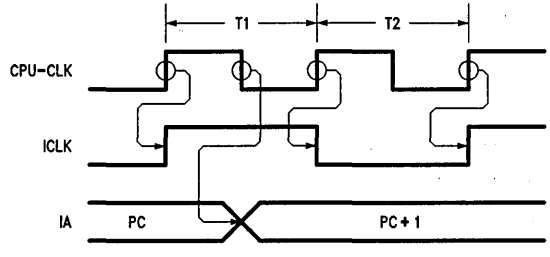
FIGURE 2-13. Changing from OCLK/2 to OCLK

2.0 CPU Description (Continued)

Figure 2-14 shows the relationship between CPU-CLK, ICLK, and IA for a two T-state instruction. The rising edge of CPU-CLK generates ICLK at the start of T1. The next falling edge of CPU-CLK increments the instruction address which appears on IA. ICLK falls one-half T-state later. The instruction completes during T2 which ends with ICLK rising, signifying the beginning of the next instruction.

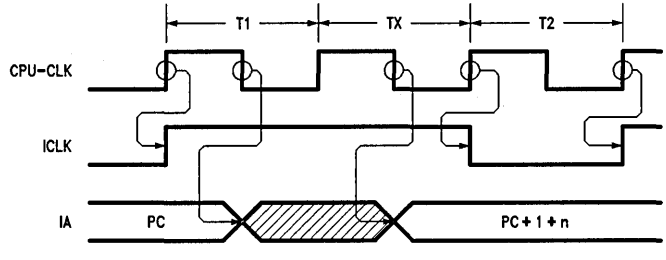
The three T-state program control instruction is similar and is shown in *Figure 2-15.* An additional T-state, TX, is added between T1 and T2. ICLK rises at the beginning of T1 as before but falls at the end of TX. The next instruction address is generated one-half T-state before the end of TX and the instruction ends with T2.

The three T-state data memory access instruction timing is shown in *Figure 2-16.* Again, TX is inserted between T1 and T2. ICLK rises at the beginning of the instruction and falls at the end of T1. The next instruction address appears on IA one-half clock cycle before ICLK falls. The address latch enable output, ALE, rises halfway through T1 and falls halfway through TX. The BCP has a 16-bit data memory address bus and an 8-bit data bus. The data bus is multiplexed with the lower 8 bits of the address bus and ALE is used to latch the lower 8 bits of the address during a data memory access. The upper 8 bits of the address become valid one-half T-state after the beginning of T1 and go invalid one-half T-state after the end of T2. The lower 8 bits of the address become valid on the address-data bus, AD, when ALE rises and goes invalid one-half T-state after ALE falls. *Figure 2-16* shows a write to data memory in which case AD switches from address to data at the beginning of T2. The data is held valid until one-half T-state after the end of T2. The write strobe, $\overline{\text{WRITE}}$, falls at the beginning of T2 and rises at the end of T2. A read of data memory is shown in *Figure 2-17.* The read timing is the same as a write except one-half T-state after ALE falls AD goes into a high impedance state allowing data to enter the BCP from data memory. AD returns to an active state at the end of T2. The read strobe, $\overline{\text{READ}}$, timing is identical to $\overline{\text{WRITE}}$.



TL/F/9336-D7

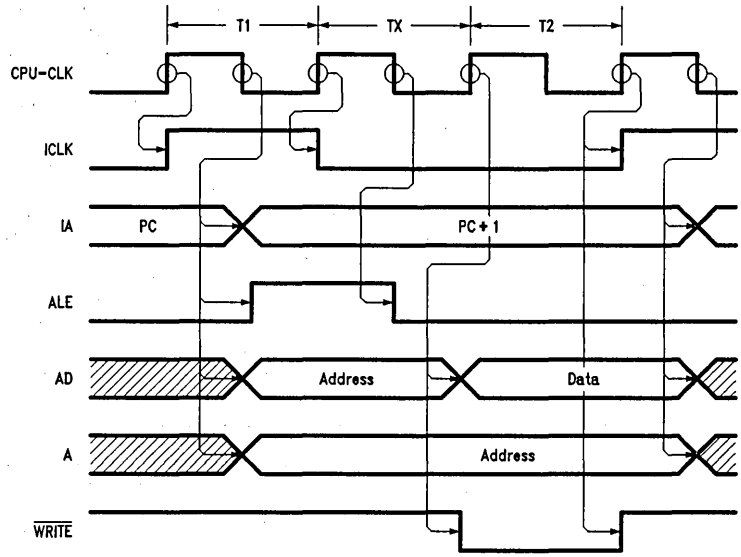
FIGURE 2-14. Two T-state Instruction



TL/F/9336-D8

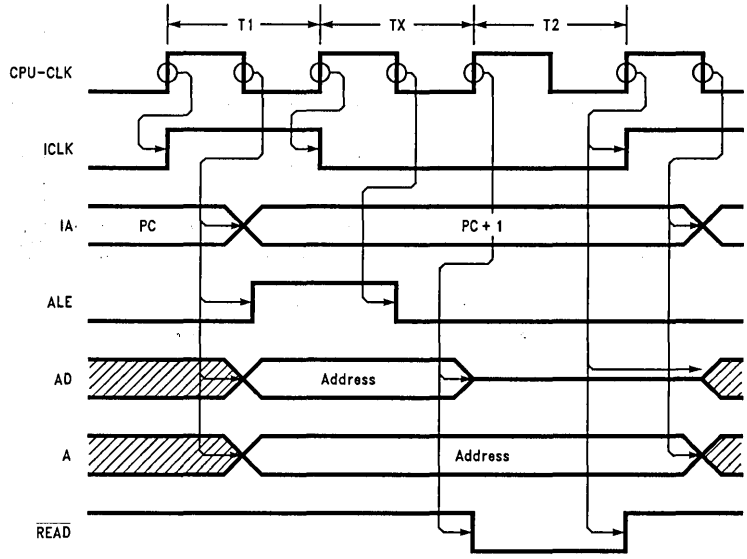
FIGURE 2-15. Three T-state Program Control Instruction

2.0 CPU Description (Continued)



TL/F/9336–D9

FIGURE 2-16. Three T-state Data Memory Write Instruction



TL/F/9336–E1

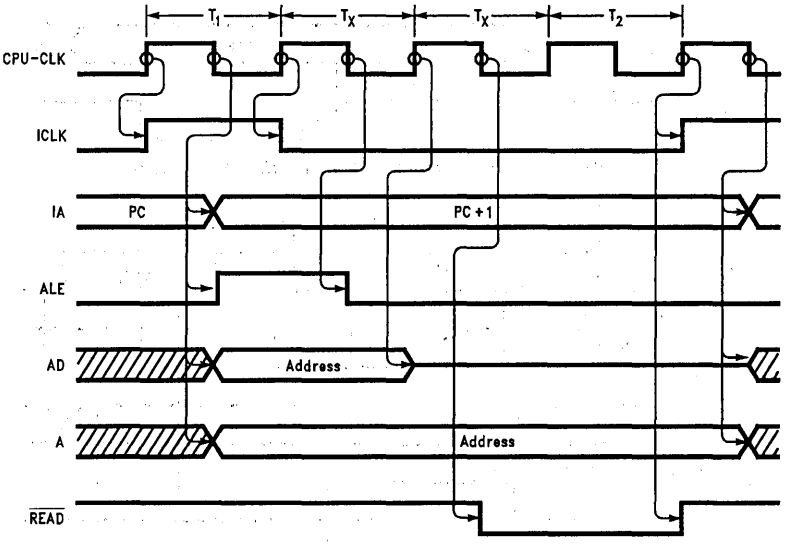
FIGURE 2-17. Three T-state Data Memory Read Instruction [4TR] = 0

2.0 CPU Description (Continued)

When the Four T-state Read mode is selected ([4TR] = 1), a second TX state is inserted before T2 and the timing of the read strobe, $\overline{READ}$, is changed such that $\overline{READ}$ falls one-half T-state after the beginning of the second TX. *Figure 2-18* shows a Four T-state Read of data memory. The extra half T-state before $\overline{READ}$ falls allows more time for the BCP to TRI-STATE the AD lines before the memory circuit begins driving those lines.

The four T-state program control instruction timing is shown in *Figure 2-19*. The instruction has two TX states inserted between T1 and T2. ICLK rises at the beginning of T1 and falls at the end of the second TX. The next instruction address becomes valid halfway through the second TX. The four T-state two word program control instruction timing is the same as two consecutive two T-state instructions and is shown in *Figure 2-20*.

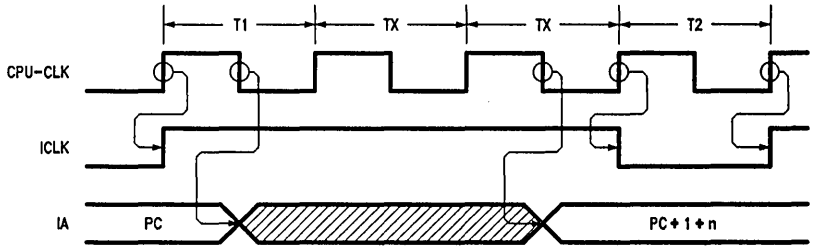
This timing describes the minimum cycle time required by each type of instruction. The BCP can be slowed down by changing the number of wait states selected in the Device Control Register, {DCR}. The BCP can be programmed for up to three instruction memory wait states (instruction wait states) and seven data memory wait states (data wait states). Instruction wait states affect all instruction types while data wait states affect only data memory access instructions. Bits three and four in {DCR} control the number of instruction wait states and bits zero, one and two are used to select the number of data wait states. The relationships between the control bits and the number of wait states selected are shown in Table 2-24 and Table 2-25. The BCP is configured with three instruction wait states and seven data wait states, and [4TR] set to zero after reset. A write to {DCR[4,3]} to change the number of instruction wait states takes effect on the following instruction if that instruction is a three T-state or four T-state program control instruction. For the other instruction types, the new number of instruction wait states will take effect on the instruction fol-



TL/F/9336–H5

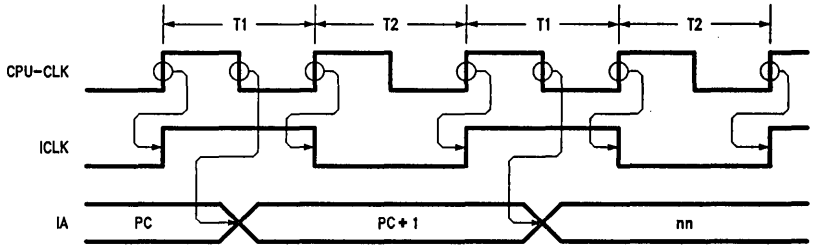
FIGURE 2-18. Four T-state Data Memory Read Instruction [4TR] = 1

2.0 CPU Description (Continued)



TL/F/9336-E2

FIGURE 2-19. Four T-state Program Control Instruction



TL/F/9336-E3

FIGURE 2-20. Four T-state Two Word Instruction

TABLE 2-24. Data Memory Wait States

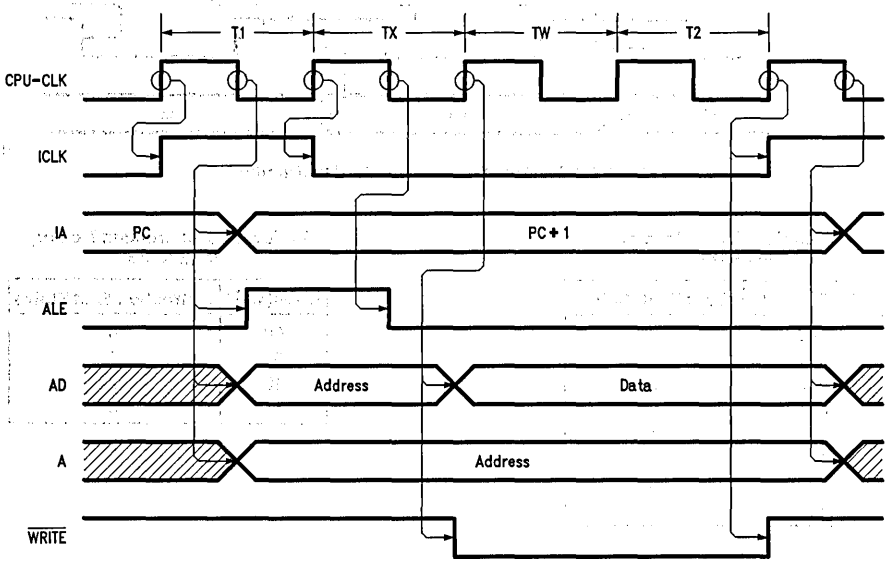
{DCR[2-0]}	Data Wait States
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

TABLE 2-25. Instruction Memory Wait States

{DCR[4,3]}	Instruction Wait States
00	0
01	1
10	2
11	3

2.0 CPU Description (Continued)

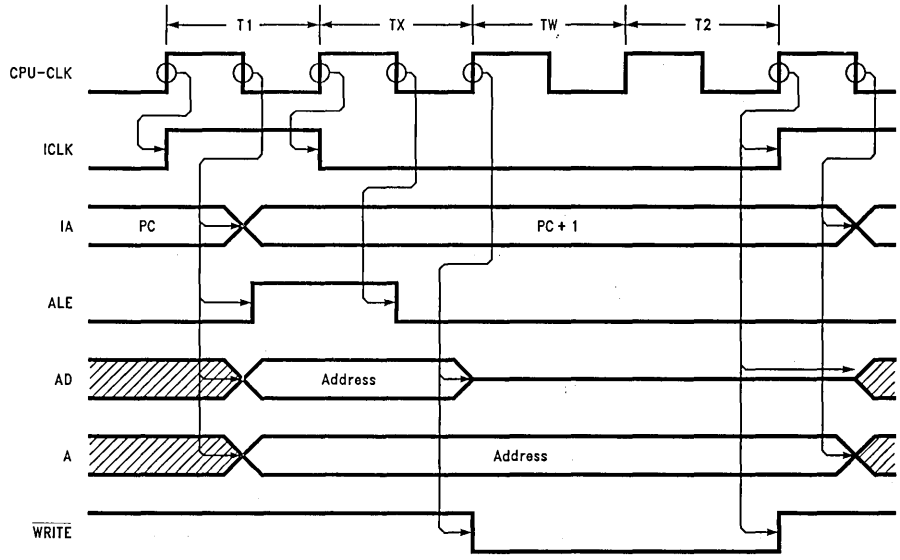
lowing the instruction after the write to {DCR}. A write to {DCR[2–0]} to change the number of data wait states will take effect on the next data memory access instruction even if it immediately follows the write to {DCR}. A write to {DCR [2–0]} to change the number of data wait states or to {ACR [4TR]} will take effect on the next data memory access instruction even if it immediately follows write to {DCR} or {ACR}. Both instruction and data wait states cause the insertion of additional T-states prior to T2 and these T-states are referred to as TW. The purpose of instruction wait states is to increase the time from instruction address generation to the beginning of the next instruction cycle. Data wait states increase the time from data memory address generation to the removal of the strobe at the end of data memory access instructions. Therefore, instruction and data wait states are counted concurrently in a data memory access instruction and TX of a data memory access instruction is counted as one instruction wait state. The actual number of wait states added to a data memory access is calculated as the maximum between the number of data wait states and one less than the number of instruction wait states. *Figure 2-21* shows a write of data memory with one wait state. This could be accomplished by selecting two instruction wait states or one data wait state. The effect of the wait state is to increase the time the write strobe is active and the data is valid on AD. The same situation for a read of data memory is shown in *Figure 2-22*. Note that if [4TR] is set to one then one data wait state has no additional affect on a read of data memory and the timing is the same as shown in *Figure 2-18*. The affect of two data memory wait states and [4TR] set to one is shown in *Figure 2-23*. A two T-state instruction with two instruction wait states is shown in *Figure 2-24* and a four T-state instruction with one instruction wait state is shown in *Figure 2-25*. As stated earlier, instruction wait states are inserted before T2. Adding wait states to a four T-state two word instruction causes the wait states to count twice when calculating total instruction cycle time. The wait states are added to each of the two words of the instruction.



TL/F/9336–E4

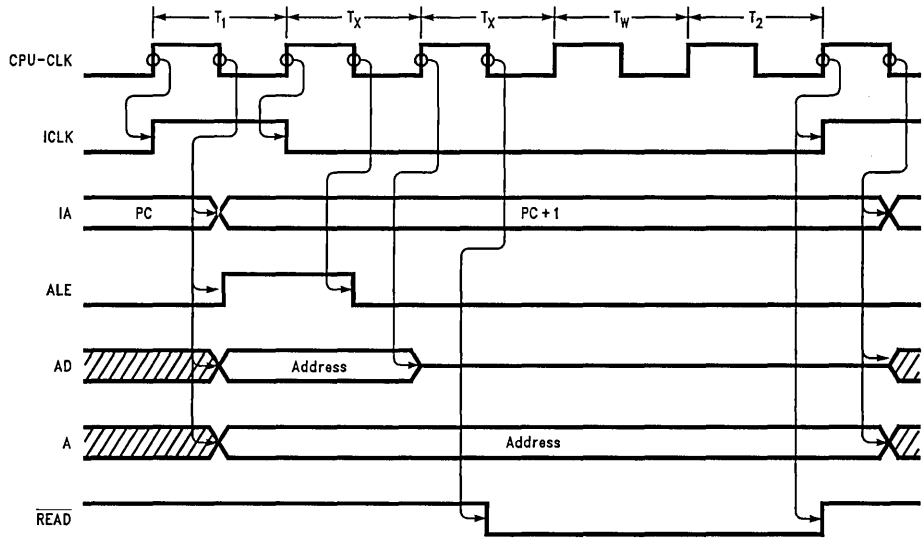
FIGURE 2-21. Data Memory Write with One Wait State

2.0 CPU Description (Continued)



TL/F/9336–E5

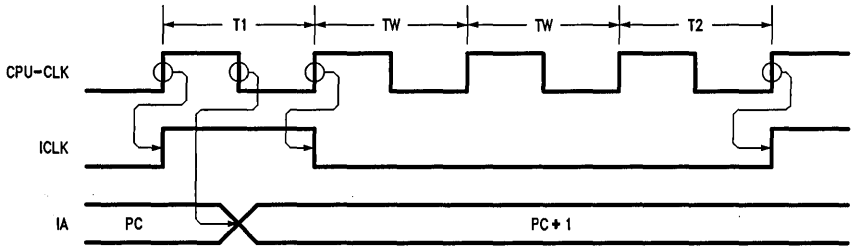
FIGURE 2-22. Data Memory Read with One Wait State and [4TR] = 0



TL/F/9336–H6

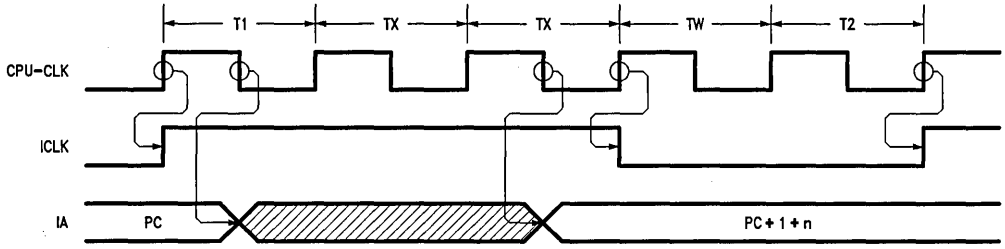
FIGURE 2-23. Data Memory Read with Two Wait States and [4TR] = 1

2.0 CPU Description (Continued)



TL/F/9336-E6

FIGURE 2-24. Two T-state Instruction with Two Wait States

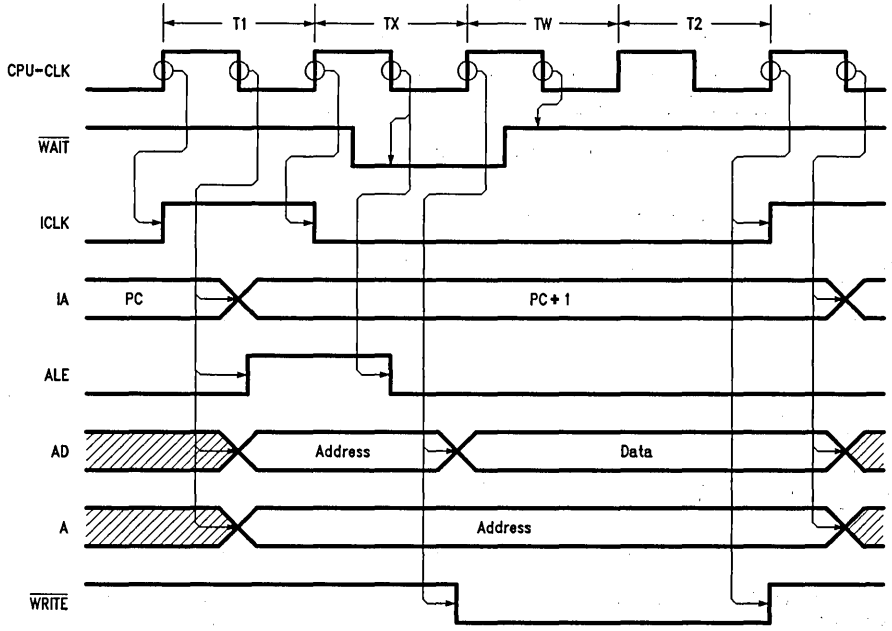


TL/F/9336-E7

FIGURE 2-25. Four T-state Instruction with One Wait State

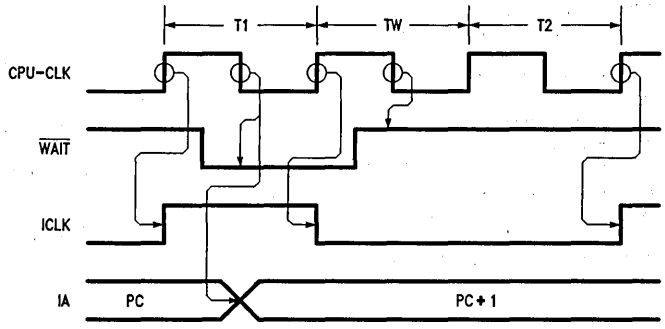
2.0 CPU Description (Continued)

The $\overline{WAIT}$ pin can also be used to add wait states to BCP instruction execution. The CPU will be waited as long as $\overline{WAIT}$ is low. To wait a given instruction, $\overline{WAIT}$ must be asserted low one-half T-state prior to the beginning of T2 in the instruction to be affected. *Figure 2-26* shows $\overline{WAIT}$ asserted during a write to data memory. In order to wait this instruction, $\overline{WAIT}$ must fall prior to the falling edge of CPU-CLK in TX. One wait state is added to the access and $\overline{WAIT}$ rises prior to the falling edge of CPU-CLK in TW which allows the access to finish. If $\overline{WAIT}$ had remained low, the access would have been held off indefinitely. Programmed wait states would delay when $\overline{WAIT}$ must be asserted since they would delay the beginning of T2. *Figures 2-27* through *Figure 2-29* depict the use of $\overline{WAIT}$ with three other instruction types. In all three cases, $\overline{WAIT}$ is asserted one-half T-state prior to when T2 would normally begin. Also, it is evident that the effect of $\overline{WAIT}$ on instruction timing is identical to adding programmed wait states.



TL/F/9336-E8

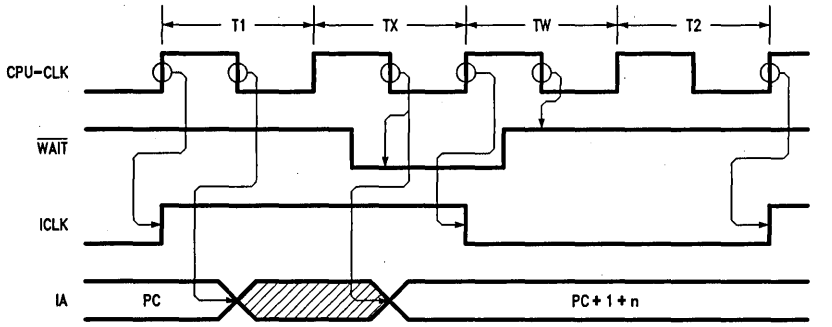
FIGURE 2-26. Data Memory Access $\overline{WAIT}$ Timing



TL/F/9336-E9

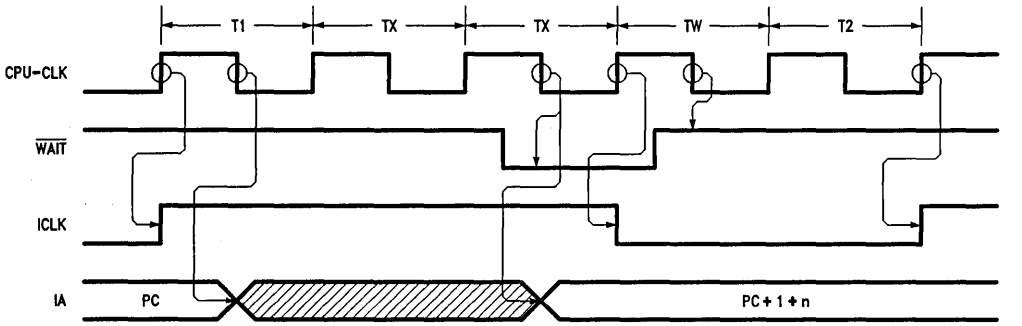
FIGURE 2-27. Two T-state Instruction $\overline{WAIT}$ Timing

2.0 CPU Description (Continued)



TL/F/9336-F1

FIGURE 2-28. Three T-state Program Control Instruction WAIT Timing



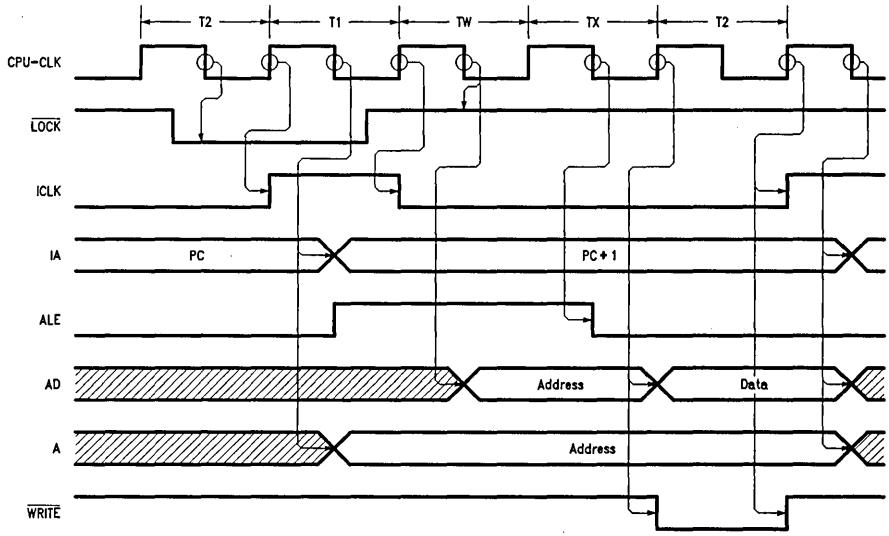
TL/F/9336-F2

FIGURE 2-29. Four T-state Program Control Instruction WAIT Timing

LOCK is another input which affects BCP instruction timing. LOCK prevents the BCP from accessing data memory. When asserted low, LOCK will cause the BCP to wait when it executes a data memory access instruction. The BCP will be waited until LOCK is taken high. To prevent a given access of data memory, LOCK must be asserted low one-half T-state prior to the beginning of the instruction accessing data memory. *Figure 2-30* shows LOCK being used to wait a write to data memory. LOCK falls prior to the falling edge of CPU-CLK before T1. In order to guarantee at least one wait state, LOCK is held low until after the falling edge of CPU-CLK in T1. This causes the insertion of TW into the cycle prior to TX. ALE remains high and the address is delayed on AD until LOCK is removed. After LOCK rises the access concludes normally with ALE falling halfway through TX and WRITE occurring during T2. Note that LOCK waits the access at a different point in the cycle than programmed wait states or WAIT. Additional wait states could occur from these sources prior to T2. *Figure 2-31* shows an example of LOCK holding off a write to data memory with one programmed wait state.

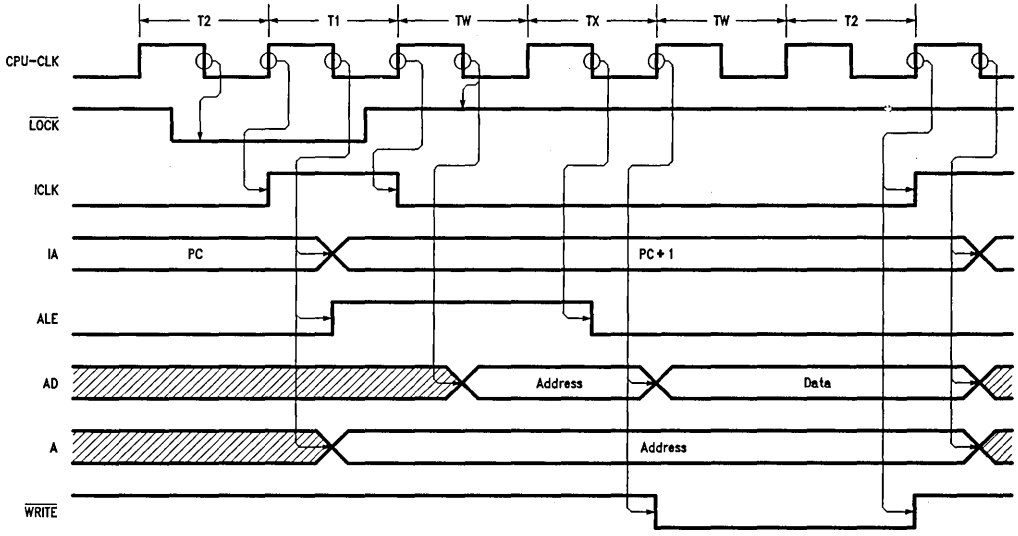
With timing similar to LOCK, the BCP will be delayed from making a data memory access by an access from the remote system. If the remote system is accessing the Remote Interface Configuration register, {RIC}, or data memory, the BCP will be waited by the Remote Interface and Arbitration System, RIAS, until the remote access is finished. The length of time the BCP is waited depends on the speed of the remote system and the type of remote access. The wait states are added prior to TX in the same manner as for LOCK shown in *Figure 2-30*. A more detailed description of the operation of RIAS can be found in Section 4.0, Remote Interface and Arbitration System.

2.0 CPU Description (Continued)



TL/F/9336-F3

FIGURE 2-30. $\overline{\text{LOCK}}$ Timing



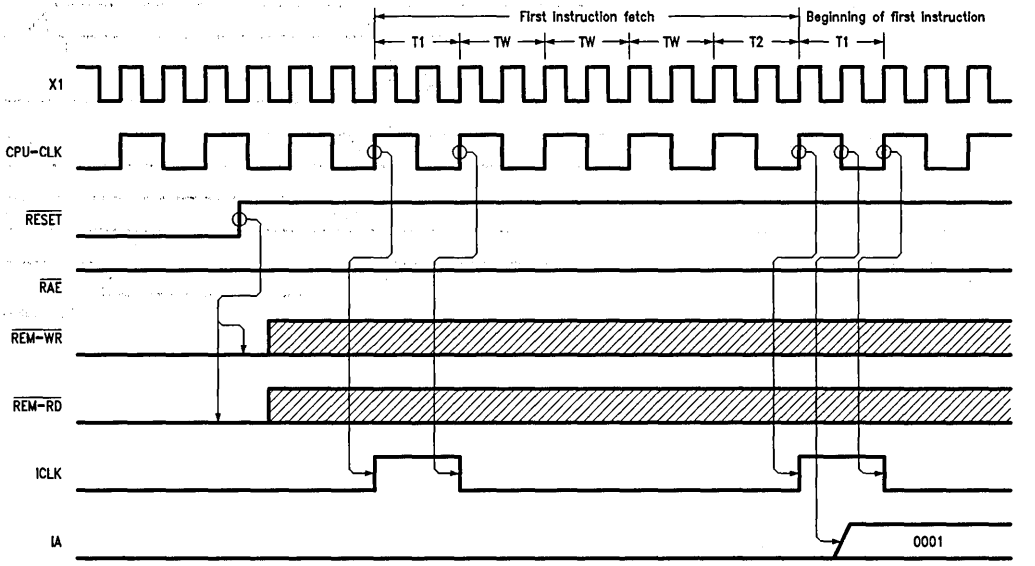
TL/F/9336-F4

FIGURE 2-31. $\overline{\text{LOCK}}$ Timing with One Wait State

2.0 CPU Description (Continued)

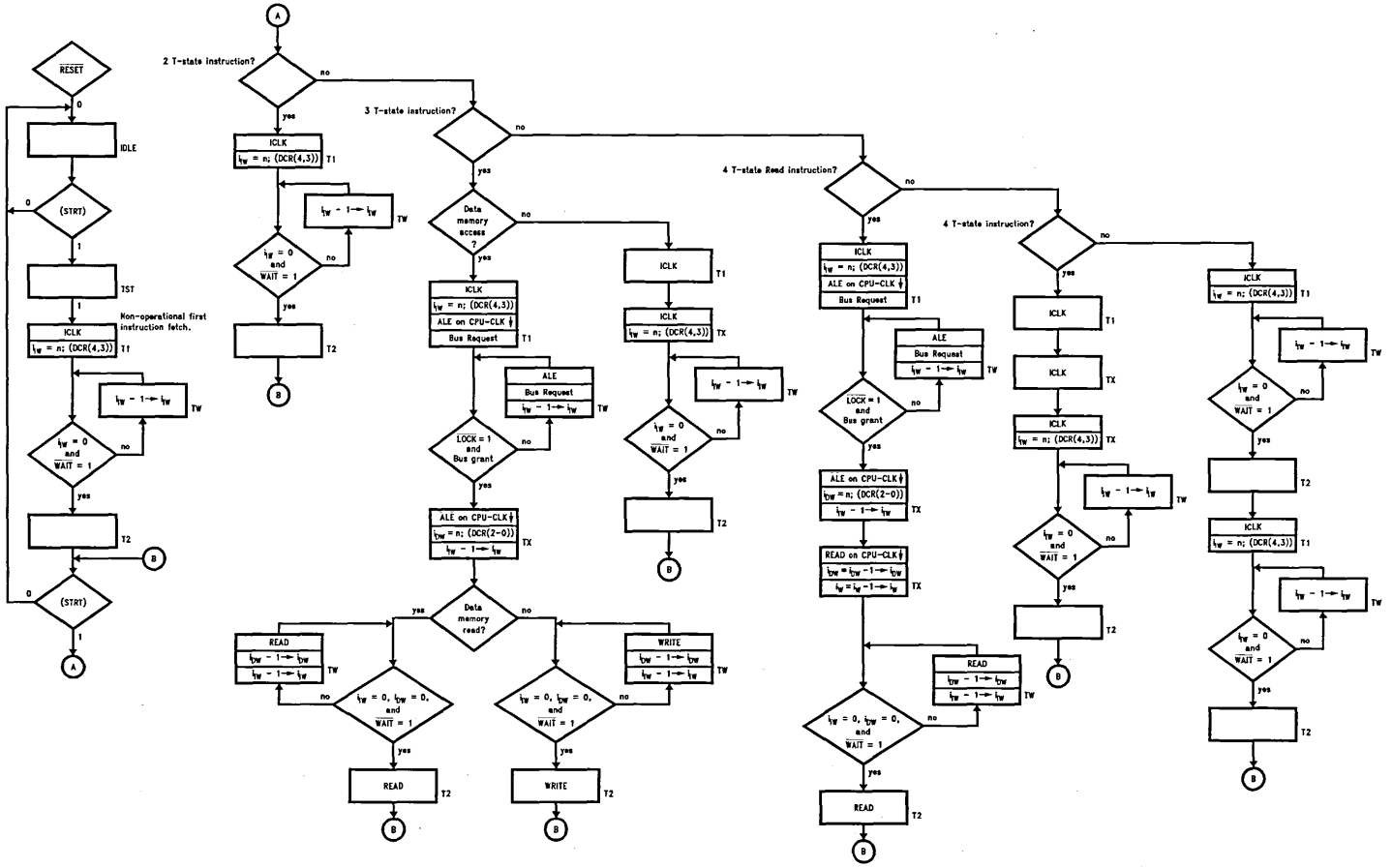
The CPU will be stopped after $\overline{RESET}$ is asserted low. The CPU can be externally controlled by changing the state of the start bit, [STRT], in {RIC}. The CPU starts executing instructions from the current address in the program control register when a one is written to [STRT] and stops when [STRT] is cleared. The CPU will complete the current instruction before stopping. Controlling the CPU from {RIC} requires a processor to access {RIC}. If no external processor is present, the CPU can be made to start automatically after reset by holding $\overline{REM\text{-}WR}$ and $\overline{REM\text{-}RD}$ low and $\overline{RAE}$ high while $\overline{RESET}$ is transitioning from low to high. The CPU "kick-starts" and will begin executing instructions from address zero. The timing for kick-starting the CPU is shown in *Figure 2-32.* ICLK rises on the rising edge of CPU-CLK one T-state after $\overline{RESET}$ is de-asserted. The falling edge of ICLK signifies the beginning of the first instruction fetch. Three instruction wait states and T2 precede the first instruction.

A functional state diagram describing the timing of the CPU is shown in *Figure 2-33.* The functional state diagram is similar to a flow chart, except that transitions to a new state (states are denoted as rectangular boxes) can only occur on the rising edge of the CPU-CLK. A state box can specify several actions, and each action is separated by a horizontal line. A signal name listed in a state box indicates that that pin will be asserted high when Timing Control has entered that state. When the signal is omitted from a box, it is asserted low. (Note: this requires using the inversion of a signal in some cases.) Decision blocks are shown as diamonds and their meaning is the same as in a flow chart. The functional state diagram is a generalized approach to determining instruction flow while allowing for any combination of wait states and control signals. Timing Control always starts from a reset in the state IDLE. After $\overline{RESET}$ goes high, Timing Control remains in IDLE until [STRT] is written high. If the BCP kick-starts, Timing Control enters TST on the next rising edge of CPU-CLK. Timing Control starts with a dummy instruction cycle in order to fetch the first instruction. ICLK goes high in T1 and the instruction wait state counter is loaded. ICLK falls when either T2 or TW is entered as determined by the value of i_{IW} and $\overline{WAIT}$. The normal instruction flow begins after T2 at B on the diagram. As an example, consider a three T-state data memory write instruction with one data wait state. The instruction cycle path for this instruction would begin at T1 following the decision block for data memory access. In T1, ICLK is asserted high, the instruction wait state counter is loaded, and a bus request to RIAS is generated. Also, ALE is asserted high on the falling edge of CPU-CLK during T1. A branch decision is now made based on the state of $\overline{LOCK}$ and the response from RIAS to the bus request. Assuming that $\overline{LOCK}$ is not asserted and a remote access is not in progress, Timing Control enters TX on the next rising edge of CPU-CLK. In TX, the data wait state counter is loaded and the instruction wait state counter is decremented. In this example, the instruction wait state counter is at zero and is not counting. The data wait state counter is loaded with one. ALE goes low on the falling edge of CPU-CLK during TX. The next decision block checks for a read of data memory. This example is a write to data memory so the decision is no and the branch is to the right. The wait state conditions are evaluated in the following decision block. i_{DW} is one and Timing Control enters TW on the next rising edge of CPU-CLK. $\overline{WRITE}$ is asserted low when TW is entered and the data wait state counter is decremented to zero. The decision on i_{DW}, i_{IW}, and $\overline{WAIT}$ is now true and T2 is entered on the next rising edge of CPU-CLK. $\overline{WRITE}$ remains low. The CPU will stop execution if [STRT] is low at B in the diagram. Otherwise, the next instruction will be executed beginning at A. To summarize, this instruction went through the following states: T1, TX, TW, and T2. The complete instruction cycle is shown in *Figure 2-21.* Any instruction cycle can be analyzed in a similar manner using this functional state diagram.



TL/F/9336-F5

FIGURE 2-32. CPU Start-Up Timing



TL/F/9336-F6

FIGURE 2-33. Functional State Diagram of CPU Timing

2.0 CPU Description (Continued)

2.2.3 Interrupts

The DP8344B has two external and four internal interrupt sources. The external interrupt sources are the Non-Maskable Interrupt pin, $\overline{NMI}$, and the Bi-directional Interrupt Request pin, $\overline{BIRQ}$.

External

A non-maskable interrupt is detected by the CPU when a falling edge is detected at the $\overline{NMI}$ pin. The interrupt is automatically cleared internally when the CPU recognizes the interrupt.

$\overline{BIRQ}$ can function as both an interrupt into the DP8344B and as an output which can be used to interrupt other devices. $\overline{BIRQ}$ is configured as an input or output according to the state of [BIC] in the Auxiliary Control Register, {ACR}. $\overline{BIRQ}$ is an input if [BIC] is a zero and an output when [BIC] is a one. The reset state of [BIC] is a zero, causing $\overline{BIRQ}$ to be an input after the BCP is reset. [BIRQ] in the Condition Code Register, {CCR}, is a read only bit which mirrors the state of $\overline{BIRQ}$ regardless of whether $\overline{BIRQ}$ is configured as an input or output. This bit is updated at the beginning of T1 of each instruction.

When $\overline{BIRQ}$ is configured as an input, an interrupt will occur if the pin is held low. $\overline{BIRQ}$ must be held low until the interrupt is recognized or the interrupt will not be processed. Due to the prioritizing of interrupts as described below, $\overline{BIRQ}$ may not be recognized by the CPU until higher priority interrupts have been serviced. $\overline{BIRQ}$ will be recognized after higher priority interrupts have been processed. The low state on $\overline{BIRQ}$ should be removed after the CPU recognizes the interrupt or the interrupt will be processed multiple times.

When $\overline{BIRQ}$ is configured as an output, its state is controlled by [IM3] in the Interrupt Control Register, {ICR}. Changing the state of this bit will change $\overline{BIRQ}$ at the beginning of T1 of the instruction following the write to [IM3]. Note that [BIRQ] in {CCR} is also updated at the beginning of T1. Therefore, there is a one instruction cycle delay from when [IM3] changes to when the new value of $\overline{BIRQ}$ is made available in [BIRQ]. [BIS] in the Remote Interface Configuration register, {RIC}, mirrors the state of [IM3]. When $\overline{BIRQ}$ is an output, writing a one to [BIS] will change the state of [IME] thus changing $\overline{BIRQ}$ and allowing a remote processor to acknowledge an interrupt from the BCP. Note, if the BCP code operates on [IM3] at the same time that the remote processor acknowledges the interrupt by writing a one to [BIS], $\overline{BIRQ}$ will toggle and then assume the state of [IM3] resulting from the BCP code operation. Therefore, if the designer chooses to operate on [IM3] while waiting for the remote processor to acknowledge a $\overline{BIRQ}$ interrupt, the designer should ensure that the remote processor is locked out from accessing [BIS] during the operation on [IM3]. This can be accomplished by setting [LOR] in {ACR}, having the BCP perform a data memory access to ensure that any current remote accesses are complete, operating an [IM3], and finally clearing [LOR]. $\overline{BIRQ}$ will change state two T-states after the end of the write to [BIS]. Writing a one to [BIS] will have no effect on [IM3] when $\overline{BIRQ}$ is an input. Table 2-26 summarizes the relationship between $\overline{BIRQ}$ and its associated register bits.

TABLE 2-26. $\overline{BIRQ}$ Control Summary

(a) $\overline{BIRQ}$ is an Input ([BIC] = 0): Remote Processor Controls the State of $\overline{BIRQ}$

[IM3]	[BIS]	$\overline{BIRQ}$	[BIRQ]
0	[IM3] = 0	Active Interrupt to the BCP: state of $\overline{BIRQ}$ controlled by the Remote Processor	Reflects the state of $\overline{BIRQ}$
1	[IM3] = 1	Masked Interrupt to the BCP: state of $\overline{BIRQ}$ controlled by the Remote Processor	Reflects the state of $\overline{BIRQ}$

(b) $\overline{BIRQ}$ is an Output ([BIC] = 1): BCP Controls the State of $\overline{BIRQ}$

[IM3]	[BIS]	$\overline{BIRQ}$	[BIRQ]
0	[IM3] = 0	State of [IM3] = 0	Reflects the state of $\overline{BIRQ}$ = 0
1	[IM3] = 1	State of [IM3] = 1	Reflects the state of $\overline{BIRQ}$ = 1

(c) $\overline{BIRQ}$ is an Output ([BIC] = 1): Remote Processor Acknowledges $\overline{BIRQ}$

[BIS]	[IM3]	[BIS]	$\overline{BIRQ}$	[BIRQ]
Remote Processor writes a 1 to [BIS]	Toggles	[IM3]	State of [IM3]	Reflects the state of $\overline{BIRQ}$

2.0 CPU Description (Continued)

Internal

The internal interrupts consist of the Transmitter FIFO Empty, TFE, interrupt, the Line Turn Around, LTA, interrupt, the Time Out, TO, interrupt, and a user selectable receiver interrupt source. The receiver interrupt source is selected from either the Receiver FIFO, Full, RFF, interrupt, the Data Available, DA, interrupt, or the Receiver Active, RA, interrupt. The receiver interrupt is selected using bits [RIS1] and [RIS0] in the Interrupt Control Register, {ICR}. See the Section 3.0, Transceiver for a description of these interrupts.

Masking

The BCP uses two levels of interrupt masking: a global interrupt mask which affects all interrupts except $\overline{NMI}$ and individual interrupt mask bits. Global enabling and disabling of the interrupts is performed by changing the state of the Global Interrupt Enable bit, [GIE], in {ACR}. The maskable interrupts are disabled when [GIE] is a zero and enabled when [GIE] is a one. [GIE] is a zero after the BCP is reset. [GIE] is a read/write register bit and may be changed by using any instruction that can write to {ACR}. In addition, the RET, RETF, and EXX instructions have option fields which can be used to alter the state of [GIE]. The EXX instruction can set or clear [GIE] as well as leaving it unchanged. The RET and RETF instructions can restore [GIE] to the value that was saved on the address stack at the time the interrupt was recognized. These instructions also provide the options of clearing or setting [GIE] or leaving it unchanged. [GIE] is set to a zero when an interrupt is recognized by the CPU. It is necessary to set [GIE] to a one if interrupts are to be recognized within an interrupt routine.

The individual interrupt mask bits are located in {ICR}. When set to a one, bits [IM0], [IM1], [IM2], [IM3], and [IM4] in {ICR} mask the receiver interrupt, TFE interrupt, LTA interrupt, $\overline{BIRQ}$ interrupt, and TO interrupt, respectively. To enable an interrupt, its mask bit must be set to a zero. The interrupts and associated mask bits are shown in Table 2-27. These bits are set to a one when the DP8344 is reset.

Masking interrupts with [GIE] or the mask bits in {ICR} prevents the CPU from acknowledging interrupts but does not prevent the interrupts from occurring. Therefore, if an interrupt is asserted, it will be processed as soon as it is unmasked by changing [GIE] to a one and/or changing the appropriate mask bit in {ICR} to a zero.

Priorites

When more than one interrupt is unmasked and asserted, the CPU processes the interrupt with the highest priority first. $\overline{NMI}$ has the highest priority followed by the receiver interrupt, TFE, LTA, $\overline{BIRQ}$, and TO. Each time the interrupts are sampled, the highest priority interrupt is processed first, regardless of how long a lower priority interrupt has been active. Interrupt priority is summarized in Table 2-27.

TABLE 2-27. {ICR} Interrupt Mask Bits and Interrupt Priority

Interrupt	Mask Bit	Priority
$\overline{NMI}$	—	Highest
RFF, DA, RA	[IMO]	
TFE	[IM1]	
LTA	[IM2]	
$\overline{BIRQ}$	[IM3]	
TO	[IM4]	Lowest

2.0 CPU Description (Continued)

A call to the interrupt address is generated when an interrupt is detected by the CPU. The address for each interrupt is constructed by concatenating the Interrupt Base Register, {IBR}, contents with the individual interrupt code as shown in Table 2-28. There is room between the interrupt addresses for a maximum of four instruction words.

TABLE 2-28. Interrupt Vector Generation

Interrupt	Code
$\overline{NMI}$	111
RFF, DA, RA	001
TFE	010
LTA	011
$\overline{BIRQ}$	100
TO	101

Interrupt Vector

{IBR} Contents	0 0 0	Code	0 0
15	8	5	2 0

Interrupts are sampled by each falling edge of the CPU clock with the last falling edge prior to the start of the next instruction determining whether an interrupt will be processed. The timing of a typical interrupt event is shown in *Figure 2-34.* The interrupt occurs during the current instruction and is sampled by the falling edge of the CPU clock. The next instruction is not operated on and its address is stored in the internal address stack along with [GIE], the ALU flags, and the register bank positions. The address stack is twelve words deep. A two T-state internal call is now executed in place of the non-executed instruction. This call will cause a branch to the interrupt address that is generated in the first half of T-state T1. Also, [GIE] is cleared at the end of the first half of T-state T1. The internal call to the interrupt address is subject to instruction wait states as configured in {DCR}.

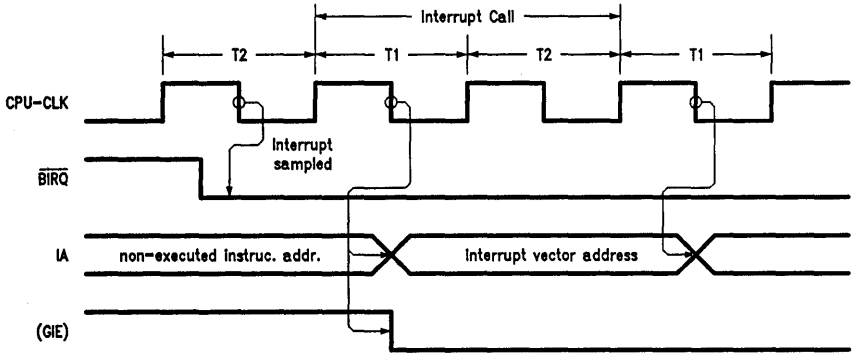
2.2.4 Oscillator

The crystal oscillator is an on-chip amplifier which may be used with an external crystal to generate accurate CPU and transceiver clocks. The input to this amplifier is X1, pin 33. The output of the amplifier is X2, pin 34. When X1 and X2 are connected to a crystal and external capacitors *(Figure 2-35),* the combined circuit forms a Pierce crystal oscillator with the crystal operating at parallel resonance. Crystals that oscillate over the frequency range of 2 MHz to 20 MHz may be used. The recommended crystal parameters for operation with the oscillator are given in Table 2-29. The external capacitor values should be chosen to provide the manufacturer's specified load capacitance for the crystal when combined with the parasitic capacitance of the trace, socket, and package. As an example, a crystal with a specified load capacitance of 20 pF used in a circuit with 13 pF per pin parasitic capacitance will require external capacitor values of 27 pF each. This provides an equivalent capacitance of 40 pF on each side of the crystal, and has a 20 pF series equivalent value across the crystal.

As an alternative to the crystal oscillator, an external clock source may be used. In this case, the external clock source should be connected to X1 and no external circuitry should be connected to X2 *(Figure 2-36).* The DP8344 can supply a clock source, equal in frequency to the crystal oscillator or external clock source, to other circuitry via pin 35, the CLKOUT output. This output is a buffered version of the signal at X1.

TABLE 2-29. Recommended Crystal Parameters

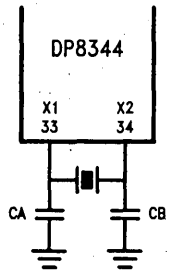
AT Cut, Parallel Resonant
Fundamental Mode
Load Capacitor = 20 pF
Series Resistance < 20Ω
Frequency Tolerance 0.005% at 25°C
Stability 0.01% 0°–70°C
Drive Level 0.5 mW Typical



TL/F/9336-F7

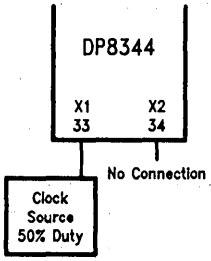
FIGURE 2-34. Interrupt Timing

2.0 CPU Description (Continued)



TL/F/9336-F8

FIGURE 2-35. DP8344B Operation with Crystal



TL/F/9336-F9

FIGURE 2-36. DP8344B Operation with External Clock

3.0 Transceiver

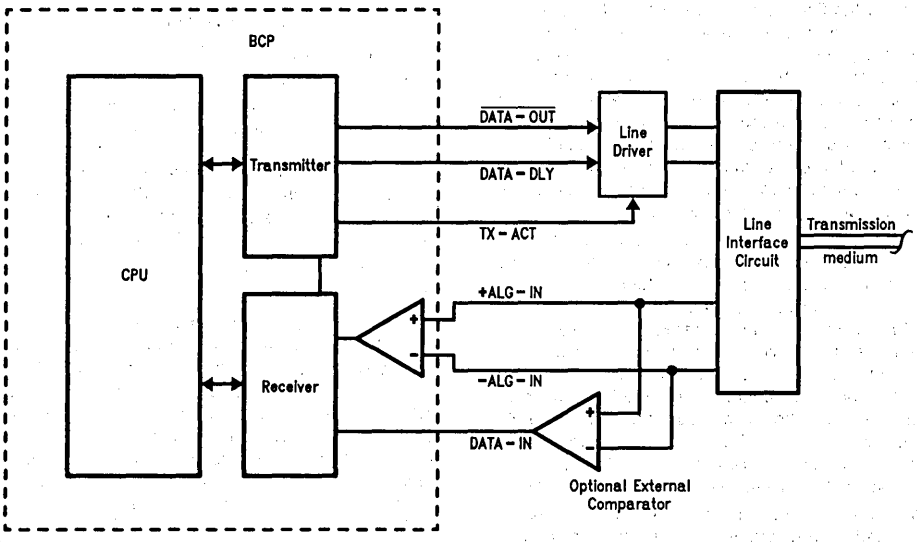
3.1 TRANSCEIVER ARCHITECTURAL DESCRIPTION

The transceiver section operates as an on-chip, independent peripheral, implementing all the necessary formatting required to support the physical layer of the following serial communications protocols:

- IBM 3270 (including 3299)
- IBM 5250
- NSC general purpose 8-bit

The CPU and transceiver are tightly coupled through the CPU register space, with the transceiver appearing to the CPU as a group of special function registers and three dedicated interrupts. The transceiver consists of separate transmitter and receiver logic sections, each capable of independent operation, communicating with the CPU via an asynchronous interface. This interface is software configurable for both polled and interrupt-driven interaction, allowing the system designer to optimize his product for the specific application.

The transceiver connects to the line through an external line interface circuit which provides the required DC and AC drive characteristics appropriate to the application. A block diagram of such an interface is shown in *Figure 3-1.* An on-chip differential analog comparator, optimized for use in a transformer coupled coax interface, is provided at the input to the receiver. Alternatively, if an external comparator is necessary, the input signal may be routed to the DATA-IN pin.



TL/F/9336-33

FIGURE 3-1. System Block Diagram, Showing Details of the Line Interface

3.0 Transceiver (Continued)

The transceiver has several modes of operation. It can be configured for single line, half-duplex operation in which the receiver is disabled while the transmitter is active. Alternatively, both receiver and transmitter can be active at the same time for multi-channel (such as repeater) or loopback operation. The transceiver has both internal and external loopback capabilities, facilitating testing of both the software and external hardware. At all times, both transmitter and receiver operate according to the same protocol definition.

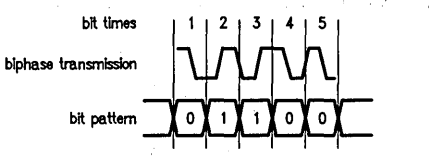
3.1.1 Protocols

In all protocols, data is transmitted serially in discrete messages containing one or more frames, each representing a single word of information. Biphase (Manchester II) encoding is used, in which the data stream is divided into discrete time intervals (bit-times) denoted by a level transition in the center of the bit-time. For the IBM 3270, 3299 and NSC general purpose 8-bit protocols, a mid-bit transition from low to high represents a biphase "1", and a mid-bit transition from high to low represents a biphase "0". For the 5250 protocol, the definition of biphase logic levels is exactly reversed, i.e. a biphase "1" is represented by a high to low transition. Depending on the bit sequence, there may or may not be a transition on the bit-time boundary. The biphase encoding of a simple bit sequence is illustrated in *Figure 3-2(a)*.

Each transmission begins with a unique start sequence consisting of 5 biphase encoded "1's", (referred to as "line quiesce pulses") followed by a 3 bit-time code violation and the sync bit of the first frame, *Figure 3-2(b)*. The three bit-time code violation does not conform to the rules of Manchester encoding and forms a unique recognition pattern for bit time synchronization by the receiver logic. The first bit of any frame is the sync bit, a biphase "1". The frame is then formatted according to the requirements of the protocol. If a multi-frame message is being transmitted, additional frames are appended to the end of the first frame—except for the 5250 protocol, where there may be an optional number of "fill bits" (biphase "0") between each frame.

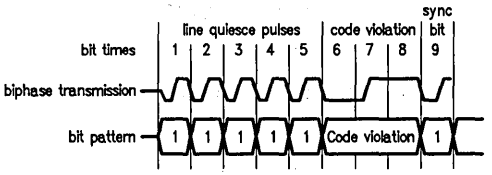
Depending on the protocol, when all data has been transmitted, the end of a message will be indicated either by the transmission of an ending sequence, or (for 5250) simply by the cessation of transitions on the differential line. Later model 5250 equipment has incorporated a "line hold" at the end of the message. The line hold maintains the final differential state on the line for several bit times to eliminate noise or reflections that could be interpreted as a continuance of the message. The ending sequence for all but 5250 protocols consists of a single biphase "0" followed by a low to high transition on the bit-time boundary and two bit-times with no transitions (two mini-code violation), *Figure 3-2(c)*.

The various protocol framing formats are shown in *Figures 3-3* through *3-5*. The diagrams use a bit pattern drawing convention which, for clarity, shows the bit-time boundaries but not the biphase transitions in the center of the bit times. The timing relationship between the biphase encoded bit stream and the bit pattern diagrams is consistent with *Figure 3-2*.



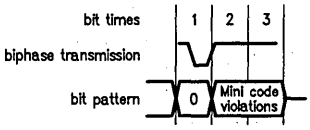
TL/F/9336-34

(a) Biphase Encoding



TL/F/9336-36

(b) Starting Sequence



TL/F/9336-35

(c) Ending Sequence

FIGURE 3-2. Biphase Encoding

3.1.1.1 IBM 3270

The framing format of the IBM 3270 coax protocol is shown in *Figures 3-3(a)* and *(b)*, for both single and multi-frame messages. Each message begins with a starting sequence and ends with an ending sequence, as shown in *Figures 3-2(b)* and *(c)*. Each 12-bit frame begins with a sync bit (B1) followed by an 8-bit data byte (MSB first), a 2-bit control field, and the frame delimiter bit (B12), representing even parity on the previous 11 bits. The bit rate on the coax line is 2.3587 MHz.

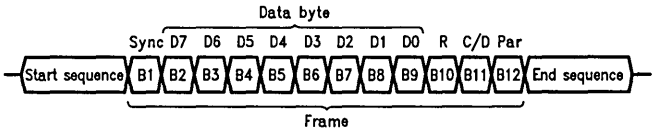
3.1.1.2 IBM 3299

Adding 3299 multiplexers to the 3270 environment requires an address to be transmitted along with each message from the controller to the multiplexer. The IBM 3299 Terminal Multiplexer protocol provides this capability by defining an additional 8-bit frame as the first frame of every message sent from the controller, as shown in *Figure 3-3(c)*. This frame contains a 6-bit data field along with the normal sync and word parity bits. The protocol currently utilizes bits B2–B4 as an address field that directs the message through the multiplexor hardware. Following the address frame, the rest of the message follows standard 3270 convention. The bit rate, 2.3587 MHz, is the same as standard 3270.

3.1.1.3 IBM 5250

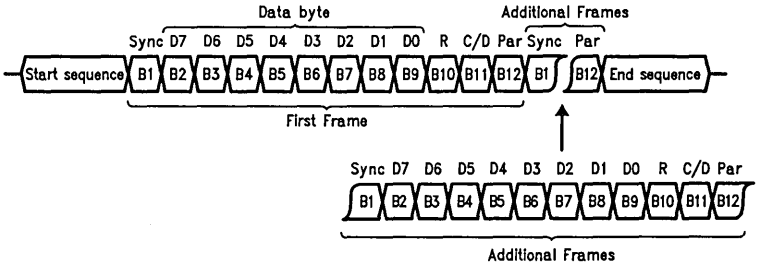
The framing format of the IBM 5250 twinax protocol is shown in *Figure 3-4*, for both single and multi-frame messages. Each message begins with the starting sequence shown in *Figure 3-2(b)*, and ends with 3 fill bits (biphase "0"). A 16-bit frame is employed, consisting of a sync bit (B15); an 8-bit data byte (B7–B14) (LSB first); a 3-bit station address field (B4–B6); and the last bit (B3) representing

3.0 Transceiver (Continued)



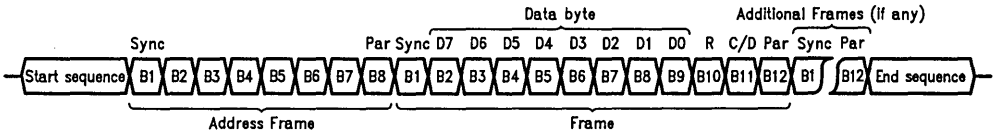
TL/F/9336-37

(a) 3270 Single-Byte Message



TL/F/9336-38

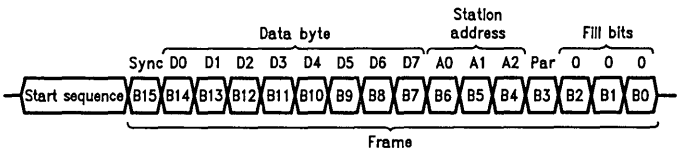
(b) 3270 Multi-Byte Message



TL/F/9336-39

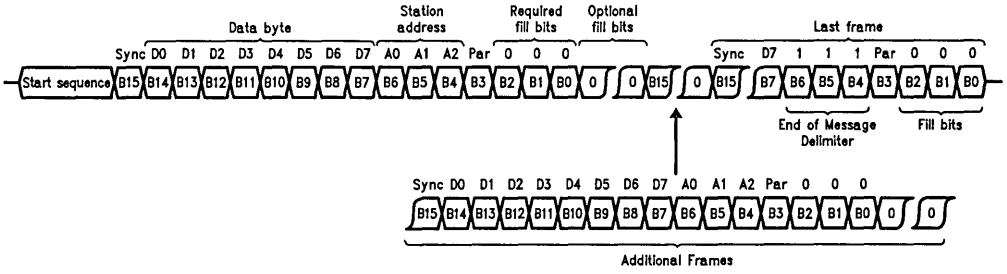
(c) 3299 Controller/Multiplexer Message

FIGURE 3-3. 3270/3299 Protocol Framing Format



TL/F/9336-40

(a) 5250 Single-Byte Message



TL/F/9336-41

(b) 5250 Multi-Byte Message

FIGURE 3-4. 5250 Protocol Framing Format

3.0 Transceiver (Continued)

even word parity on the previous 12 bits. Following the parity bit, 3 biphase "0" fill bits (B0–B2) are transmitted. Following these required fill bits, up to 240 additional fill bits can be inserted between frames before the next sync bit and the start of the next frame of a multi-byte message. The bit rate on the twinax line is 1 MHz.

3.1.1.4 General Purpose 8-Bit

The framing format of the general purpose 8-bit protocol is shown in *Figure 3-5*, for both single and multi-frame messages. It is identical to that used by the National Semiconductor DP8342 transmitter and DP8343 receiver chips. Each message begins with a starting sequence and ends with an ending sequence, as shown in *Figures 3-2(b)* and *(c)*. A 10-bit frame is employed, consisting of the sync bit (B1); an 8-bit data byte (B2–B9) (LSB first); and the last bit of the frame (B10) representing even word parity on the previous 9 bits. For multiplexed applications, the first frame can be designated as an address frame, with all 8 bits available for the logical address. (See General Purpose 8-bit Modes in this section.)

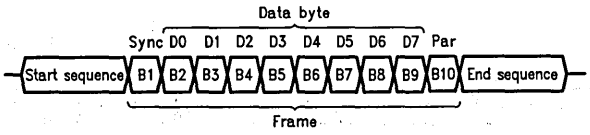
3.2 TRANSCEIVER FUNCTIONAL DESCRIPTION

A block diagram of the transceiver, revealing external inputs and outputs and details of the CPU interface, is shown in *Figure 3-6*. The transmitter and receiver are largely independent of each other, sharing only the clock, reset and protocol select signals. The transceiver is mapped into the CPU register space, thus the status of the transceiver can always be polled. In addition, the CPU/Transceiver interface can be configured for an interrupt-driven environment. (See Transceiver Interrupts in this section.)

Both transmitter and receiver are reset by a common Transceiver Reset bit, [TRES], allowing the CPU to independently reset the transceiver at any time. The Transceiver is also reset whenever the CPU reset is asserted, including the required power-up reset. When [TRES] is asserted, both transmitter and receiver FIFO's are emptied resulting in the Transmit FIFO Empty flag [TFE] being asserted and the Data Available flag [DAV] cleared. Other flags cleared by [TRES] are Transmit FIFO Full [TFF] and Transmitter Active [TA] in the transmitter and Line Active [LA], Receiver Active [RA], Receiver Error [RE], Receive FIFO Full [RFF], Data Error or Message End [DEME], [POLL], [ACK], and [RAR] command flags in the receiver. When [TRES] is asserted, external pin TX-ACT is cleared, DATA-DLY goes to a state equal to the complement of Transmitter INvert [TIN] in {TMR}, and $\overline{\text{DATA-OUT}}$ goes into a state equal to the complement of [TIN] exclusive or'ed with the Advance Transmitter Active [ATA] in {TCR}. In other words, when [TRES] is asserted, DATA-DLY = [TIN], and $\overline{\text{DATA-OUT}}$ = [TIN] ⊕ [ATA]. When [TRES] is asserted under software control, it is necessary to wait at least one instruction after asserting [TRES] before seeing the resulting reset state of the affected flags in the CPU. The transmitter and receiver are clocked by a common Transceiver Clock, TCLK, at a frequency equal to eight times the required serial data rate. TCLK can either be obtained from the on-chip oscillator divided by 1, 2 or 4, or from an external clock applied to the X-TCLK pin. TCLK selection is controlled by two Transceiver Clock Select bits, [TCS 1–0] located in the Device Control Register, {DCR}. [TCS 1–0] should only be changed when the transceiver is inactive.

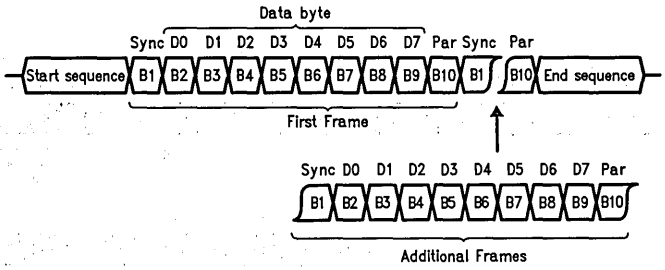
Since the TCLK source can be asynchronous with respect to the CPU clock, the CPU/Transceiver interface can be asynchronous. All flags from the Transceiver are therefore latched at the start of all instructions, and parallel data is transferred through 3 word FIFOs in both the transmitter and receiver.

Protocol selection is controlled by three Protocol Select bits, [PS2–0] in the Transceiver Mode Register, {TMR} (see Table 3-1). Enough flexibility is provided for the BCP to operate in all required positions in the network. It is not pos-



TL/F/9336–42

(a) 8-Bit Single-Byte Message



TL/F/9336–43

(b) 8-Bit Multi-Byte Message

FIGURE 3-5. General Purpose 8-Bit Protocol Framing Format

3.0 Transceiver (Continued)

sible for the transmitter and receiver to operate with different protocols at the same time. The protocol mode should only be changed when both transmitter and receiver are inactive.

If both transmitter and receiver are connected to the same line, they should be configured to operate sequentially (half-duplex). This mode of operation is achieved by clearing the RePeater ENable control bit [RPEN] in {TMR}. In this mode, an active transmitter will disable the receiver, preventing simultaneous operation of transmitter and receiver. If the transmitter FIFO is loaded while the receiver is actively processing an incoming signal, the receiver will be disabled and flag the CPU that a "Receiver Disabled While Active" error has occurred. (See Receiver Errors in this section.) On power-up/reset the transceiver defaults to this half-duplex mode.

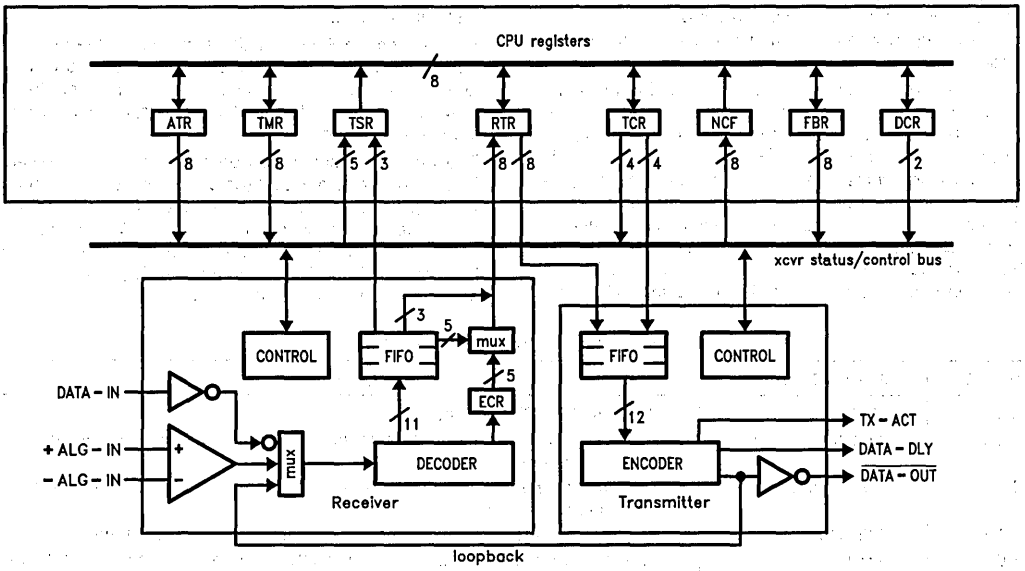
By asserting the Repeat Enable flag [RPEN], the receiver is not disabled by the transmitter, allowing both transmitter and receiver to be active at the same time. This feature provides for the implementation of a repeater function or loopback for test purposes.

The transmitter output can be connected to the receiver input, implementing a local (on-chip) loopback, by asserting [LOOP]. [RPEN] must also be asserted to enable both the transmitter and receiver at the same time. With [LOOP] asserted, the output TX-ACT is disabled, keeping the external line driver in TRI-STATE. The internal flag [TA] is still enabled, as are the serial data outputs.

TABLE 3-1. Protocol Mode Definition

PS2–0	Protocol Mode	Comments
0 0 0	3270	Standard IBM 3270 protocol.
0 0 1	3299 Multiplexer	Receiver expects first frame to be address frame. Transmitter uses standard 3270, no address frame.
0 1 0	3299 Controller	Transmitter generates address frame as first frame. Receiver expects standard 3270, no address frame.
0 1 1	3299 Repeater	Both transmitter and receiver operate with first frame as address frame.
1 0 0	5250	Non-promiscuous mode. [DAV] asserted only when first frame address matches {ATR}.
1 0 1	5250 Promiscuous	[DAV] asserted on all valid received data without regard to address field.
1 1 0	8-Bit	General-purpose 8-bit protocol with first frame address. Non-promiscuous mode. [DAV] asserted only when first frame address matches {ATR}.
1 1 1	8-Bit Promiscuous	[DAV] asserted on all valid received frames.

3.0 Transceiver (Continued)



TL/F/9336-44

KEY TO REGISTERS

RTR	Receive/Transmit Register	ATR	Auxiliary Transceiver Register
TSR	Transceiver Status Register	NCF	Network Command Register
TCR	Transceiver Command Register	FBR	Fill-Bit Register
TMR	Transceiver Mode Register	DCR	Device Control Register

FIGURE 3-6. Block Diagram of Transceiver, Showing CPU Interface

3.0 Transceiver (Continued)

3.2.1 Transmitter

The transmitter accepts parallel data from the CPU, formats it according to the desired protocol and transmits it as a serial biphase-encoded bit stream. A block diagram of the transmitter logic is shown in *Figure 3-6*. Two biphase outputs, $\overline{\text{DATA-OUT}}$, DATA-DLY, and the external line driver enable, TX-ACT, provide the data and control signals for the external line interface circuitry. The two biphase outputs are valid only when TX-ACT is asserted (high) and provide the necessary phase relationship to generate the "predistortion" waveform common to all of the transceiver protocols. See *Figure 3-7* for the timing relationships of these outputs as well as the output of the line driver. For a recommended 3270/3299 coax interface, see Section 3.2.5.1 3270 Line Interface. For a recommended 5250 twinax interface see Section 3.2.5.2 5250 Line Interface.

The capability is provided to invert $\overline{\text{DATA-OUT}}$ and DATA-DLY via the Transmitter Invert bit, [TIN], located in the Transceiver Mode Register, {TMR}. In addition, the timing relationship between TX-ACT and the two biphase outputs can be modified with the Advance Transmitter Active control, [ATA]. When [ATA] is cleared low (the power-up condition), the transmitter generates exactly five line quiesce bits at the start of each message, as shown in *Figure 3-7*. If [ATA] is asserted high, the transmitter generates a sixth line quiesce bit, adding one biphase bit time to the start sequence transmission. The line driver enable, TX-ACT, is asserted halfway through this bit time, allowing an additional half-bit to precede the first full line quiesce of the transmitted waveform. Also, the state of DATA-DLY is such that no predistortion results on the line during this first half line quiesce. This modified start sequence is depicted in the dotted lines shown in *Figure 3-7* and is used to limit the initial transient voltage amplitude when the message begins.

Data is loaded into the transmitter by writing to the Receive/Transmit Register {RTR}, causing the first location of the FIFO to be loaded with a 12-bit word (8 bits from {RTR} and 4 bits from the Transceiver Command Register {TCR}. The data byte to be transmitted is loaded into {RTR}, and {TCR} contains additional information required by the protocol. It is important to note that if {TCR} is to be changed, it must be loaded before {RTR}. A multi-frame transmission is accomplished by sequentially loading the FIFO with the required data, the transmitter taking care of all necessary frame formatting.

If the FIFO was previously empty, indicated by the Transmit FIFO Empty flag [TFE] being asserted, the first word loaded into the FIFO will asynchronously propagate to the last location in approximately 40 ns, leaving the first two locations empty. It is therefore possible to load up the FIFO with three sequential instructions, at which time the Transmit FIFO Full flag [TFF] will be asserted. If {RTR} is written while [TFF] is high, the first location of the FIFO will be over-written and that data will be destroyed.

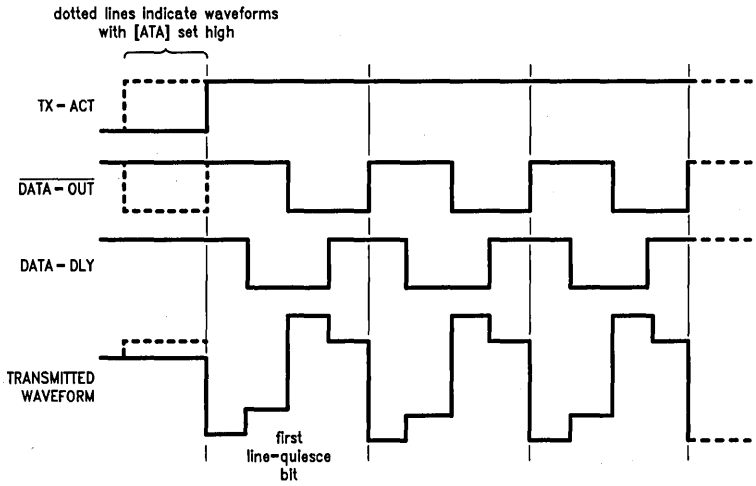
When the first word is loaded into the FIFO, the transmitter starts up from idle, asserting TX-ACT and the Transmitter Active flag [TA], and begins generating the start sequence. After a delay of approximately 16 TCLK cycles (2 biphase bit times), the word in the last location of the FIFO is loaded into the encoder and prepared for transmission. If the FIFO was full, [TFF] will be de-asserted when the encoder is loaded, allowing an additional word to be loaded into the FIFO.

When the last word in the FIFO has been loaded into the encoder, [TFE] goes high, indicating that the FIFO is empty. To ensure the continuation of a multi-frame message, more data must then be loaded into the FIFO before the encoder starts the transmission of the last bit of the current frame (the frame parity bit for 3270, 3299, and 8-bit modes; the last of the three mandatory fill bits for 5250). This maximum load time from [TFE] can be calculated by subtracting two from the number of bits in each frame of the respective protocol, and multiplying that result by the bit rate. This number represents the best case time to load—the worst case value is dependent on CPU performance. Since the CPU samples the transceiver flags and interrupts at instruction boundaries, the CPU clock rate, wait states (from programmed wait states, asserting the WAIT pin, or remote access cycles), and the type of instruction currently being executed can affect when the flag or interrupt is first presented to the CPU.

If there is no further data to transmit (or if the load window is missed), the ending sequence (3270/3299/8-bit) is generated and the transmitter returns to idle, de-asserting TX-ACT and [TA]. In 5250 mode, the three required fill bits are sent and TX-ACT and [TA] are de-asserted at a time dependent on the value of bits 7 through 3 of the Auxiliary Transceiver Register {ATR}. If {ATR[7–3]} = 00000, TX-ACT and [TA] are de-asserted at the end of the third required fill bit resulting in no additional "line hold" at the end of the message. Each increment of {ATR[7–3]} results in an additional half bit time of line hold up to a maximum of 15.5 bit times.

Data should not be loaded into the FIFO after the transmitter is committed to ending the message and before the [TA] flag is deasserted. If this occurs, the load will be missed by the transmitter control logic and the word(s) will remain in the FIFO. This condition exists when [TA] and [TFE] are both low at the same time, and can be cleared by resetting the transceiver (asserting [TRES]) or by loading more data into the FIFO, in which case the first frame(s) transmitted will contain the word(s) left in the FIFO from the previous message.

3.0 Transceiver (Continued)



TL/F/9336–45

FIGURE 3-7. Transmitter Output

3.2.2 Receiver

The receiver accepts a serial biphase-encoded bit stream, strips off the framing information, checks for errors and reformats the data for parallel transfer to the CPU. The block diagram in *Figure 3-6* depicts the data flow from the serial input(s) to the FIFO's parallel outputs. Note that the FIFO outputs are multiplexed with the Error Code Register {ECR} outputs.

The receiver and transmitter share the same TCLK, though in the receiver this clock is used only to establish the sampling rate for the incoming biphase encoded data. All control timing is derived from a clock signal extracted from this data. Several status flags and interrupts are made available to the CPU to handle the asynchronous nature of the incoming data stream. See *Figure 3-8* for the timing relationships of these flags and interrupts relative to the incoming data.

The input source to the decoder can be either the on-chip analog line receiver, the DATA-IN input or the output of the transmitter (for on-chip loopback operation). Two bits, the Select Line Receiver [SLR] and Loopback [LOOP], control this selection. For interfacing to the on-chip analog line receiver, see Section 3.2.5.1, 3270 Line Interface. An example of an external comparator circuit for interfacing to twinax cable in 5250 environments is contained in Section 3.2.5.2, 5250 Line Interface. The selected serial data input can be inverted via the Receiver Invert [RIN] control bit.

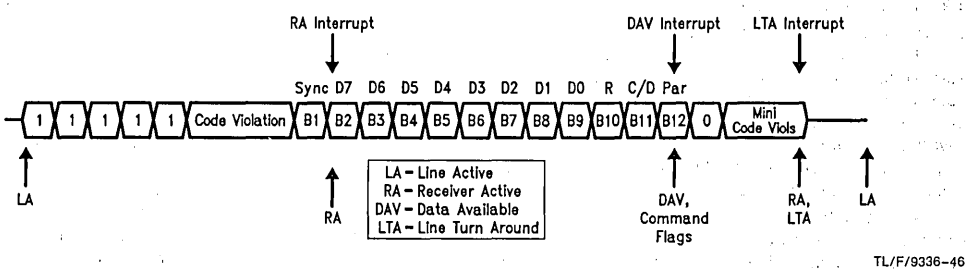
The receiver continually monitors the line, sampling at a frequency equal to eight times the expected data rate. The Line Active flag [LA] is asserted whenever an input transition is detected and will remain asserted as long as another input transition is detected within 16 TCLK cycles. If another transition is not detected in this time frame, [LA] will be de-asserted. The propagation delay from the occurrence of the edge to [LA] being set is approximately 1 transceiver clock cycle. This function is independent of the mode of operation of the transceiver; [LA] will continue to respond to input signal transitions, even if the transmitter is activated and the receiver disabled.

If the receiver is not disabled by the transmitter or by asserting [TRES], the decoder will adjust its internal timing to the incoming transitions, attempting to synchronize to valid biphase-encoded data. When synchronization occurs, the biphase clock will be extracted and the serial NRZ (Non-Return to Zero) data will be analyzed for a valid start sequence, see *Figure 3-2(b)*. The minimum number of line quiesce bits required by the receiver logic is selectable via the Receiver Line Quiesce [RLQ] control bit. If this bit is set high (the power-up condition), three line quiesce bits are required; if set low, only two are needed. Once the start sequence has been recognized, the receiver asserts the Receiver Active flag [RA] and enables the error detection circuitry. The propagation delay from the occurrence of the mid-bit edge of the sync bit in the starting sequence to [RA] being set is approximately 3 transceiver clock cycles.

The NRZ serial bit stream is now clocked into a serial to parallel shift register and analyzed according to the expected data pattern as defined by the protocol. If no errors are detected by the word parity bit, the parallel data (up to a total of 11-bits, depending on the protocol) is passed to the first location of the FIFO. It then propagates asynchronously to the last location in approximately 40 ns, at which time the Data Available flag [DAV] is asserted, indicating to the CPU that valid data is available in the FIFO. The propagation delay from the occurrence of the mid-bit edge of the parity bit of the frame to [DAV] being set is approximately 5 transceiver clock cycles.

Of the possible 11-bits in the last location of the FIFO, 8-bits (data byte) are mapped into {RTR} and the remaining bits (if any) are mapped into the Transceiver Status Register {TSR [2–0]}. The CPU accesses the data byte by reading {RTR}, and the 5250 address field or 3270 control bits by reading {TSR}. When reading the FIFO, it is important to note that {TSR} must be read before {RTR}, since reading {RTR} advances the FIFO. Once [DAV] has been recognized as set by the CPU, the data can be read by any instruction with {RTR} as the source. All instructions with {RTR} as the source (except BIT, CMP, JRMK, JMP reg-

3.0 Transceiver (Continued)



TL/F/9336-46

FIGURE 3-8. Timing of Receiver Flags Relative to Incoming Data

ister, LJMP conditional, and LCALL conditional) will result in popping the last location of the FIFO, presenting a new word (if present) for future CPU access. Data in the FIFO will propagate from one location to the next in approximately 10–15 ns, therefore the CPU is easily able to unload the FIFO with a set of consecutive instructions.

If the received bit stream is a multi-byte message, the receiver will continue to process the data and load the FIFO. After the third load (if the CPU has not accessed the FIFO), the Receive FIFO Full flag [RFF] will be asserted. The propagation delay from the occurrence of the mid-bit edge of the parity bit of the frame to [RFF] being set is approximately 5 transceiver clock cycles. If there are more than 3 frames in the incoming message, the CPU has approximately one frame time (sync bit to start of parity bit) to start unloading the FIFO. Failure to do so will result in an overflow error condition and a resulting loss of data (see Receiver Errors).

If there are no errors detected, the receiver will continue to process the incoming frames until the end of message is detected. The receiver will then return to an inactive state, clearing [RA] and asserting the Line Turn-Around flag, [LTA] indicating that a message was received with no errors. The propagation delay from the occurrence of the edge starting the first minicode violation to [RA] cleared and [LTA] set is approximately 17 transceiver clock cycles in 3270, 3299, and 8-bit modes. In 5250 modes, the assertion of [LTA] and clearing of [RA] are dependent on how the transmission line ends after the transmission of the three required fill bits (see 5250 Modes). For the 3270 and 3299 protocols, [LTA] can be used to initiate an immediate transmitter FIFO load; for the other protocols, an appropriate response delay time may be needed. [LTA] is cleared by loading the transmitter's FIFO, writing a one to [LTA] in the Network Command flag register, or by asserting [TRES].

Receiver Errors

If the Receiver Active flag, [RA], is asserted by the receiver logic, the selected receiver input source is continuously checked for errors, which are reported to the CPU by asserting the Receiver Error flag, [RE], and setting the appropriate receiver error flag in the Error Code Register {ECR}. If a condition occurs which results in multiple errors being created, only the first error detected will be latched into {ECR}. Once an error has been detected and the appropriate error flag has been set, the receiver is disabled, clearing [RA] and preventing the Line Turn-Around flag and interrupt [LTA] from being asserted. The Line Active flag [LA] remains asserted if signal transitions continue to be detected on the input.

5 error flags are provided in {ECR}:

7	6	5	4	3	2	1	0
rsv	rsv	rsv	OVF	PAR	IES	LMBT	RDIS

[OVF] **Overflow**—Asserted when the decoder writes to the first location of the FIFO while [RFF] is asserted. The word in the first location will be over-written; there will be no effect on the last two locations.

[PAR] **Parity Error**—Asserted when a received frame fails an even (word) parity check.

[IES] **Invalid Ending Sequence**—Asserted during an expected end sequence when an error occurs in the mini code-violation. Not valid in 5250 modes.

[LMBT] **Loss of Mid-Bit Transition**—Asserted when the expected biphase-encoded mid-bit transition does not occur within the expected window. Indicates a loss of receiver synchronization.

[RDIS] **Receiver Disabled While Active**—Asserted when an active receiver is disabled by the transmitter being activated.

To determine which error has occurred, the CPU must read {ECR}. This is accomplished by asserting the Select Error Codes control bit, [SEC], and reading {RTR}. The {ECR} is only 5 bits wide, therefore the upper 3 bits are still the output of the receive FIFO (see *Figure 3-6)*. All instructions with {ECR} as the source (except BIT, CMP, JRMK, JMP register, LJMP conditional, and LCALL conditional) will clear the error condition and return the receiver to idle, allowing the receiver to again monitor the incoming data stream for a new start sequence. The [SEC] control bit must be de-asserted to read the FIFO's data from {RTR}.

If data is present in the FIFO when the error occurs, the Data Available flag [DAV] is de-asserted when the error is detected and re-asserted when {ECR} is read. Data present in the FIFO before the error occurred is still available to the CPU. The flexibility is provided, therefore, to read the error type and still recover data loaded into the FIFO before the error occurred. The Transceiver Reset, [TRES] can be asserted at any time, clearing both Transceiver FIFOs and the error flags.

3.0 Transceiver (Continued)

3.2.3 Transceiver Interrupts

The transceiver has access to 3 CPU interrupt vectors, one each for the transmitter and receiver, and a third, the Line Turn-Around interrupt, providing a fast turn around capability between receiver and transmitter. The receiver interrupt is the CPU's highest priority interrupt (excluding NMI), followed by the transmitter and Line Turn-Around interrupts, respectively. The three interrupt vector addresses and a full description of the interrupts are given in Table 3-2.

The receiver interrupt is user-selectable from 4 possible sources (only 3 used at present) by specifying a 2-bit field, the Receiver Interrupt Select bits [RIS1-0] in the Interrupt Control Register {ICR}. A full description is given in Table 3-3.

The RFF + RE interrupt occurs only when the receive FIFO is full (or an error is detected). If the number of frames in a received message is not exactly divisible by 3, one or two words could be left in the FIFO at the end of the message, since the CPU would receive no indication of the presence of that data, it is recommended that this interrupt be used together with the line turn-around interrupt, whose service routine can include a test for whether any data is present in the receive FIFO.

For additional information concerning interrupts, refer to Sections 2.1.1.3, Interrupt Control Registers, and 2.2.3, Interrupts.

3.2.4 Protocol Modes

3270/3299 Modes

As shown in Table 3-1, the transceiver can operate in 4 different 3270/3299 modes, to accommodate applications of the BCP in different positions in the network. The 3270 mode is designed for use in a device or a controller which is not in a multiplexed environment. For a multiplexed network, the 3299 multiplexer and controller modes are designed for each end of the controller to multiplexer connection, the 3299 repeater mode being used for an in-line repeater situated between controller and multiplexer.

For information on how parallel data loaded into the transmit FIFO and unloaded from the receive FIFO maps into the serial bit positions, see *Figure 3-9*.

To transmit a frame, {TCR [3–0]} must first be set up with the correct control information, after which the data byte can be written to {RTR}. The resulting composite 12-bit word is loaded into the transmit FIFO where it propagates through to the last location to be loaded into the encoder and formatted for transmission.

When formatting a 3270 frame, {TCR [2]} controls whether the transmitter is required to format a data frame or a command frame. If {TCR [2]} is low, the transmitter logic calcu-

TABLE 3-2. Transceiver Interrupts

Interrupt	Vector Address	Description
Receiver	000100	User selectable from 4 possible sources, see Table 3-3.
Transmitter	001000	Set when [TFE] asserted, indicating that the transmit FIFO is empty, cleared by writing to {RTR}. **Note:** [TRES] causes [TFE] to be asserted.
Line Turn-Around	001100	Set when a valid end sequence is detected, cleared by writing to {RTR}, writing a one to [LTA], or asserting [TRES]. In 5250 modes, interrupt is set when the last fill bit has been received and no further input transitions are detected. Will not be set in 5250 or 8-bit non-promiscuous modes unless an address match was received.

The interrupt vector is obtained by concatenating {IBR} with the vector address as shown:

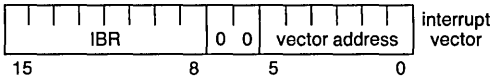


TABLE 3-3. Receiver Interrupts

Interrupt	RIS1,0	Description
RFF+RE	0 0	Set when [RFF] or [RE] asserted. If activated by [RFF], indicating that the receive FIFO is full, interrupt is cleared by reading from {RTR}. If activated by [RE], indicating that an error has been detected, interrupt is cleared by reading from {ECR}.
DAV+RE	0 1	Set when [DAV] or [RE] asserted. If activated by [DAV], indicating that valid data is present in the receive FIFO, interrupt is cleared by reading from {RTR}. If activated by [RE], indicating that an error has been detected, interrupt is cleared by reading from {ECR}.
Not Used	1 0	Reserved for future product enhancement.
RA	1 1	Set when [RA] asserted, indicating the receipt of a valid start sequence, cleared by reading {ECR} or {RTR}.

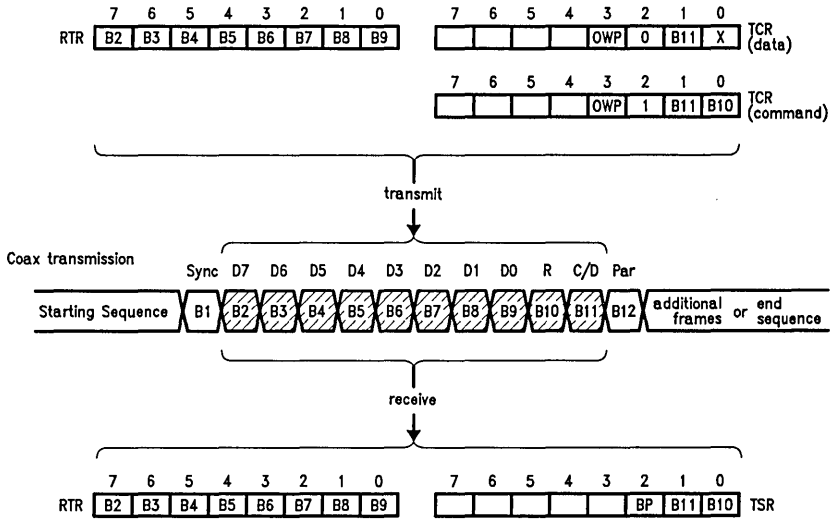
All receiver interrupts can be cleared by asserting [TRES].

3.0 Transceiver (Continued)

lates odd parity on the data byte (B2–B9) and transmits this value for B10. If {TCR [2]} is high, B10 takes the state of {TCR [0]}. Odd Word Parity [OWP] controls the type of parity calculated on B1–B11 and transmitted as B12, the frame delimiter. If [OWP] is high, odd parity is output; otherwise even parity is transmitted. In this manner the system designer is provided with maximum flexibility in defining the transmitted 3270 control bits (B10–B12).

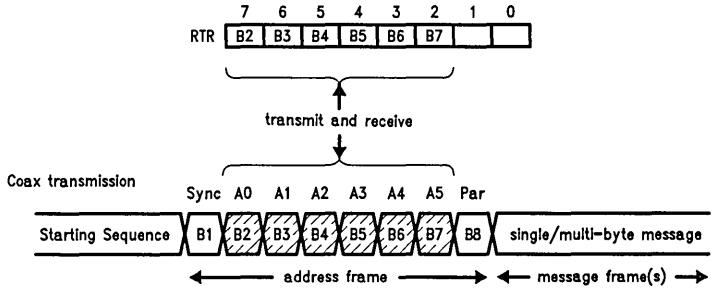
When data is written to {RTR}, the least significant 4 bits of {TCR} are loaded into the FIFO along with the data being written to {RTR}. The same {TCR} contents can therefore be used for more than one frame of a multi-frame transmission, or changed for each frame.

When a 3270 frame is received and decoded, the decoder loads the parallel data into the receive FIFO where it propagates through to the last location and is mapped into {RTR} and {TSR}. Bits B2–B11 are exactly as received; Byte Parity [BP] is odd parity on B2–B9, calculated in the decoder. Reading {RTR} will advance the receive FIFO, therefore {TSR} must be read first if this information is to be utilized.



TL/F/9336–47

(a) 3270 Data and Command Frames



TL/F/9336–48

(b) 3299 Address Frame

FIGURE 3-9. 3270/3299 Frame Assembly/Disassembly Procedure

3.0 Transceiver (Continued)

When formatting a 3299 address frame, the procedure is the same as for a 3270 frame, with {RTR [7–2]} defining the address to be transmitted. The only bit in {TCR} which has any functional meaning in this mode is [OWP], which controls the type of parity required on B1–B8. Similarly, when the receiver de-formats a 3299 address frame, the received address bits are loaded into {RTR [7–2]}; {RTR [1–0]} and {TSR [2–0]} are undefined.

The POLL, POLL/ACK and TT/AR flags in the Network Command Flag Register are valid only in 3270 and 3299 (excluding the 3299 address frame) modes. These flags are decodes of their respective coax commands as defined in Table 3-4. The Data Error or Message End [DEME] flag (also in the {NCF} register) indicates different information depending on the selected protocol. In 3270 and 3299, [DEME] is set when B10 of the received frame does not match the locally generated odd parity on bits B2–B9 of the received frame. [DEME] is not part of the receiver error logic, it functions only as a status flag to the CPU. These flags are decoded from the last location in the FIFO and are valid only when [DAV] is asserted; they are cleared by reading {RTR} and must be checked before advancing the receiver FIFO.

5250 Modes

The biphase data is inverted in the 5250 protocol relative to 3270/3299 (see the Protocol section—IBM 5250). Depending on the external line interface circuitry, the transceiver's biphase inputs and outputs may need to be inverted by asserting the [RIN] (Receiver INvert) and [TIN] (Transmitter INvert) control bits in {TMR}.

For information on how data must be organized in {TCR} and {RTR} for input to the transmitter, and how data extracted from a received frame is organized by the receiver and mapped into {TSR} and {RTR}, see *Figure 3-10.*

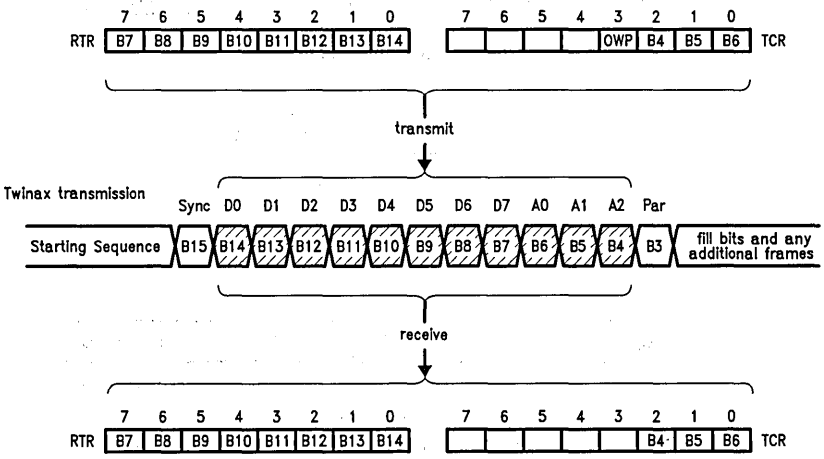
To transmit a 5250 message, the least significant 4 bits of {TCR} must first be set up with the correct address and parity control information. The station address field (B4–B6) is defined by {TCR[2–0]}, and [OWP] controls the type of parity (even or odd) calculated on B4–B15 and transmitted as B3. When the 8-bit data byte is written to {RTR}, the resulting composite 12-bit word is loaded into the transmit FIFO, starting the transmitter. The same {TCR} contents can be used for more than one frame of a multi-frame transmission, or changed for each frame.

The 5250 protocol defines bits B0–B2 as fill bits which the transmitter automatically appends to the parity bit (B3) to

TABLE 3-4. Decode of 3270 Coax Commands

Received Word										Flag	Description
B2	B3	B4	B5	B6	B7	B8	B9	B10	B11		
0	0	0	0	0	0	0	0	0	0	RAR	TT/AR (Clean Status) Received
X	X	X	1	0	0	0	1	X	1	ACK	POLL/ACK Command Received
X	X	X	0	0	0	0	1	X	1	POLL	POLL Command Received

All flags cleared by reading {RTR}.



TL/F/9336-49

FIGURE 3-10. 5250 Frame Assembly/Disassembly Description

3.0 Transceiver (Continued)

form the 16-bit frame. Additional fill bits may be inserted between frames of a multi-frame transmission by loading the fill bit register, {FBR}, with the one's complement of the number of fill bits to be transmitted. A value of FF (hex), corresponds to the addition of no extra fill bits. At the conclusion of a message the transmitter will return to the idle state after transmitting the 3 fill bits of the last frame (no additional fill bits will be transmitted).

As shown in Table 3-1, the transceiver can operate in 2 different 5250 modes, designated "promiscuous" and "non-promiscuous". The transmitter operates in the same manner in both modes.

In the promiscuous mode, the receiver passes all received data to the CPU via the FIFO, regardless of the station address. The CPU must determine which station is being addressed by reading {TSR [2–0]} before reading {RTR}.

In the non-promiscuous mode, the station address field (B4–B6) of the first frame must match the 3 least significant bits of the Auxiliary Transceiver Register, {ATR [2–0]}, before the receiver will pass the data on to the CPU. If no match is detected in the first frame of a message, and if no errors were found on that frame, the receiver will reset to idle, looking for a valid start sequence. If an address match is detected in the first frame of a message, the received data is passed on to the CPU. For the remainder of the message all received frames are decoded in the same manner as the promiscuous mode.

To maintain maximum flexibility, the receiver logic does not interpret the station address or command fields in determining the end of a 5250 message. The message typically ends with no further line transitions after the third fill bit of the last frame. This end of message must be distinguished from a loss of synchronization between frames of a multi-byte transmission condition by looking for line activity some time after the loss of synchronization occurs. When the loss of synchronization occurs during fill bit reception, the receiver monitors the Line Active flag, [LA], for up to 11 biphase bit times (11 μs at the 1 MHz data rate). If [LA] goes inactive at any point during this period, the receiver returns to the idle state, de-asserting [RA] and asserting [LTA]. If, however, [LA] is still asserted at the end of this window, the receiver interprets this as a real loss of synchronization and flags the [LMBT] error condition to the CPU. (See Receiver Errors in this section.)

In the 5250 modes, the Data-Error-or-Message-End [DEME] flag is a decode of the 111 station address (the end of message delimiter) and is valid only when [DAV] is asserted. This function allows the CPU to quickly determine when the end of message has been received.

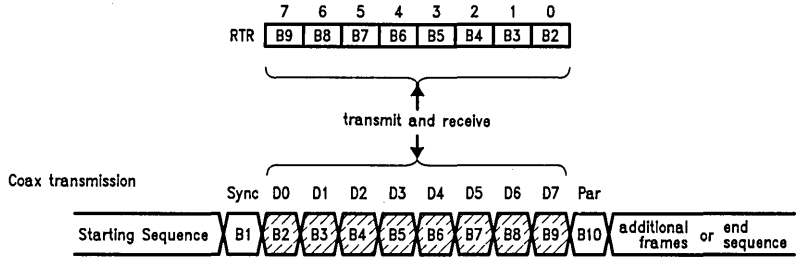
The transmitter has the flexibility of holding TX-ACT active at the end of a 5250 message, thus reducing line reflections and ringing during this critical time period. The amount of hold time is programmable from 0 μs to 15.5 μs in 500 ns increments (assuming TCLK is 8 MHz), and is set by writing the selected value to the upper 5-bits of the Auxiliary Transceiver Register, {ATR [7–3]}.

General Purpose 8-Bit Modes

As shown in Table 3-1, the transceiver can operate in 2 different 8-bit modes, designated "promiscuous" and "non-promiscuous". In the non-promiscuous mode, the first frame data byte (B2–B9) must match the contents of {ATR[7–0]} before the receiver will load the FIFO and assert [DAV]. If no match is made on the first frame, and if no errors were found on that frame, the receiver will go back to idle, looking for a valid start sequence. The address comparator logic is not enabled in the promiscuous mode, and therefore all received frames are passed through the receive FIFO to the CPU. The transmitter operates in the same manner in both modes.

The serial bit positions relative to the parallel data loaded into the transmit FIFO and presented to the CPU by the receiver FIFO are shown in *Figure 3-11*. To transmit a frame, the data byte is written to {RTR}, loading the transmit FIFO where it propagates through to the last location to be loaded into the encoder and formatted for transmission. Only [OWP] in {TCR} is loaded into the transmitter FIFO in both protocol modes; {TCR [2–0]} are don't cares. B10 is defined by a parity calculation on B1–B9; odd if [OWP] is high and even if [OWP] is low.

When a frame is received, the decoder loads the processed data into the receive FIFO where it propagates through to the last location and is mapped into {RTR}. All bits are exactly as received. Reading the data is accomplished by reading {RTR}. {TSR [2–0]} are undefined in the 8-bit modes.



TL/F/9336-50

FIGURE 3-11. General Purpose 8-Bit Frame Assembly/Disassembly Procedure

3.0 Transceiver (Continued)

3.2.5 Line Interface

3.2.5.1 3270 Line Interface

In the 3270 environment, data is transmitted between a control unit and a device via a single coax cable or twisted pair cable. The coax type is RG62AU with a maximum length of 1.5 kilometers. The twisted pair cable has become more prevalent to reduce cabling and routing costs. Typically, a 24 AWG unshielded twisted pair is used to achieve the cost reduction goals. The length of the twisted pair cable is a minimum of 100 feet to a maximum of 900 feet. The 3270 protocol utilizes a transformer to isolate the peripheral from the cabling system.

An effective line interface design must be able to accept either coax or twisted pair cabling and compensate for noise, jitter and reflections in the cabling system. There must be an adequate amount of jitter tolerance to offset the effects of filtering and noise. Some filtering is needed to reduce ambient noise caused by surrounding hardware. Such filtering must not introduce transients that the receiver comparator translates into data jitter.

An effective driver design should also attempt to compensate for the filtering effects of the cable. Higher data frequencies become attenuated more than lower frequency signals as cable length is increased, yielding greater disparity in the amplitudes of these signals. This effect generates greater jitter at the receiver. The 3270 signal format allows for a high voltage (predistorted) magnitude and a low voltage (nondistorted) magnitude within each data bit time. Increasing the predistorted-to-nondistorted signal level ratio counteracts the filtering phenomenon because the lower frequency signals contain less predistortion than do higher frequency signals. Thus, the amplitude of the higher frequency signals is "boosted" more than the lower frequency signals. Unfortunately, a low signal level is more susceptible to reflection-induced errors at short cable length. Proper impedance matching and slower edge rates must be utilized to eliminate as much reflection as possible at these lengths.

Additionally, shielded or balanced operation must be adequately supported. Shielded operation implies the use of coax cable, where balanced implies the use of twisted pair cable. Proper termination should be employed, and a termination slightly greater than the characteristic impedance of theline may actually provide more desirable waveforms than a perfectly matched termination. Board layout should make the comparator lines as short as possible. Lines should be placed closely together to avoid the introduction of differential noise. These lines should not pass near "noisy" lines. A ground plane should isolate all "noisy" lines.

BCP Design

The line interface design for the receiver is shown in *Figure 3-12.* An offset of approximately 17 mV separates the comparator inputs, making the receiver more immune to ambient noise present on the circuit board. A 2:1:1 (arranged as a 3:1) transformer increases any voltage sensitivity lost by introducing the offset. A bandpass filter is employed to reduce edge rate to the comparator and eliminate ambient noise. The bandwidth (30 kHz to 30 MHz) was chosen to provide sufficient attenuation for noise while producing minimum data jitter.

The driver design, *Figure 3-13,* incorporates a National Semiconductor DS3487 and a resistor network to generate the proper signal levels. The predistorted-to-nondistorted ratio was chosen to be about 3 to 1. The coax/twisted pair front end, *Figure 3-14,* includes an ADC brand connector to switch between coax and twisted pair cable. The coax interface has the shield capacitively coupled to ground. The 510Ω resistor and the filter loading produce a termination of about 95Ω. The twisted pair interface balances both lines and possesses an input impedance of about 100Ω. This termination is somewhat higher than the characteristic impedance (about 96Ω) of twisted pair. Terminations of this type produce reflections that do not tend to generate mid-bit errors. Such terminations have the benefit of creating a larger voltage at the receiver over longer cable lengths. For a more detailed explanation of the 3270 line interface, see Application Note "A Combined Coax/Twisted Pair 3270 Line Interface for the DP8344 Biphase Communications Processor".

3.2.5.2 5250 Line Interface

The 5250 environment utilizes twinax in a multi-drop configuration, where eight devices can be "daisy-chained" over a total distance of 5,000 feet and eleven splices, (each physical device is considered a splice). Twinax connectors are bulky and expensive, but are very sturdy. Twinaxial cable is a shielded twisted pair that is nearly ⅓ of an inch thick.

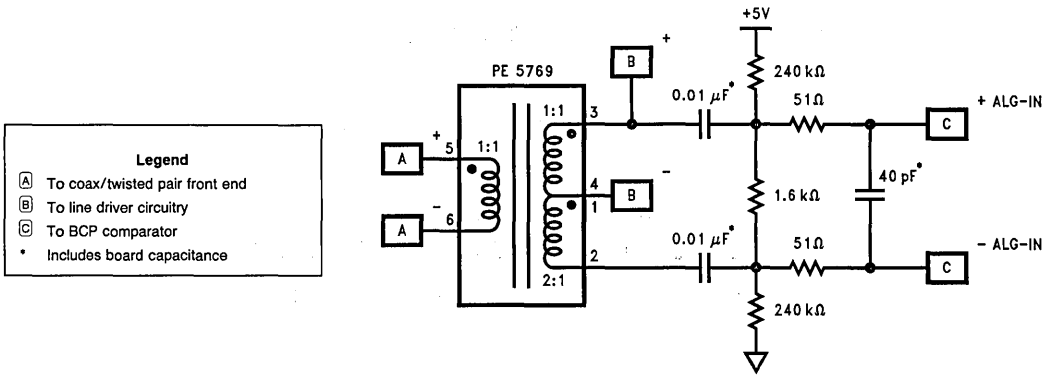


FIGURE 3-12. BCP Receiver Design

3.0 Transceiver (Continued)

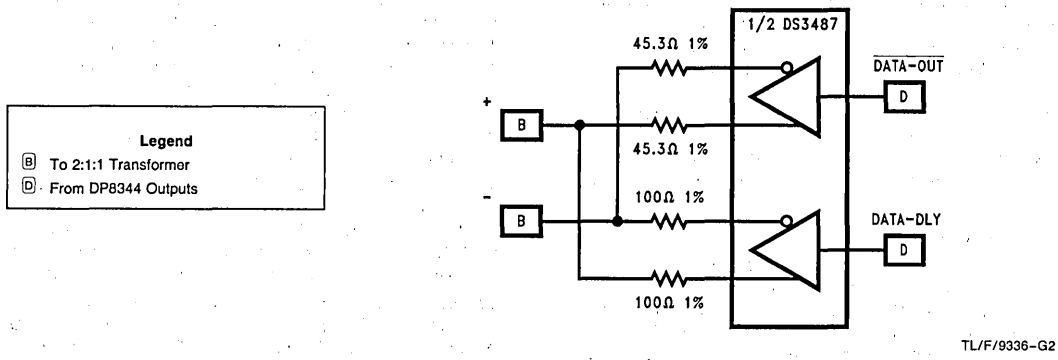


FIGURE 3-13. BCP Driver Design

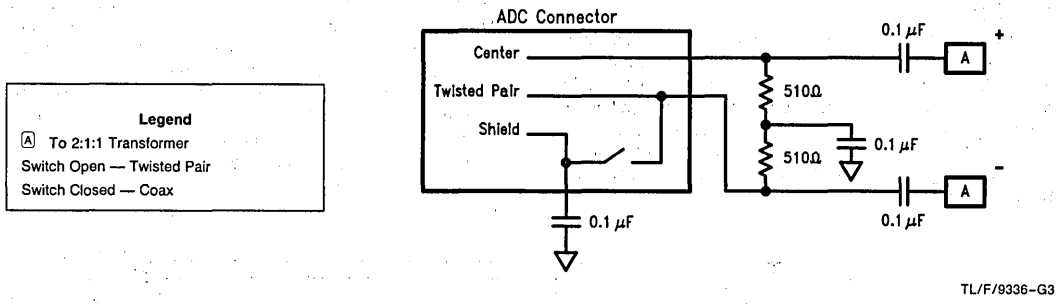


FIGURE 3-14. BCP Coax/Twisted Pair Front End

The cable shield must be continuous throughout the transmission system, and be grounded at the system unit and each station. Since twinax connectors have exposed metal connected to their shield grounds, care must be taken not to expose them to noise sources. The polarity of the two inner conductors must also be maintained throughout the transmission system.

The transmission system is implemented in a balanced current mode; every receiver/transmitter pair is directly coupled to the twinax at all times. Data is impressed on the transmission line by unbalancing the line voltage with the driver current. The system requires passive termination at both ends of the transmission line. The termination resistance value is given by:

$R_t = Z_{O/2}$; where

R_t: Termination Resistance

Z_O: Characteristic Impedance

In practice, termination is accomplished by connecting both conductors to the shield via 54.9Ω, 1% resistors; hence the characteristic impedance of the twinax cable of 107Ω ±5% at 1.0 MHz. Intermediate stations must not terminate the line; each is configured for "pass-through" instead of "terminate" mode. Stations do not have to be powered on to pass twinax signals on to other stations; all of the receiver/transmitter pairs are DC coupled. Consequently, devices must never output any signals on the twinax line during power-up or down that could be construed as data, or interfere with valid data transmission between other devices.

Driver Circuits for the DP8344B

The transmitter interface on the DP8344B is sufficiently general to allow use in 3270, 5250, and 8-bit transmission systems. Because of this generality, some external hardware is needed to adapt the outputs to form the signals necessary to drive the twinax line. The chip provides three signals: $\overline{\text{DATA-OUT}}$, DATA-DLY and TX-ACT. $\overline{\text{DATA-OUT}}$ is biphase serial data (inverted). DATA-DLY is the biphase serial data output (non-inverted) delayed one-quarter bit-time. TX-ACT, or transmitter active, signals that serial data is being transmitted when asserted. $\overline{\text{DATA-OUT}}$ and DATA-DLY can be used to form the A and B phase signals with their three levels by the circuit shown in *Figure 3-15.* TX-ACT is used as an external transmitter enable. The BCP can invert the sense of the $\overline{\text{DATA-OUT}}$ and DATA-DLY signals by asserting [TIN] {TMR[3]}. This feature allows both 3270 and 5250 type biphase data to be generated, and/or utilization of inverting on non-inverting transmitter stages.

Drivers for the 5250 environment may not place any signals on the transmission system when not activated. The power-on and off conditions of drivers must be prevented from causing noise on the system since other devices may be in operation. *Figure 3-15* shows a "DC power good" signal enabling the driver circuit. This signal will lock out conduction in the drivers if the supply voltage is out of tolerance.

Twinax signals can be viewed as consisting of two distinct phases, phase A and phase B, each with three levels, off,

3.0 Transceiver (Continued)

high and low. The off level corresponds with 0 mA current being driven, the high level is nominally 62.5 mA, +20% −30%, and the low level is nominally 12.5 mA, +20% −30%. When these currents are applied to a properly terminated transmission line the resultant voltages impressed at the driver are: off level is 0V, low level is 0.32V ±20%, high level is 1.6V ±20%. The interface must provide for switching of the A and B phases and the three levels. A bimodal constant current source for each phase can be built that has a TTL level interface for the BCP.

Receiver Circuits

The pseudo-differential mode of the twinax signals make receiver design requirements somewhat different than the coax 3270 world. Hence, the analog receiver on the BCP is not well suited to receiving twinax data. The BCP provides both analog inputs to an on-board comparator circuit as well as a TTL level serial data input, DATA-IN. The sense of this serial data can be inverted by the BCP by asserting [RIN], {TMR[4]}.

The external receiver circuit must be designed with care to ensure reliable decoding of the bit-stream in the worst environment. Signals as small as 100 mV must be detected. In order to receive the worst case signals, the input level switching threshold or hysteresis for the receiver should be nominally 29 mV ±20%. This value allows the steady state, worst case signal level of 100 mV ±66% of its amplitude before transitioning.

To achieve this, a differential comparator with complementary outputs can be applied, such as the National LM361. The complementary outputs are useful in setting the hysteresis or switching threshold to the appropriate levels. The LM361 also provides excellent common mode noise rejection and a low input offset voltage. Low input leakage current allows the design of an extremely sensitive receiver, without loading the transmission line excessively.

In addition to good analog design techniques, a low pass filter with a roll-off of approximately 1 MHz should be applied to both the A and B phases. This filter essentially conducts high frequency noise to the opposite phase, effectively making the noise common mode and easily rejectable.

Layout considerations for the LM361 include proper bypassing of the ±12V supplies at the chip itself, with as short as possible traces from the pins to 0.1 μF ceramic capacitors. Using surface mount chip capacitors reduces lead inductance and is therefore preferable in this case. Keeping the input traces as short and even in length is also important. The intent is to minimize inductance effects as well and standardize those effects on both inputs. The LM361 should have as much ground plane under and around it as possible. Trace widths for the input signals especially should be as wide as possible; 0.1 inch is usually sufficient. Finally, keep all associated discrete components nearby with short routing and good ground/supply connections.

For a more detailed explanation of the 5250 line interface, see application note "Interfacing the DP8344 to Twinax."

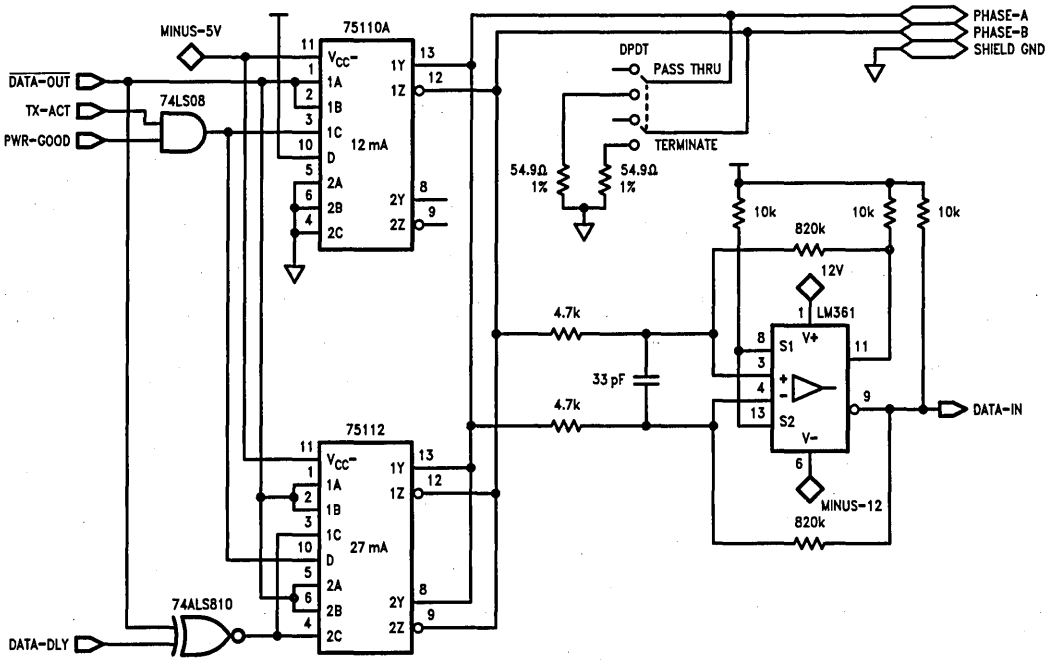


FIGURE 3-15. 5250 Line Interface Schematic

4.0 Remote Interface and Arbitration System (RIAS)

INTRODUCTION

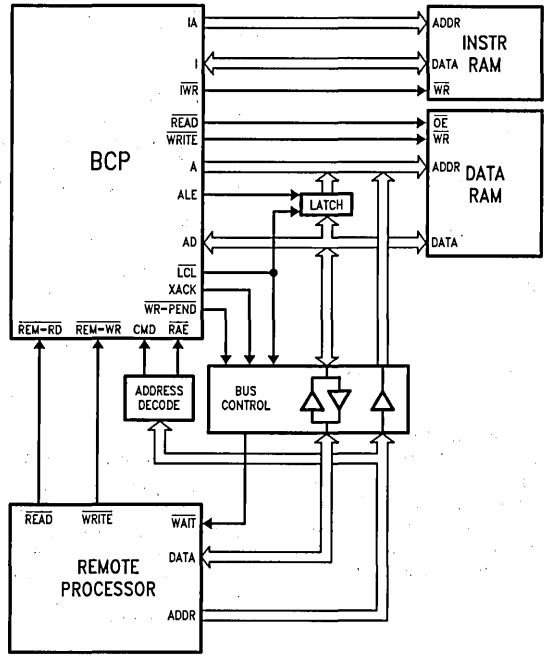
Communication with the BCP is based on the BCP's ability to share its data memory. A microprocessor (or any intelligent device) can read and write to any BCP data location while the BCP CPU is executing instructions. This capability is part of the BCP's Remote Interface and Arbitration System (RIAS). Sharing data memory is possible because RIAS's arbitration logic allocates use of the BCP's data and address buses. RIAS has been designed so that accesses of BCP data memory by another device minimally impact its performance as well as the BCP's. In addition to data memory accesses, RIAS allows another device to control how BCP programs are loaded, started and debugged.

4.1 RIAS ARCHITECTURAL DESCRIPTION

Interfacing to the BCP is accomplished with the control signals listed in Table 4-1. *Figure 4-1* shows the BCP interfaced to Instruction Memory, Data Memory, and an intelligent device, termed the Remote Processor (RP). Instruction and Data are separate memory systems with separate address buses and data paths. This arrangement allows continuous instruction fetches without interleaved data accesses. Instruction Memory (IMEM) is interfaced to the BCP through the Instruction (I) and Instruction Address (IA) buses. IMEM is 16 bits wide and can address up to 64k memory. Data Memory (DMEM) is eight bits wide and can also address up to 64k memory. The DMEM address is formed by the 8-bit upper byte (A bus) and the 8-bit lower byte (AD bus). The AD bus must be externally latched because it also serves as the path for data between the BCP and DMEM. For further information on how AD bus is used, refer to Section 2.2.2 CPU Timing.

The Remote Processor's address and data buses are connected to the BCP's address and data buses through the bus control circuitry. The RP's address lines decode a chip select for the BCP called Remote Access Enable ($\overline{RAE}$). Basically, the BCP's Data Memory has been memory mapped into the RP's memory. A Remote Access of the BCP occurs when $\overline{REM\text{-}RD}$ or $\overline{REM\text{-}WR}$, along with $\overline{RAE}$ is asserted low. $\overline{REM\text{-}RD}$ and $\overline{REM\text{-}WR}$ can be directly connected to the Remote Processor's read and write lines, or for more complicated systems the $\overline{REM\text{-}RD}$ and $\overline{REM\text{-}WR}$ signals may be controlled by a combination of address decode and the RP's read and write signals. To the RP, an access of the BCP will appear as any other memory system access. This configuration allows the RP to read and write Data Memory, read and write the BCP's Program Counter, and read and write BCP Instruction Memory. These functions are selected by control bits in the Remote Interface Configuration register {RIC}. This register can be accessed only by the RP and not by the BCP CPU. If the Remote Processor executes a remote access with the Command input (CMD) high, {RIC} is accessed through the BCP's AD bus.

In *Figure 4-1*, the Remote Processor's address lines are decoded to form the CMD input. When a remote access takes place with CMD low, the memory system designated in {RIC} is accessed. *Figure 4-2* shows the contents of {RIC}. The two least significant bits are the Memory Select bits [MS1–0] which designate the type of remote access: to Data Memory, the Program Counter, or Instruction Memory. This register also contains the BCP start bit [STRT], three interface select bits [FBW, LR, LW], the Single-Step bit [SS], and the Bi-directional Interrupt Status bit [BIS]. Refer to the RIAS Reference Section for a more detailed description of the contents of this register and the function of each bit.



TL/F/9336–19

FIGURE 4-1. BCP/Remote Processor Interface

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

TABLE 4-1. RIAS Inputs and Outputs

Signal	In/Out	Pin	Reset State	Function
CMD	In	45	X	**CoMmanD** input. When high, remote accesses are directed to the Remote Interface Configuration register, {RIC}. When low, remote accesses are directed to Data Memory, Instruction Memory or the Program Counter as determined by {RIC [1,0]}.
$\overline{\text{LCL}}$	Out	31	0	**LoCaL**. Normally low, goes high when the BCP relinquishes the data and address bus to service a remote access.
$\overline{\text{LOCK}}$	In	44	X	Asserting this input Low will **LOCK** out local (BCP) accesses to Data Memory. Once the remote processor has been granted the bus, $\overline{\text{LOCK}}$ gives it sole access to the bus and BCP accesses are "waited".
$\overline{\text{RAE}}$	In	46	X	**R**emote **A**ccess **E**nable. Setting this input low allows host access of BCP functions and memory.
$\overline{\text{REM-RD}}$	In	47	X	**REM**ote **ReaD**. When low along with $\overline{\text{RAE}}$, a remote read cycle is requested; serviced by the BCP when the data bus becomes available.
$\overline{\text{REM-WR}}$	In	48	X	**REM**ote **WR**ite. When low along with $\overline{\text{RAE}}$, a remote write cycle is requested; serviced by the BCP when the data bus becomes available.
$\overline{\text{WR-PEND}}$	Out	49	1	**WR**ite **PEND**ing. In a system configuration where remote write cycles are latched, $\overline{\text{WR-PEND}}$ will go low, indicating that the latches contain valid data which have yet to be serviced by the BCP.
XACK	Out	50	1	Transfer **ACK**nowledge. Normally high, goes low on $\overline{\text{REM-RD}}$ or $\overline{\text{REM-WR}}$ going low (if $\overline{\text{RAE}}$ low) returning high when the transfer is complete. Normally used as a "wait" signal to a remote processor. (In the Latched Write mode, XACK will only transition if a second remote access begins before the first one completes.)
$\overline{\text{WAIT}}$	In	54	X	Asserting this input low will add wait states to both remote accesses and to the BCP instruction cycle. $\overline{\text{WAIT}}$ will extend a remote access until it is set high.

7	6	5	4	3	2	1	0	
BIS	SS	FBW	LR	LW	STRT	MS1	MS0	RIC

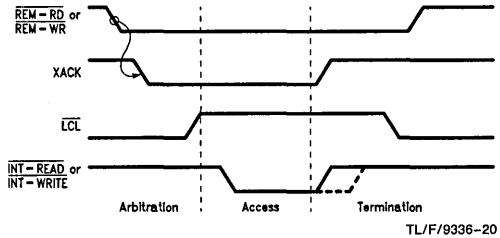
BIS —Bidirectional Interrupt Status
SS —Single-Step
FBW —Fast Buffered Write mode
LR —Latched Read mode
LW —Latched Write mode
STRT —BCP CPU start/stop
MS1–0 —Memory Selection

FIGURE 4-2. Remote Interface Control Register

4.1.1 Remote Arbitration Phases

The BCP CPU and RIAS share the internal CPU-CLK. This clock is derived from the X1 crystal input. It can be divided by two by setting [CCS] = 1 in {DCR} or run undivided by setting [CCS] = 0. The frequency at which the Remote Processor is run need not bear any relationship to the CPU-CLK. A remote access is treated as an asynchronous event and data is handshaked between the Remote Processor and the BCP.

The two key handshake signals involved in the BCP/RP interface are Transfer Acknowledge (XACK) and Local ($\overline{\text{LCL}}$). Internally, two more signals control the access timing: $\overline{\text{INT-READ}}$ and $\overline{\text{INT-WRITE}}$. The timing for a generic Remote Access is shown in *Figure 4-3*. A remote access is



TL/F/9336-20

FIGURE 4-3. Generic Remote Access ($\overline{\text{RAE}}$ = 0)

initiated by the RP asserting $\overline{\text{REM-RD}}$ or $\overline{\text{REM-WR}}$ with $\overline{\text{RAE}}$ low. There is no set-up/hold time relationship between $\overline{\text{RAE}}$ and $\overline{\text{REM-RD}}$ or $\overline{\text{REM-WR}}$. These signals are internally gated together such that if RAE (REM-RD + REM-WR) is true, a remote access will begin. A short delay later, XACK will fall. This signal can be fed back to the RP's wait line to extend its read or write cycle, if necessary. When the BCP's

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

arbitration logic determines that the BCP is not using data memory, $\overline{LCL}$ rises, relinquishing control of the address and data buses to the RP. The remote access can be delayed at most one BCP instruction (providing [LOR] is not set high). If the CPU is executing a string of data memory accesses, RIAS has an opportunity to break in at the completion of every instruction. The time period between $\overline{REM\text{-}RD}$ or $\overline{REM\text{-}WR}$ being asserted (with $\overline{RAE}$ low) and $\overline{LCL}$ rising is called the Arbitration Phase. It is a minimum of one T-state, but can be increased if the BCP CPU is accessing Data Memory (local access) or if the BCP has set the Lock Out Remote bit [LOR].

The CMD pin is internally latched on the first falling edge of the CPU-CLK after a remote access has been initiated by asserting $\overline{RAE}$ low along with asserting $\overline{REM\text{-}RD}$ or $\overline{REM\text{-}WR}$ low. If the remote interface is asynchronous, the CMD signal must be valid simultaneously or before $\overline{RAE}$ is asserted low along with $\overline{REM\text{-}RD}$ or $\overline{REM\text{-}WR}$ being asserted low. The value of CMD is only sampled once during each remote access and will remain in effect for the duration of the remote access.

After the Arbitration Phase has ended, the Access Phase begins. Either Data Memory, Instruction Memory, the Program Counter, or {RIC} is read or written in this phase. Either $\overline{INT\text{-}READ}$ or $\overline{INT\text{-}WRITE}$ will fall one T-state after $\overline{LCL}$ rises. These two signals provide the timing for the different types of accesses. $\overline{INT\text{-}READ}$ times the transitions on the AD bus for Remote Reads and forms the external $\overline{READ}$ line. $\overline{INT\text{-}WRITE}$ clocks data into the PC and {RIC} and forms the $\overline{IWR}$ and $\overline{WRITE}$ lines. $\overline{INT\text{-}READ}$ and $\overline{INT\text{-}WRITE}$ rise with XACK, or shortly after.

The duration of the Access Phase depends on the type of memory being accessed. Data Memory and Instruction Memory accesses are subject to any programmed wait states and all remote accesses are waited by asserting $\overline{WAIT}$ low. The minimum time in the Access Phase is 2 T-states.

The rising edge of XACK indicates the Access Phase has ended and the Termination Phase has begun. If the RP was doing a read operation, this edge indicates that valid data is available to the RP. During the Termination Phase the BCP is regaining control of the buses. $\overline{LCL}$ falls one T-state after XACK and since the RP is no longer being waited, it can deassert $\overline{REM\text{-}RD}$ or $\overline{REM\text{-}WR}$. The duration of this phase is a minimum of one T-state, but can be extended depending on the interface mode chosen in {RIC}.

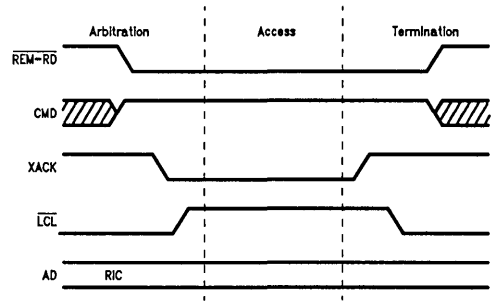
4.1.2 Access Types

There are four types of accesses an RP can make of the BCP:

—Remote Interface Control Register {RIC}
—Data Memory (DMEM)
—Program Counter (PC)
—Instruction Memory (IMEM)

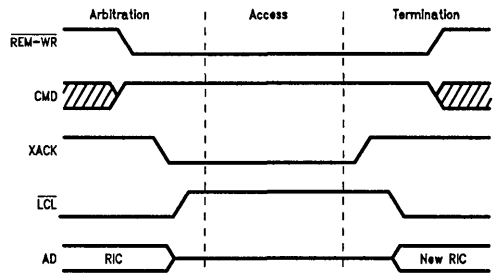
An access of {RIC} is accomplished by asserting $\overline{RAE}$ and $\overline{REM\text{-}RD}$ or $\overline{REM\text{-}WR}$ with the CMD pin asserted high. The Remote Interface Configuration register is accessed through the AD bus as shown in *Figure 4-4(c)*. A read or write of {RIC} can take place while the BCP CPU is executing instructions. Timing for this access is shown in *Figures 4-4(a)* and *(b)*. Note that in the Remote Read *Figure 4-4(a)*, AD does not transition. This is because the contents of {RIC} are active on the bus by default. The AD bus is in TRI-STATE during a Remote Write *Figure 4-4(b)* while $\overline{LCL}$ is high. The byte being written to {RIC} is latched on the rising edge of XACK and can be seen on AD after $\overline{LCL}$ falls.

The Access Phase, in this case, is always two T-states (unless $\overline{WAIT}$ is low) because {RIC} is not subject to any programmed wait states.



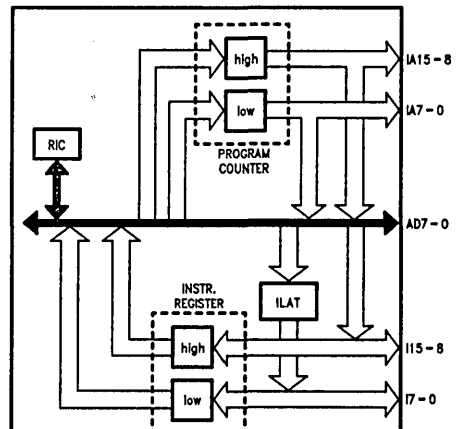
TL/F/9336-80

(a) Remote Read Timing ($\overline{RAE}$ = 0)



TL/F/9336-81

(b) Remote Write Timing ($\overline{RAE}$ = 0)



TL/F/9336-82

(c) RIC to AD Connectivity

FIGURE 4-4. Generic RIC Access

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

Remote Accesses other than to {RIC} are accomplished with the CMD pin low in conjunction with asserting $\overline{RAE}$ low along with $\overline{REM\text{-}WR}$ or $\overline{REM\text{-}RD}$ being taken low. The type of access performed is defined by the Memory Select bits in {RIC}, as shown in *Figure 4-5*.

7	6	5	4	3	2	1	0
BIS	SS	FBW	LR	LW	ST	MS1	MS0

Memory Select Bits
00 - Data Memory
01 - Instruction Memory
10 - PC low byte
11 - PC high byte

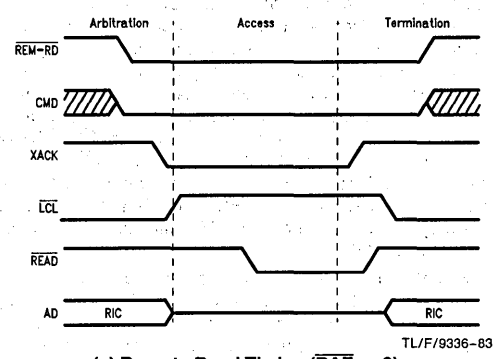
FIGURE 4-5. Memory Select Bits in {RIC}

Reads or writes of Data Memory (DMEM) are preceded by setting the Memory Select bits in {RIC} for a DMEM access: [MS1,0] = 00. After that, the RP simply reads or writes to BCP Data Memory as many times as it needs to. A DMEM access, as well as a {RIC} access, can be made while the BCP CPU is executing instructions. All other accesses must be executed with the BCP CPU stopped.

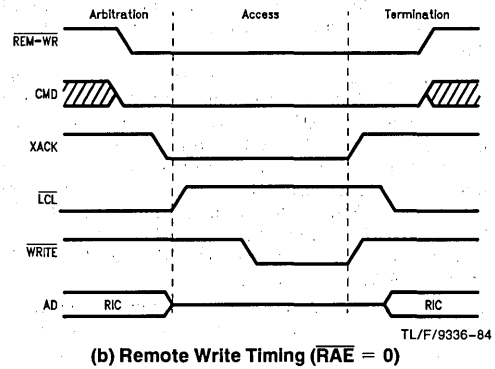
The timing for a Data Memory read and write are shown in *Figure 4-6*. The access is initiated by asserting $\overline{RAE}$ and $\overline{REM\text{-}RD}$ or $\overline{REM\text{-}WR}$ while CMD is low. The BCP responds by bringing its address and data lines into TRI-STATE and allowing the RP to control DMEM. $\overline{READ}$ is asserted in the Access Phase of a Remote Read *Figure 4-6(a)*. It will stay low for a minimum of one T-state, but can be extended by adding programmable data wait states or by taking $\overline{WAIT}$ low. $\overline{WRITE}$ is asserted in the Access Phase with a remote write. It too is a minimum of one T-state and can be increased by adding programmable wait states or by taking $\overline{WAIT}$ low.

Figure 4-7(c) shows the data path from the Program Counter to the AD bus. Both high and low PC bytes can be written or read through AD. The RP has independent control of the high and low bytes of the Program Counter—the byte being accessed is specified in the Memory Select bits. The high byte of the PC is accessed by setting [MS1–0] = 11. Setting [MS1–0] = 10 allows access to the low byte of the PC. After the Memory Select bits are set by a Remote Write to {RIC}, the byte selected can be read or written by the RP by executing a Remote Access with CMD low. Remote accesses to both the high and low bytes of the PC, as well as the instruction memory access must be executed with the BCP CPU idle. Four accesses by the RP are necessary to read or write both the high and low bytes of the PC. Timing for a PC access is shown in *Figure 4-7(a)* and *(b)*. The PC becomes valid on a Remote Read *(a)* one T-state after $\overline{LCL}$ rises and one T-state before XACK rises. AD is in TRI-STATE while $\overline{LCL}$ is high for a Remote Write *(b)*. Time in the Access Phase is two T-states if $\overline{WAIT}$ is not asserted.

Instruction memory (IMEM) is accessed through another internal path: from AD to the I bus, shown in *Figure 4-8(c)*. The memory is accessed first low byte, then high byte. Low and high bytes of the 16-bit I bus are alternately accessed for Remote Reads. An 8-bit holding register, ILAT, retains the low byte until the high byte is written by the Remote Processor for the write to IMEM. The BCP increments the PC after the high byte has been accessed.



(a) Remote Read Timing ($\overline{RAE}$ = 0)



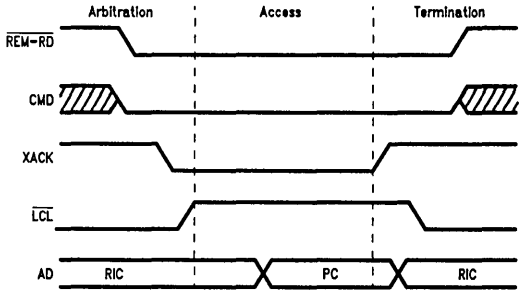
(b) Remote Write Timing ($\overline{RAE}$ = 0)

FIGURE 4-6. Generic DMEM Access

Timing for an IMEM access is shown in *Figure 4-8(a)* and *(b)*. As before, the Memory Select bits are first set to instruction memory: [MS1–0] = 01. It is only necessary to set [MS1–0] once for repeated IMEM accesses. (Instruction Memory is the power-up Memory Selection state.) A simple state machine keeps track of which instruction byte is expected next—low or high byte. The state machine powers up looking for the low instruction byte and every IMEM access causes this state machine to switch to the alternate byte. Accesses other than to IMEM will not cause the state machine to switch to the alternate byte, but writing 01 to the Memory Select bits in {RIC} (i.e. [MS1–0] = 01, pointing to IMEM) will always force the state machine to the "low byte state". This way the instruction word boundary can be reset without resetting the BCP. When the BCP is reset the state machine will also be forced to the "low byte state."

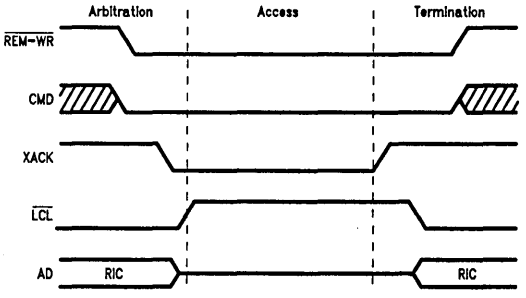
Figure 4-8(a) shows a Remote Read of Instruction memory. Both the low byte, then the high byte can be seen on back to back remote reads. An instruction byte becomes active on the AD bus one T-state after $\overline{LCL}$ rises and is valid when XACK rises. This time period will be a minimum of one T-state, but can be extended up to three more T-states by instruction wait states.

4.0 Remote Interface and Arbitration System (RIAS) (Continued)



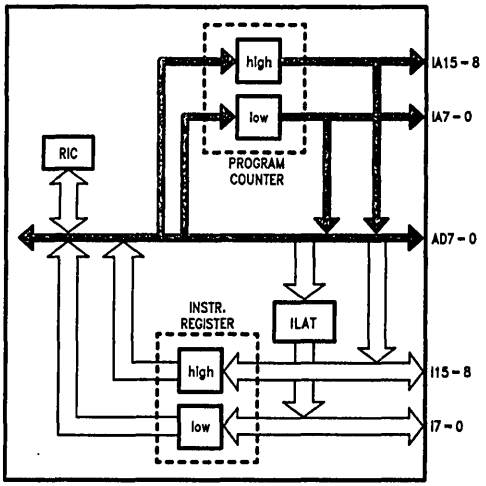
TL/F/9336-85

(a) Remote Read Timing ($\overline{RAE}$ = 0)



TL/F/9336-86

(b) Remote Write Timing ($\overline{RAE}$ = 0)

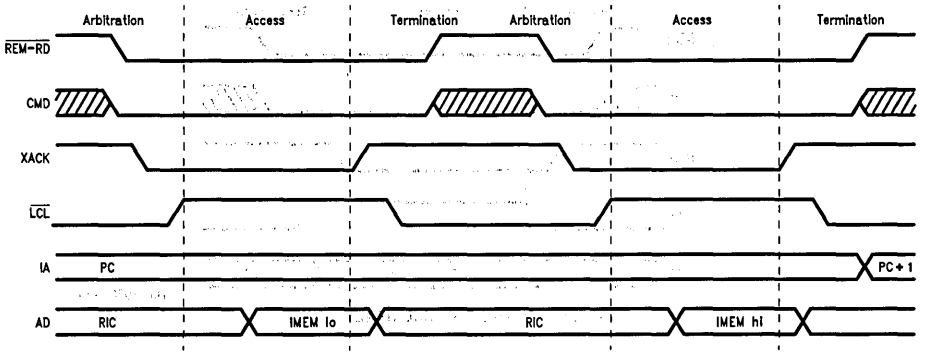


TL/F/9336-87

(c) IA to AD Connectivity

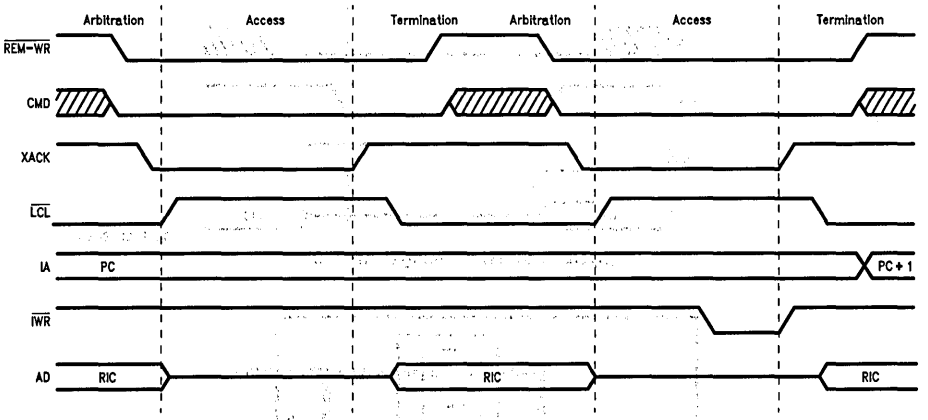
FIGURE 4-7. Generic PC Access

4.0 Remote Interface and Arbitration System (RIAS) (Continued)



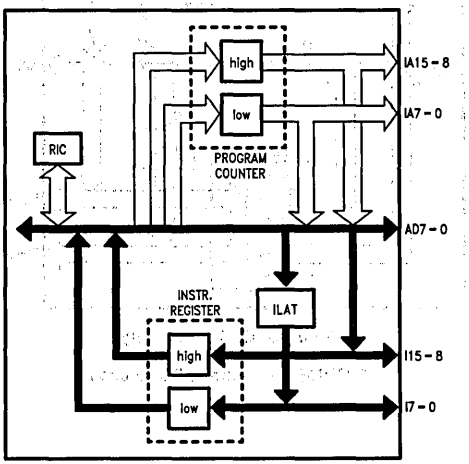
TL/F/9336-88

(a) Remote Read Timing ($\overline{RAE}$ = 0)



TL/F/9336-89

(b) Remote Write Timing ($\overline{RAE}$ = 0)



TL/F/9336-90

(c) I to AD Connectivity

FIGURE 4-8. Generic IMEM Access

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

In addition, $\overline{WAIT}$ can delay the rising edge of XACK indefinitely. One T-state after XACK rises, {RIC} will once again be active on AD. Timing is similar for a Remote Write. AD is in TRI-STATE while $\overline{LCL}$ is high. $\overline{LCL}$ is asserted for a minimum of three T-states, but can be extended by instruction wait states and the $\overline{WAIT}$ pin. $\overline{IWR}$ clocks the instruction into memory during the write of the high byte. The Instruction Address (PC) is incremented about one T-state after $\overline{LCL}$ falls on a high byte access for both Remote Reads and Writes.

Soft-loading Instruction Memory is accomplished by first setting the BCP Program Counter to the starting address of the program to be loaded. The Memory Select bits are then set to IMEM. BCP instructions can then be moved from the Remote Processor to the BCP—low byte, high byte—until the entire program is loaded.

4.1.3 Interface Modes

The Remote Interface and Arbitration System will support TRI-STATE buffers or latches between the Remote Processor and the BCP. The choice between buffers and latches depends on the type of system that is being interfaced to. Latches will help prevent the faster system from slowing to the speed of the slower system. Buffers can be used if the Remote Processor (RP) requires that data be handshaked between the systems.

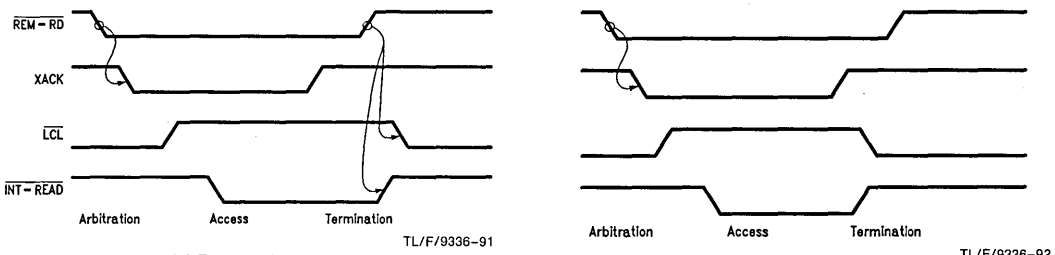
Figure 4-9 shows the timing of Remote Reads via a buffer *(a)* and a latch *(b)* (called a Buffered Read and Latched Read). The main difference in these modes is in the Termination Phase. The Buffered Read handshakes the data back to the RP. When the BCP deasserts XACK, data is valid and the RP can deassert $\overline{REM\text{-}RD}$. Only after $\overline{REM\text{-}RD}$ goes high is $\overline{LCL}$ removed. In the Latched Read *Figure 4-9(b)* XACK rises at the same time, but the Termination Phase completes without waiting for the rising edge of $\overline{REM\text{-}RD}$. One half T-state after XACK rises, $\overline{INT\text{-}READ}$ rises and one half T-state later $\overline{LCL}$ falls. The BCP can use the buses one T-state after $\overline{LCL}$ falls. The minimum time (no wait states, no arbitration delay) the BCP CPU could be prevented from using the bus is four T-states in the Latched Read Mode.

A Buffered Read prevents the BCP CPU from using the bus during the time RP is allocated the buses. This time period begins when $\overline{LCL}$ rises and ends when $\overline{REM\text{-}RD}$ is removed. If the $\overline{REM\text{-}RD}$ is asserted longer than the minimum Buffered Read execution time (four T-states), then the BCP may be unnecessarily prevented from using the buses. Therefore, if there are no overriding reasons to use the Buffered Read Mode, the Latched Read Mode is preferable.

There are three Remote Write Modes—two require buffers and one requires latches. The timing for the writes utilizing buffers is shown in *Figure 4-10*. The Slow Buffered Write *(a)* is handshaked in the same manner as the Buffered Read and thus has the same timing. The Fast Buffered Write has similar timing to the Latched Read. This timing similarity exists because the BCP terminates the remote access without waiting for the RP to deassert $\overline{REM\text{-}WR}$.

In both cases, XACK falls a short delay after $\overline{REM\text{-}WR}$ falls and $\overline{LCL}$ rises when the RP is given the buses. One T-state after $\overline{LCL}$ rises, $\overline{INT\text{-}WRITE}$ falls. The termination in the Slow Buffered Write mode keys off $\overline{REM\text{-}WR}$ rising, as shown in *Figure 4-10(a)*. $\overline{INT\text{-}WRITE}$ rises a prop-delay later and $\overline{LCL}$ falls one T-state later. The Fast Buffered Write, shown in *Figure 4-10(b)*, begins the Termination Phase with the rising edge of XACK. $\overline{INT\text{-}WRITE}$ rises at the same time as XACK, and $\overline{LCL}$ falls one T-state later. The BCP can begin a local access one T-state after $\overline{LCL}$ transitions.

A Fast Buffered Write is preferable to the Slow Buffered Write if RP's write cycles are slow compared to the minimum Fast Buffered Write execution time. The Fast Buffered Write assumes, though, that data is available to the BCP by the time $\overline{INT\text{-}WRITE}$ rises.



(a) Buffered Read

(b) Latched Read

FIGURE 4-9. Read from Remote Processor

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

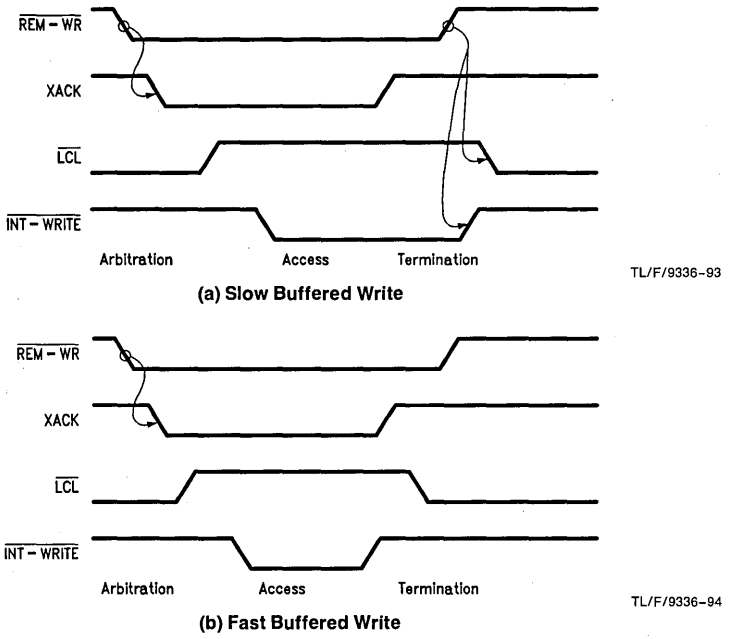


FIGURE 4-10. Buffered Write from Remote Processor

In both Buffered Write Modes, XACK is asserted to wait the RP. The Latched Write Mode makes it possible for the RP to write to the BCP without getting waited. The timing for the Latched Write Mode is shown in *Figure 4-11*. When the Remote Processor writes to the BCP, its address and data buses are externally latched on the rising edge of $\overline{\text{REM-WR}}$. Even though $\overline{\text{REM-WR}}$ has been asserted XACK does not switch. The BCP only begins remote access execution after the trailing edge of $\overline{\text{REM-WR}}$. Since the RP is not requesting data back from the BCP, it can continue execution without waiting for the BCP to complete the remote access. After $\overline{\text{REM-WR}}$ is deasserted, $\overline{\text{WR-PEND}}$ is taken low to prevent overwrite of the latches. A minimum of two T-states later $\overline{\text{LCL}}$ switches and AD, A, and the external address latch go into TRI-STATE, allowing the latches which contain the remote address and data to become active. If the RP attempts to initiate another access before the current write is complete, XACK is taken low to wait the RP and the address and the data are safe because $\overline{\text{WR-PEND}}$ prevents the latches from opening. The Access Phase ends when $\overline{\text{INT-WRITE}}$ rises and the data is written. One T-state later, $\overline{\text{LCL}}$ falls and one T-state after that $\overline{\text{WR-PEND}}$ rises. If another access is pending, it can begin in the next T-state. This is indicated by XACK rising when $\overline{\text{WR-PEND}}$ rises.

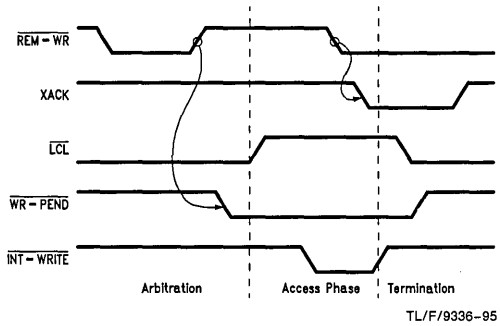
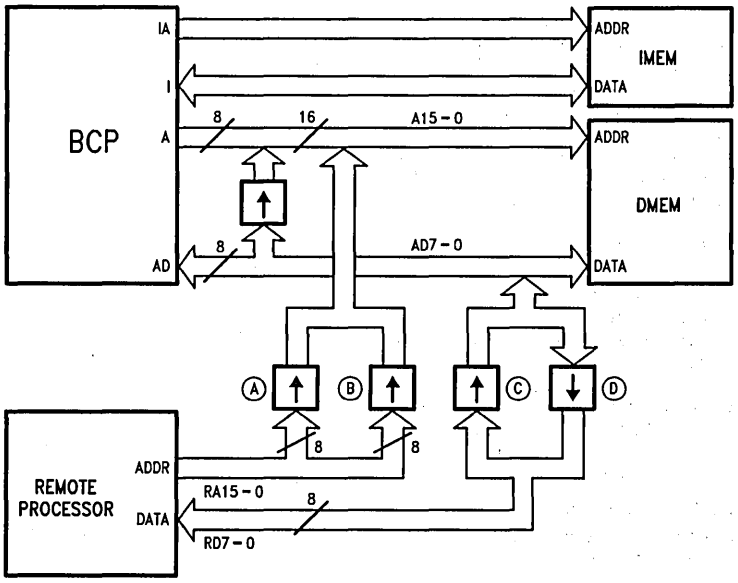


FIGURE 4-11. Latched Write from Remote Processor

A minimum BCP/RP interface utilizes four TRI-STATE buffers or latches. A block diagram of this interface is shown in *Figure 4-12*. The blocks A, B, C, and D indicate the location of buffers or latches. Blocks A and B isolate 16 bits of the RP's address bus from the BCP's Data Address bus. Two more blocks, C and D, bidirectionally isolate 8 bits of the RP's data bus from the BCP AD bus.

4.0 Remote Interface and Arbitration System (RIAS) (Continued)



TL/F/9336–96

FIGURE 4-12. Minimum BCP/Remote Processor Interface

The BCP Remote Arbitrator State Machine (RASM) must know what hardware interfaces to the RP in order to time the remote accesses correctly. To accomplish this, three Interface Mode bits in {RIC} are used to define the hardware interface. These bits are the Latched Write bit [LW], the Latched Read bit [LR] and the Fast Buffered Write bit [FBW]. See *Figure 4-13.*

7	6	5	4	3	2	1	0
BIS	SS	FBW	LR	LW	ST	MS1	MS0

Interface Mode Bits

– 0 – - Buffered Read

– 1 – - Latched Read

0 – 0 - Slow Buffered Write

1 – 0 - Fast Buffered Write

X – 1 - Latched Write

FIGURE 4-13. Interface Mode Bits

All combinations of Remote Reads or Writes with buffers or latches can be configured via the Interface Mode bits. A Buffered Read is accomplished by using a buffer for block D and setting [LR] = 0. Conversely, using a latch for block D and setting [LR] = 1 configures the RASM for Latched Reads. Using buffers for blocks A, B, and C and setting [LW] = 0 allows either a Slow or Fast Buffered Write. Setting [FBW] = 0 configures RASM for a Slow Buffered Write and [FBW] = 1 designates a Fast Buffered Write. A Latched Write is accomplished by using latches for blocks A, B, and C and setting [LW] = 1.

4.1.4 Execution Control

The BCP can be started and stopped in two ways. If the BCP is not interfaced to another processor, it can be started by pulsing $\overline{\text{RESET}}$ low while both $\overline{\text{REM-RD}}$ and $\overline{\text{REM-WR}}$ are low. Execution then begins at location zero. If there is a Remote Processor interfaced to the BCP, a write to {RIC} which sets the start bit [STRT] high will begin execution at the current PC location. Writing a zero to [STRT] stops execution after the current instruction is completed. A Single-Step is accomplished by writing a one to the Single-Step bit [SS] in {RIC}. This will execute the instruction at the current PC, increment the PC, and then return to idle. [SS] returns low after the single-stepped instruction has completed. [SS] is a write only bit and will always appear low when {RIC} is read.

Two pins ($\overline{\text{WAIT}}$ and $\overline{\text{LOCK}}$), and one register bit, [LOR], can also affect the BCP CPU or RIAS execution. The $\overline{\text{WAIT}}$ pin can be used to add wait states to a remote access. When $\overline{\text{WAIT}}$ must be asserted low to add wait states is dependent on which remote access mode is being used. The information needed to calculate when $\overline{\text{WAIT}}$ must be asserted to add wait states, is contained within the individual descriptions of the modes in the next section (4.2 RIAS Functional Description).

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

Programmed wait states delay when $\overline{WAIT}$ must be asserted since programmed wait states are inserted before $\overline{WAIT}$ is tested to see if any more wait states should be added. $\overline{LOCK}$ prevents local accesses of Data Memory. If $\overline{LOCK}$ is asserted a half T-state before T1 of a BCP instruction cycle, further local accesses will be prevented by waiting the Timing Control Unit. The Timing Control Unit (TCU) is the BCP CPU sub system responsible for timing each instruction. For a more detailed description of the operation of $\overline{LOCK}$, refer to the CPU Timing section. [LOR] allows the BCP to prevent remote accesses. Once [LOR], located in {ACR}, is set high, further remote accesses are waited by XACK remaining low.

Though the BCP CPU runs independently of RIAS there is some interaction between the two systems. [LOR] is one such interaction. In addition, two bits allow the BCP CPU to keep track of remote accesses. These bits are the Remote Write bit [RW] and the Remote Read bit [RR], and are located in {CCR[6–5]}. Each bit goes high when its respective remote access to DMEM reaches its Termination Phase. Once one of these bits has been set, it will remain high until a "1" is written to that bit to reset it low.

4.2 RIAS FUNCTIONAL DESCRIPTION

In this section, the operation of the Remote Arbitration State Machine (RASM), is described in detail. Discussed, among other things, are the sequence of events in a remote access, arbitration of the data buses, timing of external signals, when inputs are sampled, and when wait states are added. Each of the five Interface Modes is described in functional state machine form. Although each interface mode is broken out in a separate flow chart, they are all part of a single state machine (RASM). Thus the first state in each flow chart is actually the same state.

The functional state machine form is similar to a flow chart, except that transitions to a new state (states are denoted as rectangular boxes) can only occur on the rising edge of the internal CPU clock (CPU-CLK). CPU-CLK is high during the first half of its cycle. A state box can specify several actions, and each action is separated by a horizontal line. A signal name listed in a state box indicates that that pin will be asserted high when RASM has entered that state. Signals not listed are assumed low.

Note: This sometimes necessitates using the inversion of the external pin name.

This same rule applies to the A and AD buses. By default, these buses are active. The A bus will have the upper byte of the last used data address. The AD bus will display {RIC}. When one of these buses appears in a state box, the condition specified will be in effect only during that state. Decision blocks are shown as diamonds and their meaning is the same as in a flow chart. The hexagon box is used to denote a conditional state—not synchronous with the clock. When the path following a decision block encounters a conditional state, the action specified inside the hexagon box is executed immediately.

Also provided is a memory arbitration example in the form of a timing diagram for each of the five modes. These examples show back to back local accesses punctuated by a remote access. Both the state of RASM and the Timing Control Unit are listed for every clock at the top of each timing diagram. The RASM states listed correspond to the flow charts. The Timing Control Unit states are described in Section 2.2.2, Timing portion of the data sheet.

4.2.1 Buffered Read

The unique feature of this mode is the extension of the read until $\overline{REM\text{-}RD}$ is deasserted high. The complete flow chart for the Buffered Read mode is shown in *Figure 4-14.* Until a Remote Read is initiated (RAE*REM-RD true), the state machine (RASM) loops in state RS_{A1}. If a Remote Read is initiated and [LOR] is set high, RASM will move to state RS_{A2}. Likewise, if a Remote Read is initiated while the buses have been granted locally (i.e., Local Bus Request = 1), RASM will move to state RS_{A2}. The state machine will loop in state RS_{A2} as long as [LOR] is set high or the buses are granted locally. If the BCP CPU needs to access Data Memory while in either RS_A state (and $\overline{LOCK}$ is high), it can still do so. A local access is requested by the Timing Control Unit asserting the Local Bus Request (LCL-BREQ) signal. A local bus grant will be given by RASM if the buses are not being used (as is the case in the RS_A states).

XACK is taken low as soon as RAE*REM-RD is true, regardless of an ongoing local access. If [LOR] is low, RASM will move into RS_B on the next clock after RAE*REM-RD is true and there is no local bus request. No further local bus requests will be granted until the remote access is complete and RASM returns to RS_A. Half a T-state after entering RS_B the A bus (and AD bus if the access is to Data Memory) goes into TRI-STATE.

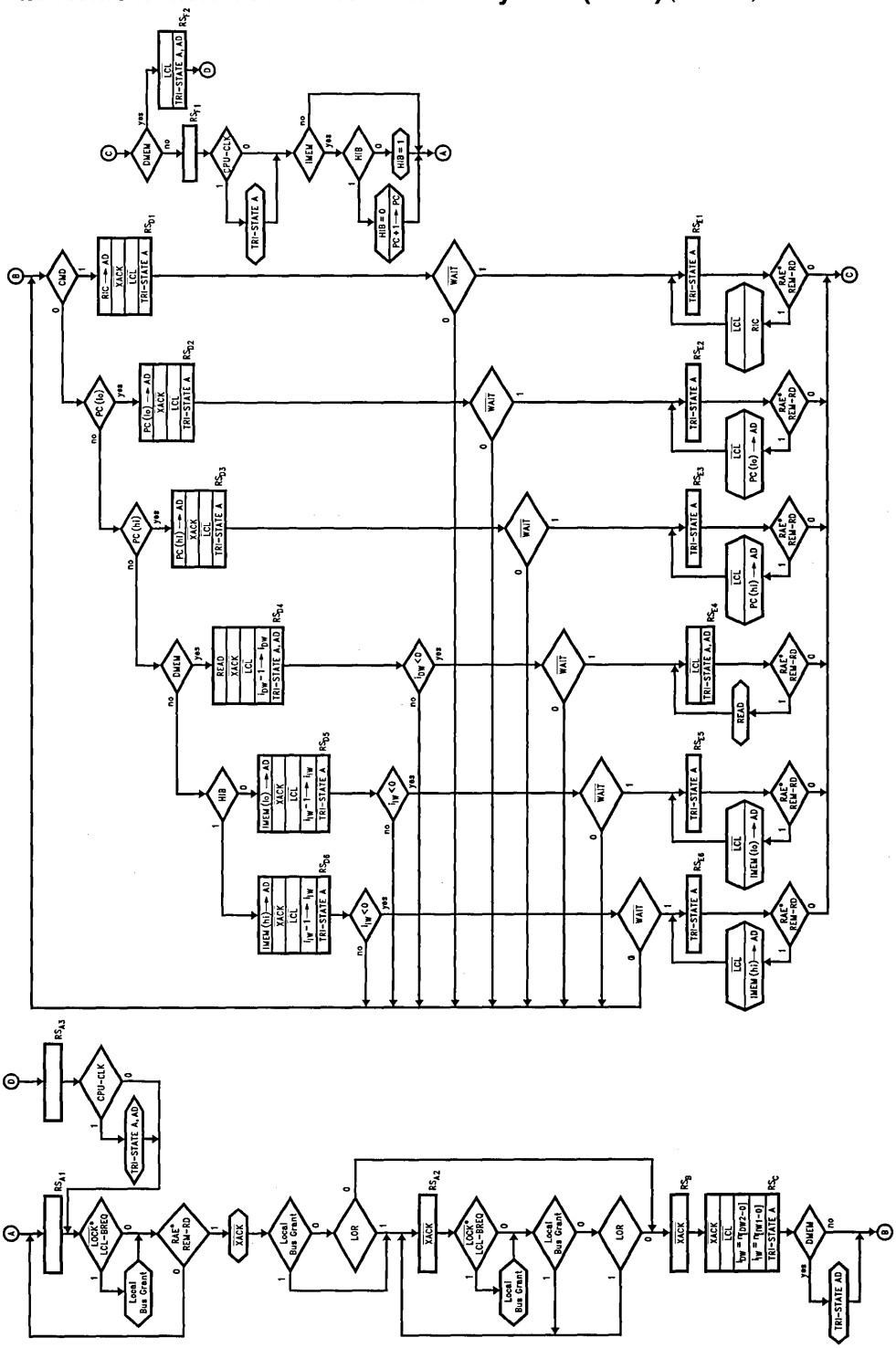
On the next CPU-CLK, RASM enters RS_C and $\overline{LCL}$ is taken high while XACK remains low. The wait state counters, i_{IW} and i_{DW}, are loaded in this state from [IW1–0] and [DW2–0], respectively, in {DCR}. The A bus (and AD if the access is to Data Memory) remains in TRI-STATE and the Access Phase begins.

The state machine can move into one of several states, depending on the state of CMD and [MS1–0], on the next clock. XACK remains low and $\overline{LCL}$ remains high in all the possible next states. If CMD is high, the access is to {RIC} and the next state will be RS_{D1}. Since the default state of AD is {RIC}, it will not transition in this state.

The five other next states all have CMD low and depend on the Memory Select bits. If [MS1–0] is 10 or 11 the state machine will enter either RS_{D2} or RS_{D3} and the low or high bytes of the Program Counter, respectively, will be read.

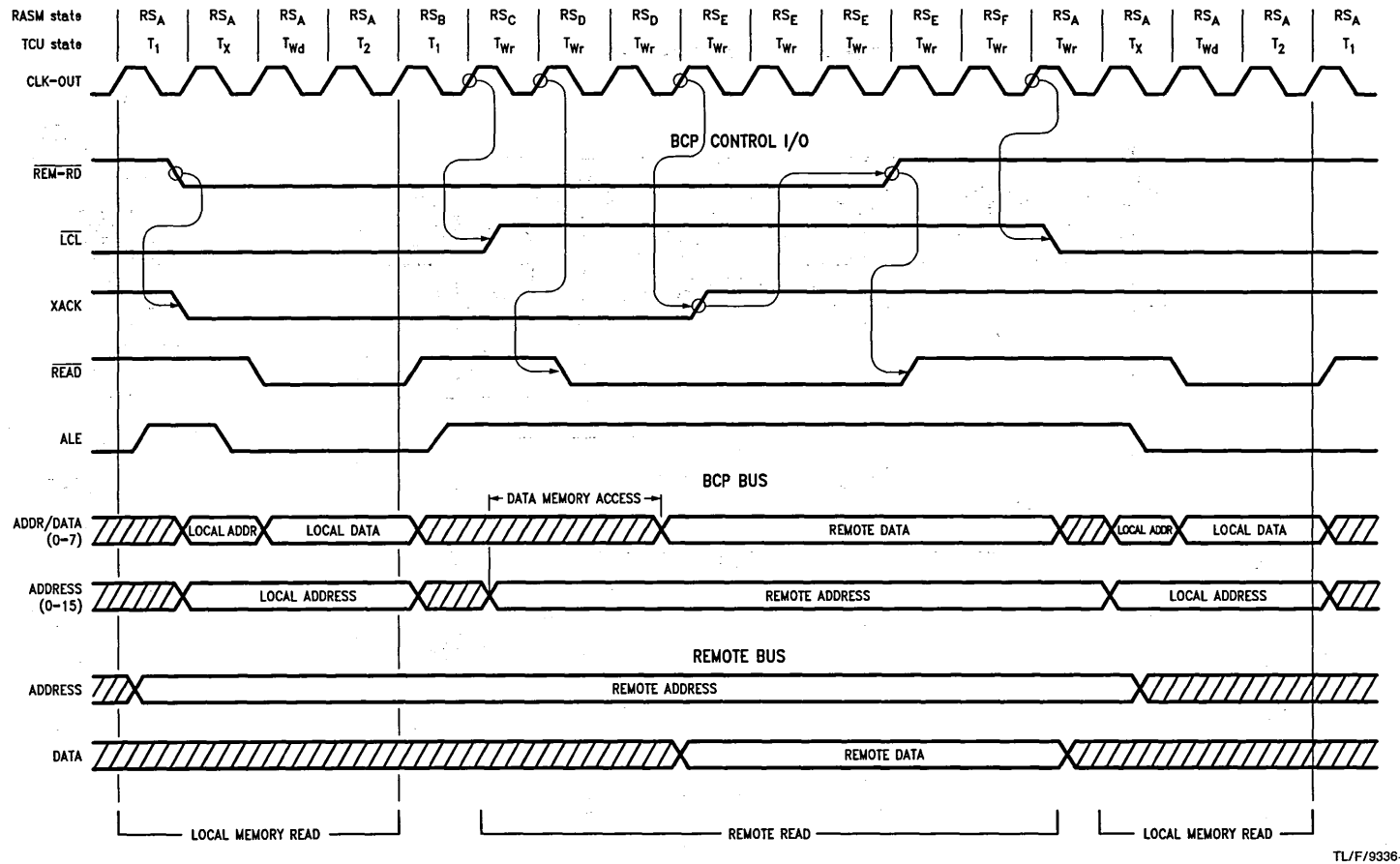
[MS1–0] = 00 designates a Data Memory access and moves RASM into RS_{D4}. $\overline{READ}$ will be asserted in this state and A and AD continue to be in TRI-STATE. This allows the Remote Processor to drive the Data Memory address for the read. Since DMEM is subject to wait states, RS_{D4} is looped upon until all the wait states have been inserted.

4.0 Remote Interface and Arbitration System (RIAS) (Continued)



TL/F/9336-97

FIGURE 4-14. Flow Chart of Buffered Read Mode



Register Configuration:

—One Wait-State Programmed for Data-Memory
—Zero Wait-States Programmed for Instruction-Memory
—{RIC} Contents: XXX0X100
—[LOR] = 0

Other BCP Control Signals:

$\overline{RAE}$	=0
CMD	=0
$\overline{REM\text{-}WR}$	=1
$\overline{LOCK}$	=1

FIGURE 4-15. Buffered Read of Data Memory by Remote Processor

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

The last possible Memory Selection is Instruction Memory, [MS1–0] = 01. The two possible next states for an IMEM access depend on if RASM is expecting the low byte or high byte. Instruction words are accessed low byte then high byte and RASM powers up expecting the low Instruction byte. The internal flag that keeps track of the next expected Instruction byte is called the High Instruction Byte flag (HIB). If HIB is low, the next state is RS_{D5} and the low instruction byte is MUXed to the AD bus. If HIB is high, the high instruction byte is MUXed to AD and RS_{D6} is entered. An IMEM access, like a DMEM access, is subject to wait states and these states will be looped on until all programmed instruction memory wait states have been inserted.

Note: Resetting the BCP will reset HIB (i.e., HIB = 0). Writing 01 to the Memory Select bits in {RIC} (i.e., [MS1–0] = 01, pointing to IMEM) will also force HIB to zero. This way the instruction word boundary can be reset without resetting the BCP.

After all of the programmed wait states are inserted in the RS_D states, more wait states may be added by asserting $\overline{WAIT}$ low a half T-state before the end of the last programmed wait state. If there are no programmed wait states, $\overline{WAIT}$ must be asserted low a half T-state before the end of RS_D to add wait states. If $\overline{WAIT}$ remains low, the remote access is extended indefinitely. All the RS_D states move to their corresponding RS_E states on the CPU-CLK after the programmed wait state conditions are met and $\overline{WAIT}$ is high. The RS_E states are looped upon until RAE* REM-RD is deasserted. $\overline{LCL}$ remains high in all RS_E states and A remains in TRI-STATE. AD will also stay in TRI-STATE if the access was to DMEM. XACK is taken back high to indicate that data is now valid on the read. If XACK is connected to a Remote Processor wait pin, it is no longer waited and can now terminate its read cycle. This state begins the Termination Phase. The action specified in the conditional box is only executed while RAE*REM-RD is asserted—a clock edge is not necessary. In all RS_E states except RS_{E4} (DMEM) LCL will fall a propagation delay after RAE*REM-RD is deasserted. In RS_{E4}, $\overline{LCL}$ remains high through the whole state.

On the CPU-CLK after RAE*REM-RD is deasserted. RASM, enters RS_{F1} from every RS_E state except RS_{E4} (DMEM). In RS_{F1}, $\overline{LCL}$ remains low and A remains in TRI-STATE while CPU-CLK is high (i.e., for the first half T-state of RS_{F1}).

From RS_{E4}, RASM enters RS_{F2} on the CPU-CLK after RAE*REM-RD is deasserted. In RS_{F2}, $\overline{LCL}$ remains high while both A and AD remain in TRI-STATE.

From RS_{F1}, the next clock will return the state machine back to state RS_{A1} where it will loop until another Remote Access is initiated. If the access was to IMEM, then the last action of the remote access before returning to RS_A is to switch HIB and increment the PC if the high byte was read.

From RS_{F2}, the next CPU-CLK returns to state RS_{A3} where $\overline{LCL}$ returns low, but A and AD remain TRI-STATE for the first half T-state of RS_{A3}. If no Remote Access is initiated the next state will be RS_{A1} where it will loop until another Remote Access is initiated.

The example in *Figure 4-15* shows the BCP executing the first of two consecutive Data Memory reads when $\overline{REM\text{-}RD}$ goes low. In response, XACK goes low waiting the remote processor. At the end of the first instruction, although the BCP begins its second read by taking ALE high, the RASM now takes control of the bus and takes $\overline{LCL}$ high at the end of T_1. A one T-state delay is built into this transfer to ensure that $\overline{READ}$ has been deasserted before the data bus is switched. The Timing Control Unit is now waited, inserting remote access wait states, T_{Wr}, as RASM takes over.

The remote address is permitted one T-state to settle on the BCP address bus before $\overline{READ}$ goes low, XACK then returns high one T-state plus the programmed Data Memory wait state, T_{Wd} later, having satisfied the memory access time. The Remote Processor will respond by deasserting $\overline{REM\text{-}RD}$ high to which the BCP in turn responds by deasserting $\overline{READ}$ high. Following $\overline{READ}$ being deasserted high, the BCP waits till the end of the next T-state before taking $\overline{LCL}$ low, again ensuring that the read cycle has concluded before the bus is switched. Control is then returned to the Timing Control Unit and the local memory read continues.

4.2.2 Latched Read

This mode differs from the Buffered Read mode in the way the access is terminated. A latched Read cycle ends after the data being read is valid and the termination doesn't wait for the trailing edge of $\overline{REM\text{-}RD}$. Therefore the Arbitration and Access Phases of the Latched Read mode are the same as for the Buffered Read mode. The complete flow chart for the Latched Read mode is shown in *Figure 4-16*.

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

Until a Remote Read is initiated (RAE*REM-RD true), the state machine (RASM) loops in state RS_{A1}. If a Remote Read is initiated and [LOR] is set high, RASM will move to state RS_{A2}. Likewise, if a Remote Read is initiated while the buses have been granted locally (i.e., Local Bus Grant = 1), RASM will move to state RS_{A2}. The state machine will loop in state RS_{A2}, as long as [LOR] is set high or the buses are granted locally. If the BCP CPU needs to access Data Memory while in either RS_A state (and $\overline{LOCK}$ is high), it can still do so. A local access is requested by the Timing Control Unit asserting the Local Bus Request (LCL-BREQ) signal. A local bus grant will be given by RASM if the buses are not being used (as is the case in RS_A).

XACK is taken low as soon as RAE*REM-RD is true, regardless of an ongoing local access. If [LOR] is low, RASM will move into RS_B on the next clock after RAE*REM-RD is asserted and there is no local bus request. No further local bus requests will be granted until RASM enters the Termination Phase. If the BCP CPU initiates a Data Memory access after RS_A, the Timing Control Unit will be waited and the BCP CPU will remain in state T_{Wr} until the remote access reaches the Termination Phase. Half a T-state after entering RS_B the A bus (and AD bus if the access is to Data Memory) goes into TRI-STATE.

On the next clock, RASM enters RS_C and $\overline{LCL}$ is taken high while XACK remains low. The wait state counters, i_{IW} and i_{DW}, are loaded in this state from [IW1–0] and [DW2–0], respectively, in {DCR}. The A bus (and AD if the access is to Data Memory) now remains TRI-STATE and the Access Phase begins.

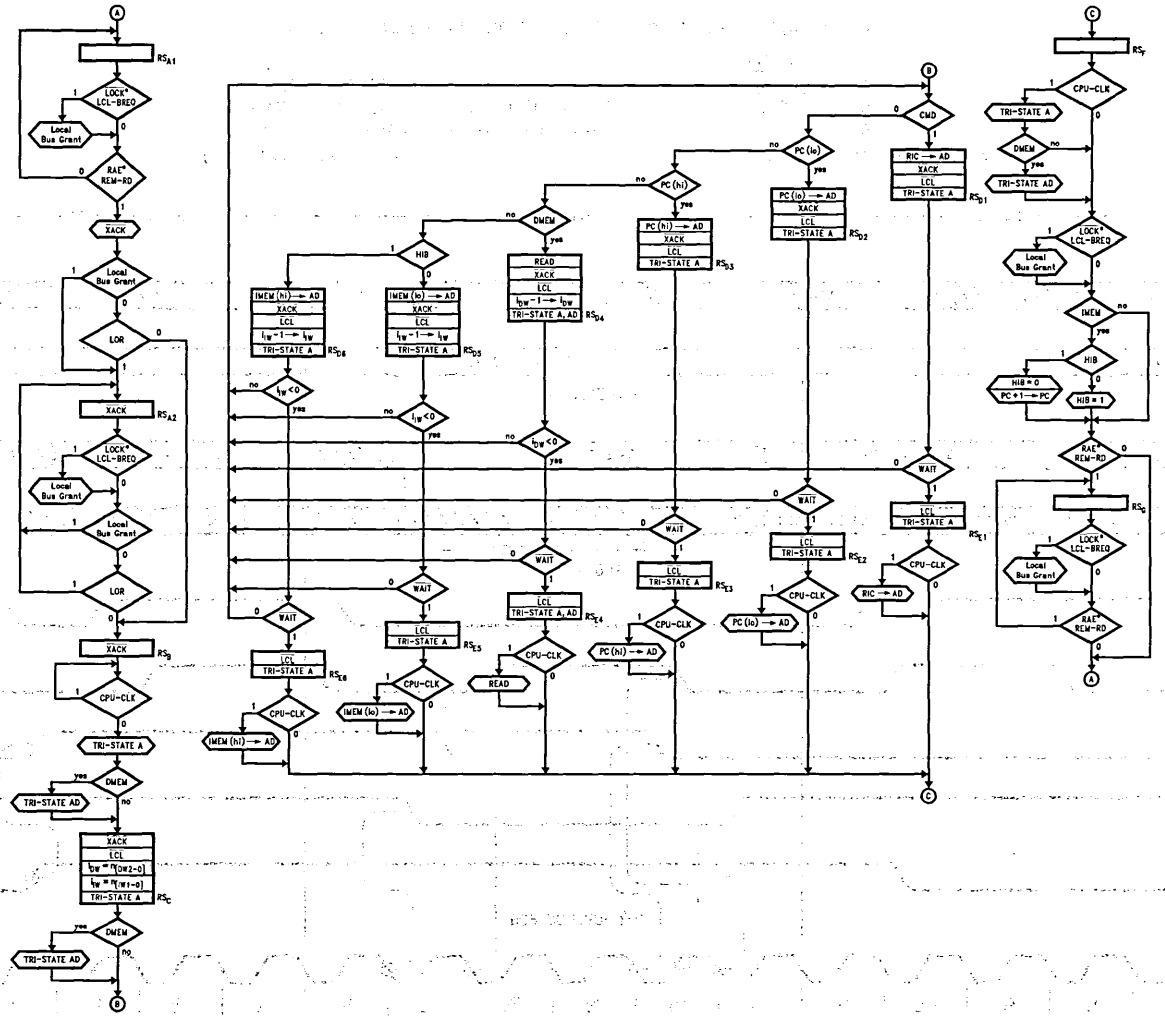
The state machine can move into one of several states, depending on the state of CMD and [MS1–0], on the next clock. XACK remains low and $\overline{LCL}$ remains high in all the possible next states. If CMD is high, the access is to {RIC} and the next state will be RS_{D1}. Since the default state of AD is {RIC}, it will not transition in this state. The five other next states all have CMD low and depend on the Memory Select bits. If [MS1–0] is 10 or 11 the state machine will enter either RS_{D2} or RS_{D3} and the low or high bytes of the Program Counter, respectively, will be read.

[MS1–0] = 00 designates a Data Memory access and moves RASM into RS_{D4}. $\overline{READ}$ will be asserted low in this state and A and AD continue to be tri-stated. This allows the Remote Processor to drive the Data Memory address for the read. Since DMEM is subject to wait states, RS_{D4} is looped upon until all the wait states have been inserted.

The last possible Memory Selection is Instruction Memory, [MS1–0] = 01. The two possible next states for the IMEM access depend on if RASM is expecting the low byte or high byte. Instruction words are accessed low byte then high byte and RASM powers up expecting the low Instruction byte. The internal flag that keeps track of the next expected Instruction byte is called the High Instruction Byte flag (HIB). If HIB is low, the next state is RS_{D5} and the low instruction byte is MUXed to the AD bus. If HIB is high, the high instruction byte is MUXed to AD and RS_{D6} is entered. An IMEM access, like a DMEM access, is subject to wait states and these states will be looped on until all programmed instruction memory wait states have been inserted.

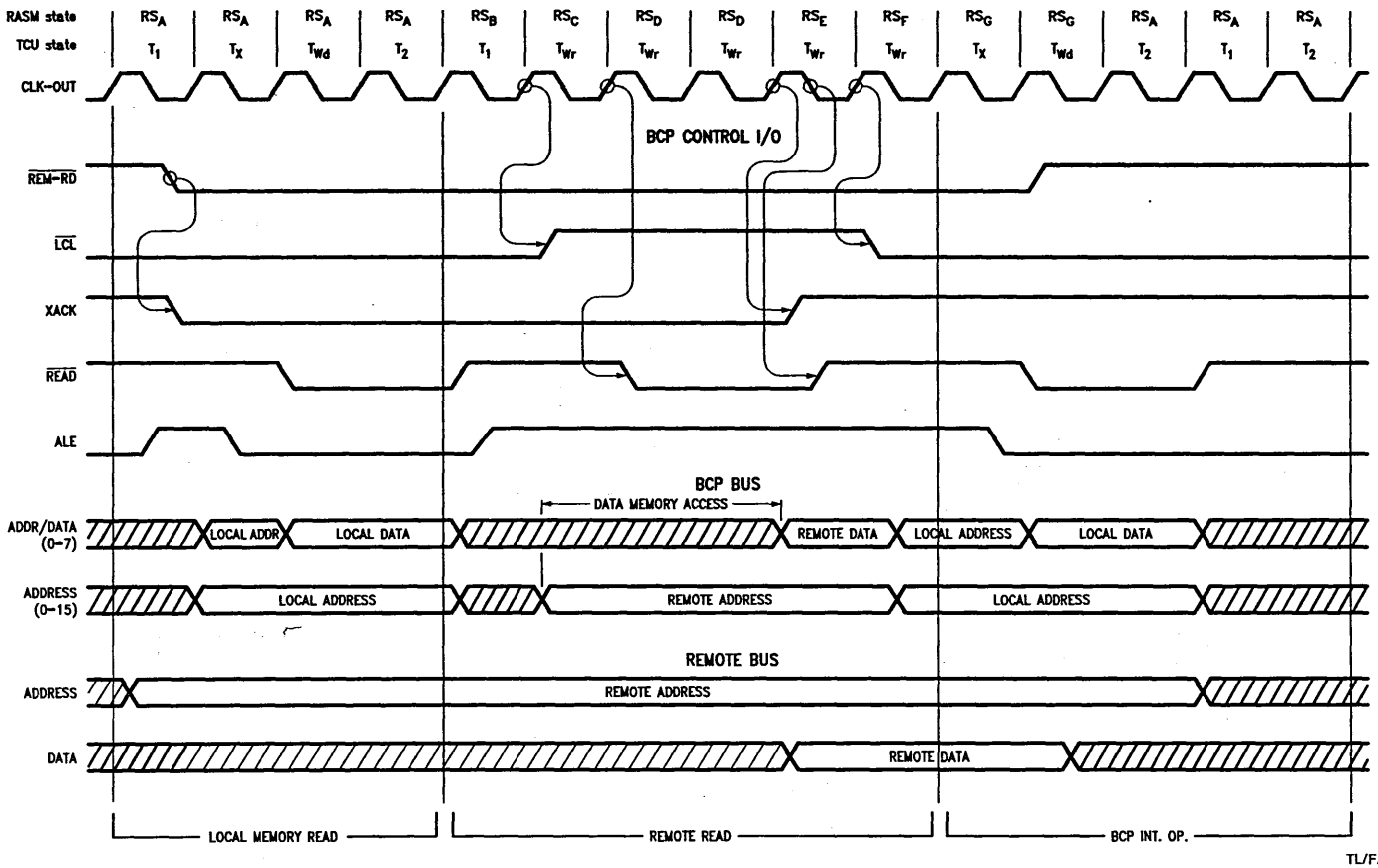
Note: Resetting the BCP will reset HIB (i.e., HIB = 0). Writing 01 to the Memory Select bits in {RIC} (i.e., [MS1–0] = 01, pointing to IMEM) will also force HIB to zero. This way the instruction word boundary can be reset without resetting the BCP.

After all of the programmed wait states are inserted in the RS_D states, more wait states may be added by asserting $\overline{WAIT}$ low a half T-state before the end of the last programmed wait state. If there are no programmed wait states $\overline{WAIT}$ must be asserted low a half T-state before the end of RS_D to add wait states. If $\overline{WAIT}$ remains low, the remote access is extended indefinitely. All the RS_D states move to their corresponding RS_E states on the CPU-CLK after the programmed wait state conditions are met and $\overline{WAIT}$ is high. $\overline{LCL}$ remains high in all RS_E states and A remains in TRI-STATE (and AD if the access is to Data Memory). XACK returns high in this state, indicating that data is valid so that it can be externally latched. The action specific to each RS_D state remains in effect during the first half of the RS_E cycle (i.e. $\overline{READ}$ is asserted in the first half of RS_{E4}). This half T-state of hold time is provided to guarantee data is latched when XACK goes high. This state begins the Termination Phase.



TL/F/9336–98

FIGURE 4-16. Flow Chart of Latched Read Mode



TL/F/9336–30

Register Configuration:

—One Wait-State Programmed for Data-Memory
—Zero Wait-States Programmed for Instruction-Memory
—{RIC} Contents: XXX1X100
—[LOR] = 0

Other BCP Control Signals:

$\overline{RAE}$ = 0
CMD = 0
$\overline{REM\text{-}WR}$ = 1
$\overline{LOCK}$ = 1

FIGURE 4-17. Latched Read of Data Memory by Remote Processor

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

On the next clock the state machine will enter RS_F and $\overline{LCL}$ will return low. The A bus (and AD bus if the access is to data memory) remains in TRI-STATE for the first half T-state of RS_F. After the first half of RS_F, the Remote Processor is no longer using the buses and the BCP CPU will be granted the buses if LCL-BREQ is asserted. If a local bus request is made, a local bus grant will be given to the Timing Control Unit. If the preceding access was a read of IMEM, then HIB is switched and if the access was to the high byte of IMEM then the PC is incremented. If RAE* REM-RD is deasserted at this point, the next clock will bring RASM back to RS_A where it will loop until another Remote Access is initiated. RS_G is entered if RAE*REM-RD is still true. RASM will loop in RS_G until RAE*REM-RD is no longer active at which time the state machine will return to RS_A.

In *Figure 4-17*, the BCP is executing the first of two Data Memory reads when $\overline{REM\text{-}RD}$ goes low. In response, XACK goes low, waiting the Remote Processor. At the end of the first instruction, although the BCP begins its second write by taking ALE high, the RASM now takes control of the bus and deasserts $\overline{LCL}$ high at the end of T_1. A one T-state delay is built into this transfer to ensure that $\overline{READ}$ has been deasserted high before the data bus is switched. The Timing Control Unit is now waited, inserting remote access wait states, T_{Wr}, as RASM takes over.

The remote address is permitted one T-state to settle on the BCP address bus before $\overline{READ}$ goes low, XACK then returns high one T-state plus the programmed Data Memory wait state, T_{Wd} later, having satisfied the memory access time. $\overline{READ}$ returns high a half T-state later, ensuring sufficient hold time, followed by $\overline{LCL}$ being reasserted low after an additional half T-state, transferring bus control back to the BCP. The Remote Processor responds to XACK returning high by deasserting $\overline{REM\text{-}RD}$ high, although by this time the BCP is well into its own memory read.

4.2.3 Slow Buffered Write

The timing for this mode is the same as the Buffered Read mode. The complete flow chart for the Slow Buffered Write mode is shown in *Figure 4-18*. Until a Remote Write is initiated (RAE*REM-WR true), the state machine (RASM) loops in state RS_{A1}. If a Remote Write is initiated and [LOR] is set high, RASM will move to state RS_{A2}. Likewise, if a Remote Write is initiated while the buses have been granted locally (i.e., Local Bus Grant = 1), RASM will move to state RS_{A2}. The state machine will loop in state RS_{A2} as long as [LOR] is set high or the buses are granted locally. If the BCP CPU needs to access Data Memory while in either RS_A state (and $\overline{LOCK}$ is high), it can still do so. A local access is requested by the Timing Control Unit asserting the Local Bus Request (LCL-BREQ) signal. A local bus grant will be given by RASM if the buses are not being used (as is the case in the RS_A state).

XACK is taken low as soon as RAE*REM-WR is true, regardless of an ongoing local access. RASM will move into RS_B on the next clock after RAE*REM-WR is asserted and there is no local bus request and [LOR] = 0. No further local bus requests will be granted until the remote access is complete and RASM returns to RS_A. If the BCP CPU initiates a Data Memory access after RS_A, the Timing Control Unit will be waited and the BCP CPU will remain in state T_{Wr} until completion of the remote access. Half a T-state after entering RS_B the A and AD buses go into TRI-STATE.

On the next CPU-CLK, RASM enters RS_C and $\overline{LCL}$ is taken high while XACK remains low. The wait state counters, i_{IW} and i_{DW}, are loaded in this state from [IW1–0] and [DW2–0], respectively, in {DCR}. The A and AD buses now remain in TRI-STATE and the Access Phase begins. If the Remote Access is to IMEM and the high instruction byte flag is set (i.e., HIB = 1), then $\overline{IWR}$ is asserted low in RS_C. The state machine can move into one of several states, depending on the state of CMD and [MS1–0], on the next clock. XACK remains low and $\overline{LCL}$ remains high in all the possible next states. If CMD is high, the access is to {RIC} and the next state will be RS_{D1}. The path from AD to {RIC} opens in this state. Any remote access mode changes made by this write will not take effect until one T-state after the completion of the present write.

The five other next states all have CMD low and depend on the Memory Select bits. If [MS1–0] is 10 or 11, the state machine will enter either RS_{D2} or RS_{D3} and the low or high bytes of the Program Counter, respectively, will be written.

[MS1–0] equal to 00 designates a Data Memory access and moves RASM into RS_{D4}. $\overline{WRITE}$ will be asserted in this state and A and AD continue to be tri-stated. This allows the Remote Processor to drive the Data Memory address and data buses for the write. Since DMEM is subject to wait states, RS_{D4} is looped upon until all the programmed data memory wait states have been inserted.

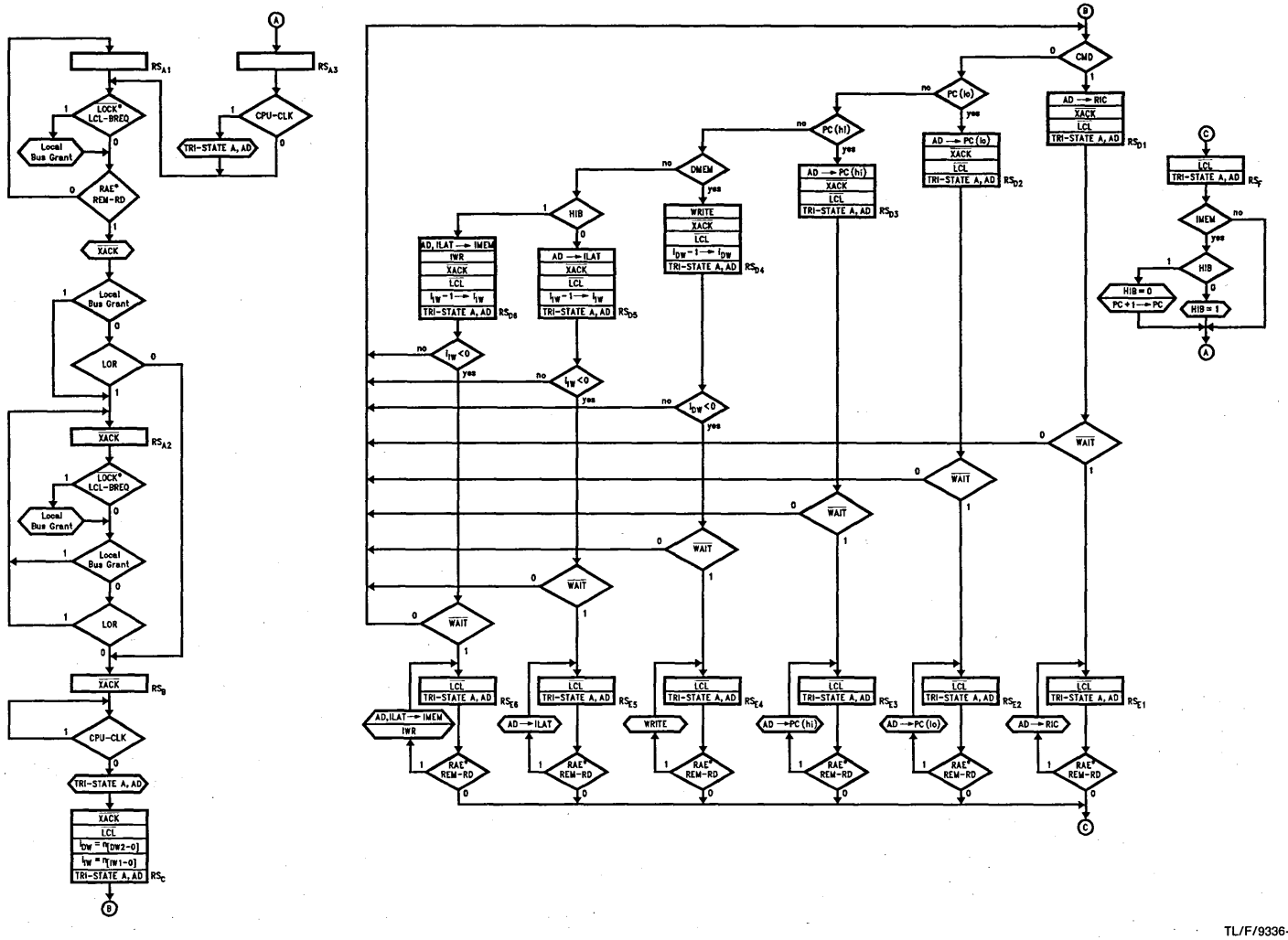
The last possible Memory Selection is Instruction Memory, [MS1–0] = 01. The two possible next states for IMEM depend on whether RASM is expecting the low byte or high byte. Instruction words are accessed low byte, then high byte and RASM powers up expecting the low Instruction byte. The internal flag that keeps track of the next expected Instruction byte is called the High Instruction Byte flag (HIB). If HIB is low, the next state is RS_{D5} and the low instruction byte is written into the holding register, ILAT. If HIB is high, the high instruction byte is moved to I15–8 and the value in ILAT is moved to I7–0. At the same time, $\overline{IWR}$ is asserted low, beginning the write to instruction memory. An IMEM access, like a DMEM access, is subject to wait states and these states will be looped on until all programmed Instruction Memory wait states have been inserted.

Note: Resetting the BCP will reset HIB (i.e., HIB = 0). Writing 01 to the Memory Select bits in {RIC} (i.e., [MS1–0] = 01, pointing to IMEM) will also force HIB to zero. This way the instruction word boundary can be reset without resetting the BCP.

After all of the programmed wait states are inserted in the RS_D states, more wait states may be added by asserting $\overline{WAIT}$ low a half T-state before the end of the last programmed wait state. If there are no programmed wait states, $\overline{WAIT}$ must be asserted low a half T-state before the end of RS_D to add wait states. If $\overline{WAIT}$ remains low, the remote access is extended indefinitely. All the RS_D states move to their corresponding RS_E states on the CPU-CLK after the programmed wait state conditions are met and $\overline{WAIT}$ is high. The RS_E states are looped upon until RAE* REM-WR is deasserted. $\overline{LCL}$ remains high in all RS_E states, but XACK is taken back high to indicate that the remote access can be terminated. If XACK is connected to a Remote Processor wait pin, it can now terminate its write cycle. This state begins the Termination Phase. The action specified in the conditional box is only executed while RAE*REM-WR is asserted—a clock edge is not necessary.

On the CPU-CLK after RAE*REM-WR is deasserted, RASM enters RS_F, where $\overline{LCL}$ remains high and the BCP A and AD buses are still in TRI-STATE. The next CPU-CLK causes RASM to move to RS_{A3}. If the access was to IMEM, then

4.0 Remote Interface and Arbitration System (RIAS) (Continued)



TL/F/9336-99

FIGURE 4-18. Flow Chart of Slow Buffered Write Mode

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

the last action of the remote access before moving to RS_{A3} is to switch HIB and increment the PC if the high byte was written. In RS_{A3}, $\overline{LCL}$ goes low while A and AD remain in TRI-STATE for the first half of RS_{A3}. If no new Remote access is initiated the next clock brings the state machine back to RS_{A1} where it will loop until a Remote Access is initiated.

In *Figure 4-19,* the BCP is executing the first of two consecutive Slow Buffered Writes to Data Memory when $\overline{REM\text{-}WR}$ goes low. In response, XACK goes low, waiting the Remote Processor. At the end of the first instruction, although the BCP begins its second write by taking ALE high, RASM now Takes control of the bus and deasserts $\overline{LCL}$ high at the end of T_1. A one T-state delay is built into this transfer to ensure that $\overline{WRITE}$ has been deasserted high before the data bus is switched. The Timing Control Unit is now waited, inserting remote access wait states, T_{Wr}, as RASM takes over.

The remote address is permitted one T-state to settle on the BCP address bus before $\overline{WRITE}$ goes low, XACK then returns high one T-state plus the programmed Data Memory wait state, T_{Wd} later, having satisfied the memory access time. The Remote Processor will respond by deasserting $\overline{REM\text{-}WR}$ high to which the BCP in turn responds by deasserting $\overline{WRITE}$ high. Following $\overline{WRITE}$ being deasserted high, the BCP waits till the end of the next T-state before asserting $\overline{LCL}$ low, again ensuring that the write cycle has concluded before the bus is switched. Control is then returned to the Timing Control Unit and the local memory write continues.

4.2.4 Fast Buffered Write

The timing for the Fast Buffered Write mode is very similar to the timing of the Latched Read. The major difference is the additional half clock that AD is active in the Latched Read mode that is not present in the Fast Buffered Write mode. The Fast Buffered Write cycle ends after the data is written and the termination doesn't wait for the trailing edge of $\overline{REM\text{-}WR}$. Therefore the Arbitration and Access Phases of the Fast Buffered Write mode are the same as for the Latched Read mode.

The complete flow chart for the Fast Buffered Write mode is shown in *Figure 4-20.* Until a Remote Write is initiated (RAE*REM-WR true), the state machine (RASM) loops in state RS_{A1}. If a Remote Write is initiated and [LOR] is set high, RASM will move to state RS_{A2}. Likewise, if a Remote Write is initiated while the buses have been granted locally (i.e., Local Bus Grant = 1), RASM will move to state RS_{A2}. The state machine will loop in state RS_{A2} as long as [LOR] is set high or the buses are granted locally. If the BCP CPU needs to access Data Memory while in either RS_A state (and $\overline{LOCK}$ is high), it can still do so. A local access is requested by the Timing Control Unit asserting the Local Bus Request (LCL-BREQ) signal. A local bus grant will be given by RASM if the buses are not being used (as is the case in the RS_A states).

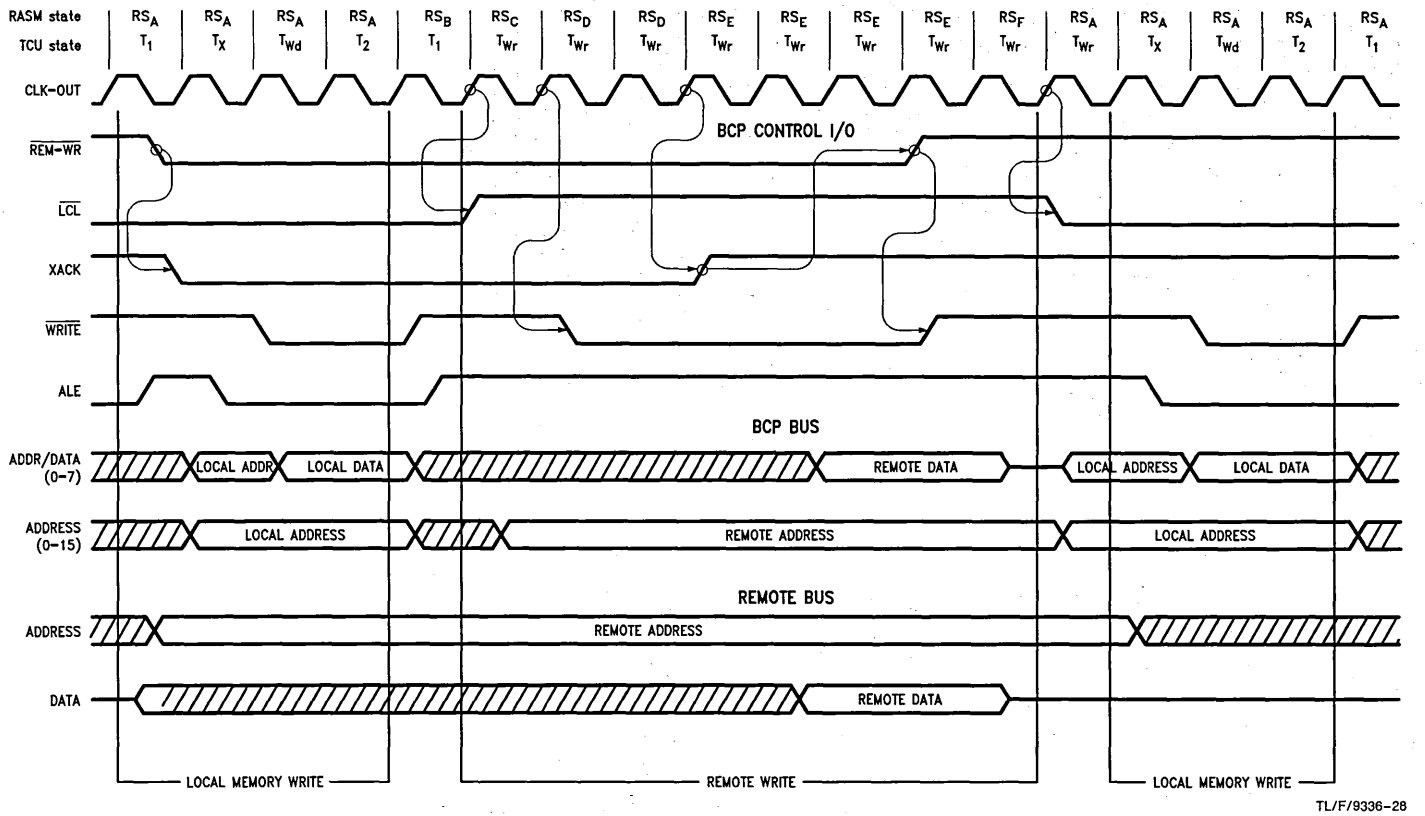
XACK is taken low as soon as RAE*REM-WR is true, regardless of an ongoing local access. If [LOR] is low, RASM will move into RS_B on the next clock after RAE*REM-WR is asserted and there is no local bus request. No further local bus requests will be granted until the BCP enters the Termination Phase. If the BCP CPU initiates a Data Memory access after RS_A, the Timing Control Unit will be waited and the BCP CPU will remain in state T_{Wr} until the remote access reaches the Termination Phase. Half a T-state after entering RS_B the A and AD buses go into TRI-STATE.

On the next CPU-CLK, RASM enters RS_C and $\overline{LCL}$ is taken high while XACK remains low. The wait state counters, i_{IW} and i_{DW}, are loaded in this state from [IW1–0] and [DW2–0], respectively, in {DCR}. The A and AD buses remain in TRI-STATE and the Access Phase begins. If the Remote Access is to IMEM and the high instruction byte flag is set (i.e., HIB = 1), then $\overline{IWR}$ is asserted low in RS_C.

The state machine can move into one of several states depending on the state of CMD and [MS1–0] on the next clock. XACK and $\overline{LCL}$ in all the possible next states. If CMD is high, the access is to {RIC} and the next state will be RS_{D1}. The path from AD to {RIC} opens in this state. Any remote access mode changes made by this write will not take effect until one T-state after the completion of the present write.

The five other next states all have CMD low and depend on the Memory Select bits. If [MS1–0] is 10 or 11 the state machine will enter either RS_{D2} or RS_{D3} and the low or high bytes of the Program Counter, respectively, will be written.

[MS1–0] = 00 designates a Data Memory access and moves RASM into RS_{D4}. $\overline{WRITE}$ will be asserted in this



TL/F/9336-28

Register Configuration:

—One Wait-State Programmed for Data-Memory
—Zero Wait-States Programmed for Instruction-Memory
—{RIC} Contents: XX0X0100
—[LOR] = 0

Other BCP Control Signals:

$\overline{RAE}$	= 0
CMD	= 0
$\overline{REM\text{-}RD}$	= 1
$\overline{LOCK}$	= 1

FIGURE 4-19. Slow Buffered Write to Data Memory by Remote Processor

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

state and A and AD continue to be tri-stated. This allows the Remote Processor to drive the Data Memory address and data buses for the write. Since DMEM is subject to wait states, RS_{D4} is looped upon until all the programmed Data Memory wait states have been inserted.

The last possible Memory Selection is Instruction Memory, [MS1–0] = 01. The two possible next states for IMEM depend on whether RASM is expecting the low byte or high byte. Instruction words are accessed low byte then high byte and RASM powers up expecting the low Instruction byte. The internal flag that keeps track of the next expected Instruction byte is called the High Instruction Byte flag (HIB). If HIB is low, the next state is RS_{D5} and the low instruction byte is written into the holding register, ILAT. If HIB is high, the high instruction byte is moved to I15–8 and ILAT is moved to I7–0. At the same time $\overline{IWR}$ is asserted low, beginning the write to instruction memory. An IMEM access, like a DMEM access, is subject to wait states and these states will be looped on until all programmed instruction memory wait states have been inserted.

Note: Resetting the BCP will reset HIB (i.e., HIB = 0). Writing 01 to the Memory Select bits in (RIC) (i.e., [MS1–0] = 01, pointing to IMEM) will also force HIB to zero. This way the instruction word boundary can be reset without resetting the BCP.

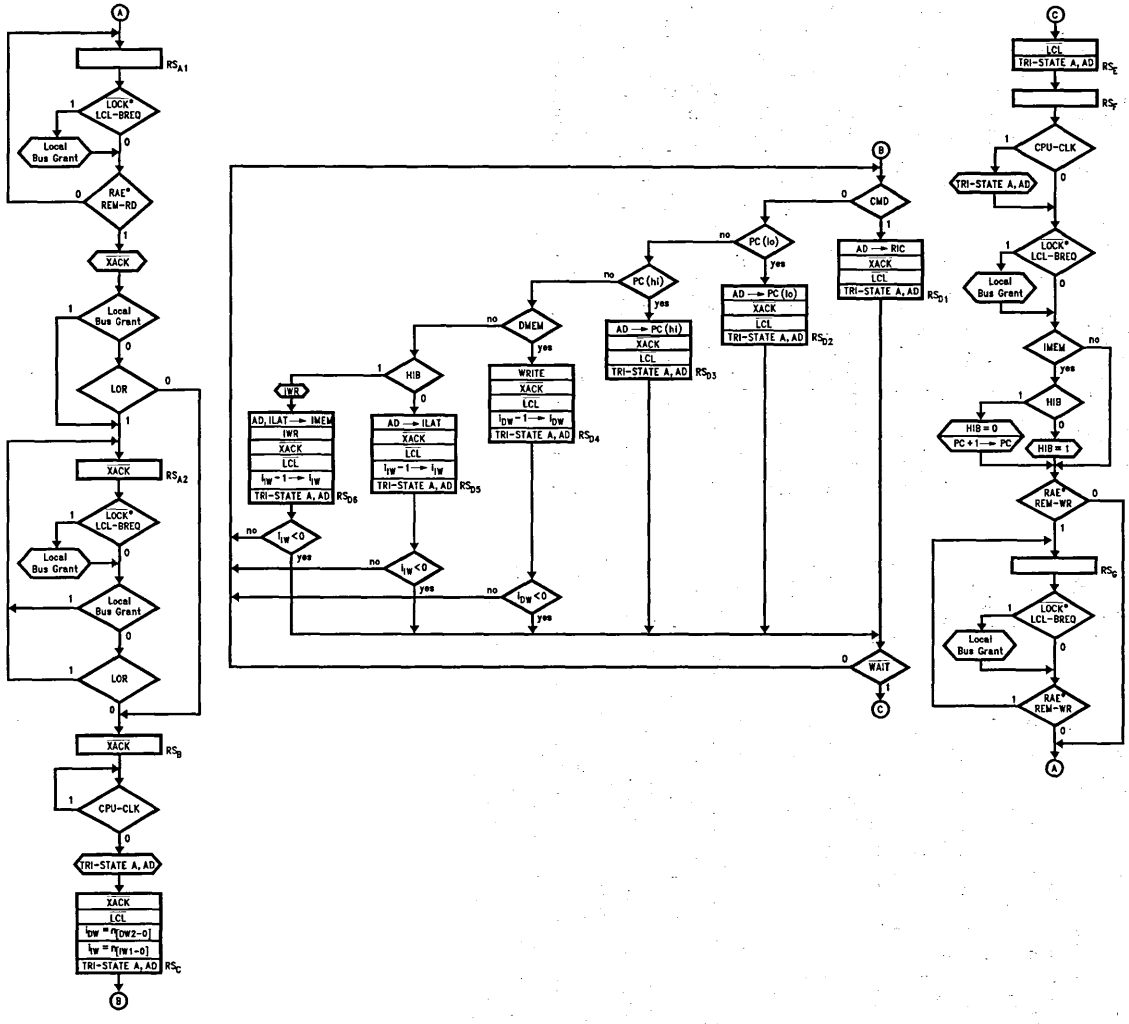
After all of the programmed wait states are inserted into RS_D states, more wait states may be added by asserting $\overline{WAIT}$ low a half T-state before the end of the last programmed wait state. If there are no programmed wait states $\overline{WAIT}$ must be asserted low a half T-state before the end of RS_D to add wait states. If $\overline{WAIT}$ remains low, the remote access is extended indefinitely. All the RS_D states converge to state RS_E on the next CPU-CLK after the programmed wait state conditions are met and $\overline{WAIT}$ is high. $\overline{LCL}$ remains high in all RS_E states and A and AD remain in TRI-STATE as well. XACK returns high in this state, indicating that the data is written and the cycle can be terminated by the RP. This state begins the Termination Phase.

On the next clock the state machine will enter RS_F and $\overline{LCL}$ will return low. The A and AD buses remain in TRI-STATE for the first half T-state of RS_F. After the first half of RS_F, the Remote Processor is no longer using the buses and the BCP CPU can make an access to Data Memory by asserting LCL-BREQ. If a local bus request is made, a local bus grant will be given to the Timing Control Unit. If the preceding access was a write of IMEM, then HIB is switched and if the access was to the high byte of IMEM then the PC is incremented. If RAE*REM-WR is deasserted at this point, the next clock will bring RASM back to RS_A where it will loop until another remote access is initiated. RS_G is entered if RAE*REM-WR is still true. RASM will loop in RS_G until RAE*REM-WR is no longer active at which time the state machine will return to RS_A.

In *Figure 4-21*, the BCP is executing the first of two Data Memory writes when $\overline{REM\text{-}WR}$ goes low. In response, XACK goes low, waiting the Remote Processor. At the end of the first instruction, although the BCP begins its second write by taking ALE high, RASM now takes control of the bus and deasserts $\overline{LCL}$ high at the end of T_1. A one T-state delay is built into this transfer to ensure that $\overline{WRITE}$ has been deasserted high before the data bus is switched. The Timing Control Unit is now waited, inserting remote access wait states, T_{Wr}, as RASM takes over.

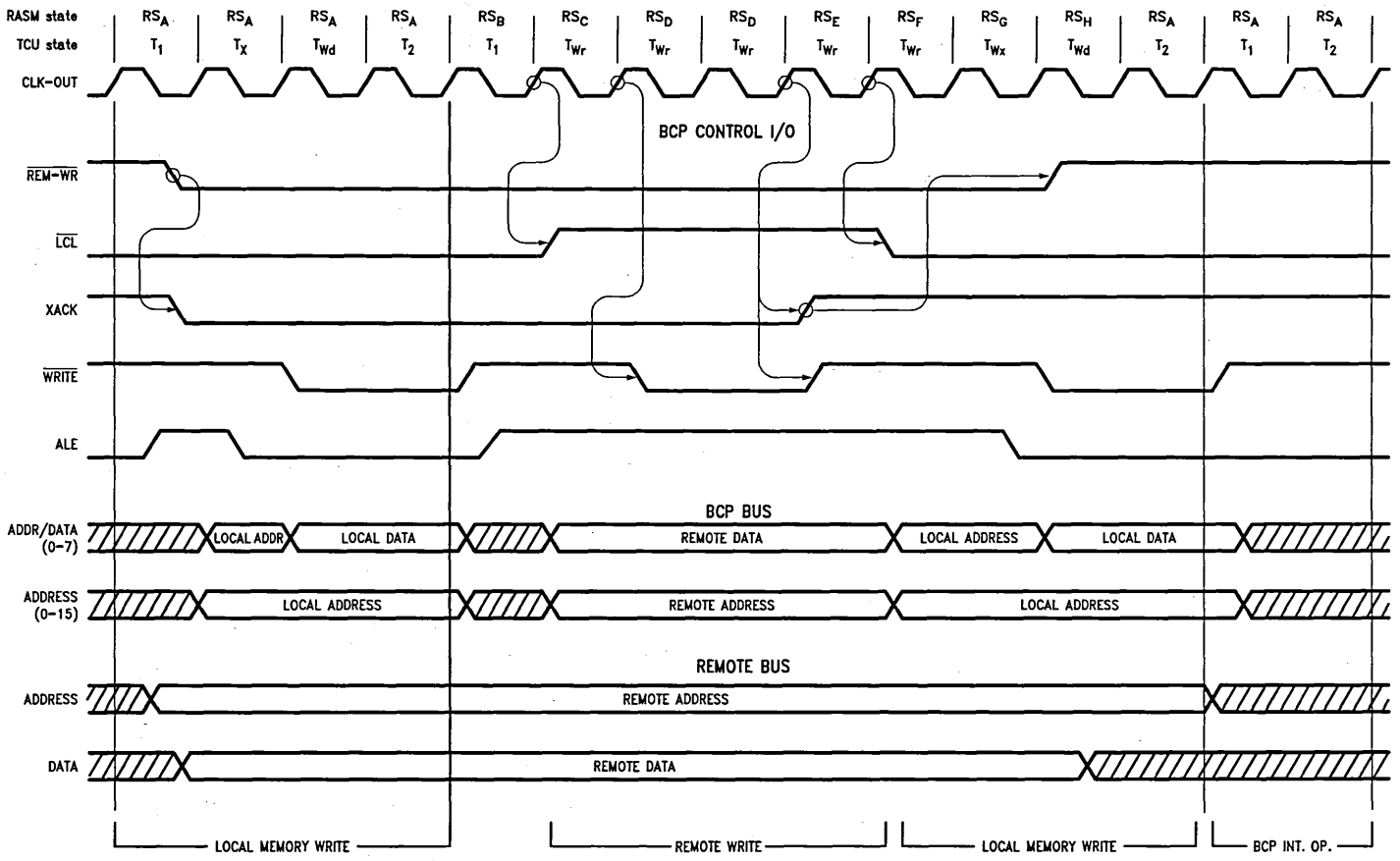
The remote access is permitted one T-state to settle on the BCP address bus before $\overline{WRITE}$ goes low, XACK then returns high one T-state plus the programmed Data Memory wait state, T_{Wd} later, having satisfied the memory access time. $\overline{WRITE}$ returns high at the same time, and one T-state later $\overline{LCL}$ returns low, transferring bus control back to the BCP. The remote processor responds to XACK returning high by deasserting $\overline{REM\text{-}WR}$ high, although by this time the BCP is well into its own memory write.

4.0 Remote Interface and Arbitration System (RIAS) (Continued)



TL/F/9336-A0

FIGURE 4-20. Flow Chart of Fast Buffered Write Mode



TL/F/9336-29

Register Configuration:

—One Wait-State Programmed for Data-Memory
—Zero Wait-States Programmed for Instruction-Memory
—{RIC} Contents: XX1X0100
—[LOR] = 0

Other BCP Control Signals:

$\overline{RAE}$	= 0
CMD	= 0
$\overline{REM\text{-}RD}$	= 1
$\overline{LOCK}$	= 1

FIGURE 4-21. Fast Buffered Write to Data Memory by Remote Processor

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

4.2.5 Latched Write

This mode executes a write without waiting the Remote Processor—XACK isn't normally taken low. The complete flow chart for the Latched Write mode is shown in *Figure 4-22*. Until a Remote Write is initiated (RAE*REM-WR true), the state machine (RASM) loops in state RS_A. If the BCP CPU needs to access Data Memory at this time (and $\overline{LOCK}$ is high), it can still do so. A local access is requested by the Timing Control Unit asserting the Local Bus Request (LCL-BREQ) signal. A local bus grant will be given by RASM if the buses are not being used (as is the case in RS_A).

RASM will move into RS_B on the next clock after RAE*REM-WR is asserted. XACK is not taken low and therefore the RP is not waited. The state machine will loop in RS_B until the RP terminates its write cycle—until RAE*REM-WR is no longer true. The external address and data latches are typically latched on the trailing edge of $\overline{REM\text{-}WR}$. A local bus request will still be serviced in this state.

Next, RASM enters RS_C and $\overline{WR\text{-}PEND}$ is asserted to prevent overwrite of the external latches. Since the RP has completed its write cycle, another write or read can happen at any time. Any Remote Read cycle (RAE*REM-RD) or Remote Write cycle (RAE*REM-WR) occurring after the state machine enters RS_C will take XACK low. A local access initiated before or during this state must be completed before RASM can move to RS_D. Once RS_D is entered, though, no further local bus requests will be granted until RASM enters the Termination Phase. If the BCP CPU initiates a Data Memory access after RS_C, the Timing Control Unit will be waited and the BCP CPU will remain in state T_{Wr} until the RASM enters RS_H. Half a T-state after entering RS_B the A and AD buses go into TRI-STATE.

On the next clock, the state machine enters RS_E and $\overline{LCL}$ is taken high. $\overline{WR\text{-}PEND}$ continues to be asserted low in this state and the data and instruction wait state counters, i_{DW} and i_{IW}, are loaded from [DW2–0] and [IW1–0], respectively, in {DCR}. The A and AD buses remain in TRI-STATE and the Access Phase begins. Any remote accesses now occurring will take XACK low and wait the Remote Processor. If the Remote Access is to IMEM and the high instruction byte flag is set (i.e., HIB = 1), then $\overline{IWR}$ is asserted low in RS_E.

The state machine will move into one of several states on the next clock, depending on the state of CMD and [MS1–0]. $\overline{WR\text{-}PEND}$ remains low and $\overline{LCL}$ remains high in all the possible next states. If CMD is high, the access is to {RIC} and the next state will be RS_{F1}. The path from AD to {RIC} opens in this state. Any remote access mode changes made by this write will not take effect until one T-state after the completion of the present write.

The five other next states all have CMD low and depend on the Memory Select bits. If [MS1–0] is 10 or 11 the state machine will enter either RS_{F2} or RS_{F3} and the low or high bytes of the Program Counter, respectively, will be loaded.

[MS1–0] = 00 designates a Data Memory access and moves RASM into RS_{F4}. $\overline{WRITE}$ will be asserted low in this state and A and AD continue to be tri-stated. This allows the Remote Processor to drive the Data Memory address and data for the write. Since DMEM is subject to wait states, RS_{F4} is looped upon until all the programmed Data Memory wait states have been inserted.

The last possible Memory Selection is Instruction Memory, [MS1–0] = 01. The two possible next states for IMEM depend on if RASM is expecting the low byte or high byte. Instruction words are accessed low byte then high byte and RASM powers up expecting the low Instruction byte. The internal flag that keeps track of the next expected Instruction byte is called the High Instruction Byte flag (HIB). If HIB is low, the next state is RS_{F5} and the low instruction byte is written into the holding register, ILAT. If HIB is high, the high instruction byte is moved to I15–8 and the value in ILAT is moved to I7–0. At the same time, $\overline{IWR}$ is asserted low and the write to Instruction Memory is begun. An IMEM access, like a DMEM access, is subject to wait states and these states will be looped on until all programmed instruction memory wait states have been inserted.

Note: Resetting the BCP will reset HIB (i.e., HIB = 0). Writing 01 to the Memory Select bits in {RIC} (i.e., [MS1–0] = 01, pointing to IMEM) will also force HIB to zero. This way the instruction word boundary can be reset without resetting the BCP.

All the RS_F states converge to a single decision box that tests $\overline{WAIT}$. If $\overline{WAIT}$ is low then the state machine loops back to RS_F, otherwise RASM will move on to RS_G. $\overline{LCL}$ remains high and $\overline{WR\text{-}PEND}$ remains low in this state but the actions specific to the RS_F states have ended (i.e. $\overline{WRITE}$ will no longer be asserted low).

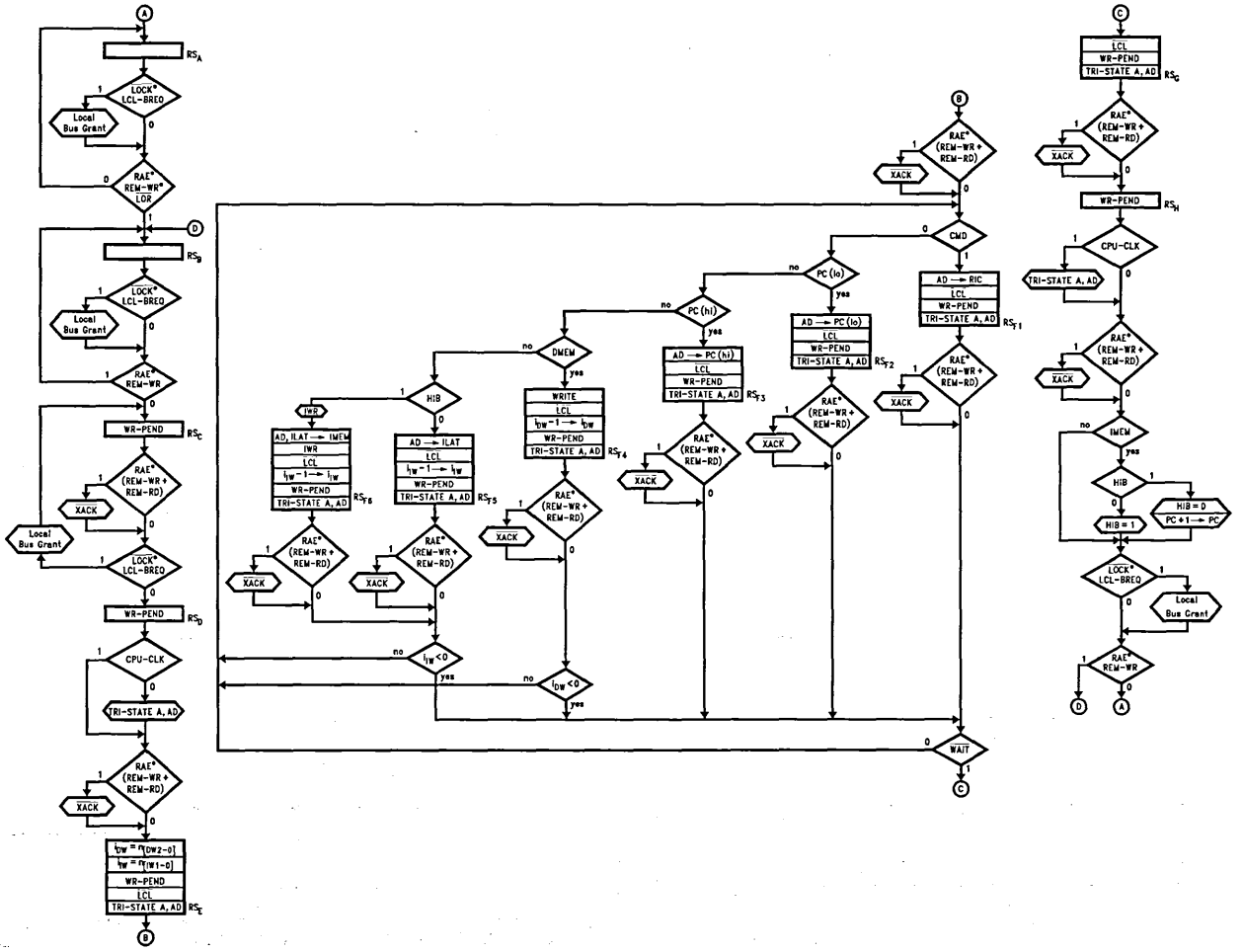
The next CPU-CLK moves RASM into RS_H, the last state in the state machine. $\overline{LCL}$ returns low but $\overline{WR\text{-}PEND}$ is still low. The A and AD buses remain in TRI-STATE for the first half of RS_H. XACK will be taken low if a Remote Access is initiated. If the just completed access was to IMEM, HIB will be switched. Also, the PC will be incremented if the high byte was written. A local access will be granted if LCL-BREQ is asserted in this state.

If another Remote Write is pending, the state machine takes the path to RS_B where that write will be processed. A pending Remote Read will return to the RS_A in either the Buffered or Latched Read sections (not shown in *Figure 4-22*) of the state machine. And if no Remote Access is pending, the machine will loop in RS_A until the next access is initiated.

In *Figure 4-23*, the BCP is executing the first of two Data Memory writes when $\overline{REM\text{-}WR}$ goes low. The BCP takes no action until $\overline{REM\text{-}WR}$ goes back high, latching the data and making a remote access request. The BCP responds to this by taking $\overline{WR\text{-}PEND}$ low. At the end of the first instruction, although the BCP begins its second write by taking ALE high, RASM now takes control of the bus and deasserts $\overline{LCL}$ high at the end of T_1. A one T-state delay is built into this transfer to ensure that $\overline{WRITE}$ has been deasserted high before the data bus is switched. Timing Control Unit is now waited, inserting remote access wait states, T_{Wr}, as RASM takes over.

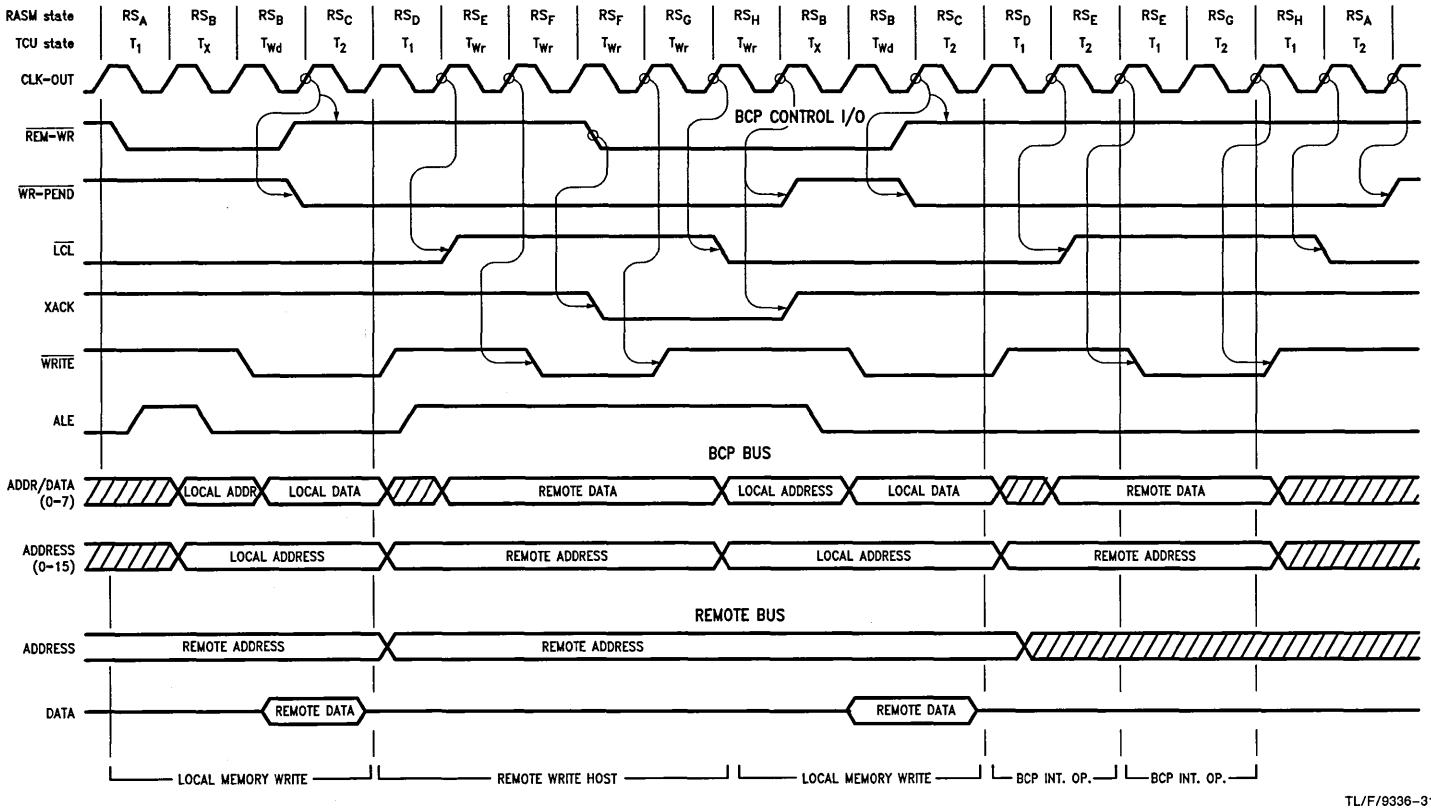
The remote address is permitted one T-state to settle on the BCP address bus before $\overline{WRITE}$ goes low. $\overline{WRITE}$ then returns high one T-state plus the programmed Data Memory wait state, T_{Wd} later, having satisfied the memory access time, and one T-state later $\overline{LCL}$ is reasserted low, transferring bus control back to the BCP.

In this example, $\overline{REM\text{-}WR}$ goes low again during the remote write cycle which, since $\overline{WR\text{-}PEND}$ is still low, causes XACK to go low to wait the Remote Processor. Then $\overline{LCL}$ goes low, allowing the second data byte to be latched on the next trailing edge of $\overline{REM\text{-}WR}$. One T-state later. XACK and $\overline{WR\text{-}PEND}$ go back high at the same time.



TL/F/9336-A1

FIGURE 4-22. Flow Chart of Latched Write Mode



TL/F/9336-31

Register Configuration:

—One Wait-State Programmed for Data-Memory
—Zero Wait-States Programmed for Instruction-Memory
—{RIC} Contents: XXXX1100
—[LOR] = 0

Other BCP Control Signals:

$\overline{RAE}$	= 0
CMD	= 0
$\overline{REM\text{-}RD}$	= 1
$\overline{LOCK}$	= 1

FIGURE 4-23. Latched Write to Data Memory by Remote Processor

4.0 Remote Interface and Arbitration System (RIAS) (Continued)

The BCP is now shown executing a local memory write, with remote data still pending in the latch. At the end of this instruction, the BCP begins executing a series of internal operations which do not require the bus. RASM therefore takes over and, without waiting the Timing Control Unit, executes the Remote Write.

4.2.6 Remote Rest Time

For the BCP to operate properly, remote accesses to the BCP must be separated by a minimal amount of time. This minimal amount of time has been termed "rest time".

There are two causes for remote rest time. The first cause is implied in the functional state machine forms for remote accesses and can be explained as follows: At the beginning of every T-state the validity of a remote access is sampled for that T-state. To guarantee that the BCP recognizes the end of a remote cycle, the time between remote accesses must be a minimum of one T-state plus set up and hold times.

In the case of Latched Read and Fast Buffered Write, the validity of a remote access is not sampled on the first rising edge of the CPU-CLK following XACK rising. However, on all subsequent rising edges of the CPU-CLK the validity of the remote access is sampled. As a result, if the remote processor can terminate its remote access quickly after XACK rises (within a T-state), up to a T-state may be added to the above equation for Latched Read and Fast Buffered Write modes (i.e., a second remote access should not begin for two T-states plus set up and hold times after XACK rises in Latched Read and Fast Buffered Write modes). On the other hand, if the remote processor does not terminate its remote access within a T-state of XACK rising, the above equation (one T-state plus set up and hold times between remote accesses) remains valid for Latched Read and Fast Buffered Write modes.

If these specifications are not adhered to, the BCP may sample the very end of one valid remote access and one T-state later sample the very beginning of a second remote access. Thus, the BCP will treat the second access as a continuation of the first remote access and will not perform the second read/write. The second access will be ignored. (Reference *Figure 4-24* for the timing diagrams which demonstrate how two remote accesses can be mistaken as one.)

The second source of remote rest time is due to the manner in which the BCP samples the CMD signal. CMD is sampled once at the beginning of each remote access. Due to the manner in which CMD is sampled, CMD will not be sampled again if a second remote access begins within 1.5 T-states plus a hold time, after the BCP recognizes the end of the first remote access. If this happens, the BCP will use the value of CMD from the previous remote access during the second remote access. If the value of CMD is the same for both accesses, the second access will proceed as intended. However, if the value of CMD is different for the two remote accesses, the second remote access will read/write the wrong location.

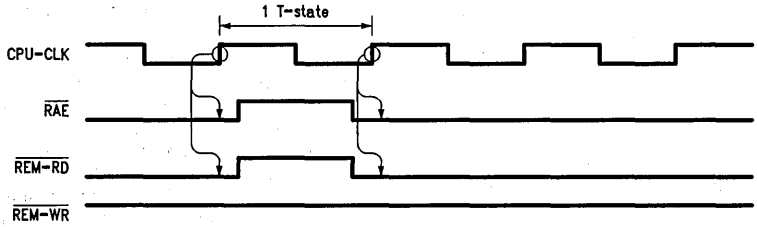
The reader should note that the timing of the second source of rest time begins at the same time that the BCP first samples the end of the previous remote access. Thus when the first source of rest time ends, the second source of rest time begins. (Reference *Figure 4-25* for timing diagrams for rest time in all modes except Latched Write mode).

Latched Write Mode

Latched Write mode is a special case of rest time and needs to be discussed separately from the other modes. The first cause of rest time affects every mode including Latched Write. In regards to the second source of rest time, Latched Write mode was designed to allow a second remote access to start while a write is still pending (i.e., $\overline{\text{WR-PEND}}$ = 0). Thus, when $\overline{\text{WR-PEND}}$ rises (signaling the end of the previous write) the value of CMD is sampled for the second remote access. This allows Latched Write to avoid the second cause of rest time discussed above.

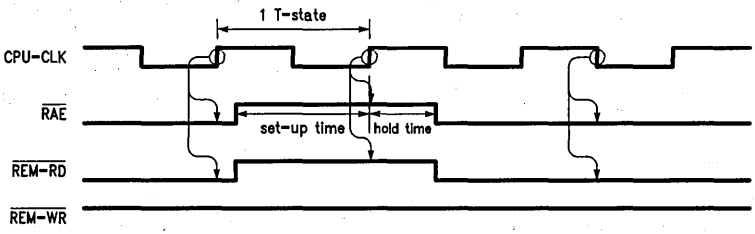
However, if a remote access begins within one half a T-state after $\overline{\text{WR-PEND}}$ rises, CMD will not be sampled again. For this case, if the value of CMD changes just after $\overline{\text{WR-PEND}}$ rose and at the same time the remote access begins, the BCP will read/write the wrong location. (Reference *Figure 4-26* for timing diagrams of rest time for latched write mode.)

4.0 Remote Interface and Arbitration System (RIAS) (Continued)



TL/F/9336-G5

(a) This timing diagram shows two remote accesses within one T-state. The first set of arrows shows the BCP sampling a valid remote read. The next time the BCP samples the validity of the remote access is shown by the second set of arrows (1 T-state later). In this case, it will sample the second remote access and mistake it as a continuation of the first remote access.

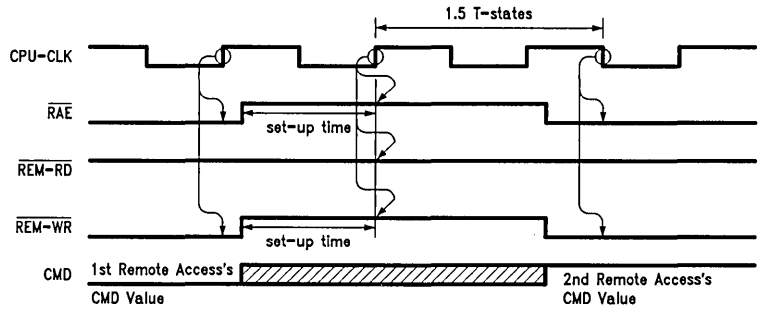


TL/F/9336-G6

(b) This timing diagram shows the timing necessary for the BCP to recognize both accesses as separate accesses. The first set of arrows shows the BCP sampling a valid remote read. One T-state later at the second set of arrows the BCP will sample the end of the first remote access. Another T-state later at the third set of arrows the BCP will sample the beginning of the second remote access.

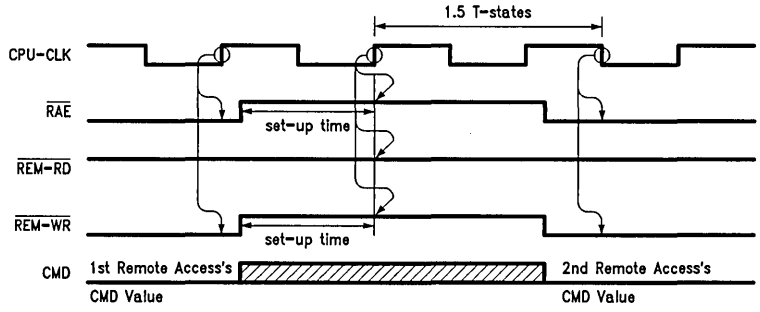
FIGURE 4-24. Mistaking Two Remote Accesses as Only One

4.0 Remote Interface and Arbitration System (RIAS) (Continued)



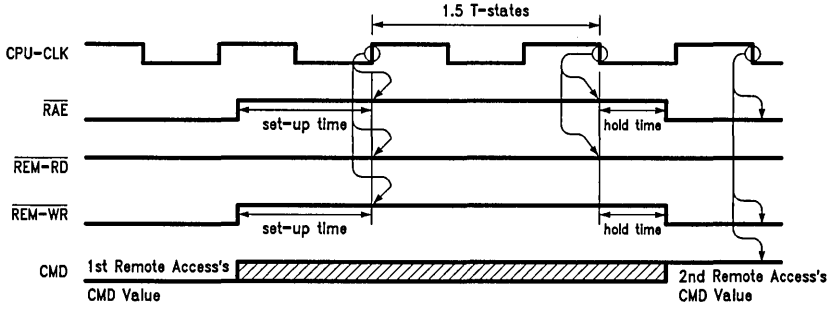
TL/F/9336-G7

(a) This timing diagram shows the second remote access violating rest time. The first set of arrows shows the BCP sampling a valid remote write. The second set of arrows (1 T-state later), shows the BCP sampling the end of the first remote access. If a second remote access starts before the position of the third set of arrows (another 1.5 T-states later), the value of CMD will not be sampled. The value of CMD has changed from the first remote access, so the BCP will write to the wrong location during the second access.



TL/F/9336-G8

(b) This timing diagram shows the second remote access violating rest time. The first set of arrows shows the BCP sampling a valid remote write. The second set of arrows (1 T-state later), shows the BCP sampling the end of the first remote access. If a second remote access starts before the position of the third set of arrows (another 1.5 T-states later), the value of CMD will not be sampled. The value of CMD does not change from the first remote access, so the BCP will write to the intended location during the second remote access.

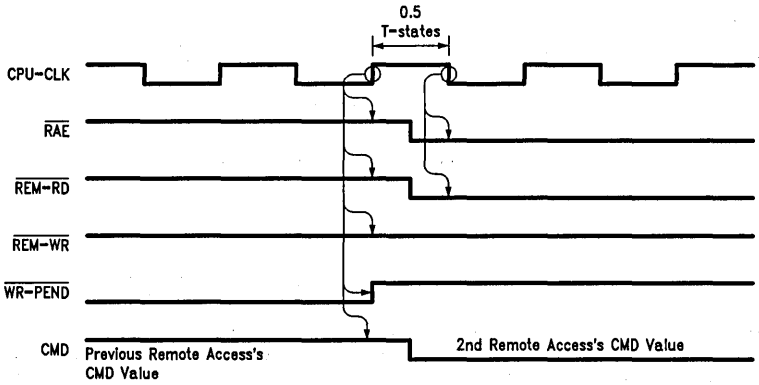


TL/F/9336-G9

(c) This timing diagram shows the timing needed to avoid violating rest time for all modes except latched write. The first set of arrows shows the BCP sampling the end of the first remote access. The second set of arrows (1.5 T-states later), shows the BCP recognizing no remote access has started and the value of CMD will be sampled for the next remote access. The third set of arrows shows the BCP sampling the correct value of CMD for the second remote access.

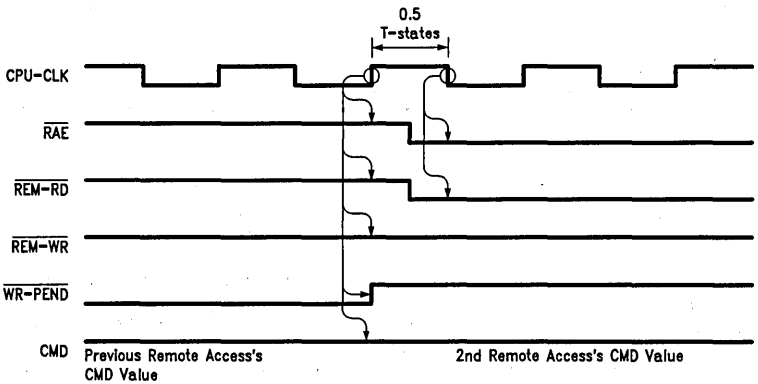
FIGURE 4-25. Remote Rest Time for All Modes except Latched Write

4.0 Remote Interface and Arbitration System (RIAS) (Continued)



TL/F/9336–H1

(a) This timing diagram shows a remote access violating remote rest time. The first set of arrows shows the BCP sampling the value of CMD when WR-PEND rises. If a remote access begins after WR-PEND rises and before the position of the second set of arrows (0.5 T-states later), the value of CMD will not be sampled again. The value of CMD has changed since WR-PEND rose, so the BCP will read the wrong location.

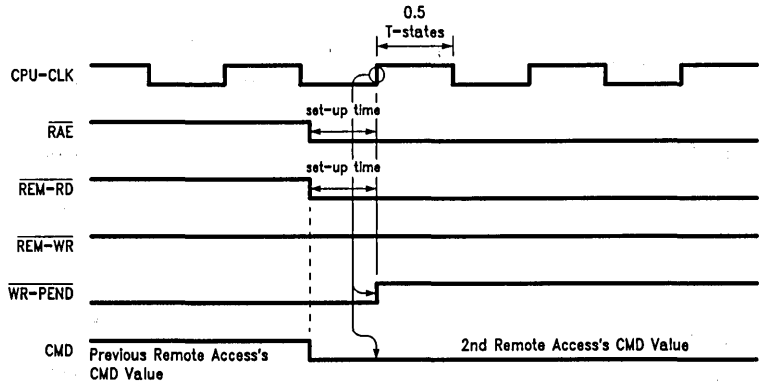


TL/F/9336–H2

(b) This timing diagram shows a remote access violating remote rest time. The first set of arrows shows the BCP sampling the value of CMD when WR-PEND rises. If a remote access begins after WR-PEND rises and before the position of the second set of arrows (0.5 T-states later), the value of CMD will not be sampled again. The value of CMD has not changed since WR-PEND rose, so the BCP will read the intended location.

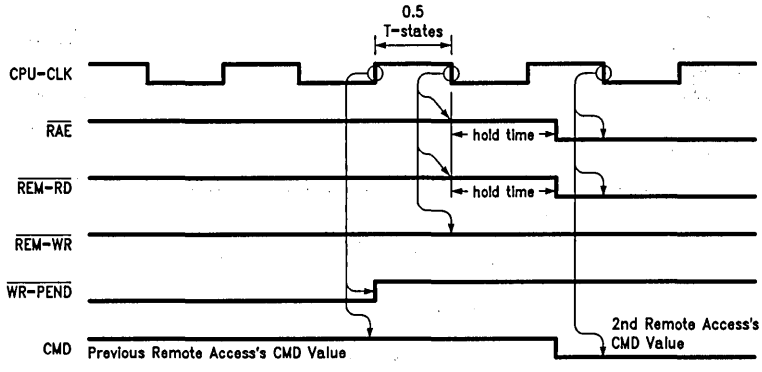
FIGURE 4-26. Rest Time for Latched Write Mode

4.0 Remote Interface and Arbitration System (RIAS) (Continued)



TL/F/9336-H3

(c) This timing diagram shows a remote access setting up in time for $\overline{\text{WR-PEND}}$ rising to latch in the proper value of CMD. The only set of arrows shows the BCP sampling the second remote access's CMD value when $\overline{\text{WR-PEND}}$ rises. The value of CMD will not be sampled again. The BCP will carry out the second remote access as it was intended.



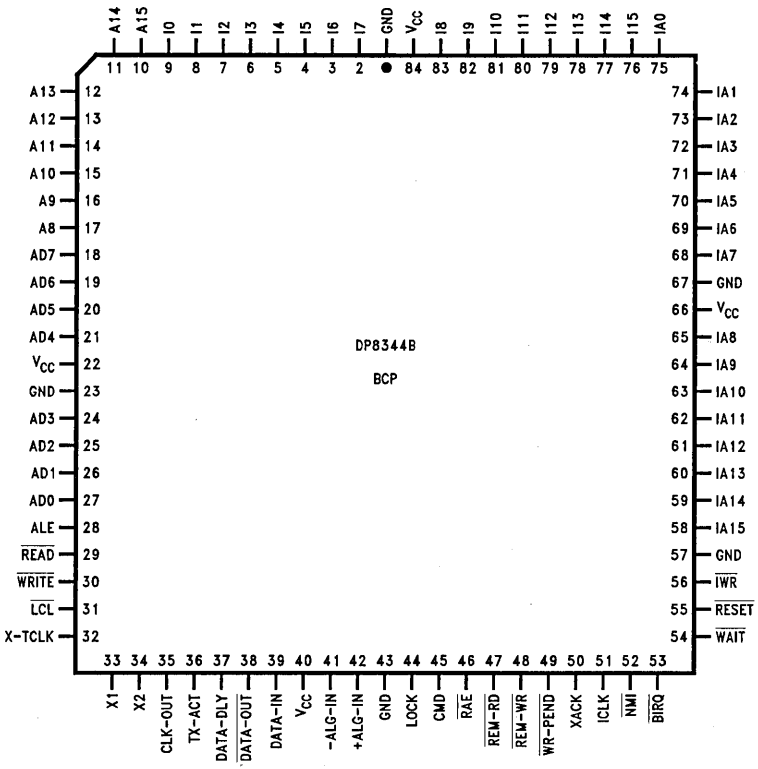
TL/F/9336-H4

(d) This timing diagram shows a remote access starting after a half T-state plus a hold time since $\overline{\text{WR-PEND}}$ rose. The first set of arrows shows the BCP sampling the value of CMD when $\overline{\text{WR-PEND}}$ rises. The second set of arrows shows the BCP recognizing that no remote access has started and the value of CMD will be sampled for the next remote access. The third set of arrows shows the BCP sampling the correct value of CMD for the second remote access. The BCP will carry out the second remote access as it was intended.

FIGURE 4-26. Rest Time for Latched Write Mode (Continued)

5.0 Device Specifications

Plastic Chip Carrier



TL/F/9336-2

FIGURE 5-1. Top View

Order Number DP8344B
See NS Package Number V84A

5.1 PIN DESCRIPTIONS

Signal	In/Out	Pin	Reset State	Description
5.1.1 TIMING/CONTROL SIGNALS				
X1 X2	In Out	33 34	X $\overline{X1}$	Input and output of the on-chip crystal oscillator amplifier. Connect a crystal across these pins, or apply an external clock to X1, with X2 left open.
CLK-OUT	Out	35	X1	Buffered **CL**ock oscillator **OUT**put, at the crystal frequency.
X-TCLK	In	32	X	**EX**ternal Transceiver **CL**ock input.
$\overline{\text{WAIT}}$	In	54	X	CPU **WAIT**. When active, waits processor and remote interface controller.
$\overline{\text{RESET}}$	In	55	0	Master **RESET**. Parallel reset to all sections of the chip.
5.1.2 INSTRUCTION MEMORY INTERFACE **Instruction Address Bus:**				
IA15 (MSB) IA14 IA13 IA12 IA11 IA10	Out Out Out Out Out Out	58 59 60 61 62 63	0 0 0 0 0 0	16-bit Instruction memory **A**ddress bus.

5.0 Device Specifications (Continued)

Signal	In/Out	Pin	Reset State	Description
5.1.2 INSTRUCTION MEMORY INTERFACE (Continued)				
Instruction Address Bus: (Continued)				
IA9	Out	64	0	16-bit Instruction memory Address bus.
IA8	Out	65	0	
IA7	Out	68	0	
IA6	Out	69	0	
IA5	Out	70	0	
IA4	Out	71	0	
IA3	Out	72	0	
IA2	Out	73	0	
IA1	Out	74	0	
IA0 (LSB)	Out	75	0	
Instruction Bus:				
I15 (MSB)	In/Out	76	In	16-bit Instruction memory data bus.
I14	In/Out	77	In	
I13	In/Out	78	In	
I12	In/Out	79	In	
I11	In/Out	80	In	
I10	In/Out	81	In	
I9	In/Out	82	In	
I8	In/Out	83	In	
I7	In/Out	2	In	
I6	In/Out	3	In	
I5	In/Out	4	In	
I4	In/Out	5	In	
I3	In/Out	6	In	
I2	In/Out	7	In	
I1	In/Out	8	In	
I0 (LSB)	In/Out	9	In	
Timing Control:				
$\overline{IWR}$	Out	56	1	Instruction **WR**ite. Instruction memory write strobe.
ICLK	Out	51	0	Instruction **CL**oc**K**. Delimits instruction fetch cycles. Rises during the first half of T1, signifying the start of an instruction cycle, and falls when the next instruction address is valid.
5.1.3 DATA MEMORY INTERFACE				
Address Bus:				
A15 (MSB)	Out	10	X	High byte of 16-bit memory Address.
A14	Out	11	X	
A13	Out	12	X	
A12	Out	13	X	
A11	Out	14	X	
A10	Out	15	X	
A9	Out	16	X	
A8	Out	17	X	
Multiplexed Address/Data Bus:				
AD7	In/Out	18	1	Low byte of 16-bit data memory Address, multiplexed with 8-bit Data bus.
AD6	In/Out	19	0	
AD5	In/Out	20	0	
AD4	In/Out	21	0	
AD3	In/Out	24	0	
AD2	In/Out	25	0	
AD1	In/Out	26	0	
AD0 (LSB)	In/Out	27	1	

5.0 Device Specifications (Continued)

Signal	In/Out	Pin	Reset State	Description
5.1.3 DATA MEMORY INTERFACE (Continued) **Timing/Control:**				
ALE	Out	28	0	**A**ddress **L**atch **E**nable. Demultiplexes AD bus. Address should be latched on the falling edge.
$\overline{\text{READ}}$	Out	29	1	Data memory **READ** strobe. Data is latched on the rising edge.
$\overline{\text{WRITE}}$	Out	30	1	Data memory **WRITE** strobe. Data is presented on the rising edge.
5.1.4 TRANSCEIVER INTERFACE				
DATA-IN	In	39	X	Logic level serial **DATA IN**put.
+ALG-IN	In	42	X	Non-inverting **AnaLoG IN**put for biphase serial data.
−ALG-IN	In	41	X	Inverting **AnaLoG IN**put for biphase serial data.
$\overline{\text{DATA-OUT}}$	Out	38	1	Biphase serial **DATA OUT**put (inverted).
DATA-DLY	Out	37	1	Biphase serial **DATA** output **DeLaY**ed by one-quarter bit time.
TX-ACT	Out	36	0	**T**ransmitter **ACT**ive. Normally low, goes high to indicate serial data is being transmitted. Used to enable external line drive circuitry.
5.1.5 REMOTE INTERFACE				
$\overline{\text{RAE}}$	In	46	X	**R**emote **A**ccess **E**nable. A "chip-select" input to allow host access of BCP functions and memory.
CMD	In	45	X	**CoMmanD** input. When high, remote accesses are directed to the Remote Interface Configuration register {RIC}. When low, remote accesses are directed to data-memory, instruction-memory or program counter as determined by {RIC}.
$\overline{\text{REM-RD}}$	In	47	X	**REM**ote **R**ea**D**. When active along with $\overline{\text{RAE}}$, a remote read cycle is requested; serviced by the BCP when the data bus becomes available.
$\overline{\text{REM-WR}}$	In	48	X	**REM**ote **WR**ite. When active along with $\overline{\text{RAE}}$, a remote write cycle is requested; serviced by the BCP when the data bus becomes available.
XACK	Out	50	1	Transfer **ACK**nowledge. Normally high, goes low on $\overline{\text{REM-RD}}$ or $\overline{\text{REM-WR}}$ going low (if $\overline{\text{RAE}}$ low), returning high when the transfer is complete. Normally used as a "wait" signal to a remote processor.
$\overline{\text{WR-PEND}}$	Out	49	1	**WR**ite **PEND**ing. In a system configuration where remote write cycles are latched, indicates when the latches contain valid data which is yet to be serviced by the BCP.
$\overline{\text{LOCK}}$	In	44	X	The remote processor uses this input to **LOCK** out local (BCP) accesses to data-memory. Once the remote processor has been granted the bus, $\overline{\text{LOCK}}$ gives it sole access to the bus and BCP accesses are "waited".
$\overline{\text{LCL}}$	Out	31	0	**LoCaL**. Normally low, goes high when the BCP relinquishes the data and address bus to service a Remote Access.
5.1.6 EXTERNAL INTERRUPTS				
$\overline{\text{BIRQ}}$	In/Out	53	In	**B**i-directional **I**nterrupt **ReQ**uest. As an input, can be used as an active low interrupt input (maskable and level-sensitive). As an output, can be used to generate remote system interrupts, reset via {RIC}.
$\overline{\text{NMI}}$	In	52	X	**N**on-**M**askable **I**nterrupt. Negative edge sensitive interrupt input.

5.0 Device Specifications (Continued)

5.2 ABSOLUTE MAXIMUM RATINGS (Notes 1 & 2)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	−0.5V to +7.0V
DC Input Voltage (V_{IN}) or	−0.5V to V_{CC} + 0.5V
DC Input Diode Current	±20 mA
DC Output Voltage (V_{OUT}) or	−0.5V to V_{CC} + 0.5V
DC Output Current, per Pin (I_{OUT})	±20 mA
DC V_{CC} or GND Current, per Pin	±50 mA
Storage Temperature Range (T_{STG})	−65°C to +150°C
Power Dissipation (PD)	500 mW
Lead Temperature (Soldering, 10 sec)	260°C
ESD Tolerance: C_{ZAP} = 120 pF, R_{ZAP} = 1500Ω	2.0 kV

5.3 OPERATING CONDITIONS

	Min	Max	Units
Supply Voltage (V_{CC})	4.5	5.5	V
DC Input or Output Voltage (V_{IN}, V_{OUT})	0.0	V_{CC}	V
Operating Temp. Range (T_A)	0	70	°C
Input Rise or Fall Times (t_r, t_f)		500	ns
Oscillator Crystal R_S		20	Ω
V_{CC} Power Up Ramp	6		ms

DC ELECTRICAL CHARACTERISTICS V_{CC} = 5V ±10% (unless otherwise specified)

Symbol	Parameter	Conditions	Guaranteed Limits 0–70°C	Units
V_{IH}	Minimum High Level Input Voltage			
	X1 (Note 3)		3.5	V
	All Other Inputs Except −ALG-IN, +ALG-IN		2.0	V
V_{IL}	Maximum Low Level Input Voltage			
	X1 (Note 3)		1.7	V
	All Other Inputs Except −ALG-IN, +ALG-IN		0.8	V
$V_{IH}-V_{IL}$	Minimum DATA-IN Hysteresis		0.1	V
V_{SENS}	Minimum Analog Input IN+, IN− Differential Sensitivity	*Figure 5-8b*	20	mV
V_{BIAS}	Common Mode Analog Input Bias Voltage	User Provided Bias Voltage	Min 2.25	V
			Max 2.75	V
V_{OH}	Minimum High Level	$V_{IN} = V_{IH}$ or V_{IL}		
	Output Voltage	$\|I_{OUT}\|$ = 20 μA	V_{CC} − 0.1	V
	IA, A, AD	$\|I_{OUT}\|$ = 4.0 mA, V_{CC} = 4.5V	3.5	V
	All Other Outputs	$\|I_{OUT}\|$ = 1.0 mA, V_{CC} = 4.5V	3.5	V
V_{OL}	Maximum Low Level	$V_{IN} = V_{IH}$ or V_{IL}		
	Output Voltage	$\|I_{OUT}\|$ = 20 μA	0.1	V
	IA, A, AD	$\|I_{OUT}\|$ = 4.0 mA, V_{CC} = 4.5V	0.4	V
	All Other Outputs	$\|I_{OUT}\|$ = 1.0 mA, V_{CC} = 4.5V	0.4	V
I_{IN}	Maximum Input Current	$V_{IN} = V_{CC}$ or GND		
		−ALG-IN, +ALG-IN	±10	μA
		X1 (Note 3)	±20	μA
		All Others	±10	μA
I_{OZ}	Maximum TRI-STATE® Output Leakage Current	$V_{OUT} = V_{CC}$ or GND	±10	μA
I_{CC}	Maximum Operating Supply Current Total to 4 V_{CC} Pins (Note 4)	$V_{IN} = V_{CC}$ or GND		
		TCLK = 8 MHz, CPU-CLK = 16 MHz		
		Xcvr and CPU Operating	61	mA
		Xcvr Idle, CPU Waited	29	mA
		$V_{IN} = V_{CC}$ or GND		
		TCLK = 20 MHz, CPU-CLK = 20 MHz		
		Xcvr and CPU Operating	71	mA
		Xcvr Idle, CPU Waited	31	mA

Note 1: Absolute Maximum Ratings are those values beyond which damage to the device may occur.
Note 2: Unless otherwise specified, all voltages are referenced to ground.
Note 3: X2 is an internal node with ESD protection. Do not use other than with crystal oscillator application.
Note 4: No DC loading, with X1 driven, no crystal. AC load per Test Circuit for Output Tests.

5.0 Device Specifications (Continued)

5.5 SWITCHING CHARACTERISTICS

The following specifications apply for V_{CC} = 4.5V to 5.5V, T_A = 0°C to 70°C.

5.5.1 Definitions

The timing specifications for the BCP are provided in the following tables and figures. The tables consist of five sections which are the following: the timing parameter symbol, the parameter ID#, the parameter description, the formula for the parameter, and the timing specification for the parameter. Below each table is a figure containing the waveforms for the parameters in the table.

The parameter symbol is composed of the type of timing specification and the signal or signals involved. Note that the symbols are unique only within a given table. The following symbol conventions are used for the type of timing specification.

- t_W — Pulse width specification
- t_{PD} — Propagation delay specification
- t_H — Hold time specification
- t_{SU} — Setup time specification
- t_{ZA} — High impedance to active delay specification (enable time)
- t_{AZ} — Active to high impedance delay specification (disable time)
- t_{ACC} — Access time specification
- t_T — Clock period specification

The parameter ID# is used to cross reference the timing parameter to the appropriate timing relationship in the accompanying figure. The waveforms in the figures are shown with the CPU clock running full speed ([CCS] = 0). For this case, CPU-CLK and CLK-OUT are equivalent. If CPU-CLK/2 is selected ([CCS] = 1), the effect on the waveforms with CLK-OUT is for CLK-OUT to double in frequency. The same is true for waveforms with X1. Note that CLK-OUT is always running at the crystal frequency and it is the CPU-CLK that is changing to half speed.

The parameter description defines the timing relationship being specified. BCP pin references are capitalized in the description.

Many of the timing specifications are dependent on variables such as operating frequency and number of programmed wait states. The formula for the parameter allows an accurate timing specification to be calculated for any combination of these variables. The formula represents the part of the timing specification that is synchronized to the internal CPU clock. This value is calculated and then added to the value specified under the Min or Max column to create the minimum or maximum guaranteed timing specification for the parameter.

The following acronyms are used in the tables:

DMEM refers to data memory

IMEM refers to instruction memory

RIC refers to the Remote Interface Control register

PC refers to the BCP Program Counter

T refers to the CPU clock period in ns

T_H refers to first half pulse width (high time) of the CPU clock in ns

T_L refers to second half pulse width (low time) of the CPU clock in ns.

C refers to the transceiver clock period in ns

n_{IW} is the number of instruction memory wait states programmed in DCR

n_{DW} is the number of data memory wait states programmed in DCR

n_{LW} is the number of remote wait states due to a BCP local data memory access

n_{RW} is the number of CPU wait states due to a remote access

MAX(A,B) means take the greater value of A or B

The following table is an example of the format used for the timing specifications. In this example, $t_{W\text{-}RD}$ indicates a pulse width specification for the output pin $\overline{READ}$. The ID# for locating the parameter in the timing waveforms is 10. The formula for this specification involves data and instruction memory wait states and the CPU clock period. For the case of 3 data memory wait states and 0 instruction memory wait states and a CPU clock period of 50 ns, the READ low minimum pulse width would be calculated as:

$(MAX(3,0-1)+1)T+(-10) = 4T - 10 = 190$ ns

For the case of 1 data memory wait state and 3 instruction memory wait states and a CPU clock period of 50 ns, the $\overline{READ}$ low minimum pulse width would be calculated as:

$(MAX(1,3-1)+1)T+(-10) = 3T - 10 = 140$ ns

To calculate n_{LW} the following two equations are needed:

n_{LW} (min) = 0

n_{LW} (max) = MAX(n_{DW}, n_{IW}−1) + Data Memory Access Cycle

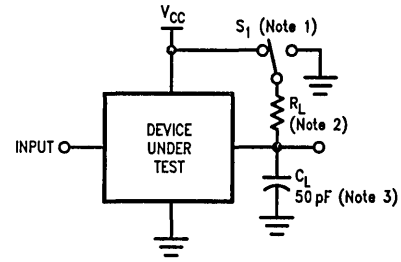
Data Memory Access Cycle is normally 3 T-states if [4TR] = 0 and 4 T-states if [4TR] = 1. Keep in mind that both [LOR] and $\overline{WAIT}$ can extend n_{LW}.

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{W\text{-}RD}$	10	$\overline{Read}$ Low	$(MAX(n_{DW},n_{IW}-1)+1)T+$	−10	10	ns

5.0 Device Specifications (Continued)

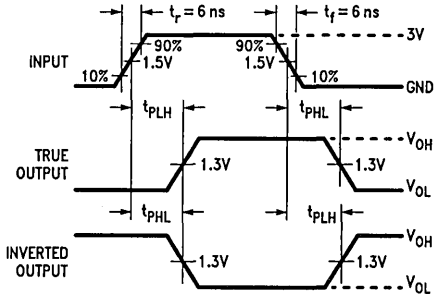
Note 1: S_1 = V_{CC} for t_{PZL}, and t_{PLZ} measurements
S_1 = GND for t_{PZH}, and t_{PHZ} measurements
S_1 = Open for push pull outputs
Note 2: R_L = 1.1k for 4 mA outputs
R_L = 4.4k for 1 mA outputs
Note 3: C_L includes scope and jig capacitance.

Test Circuit for Output Tests



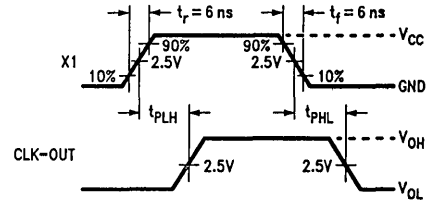
TL/F/9336–A2

Propagation Delay Waveforms Except for Oscillator



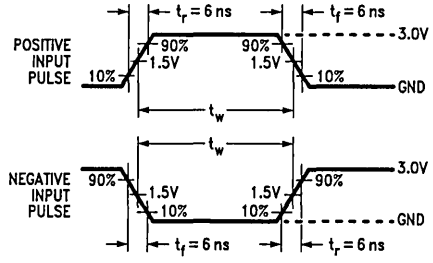
TL/F/9336–A3

Propagation Delay Waveform for Oscillator



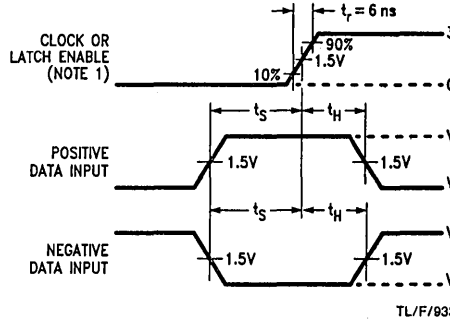
TL/F/9336–A4

Input Pulse Width Waveforms



TL/F/9336–A5

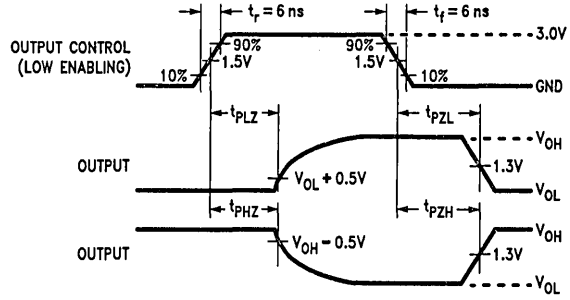
Setup and Hold Time Waveforms



TL/F/9336–A6

Note 1: Waveform for negative edge sensitive circuits will be inverted.

TRI-STATE Output Enable and Disable Waveforms



TL/F/9336–A7

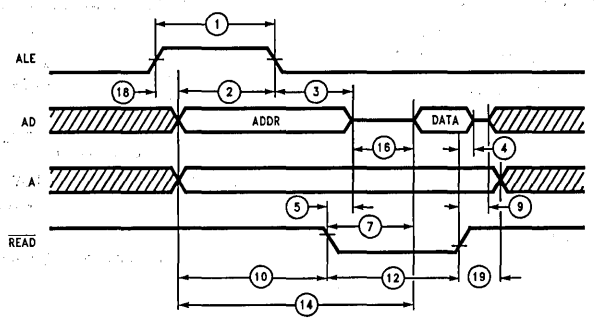
FIGURE 5-2. Switching Characteristic Measurement Waveforms

5.0 Device Specifications (Continued)

TABLE 5-3. Data Memory Read Timing (Note 1)

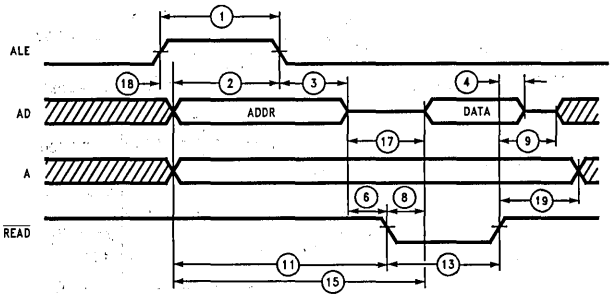
Symbol	ID #	Parameter	Formula	Min	Max	Units
$t_{W\text{-}ALE}$	1	ALE High	$(n_{RW}+1)T+$	−10	12	ns
$t_{PD\text{-}AAD\text{-}ALE}$	2	A, AD (Data Address) Valid to ALE Falling	$T+$	−22		ns
$t_{PD\text{-}ALE\text{-}AD}$	3	ALE Falling to AD (Data Address) Invalid	T_L+	−2		ns
$t_{H\text{-}RD\text{-}DATA}$	4	Data Valid after $\overline{READ}$ Rising		0		ns
$t_{AZ\text{-}RD\text{-}AD}$	5	$\overline{READ}$ Falling to AD Disabled ([4TR] = 0)			20	ns
$t_{AZ\text{-}AD\text{-}RD}$	6	AD Disabled before $\overline{READ}$ Falling ([4TR] = 1)	T_H+	−20		ns
$t_{SU\text{-}RD\text{-}DATA}$	7	$\overline{READ}$ Falling to AD (Data) Setup ([4TR] = 0)	$(MAX(n_{DW},n_{IW}-1)+1)T+$		−22	ns
$t_{SU\text{-}RD\text{-}DATA}$	8	$\overline{READ}$ Falling to AD (Data) Setup ([4TR] = 1)	$(MAX(n_{DW}-1,n_{IW}-1)+1)T+T_L+$		−21	ns
$t_{ZA\text{-}RD\text{-}AD}$	9	$\overline{READ}$ Rising to AD Enabled	T_H+	−2		ns
$t_{PD\text{-}AAD\text{-}RD}$	10	A, AD (Data Address) Valid before $\overline{READ}$ Falling ([4TR] = 0)	$T+T_L+$	−27		ns
$t_{PD\text{-}AAD\text{-}RD}$	11	A, AD (Data Address) Valid before $\overline{READ}$ Falling ([4TR] = 1)	$2T+$	−27		ns
$t_{W\text{-}RD}$	12	$\overline{READ}$ Low ([4TR] = 0)	$(MAX(n_{DW},n_{IW}-1)+1)T+$	−10	10	ns
$t_{W\text{-}RD}$	13	$\overline{READ}$ Low ([4TR] = 1)	$(MAX(n_{DW}-1,n_{IW}-1)+1)T+T_L+$	−10	10	ns
$t_{ACC\text{-}D}$	14	Data Memory Read Time ([4TR]) = 0)	$(MAX(n_{DW},n_{IW}-1)+2)T+T_L+$		−40	ns
$t_{ACC\text{-}D}$	15	Data Memory Read Time ([4TR]) = 1)	$(MAX(n_{DW}-1,n_{IW}-1)+3)T+T_L+$		−40	ns
$t_{SU\text{-}AD\text{-}DATA}$	16	AD Disabled to AD (Data) Setup ([4TR] = 0)	$(MAX(n_{DW},n_{IW}-1)+1)T+$	−33		ns
$t_{SU\text{-}AD\text{-}DATA}$	17	AD Disabled to AD (Data) Setup ([4TR] = 1)	$(MAX(n_{DW}-1,n_{IW}-1)+2)T+$	−33		ns
$t_{PD\text{-}ALE\text{-}AAD}$	18	ALE Rising to A, AD (Data Address) Valid	$(n_{RW})T+$		24	ns
$t_{PD\text{-}RD\text{-}A}$	19	$\overline{READ}$ Rising to A Invalid	T_H+	0		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.



TL/F/9336-52

(a) Read Timing with ([4TR] = 0)



TL/F/9336-H7

(b) Read Timing with ([4TR] = 1)

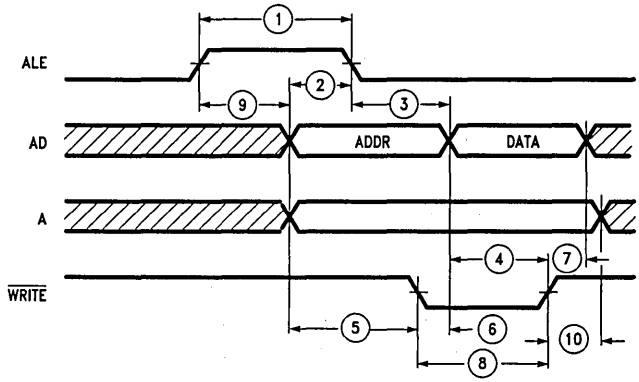
FIGURE 5-3. Data Memory Read Timing

5.0 Device Specifications (Continued)

TABLE 5-4. Data Memory Write Timing (Note 1)

Symbol	ID #	Parameter	Formula	Min	Max	Units
$t_{W\text{-}ALE}$	1	ALE High	$(n_{RW}+1)T+$	−10	12	ns
$t_{PD\text{-}AAD\text{-}ALE}$	2	A, AD (Data Address) Valid to ALE Falling	$T+$	−22		ns
$t_{PD\text{-}ALE\text{-}AD}$	3	ALE Falling to AD (Data Address) Invalid	T_L+	−2		ns
$t_{PD\text{-}DATA\text{-}WR}$	4	AD (Data) Valid to $\overline{\text{WRITE}}$ Rising	$(\text{MAX}(n_{DW},n_{IW}-1)+1)T+$	−20		ns
$t_{PD\text{-}AAD\text{-}WR}$	5	A, AD (Data Address) Valid to $\overline{\text{WRITE}}$ Falling	$1.5T+$	−28		ns
$t_{PD\text{-}WR\text{-}DATA}$	6	$\overline{\text{WRITE}}$ Falling to AD (Data) Valid			19	ns
$t_{PD\text{-}WR\text{-}DATAz}$	7	$\overline{\text{WRITE}}$ Rising to AD (Data) Invalid	T_H+	−4		ns
$t_{W\text{-}WR}$	8	$\overline{\text{WRITE}}$ Low	$(\text{MAX}(n_{DW},n_{IW}-1)+1)T+$	−10	10	ns
$t_{PD\text{-}ALE\text{-}AAD}$	9	ALE Rising to A, AD (Data Address) Valid	$(n_{RW})T+$		24	ns
$t_{PD\text{-}WR\text{-}A}$	10	$\overline{\text{WRITE}}$ Rising to A Invalid	T_H+	−2		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.



TL/F/9336-53

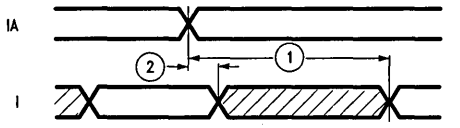
FIGURE 5-4. Data Memory Write Timing

5.0 Device Specifications (Continued)

TABLE 5-5. Instruction Memory Read Timing (Note 1)

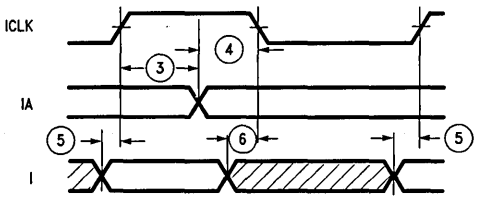
Symbol	ID#	Parameter	Formula	Min	Max	Units
t_{ACC-I}	1	Instruction Memory Read Time	$(n_{IW}+1)T+T_L+$		−19	ns
t_{H-IA-I}	2	IA Invalid to I Invalid		0		ns
$t_{PD-ICLK-IA}$	3	ICLK Rising to IA Invalid	T_H+	−13		ns
$t_{PD-IA-ICLK}$	4	Next IA Valid before ICLK Falling	T_L+	−12		ns
$t_{PD-IAz-ICLK}$		IA Invalid before ICLK Falling			17	ns
$t_{SU-I-ICLK}$	5	I Valid before ICLK Rising		20		ns
$t_{H-I-ICLK}$	6	I Invalid before ICLK Falling	T_L+		0	ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.



TL/F/9336-A9

(a) Instruction Memory Read Timing



TL/F/9336-54

(b) Instruction ICLK Timing

FIGURE 5-5. Instruction Memory Timing

5.0 Device Specifications (Continued)

TABLE 5-6. Clock Timing (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{T\text{-}X1}$	1	X1 Period (Note 2)		50	500	ns
$t_{PD\text{-}X1\text{-}CO}$	2	X1 to CLK-OUT (Note 2)			37	ns
$t_{PD\text{-}CO\text{-}ICLKr}$	3	CLK-OUT Rising to ICLK Rising			15	ns
$t_{PD\text{-}CO\text{-}ICLKf}$	4	CLK-OUT Rising to ICLK Falling (Note 3)			15	ns
$t_{T\text{-}XT}$	5	X-TCLK Period (Note 4)		50	500	ns
$t_{W\text{-}X1HL}$	6	X1 High and Low time Pulse Widths (Note 5)		21		ns
$t_{W\text{-}XTHL}$	7	XTCLK High and Low Time Pulse Widths		15		ns

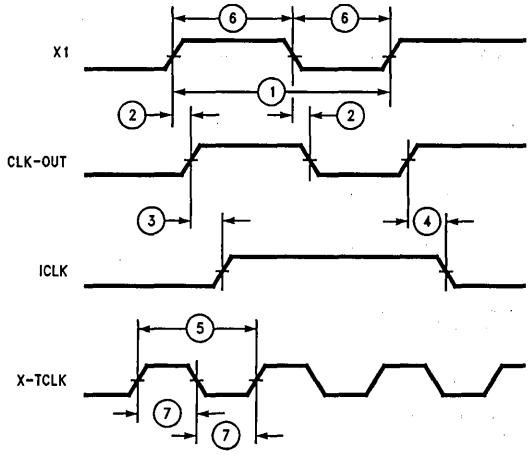
Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: Measurement thresholds at 2.5V.

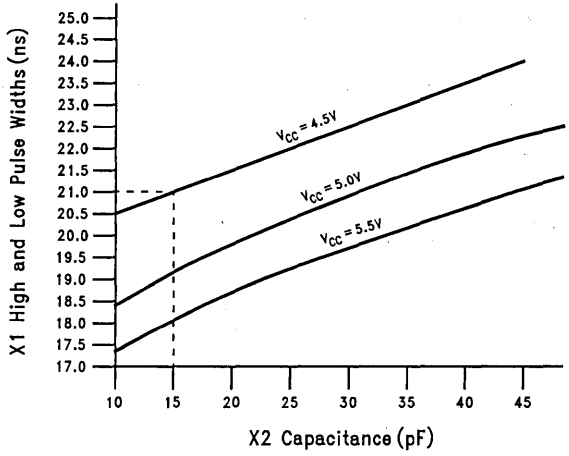
Note 3: The falling edge of ICLK occurs only after the next IA becomes valid. The CLK-OUT cycle in which this occurs depends on the instruction being executed and the number of programmed instruction wait states.

Note 4: There is no relationship between X1 and X-TCLK. X-TCLK is fully asynchronous.

Note 5: External loading on pin X2 equal to 15 pF. See *Figure 5-6b* for affect of X2 loading in non-crystal applications (i.e., an external oscillator driving X1).



TL/F/9336-55



TL/F/9336-H8

FIGURE 5-6. Clock Timing

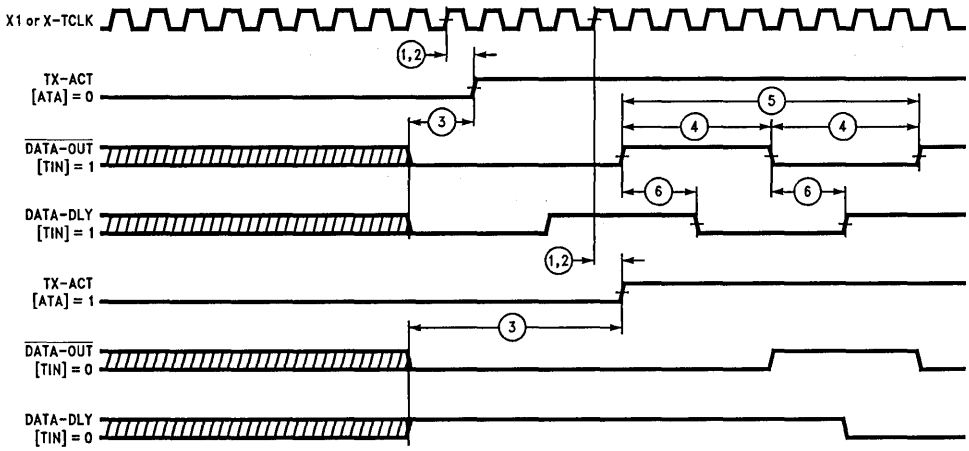
5.0 Device Specifications (Continued)

TABLE 5-7. Transceiver Timing (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{PD\text{-}X1\text{-}TA}$	1	X1 Rising to TX-ACT Rising/Falling		10	65	ns
$t_{PD\text{-}XTCLK\text{-}TA}$	2	X-TCLK Rising to TX-ACT Rising/Falling		7	49	ns
$t_{PD\text{-}DODD\text{-}TA}$	3	$\overline{\text{DATA-OUT}}$, DATA-DLY Valid to TX-ACT Rising	C+	16		ns
$t_{W\text{-}DO\text{-}HB}$	4	$\overline{\text{DATA-OUT}}$ Half Bit Cell Width	4C+	−10	10	ns
$t_{W\text{-}DO\text{-}FB}$	5	$\overline{\text{DATA-OUT}}$ Full Bit Cell Width	8C+	−10	10	ns
$t_{PD\text{-}DO\text{-}DD}$	6	$\overline{\text{DATA-OUT}}$ Falling/Rising to DATA-DLY Rising/Falling (Note 3)	2C+	−10	10	ns

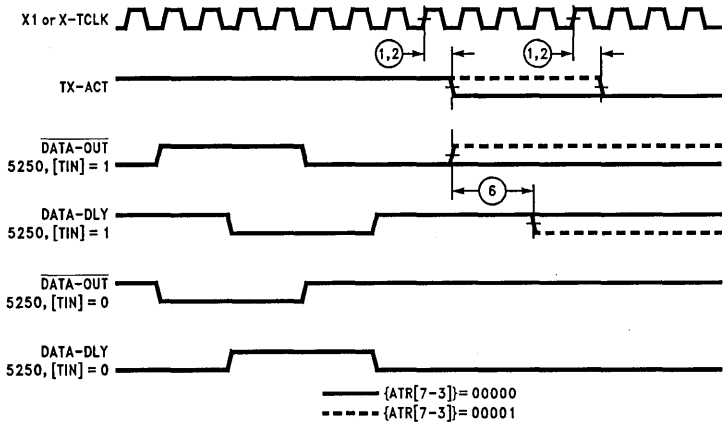
Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: When [ATA] = 1, TX-ACT is delayed by 4C and an additional line quiescent is generated resulting in 5½ line quiescent pulses after the line interface logic. The additional delay relative to a message with [ATA] = 0 is 8C (one bit time).



TL/F/9336-56

(a) Transmission Beginning Timing (Note 2)



TL/F/9336-57

(b) Transmission Ending Timing

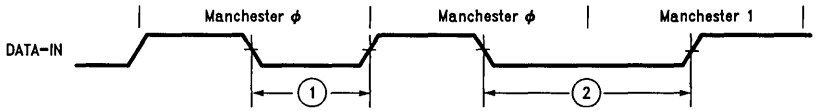
FIGURE 5-7. Transceiver Timing

5.0 Device Specifications (Continued)

TABLE 5-8. Analog and DATA-IN Timing (Note 1)

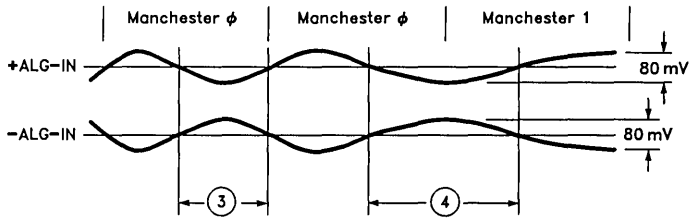
Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{W\text{-}DI\text{-}hb}$	1	DATA-IN Data, Half Bit Width	3C+	12		ns
			5C+		−12	ns
$t_{W\text{-}DI\text{-}fb}$	2	DATA-IN Data, Full Bit Width	7C+	12		ns
			9C+		−12	ns
$t_{W\text{-}AI\text{-}hb}$	3	Analog Data, Half Bit Width (−ALG-IN or +ALG-IN)	3C+	20		ns
			5C+		−20	ns
$t_{W\text{-}AI\text{-}fb}$	4	Analog Data, Full Bit Width (−ALG-IN or +ALG-IN)	7C+	20		ns
			9C+		−20	ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.



TL/F/9336-58

(a) DATA-IN Jitter Timing (3270)



TL/F/9336-59

(b) Analog Jitter Timing (3270)

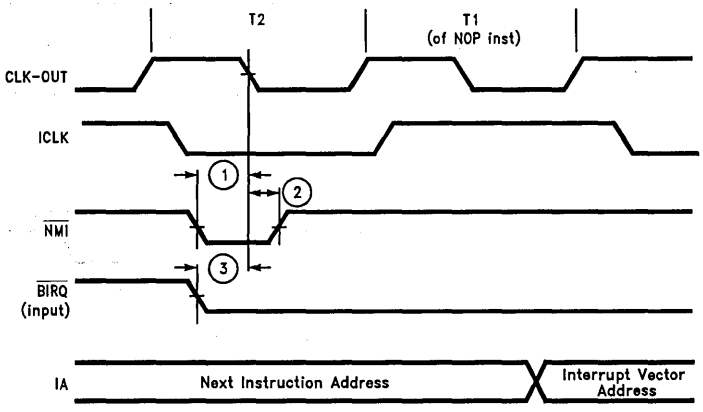
FIGURE 5-8. Analog and DATA-IN Timing

5.0 Device Specifications (Continued)

TABLE 5-9. Interrupt Timing (Note 1)

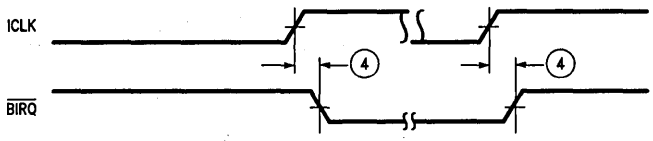
Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}NMI\text{-}CO}$	1	$\overline{NMI}$ Falling before CLK-OUT Falling		12		ns
$t_{H\text{-}NMI\text{-}CO}$	2	$\overline{NMI}$ Hold after CLK-OUT Falling		8		ns
$t_{SU\text{-}BQ\text{-}CO}$	3	$\overline{BIRQ}$ (Input) Falling before CLK-OUT Falling		13		ns
$t_{PD\text{-}ICLK\text{-}BQ}$	4	ICLK Rising to $\overline{BIRQ}$ (Output) Rising/Falling			24	ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.



TL/F/9336-60

(a) Interrupt Timing



TL/F/9336-61

(b) $\overline{BIRQ}$ Output Timing

FIGURE 5-9. Interrupt Timing

5.0 Device Specifications (Continued)

TABLE 5-10. Control Pin Timing (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{W\text{-}RST}$	1	$\overline{RESET}$ Low	5T+	0		ns
$t_{PD\text{-}RST\text{-}ICLK}$	2	$\overline{RESET}$ Rising to ICLK Rising	4T+		0	ns
$t_{SU\text{-}ALE\text{-}WT}$	3	$\overline{WAIT}$ Low after ALE High to Extend Cycle	$(MAX(n_{DW},n_{IW}-1)+1)T+$		−21	ns
$t_{H\text{-}WT\text{-}ALE}$	4	$\overline{WAIT}$ Rising after ALE Falling (Note 2)		0		ns
			$(MAX(n_{DW},n_{IW}-1)+1)T+$		−28	ns
$t_{PD\text{-}WT\text{-}RDWR}$	5	$\overline{WAIT}$ Rising to $\overline{READ}$ or $\overline{WRITE}$ Rising	$T+T_L+$	−22		ns
			$2T+T_L+$		2	ns
$t_{SU\text{-}RRW\text{-}RST}$	6	$\overline{REM\text{-}RD}$, $\overline{REM\text{-}WR}$ Low to $\overline{RESET}$ Rising for BCP to Start		15		ns
$t_{H\text{-}RST\text{-}RRW}$	7	$\overline{REM\text{-}RD}$, $\overline{REM\text{-}WR}$ Low after $\overline{RESET}$ Rising for BCP to Start		5		ns
$t_{SU\text{-}LK\text{-}ICLK}$	8	$\overline{LOCK}$ Low before ICLK High (Note 3)	T_L+	19		ns
$t_{PD\text{-}LK\text{-}ALE}$	9	$\overline{LOCK}$ High to ALE Low	T+	−2		ns
			3T+		20	ns
$t_{SU\text{-}WT\text{-}ICLK}$	10	$\overline{WAIT}$ Low after ICLK Rising to Extend Cycle (Note 4)	$(MAX(n_{DW},n_{IW}-1))T+T_H+$		−22	ns
$t_{H\text{-}WT\text{-}ICLK}$	11	$\overline{WAIT}$ High after ICLK Rising (Notes 2, 4)	$(MAX(n_{DW},n_{IW}-1))T+T_H+$	2		ns
			$(MAX(n_{DW},n_{IW}-1)+1)T+T_H+$		−20	ns
$t_{H\text{-}LK\text{-}ICLK}$	12	$\overline{LOCK}$ Rising after ICLK High	T_H+	2		ns
$t_{PD\text{-}AD\text{-}ALE}$	13	AD to ALE Falling after $\overline{LOCK}$ Rising	T+	−33		ns
$t_{SU\text{-}WT\text{-}ALEf}$	14	$\overline{WAIT}$ Low before ALE Falling to Extend Cycle		23		ns

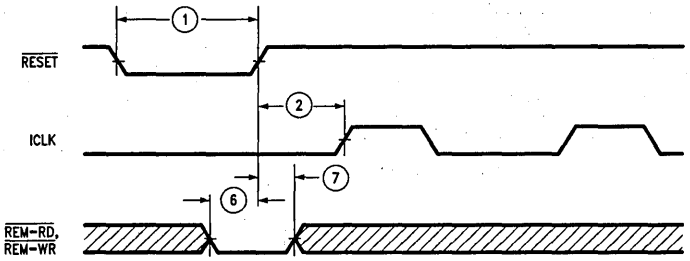
Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: The maximum value for this parameter is the lastest $\overline{WAIT}$ can be removed without adding an additional T-state. The formula assumes a minimum externally generated wait of one T-state.

Note 3: If $t_{SU\text{-}LK\text{-}ICLK}$ is not met, the maximum time from $\overline{LOCK}$ low till no more local accesses is $(MAX(n_{DW}, n_{IW}-1)+3)T$.

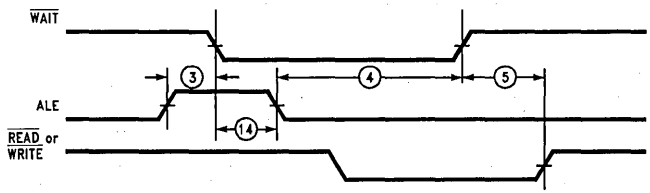
Note 4: The formula(s) apply to a 2 T-state instruction. For a 3 T-state instruction, add one T-state; for a 4 T-state instruction, add two T-states.

5.0 Device Specifications (Continued)



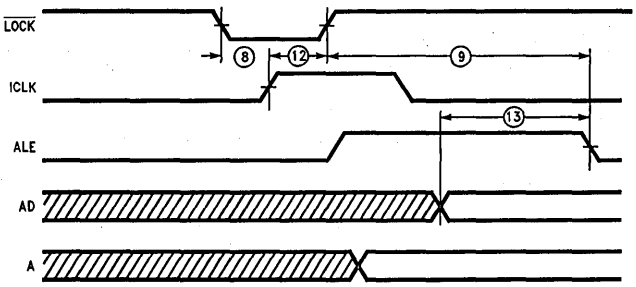
TL/F/9336-62

(a) Reset Timing



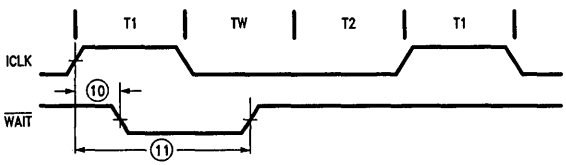
TL/F/9336-63

(b) BCP Access WAIT Timing



TL/F/9336-64

(c) LOCK Timing



TL/F/9336-A8

(d) Instruction WAIT Timing

FIGURE 5-10. Control Pin Timing

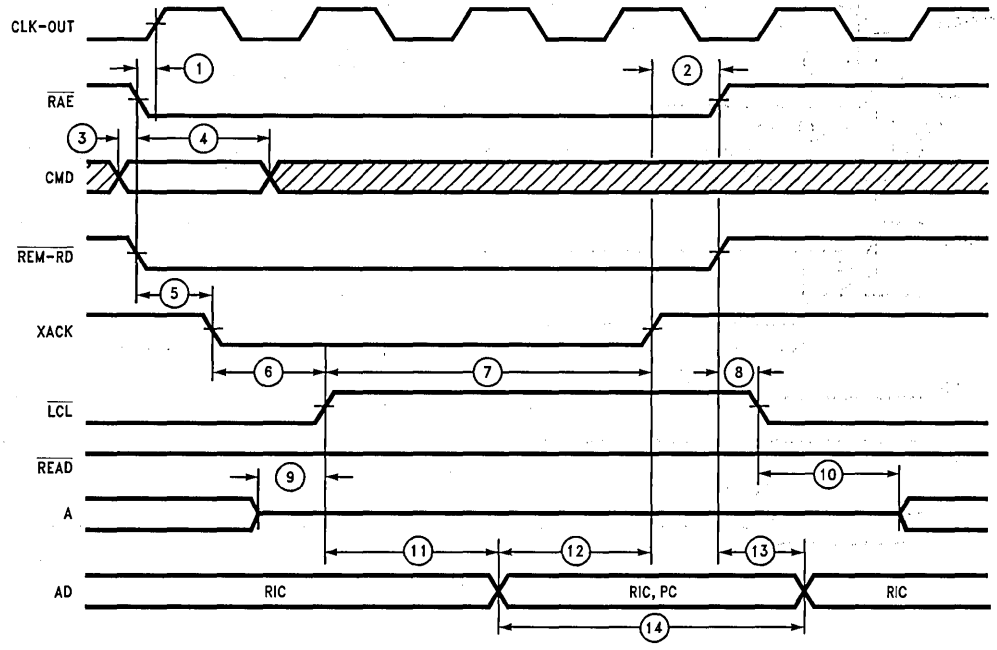
5.0 Device Specifications (Continued)

TABLE 5-11. Buffered Read of PC, RIC (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRR\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling before CLK-OUT Rising		22		
$t_{H\text{-}RRR\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising after XACK Rising (Note 2)		0		ns
			2T+		−34	ns
$t_{SU\text{-}CMD\text{-}RRR}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRR}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling	T+	26		ns
$t_{PD\text{-}RRR\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}X\text{-}LCL}$	6	XACK Falling to $\overline{LCL}$ Rising	$(n_{LW}+1)T+$	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	2T+	−10	8	ns
$t_{PD\text{-}RRR\text{-}LCL}$	8	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising to $\overline{LCL}$ Falling		3		ns
$t_{AZ\text{-}A\text{-}LCL}$	9	A Disabled before $\overline{LCL}$ Rising	T_L+	−18		ns
$t_{ZA\text{-}LCL\text{-}A}$	10	A Enabled after $\overline{LCL}$ Falling	T_H+	15		ns
$t_{PD\text{-}LCL\text{-}PC}$	11	$\overline{LCL}$ Rising to AD (PC) Valid	T+		22	ns
$t_{PD\text{-}PC\text{-}X}$	12	AD (PC, RIC) Valid before XACK Rising	T+	−24		ns
$t_{PD\text{-}PC\text{-}RRR}$	13	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising to AD (PC) Invalid		6		ns
$t_{W\text{-}PC}$	14	AD (PC, RIC) Valid Time	T+	−2		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: The maximum value for this parameter is the latest $\overline{RAE}$, $\overline{REM\text{-}RD}$ can be removed without adding a T-state to the remote access.



TL/F/9336-65

FIGURE 5-11. Buffered Read of PC, RIC

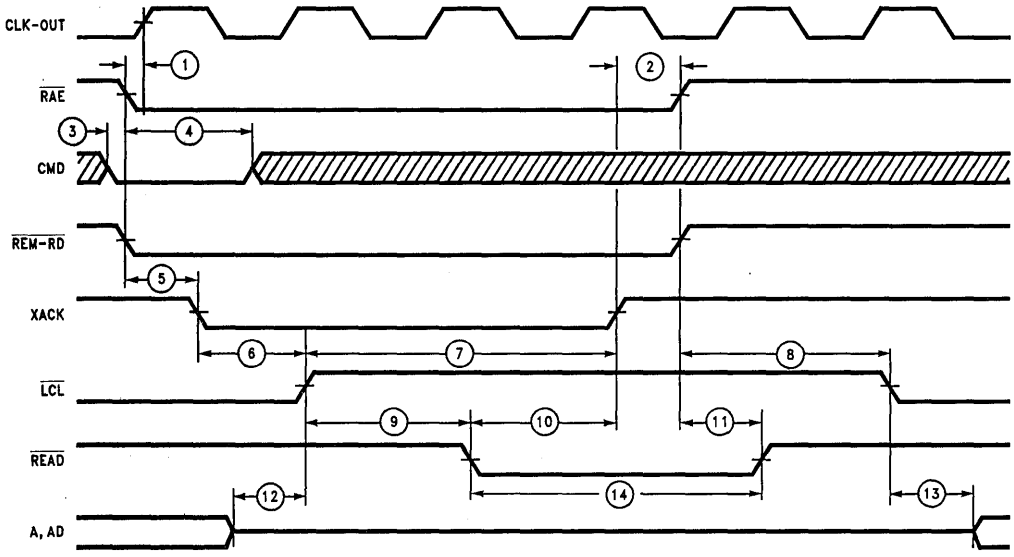
5.0 Device Specifications (Continued)

TABLE 5-12. Buffered Read of DMEM (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRR\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling before CLK-OUT Rising		22		ns
$t_{H\text{-}RRR\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising after XACK Rising (Note 2)		0		ns
			T+		−32	ns
$t_{SU\text{-}CMD\text{-}RRR}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRR}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling	T+	26		ns
$t_{PD\text{-}RRR\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}X\text{-}LCL}$	6	XACK Falling to $\overline{LCL}$ Rising	$(n_{LW}+1)T+$	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	$(n_{DW}+2)T+$	−10	8	ns
$t_{PD\text{-}RRR\text{-}LCL}$	8	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising to $\overline{LCL}$ Falling	T+	3		ns
$t_{PD\text{-}LCL\text{-}RD}$	9	$\overline{LCL}$ Rising to $\overline{READ}$ Falling	T+	−5	16	ns
$t_{PD\text{-}RD\text{-}X}$	10	$\overline{READ}$ Falling to XACK Rising	$(n_{DW}+1)T+$	−15		ns
$t_{PD\text{-}RRR\text{-}RD}$	11	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising to $\overline{READ}$ Rising		1	28	ns
$t_{AZ\text{-}AAD\text{-}LCL}$	12	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{ZA\text{-}LCL\text{-}AAD}$	13	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{W\text{-}RD}$	14	Read Low	$(n_{DW}+1)T+$	−4		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: The maximum value for this parameter is the latest $\overline{RAE}$, $\overline{REM\text{-}RD}$ can be removed without adding a T-state to the remote access.



TL/F/9336-66

FIGURE 5-12. Buffered Read of DMEM

5.0 Device Specifications (Continued)

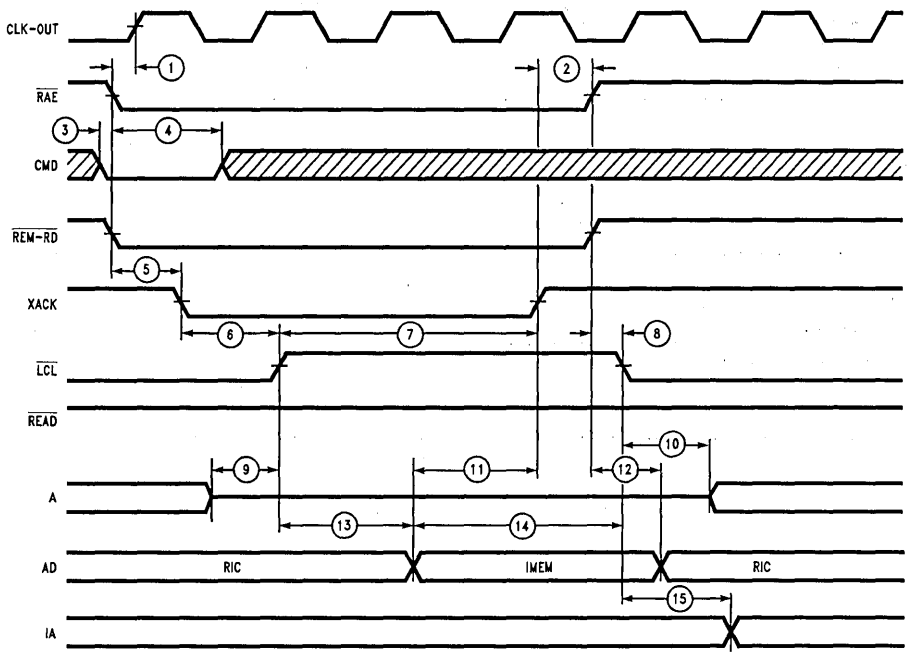
TABLE 5-13. Buffered Read of IMEM (Note 1)

Symbol	ID #	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRR\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling before CLK-OUT Rising		22		ns
$t_{H\text{-}RRR\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising after XACK Rising (Note 2)		0		ns
			T+		−32	ns
$t_{SU\text{-}CMD\text{-}RRR}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRR}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling	T+	26		ns
$t_{PD\text{-}RRR\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}X\text{-}LCL}$	6	XACK Falling to $\overline{LCL}$ Rising	T+	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	$(n_{IW}+2)T+$	−10	8	ns
$t_{PD\text{-}RRR\text{-}LCL}$	8	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising to $\overline{LCL}$ Falling		3		ns
$t_{AZ\text{-}LCL\text{-}A}$	9	A Disabled after $\overline{LCL}$ Rising	T_L+	−18		ns
$t_{ZA\text{-}A\text{-}LCL}$	10	A Enabled before $\overline{LCL}$ Falling	T_H+	15		ns
$t_{PD\text{-}IMEM\text{-}X}$	11	AD (IMEM) Valid before XACK Rising	$(n_{IW}+1)T+$	−25		ns
$t_{PD\text{-}RRR\text{-}IMEM}$	12	AD (IMEM) Invalid after $\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising		10		ns
$t_{PD\text{-}LCL\text{-}IMEM}$	13	$\overline{LCL}$ Rising to AD (IMEM) Valid	T+		22	ns
$t_{W\text{-}IMEM}$	14	(IMEM) Valid	$(n_{IW}+1)T+$	0		ns
$t_{PD\text{-}LCL\text{-}IA}$	15	$\overline{LCL}$ Falling to Next IA Valid (Note 3)	T_H+	8		ns
			$T+T_H+$		44	ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: The maximum value for this parameter is the latest $\overline{RAE}$, $\overline{REM\text{-}RD}$ can be removed without adding a T-state to the remote access.

Note 3: Two remote reads from instruction memory are necessary to read a 16-bit instruction word from IMEM—low byte followed by high byte. The timing for the two reads are the same except that IA is incremented after the high instruction memory byte is read.



TL/F/9336-67

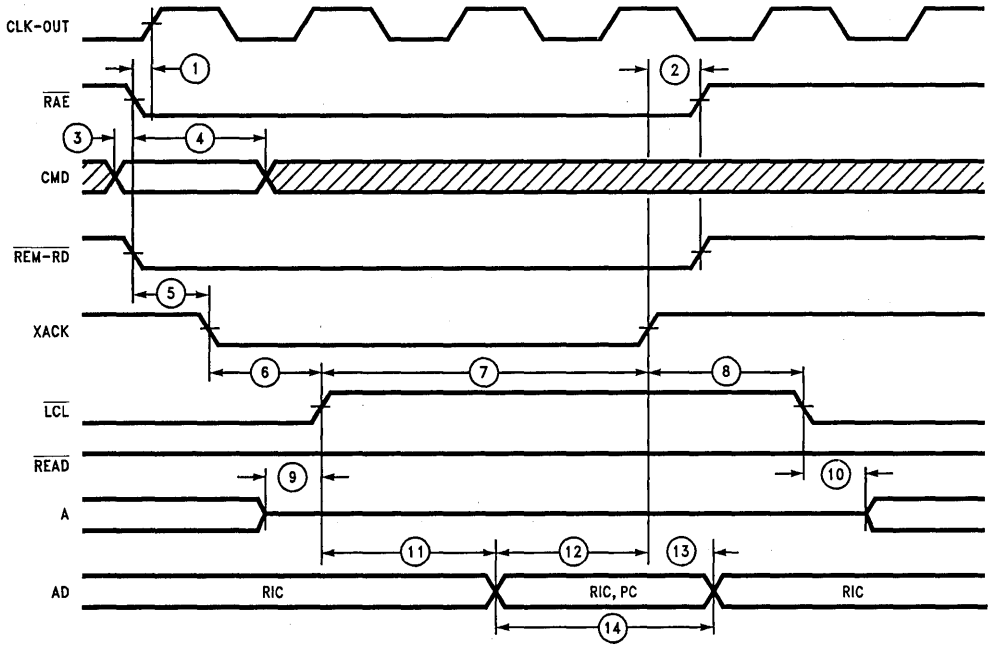
FIGURE 5-13. Buffered Read of IMEM

5.0 Device Specifications (Continued)

TABLE 5-14. Latched Read of PC, RIC (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRR\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling before CLK-OUT Rising		22		ns
$t_{H\text{-}RRR\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising after XACK Rising		0		ns
$t_{SU\text{-}CMD\text{-}RRR}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRR}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling	T+	26		ns
$t_{PD\text{-}RRR\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}Xf\text{-}LCLr}$	6	XACK Falling to $\overline{LCL}$ Rising	$(n_{LW}+1)T+$	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	2T+	−10	8	ns
$t_{PD\text{-}Xr\text{-}LCLf}$	8	XACK Rising to $\overline{LCL}$ Falling	T+	−11	11	ns
$t_{AZ\text{-}A\text{-}LCL}$	9	A Disabled before $\overline{LCL}$ Rising	T_L+	−18		ns
$t_{ZA\text{-}LCL\text{-}A}$	10	A Enabled after $\overline{LCL}$ Falling	T_H+	−12		ns
$t_{PC\text{-}LCL\text{-}PC}$	11	$\overline{LCL}$ Rising to AD (PC) Valid	T+		20	ns
$t_{PD\text{-}PC\text{-}X}$	12	AD (PC) Valid before XACK Rising	T+	−22		ns
$t_{PD\text{-}X\text{-}PC}$	13	XACK Rising to AD (PC) Invalid	T_H+	0		ns
$t_{W\text{-}PC}$	14	AD (PC, RIC) Valid	$T+T_H+$	−12		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.



TL/F/9336-68

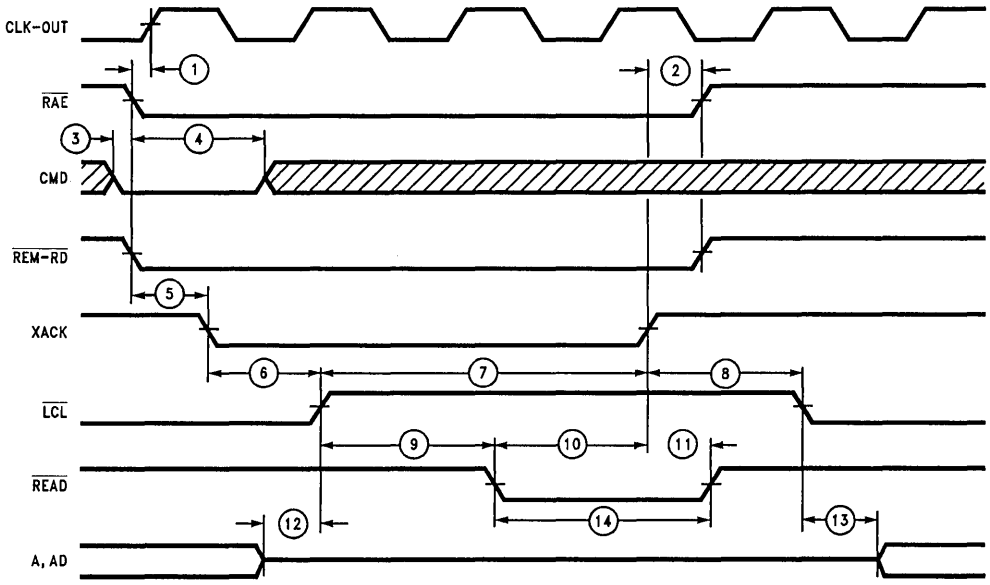
FIGURE 5-14. Latched Read of PC, RIC

5.0 Device Specifications (Continued)

TABLE 5-15. Latched Read of DMEM (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRR\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling before CLK-OUT Rising		22		ns
$t_{H\text{-}RRR\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising after XACK Rising		0		ns
$t_{SU\text{-}CMD\text{-}RRR}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRR}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling	T+	26		ns
$t_{PD\text{-}RRR\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}Xf\text{-}LCLr}$	6	XACK Falling to $\overline{LCL}$ Rising	$(n_{LW}+1)T+$	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	$(n_{DW}+2)T+$	−10	8	ns
$t_{PD\text{-}Xr\text{-}LCLf}$	8	XACK Rising to $\overline{LCL}$ Falling	T+	−11	11	ns
$t_{PC\text{-}LCL\text{-}RD}$	9	$\overline{LCL}$ Rising to $\overline{READ}$ Falling	T+	−5	16	ns
$t_{PD\text{-}RD\text{-}X}$	10	$\overline{READ}$ Falling before XACK Rising	$(n_{DW}+1)T+$	−15		ns
$t_{PD\text{-}X\text{-}RD}$	11	XACK Rising to $\overline{READ}$ Rising	T_H+	−7	12	ns
$t_{AZ\text{-}AAD\text{-}LCL}$	12	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{ZA\text{-}LCL\text{-}AAD}$	13	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{W\text{-}RD}$	14	$\overline{READ}$ Low	$(n_{DW}+1)T+T_H+$	−12		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.



TL/F/9336-69

FIGURE 5-15. Latched Read of DMEM

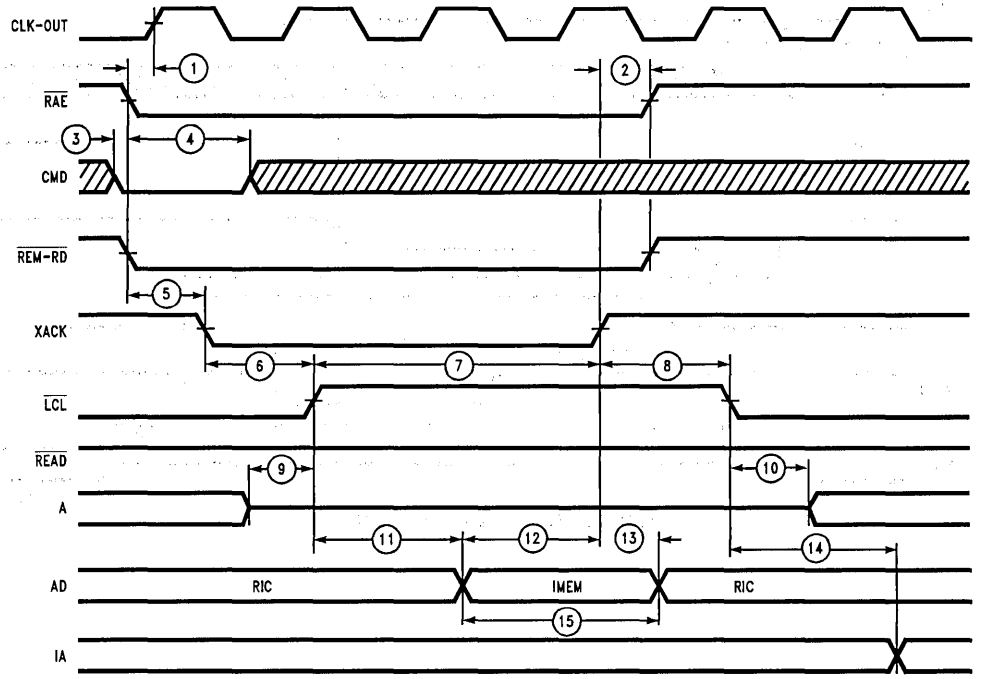
5.0 Device Specifications (Continued)

TABLE 5-16. Latched Read of IMEM (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRR\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling before CLK-OUT Rising		22		ns
$t_{H\text{-}RRR\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Rising after XACK Rising		0		ns
$t_{SU\text{-}CMD\text{-}RRR}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRR}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling	T+	26		ns
$t_{PD\text{-}RRR\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}RD}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}Xf\text{-}LCLr}$	6	XACK Falling to $\overline{LCL}$ Rising	T+	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	$(n_{IW}+2)T+$	−10	8	ns
$t_{PD\text{-}Xr\text{-}LCLf}$	8	XACK Rising to $\overline{LCL}$ Falling	T+	−11	11	ns
$t_{AZ\text{-}A\text{-}LCL}$	9	A Disabled before $\overline{LCL}$ Rising	T_L+	−18		ns
$t_{ZA\text{-}LCL\text{-}A}$	10	A Enabled after $\overline{LCL}$ Falling	T_H+	−12		ns
$t_{PD\text{-}LCL\text{-}IMEM}$	11	$\overline{LCL}$ Rising to AD (IMEM) Valid	T+		22	ns
$t_{PD\text{-}IMEM\text{-}X}$	12	AD (IMEM) Valid to XACK Rising	$(n_{IW}+1)T+$	−23		ns
$t_{PD\text{-}X\text{-}IMEM}$	13	XACK Rising to AD (IMEM) Invalid	T_H+	1		ns
$t_{PD\text{-}LCL\text{-}IA}$	14	$\overline{LCL}$ Falling to Next IA Valid (Note 2)	$T+T_H+$	−19	5	ns
$t_{W\text{-}IMEM}$	15	IMEM Valid	$(n_{IW}+1)T+T_H+$	−9		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: Two remote reads from instruction memory are necessary to read a 16-bit instruction word from IMEM—low byte followed by high byte. The timing for the two reads are the same except that IA is incremented after the high instruction memory byte is read.



TL/F/9336-70

FIGURE 5-16. Latched Read of IMEM

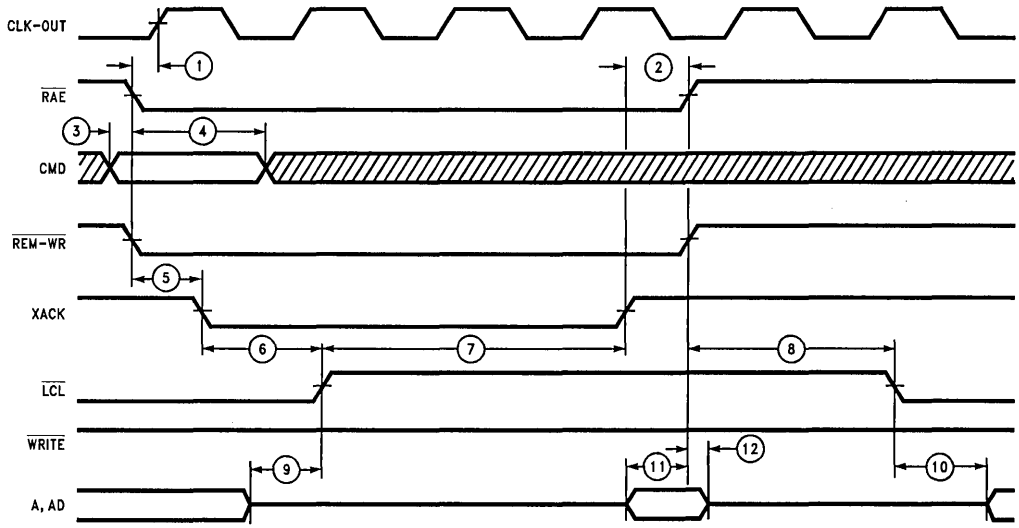
5.0 Device Specifications (Continued)

TABLE 5-17. Slow Buffered Write of PC, RIC (Note 1)

Symbol	ID #	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRW\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling before CLK-OUT Rising		24		ns
$t_{H\text{-}RRW\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after XACK Rising (Note 2)		0		ns
			T+		−37	ns
$t_{SU\text{-}CMD\text{-}RRW}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRW}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling	T+	26		ns
$t_{PD\text{-}RRW\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}X\text{-}LCL}$	6	XACK Falling to $\overline{LCL}$ Rising	$(n_{LW}+1)T+$	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	2T+	−10	8	ns
$t_{PD\text{-}RRW\text{-}LCL}$	8	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising to $\overline{LCL}$ Falling	T+	5		ns
$t_{AZ\text{-}AAD\text{-}LCL}$	9	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{ZA\text{-}LCL\text{-}AAD}$	10	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{SU\text{-}RDAT\text{-}RRW}$	11	AD (Data) Valid before $\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising		12		ns
$t_{H\text{-}RDAT\text{-}RRW}$	12	AD (Data) Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising		10		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: The maximum value for this parameter is the latest $\overline{RAE}$, $\overline{REM\text{-}WR}$ can be removed without adding a T-state to the remote access.



TL/F/9336-71

FIGURE 5-17. Slow Buffered Write of PC, RIC

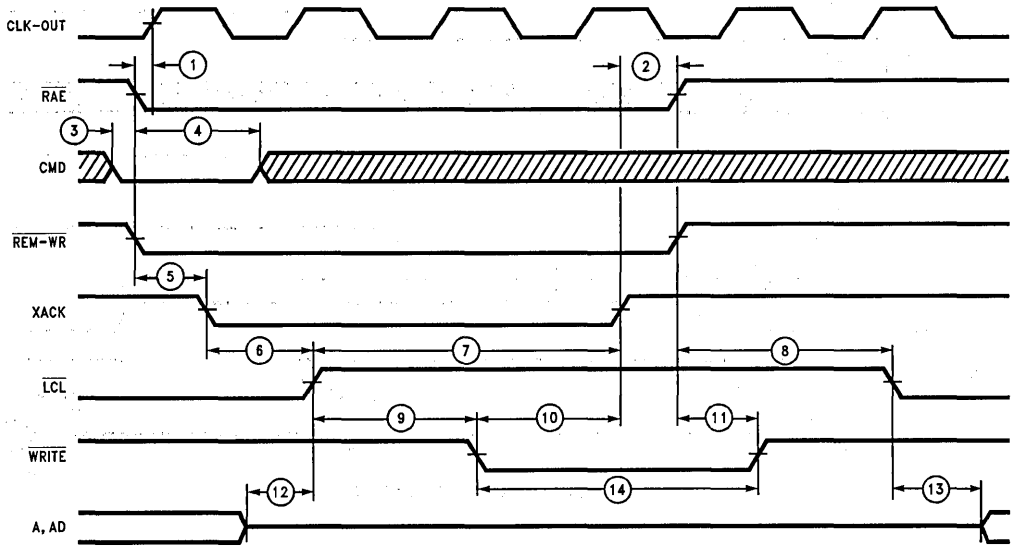
5.0 Device Specifications (Continued)

TABLE 5-18. Slow Buffered Write of DMEM (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRW\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling before CLK-OUT Rising		24		ns
$t_{H\text{-}RRW\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after XACK Rising (Note 2)		0		ns
			T+		−34	ns
$t_{SU\text{-}CMD\text{-}RRW}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRW}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling	T+	26		ns
$t_{PD\text{-}RRW\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}X\text{-}LCL}$	6	XACK Falling to $\overline{LCL}$ Rising	$(n_{LW}+1)T+$	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	$(n_{DW}+2)T+$	−10	8	ns
$t_{PD\text{-}RRW\text{-}LCL}$	8	$\overline{RAE}$, $\overline{REM\text{-}WR}$ to $\overline{LCL}$ Falling	T+	5		ns
$t_{PD\text{-}LCL\text{-}WR}$	9	$\overline{LCL}$ Rising to $\overline{WRITE}$ Falling	T+	−5		ns
$t_{PD\text{-}WR\text{-}X}$	10	$\overline{WRITE}$ Falling to XACK Rising	$(n_{DW}+1)T+$	−17		ns
$t_{PD\text{-}RRW\text{-}WR}$	11	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising to $\overline{WRITE}$ Rising		2	28	ns
$t_{AZ\text{-}AAD\text{-}LCL}$	12	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{AZ\text{-}LCL\text{-}AAD}$	13	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{W\text{-}WR}$	14	$\overline{WRITE}$ Low	$(n_{DW}+1)T+$	−3		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: The maximum value for this parameter is the latest $\overline{RAE}$, $\overline{REM\text{-}WR}$ can be removed without adding a T-state to the remote access.



TL/F/9336-72

FIGURE 5-18. Slow Buffered Write of DMEM

5.0 Device Specifications (Continued)

TABLE 5-19. Slow Buffered Write of IMEM (Notes 1, 2)

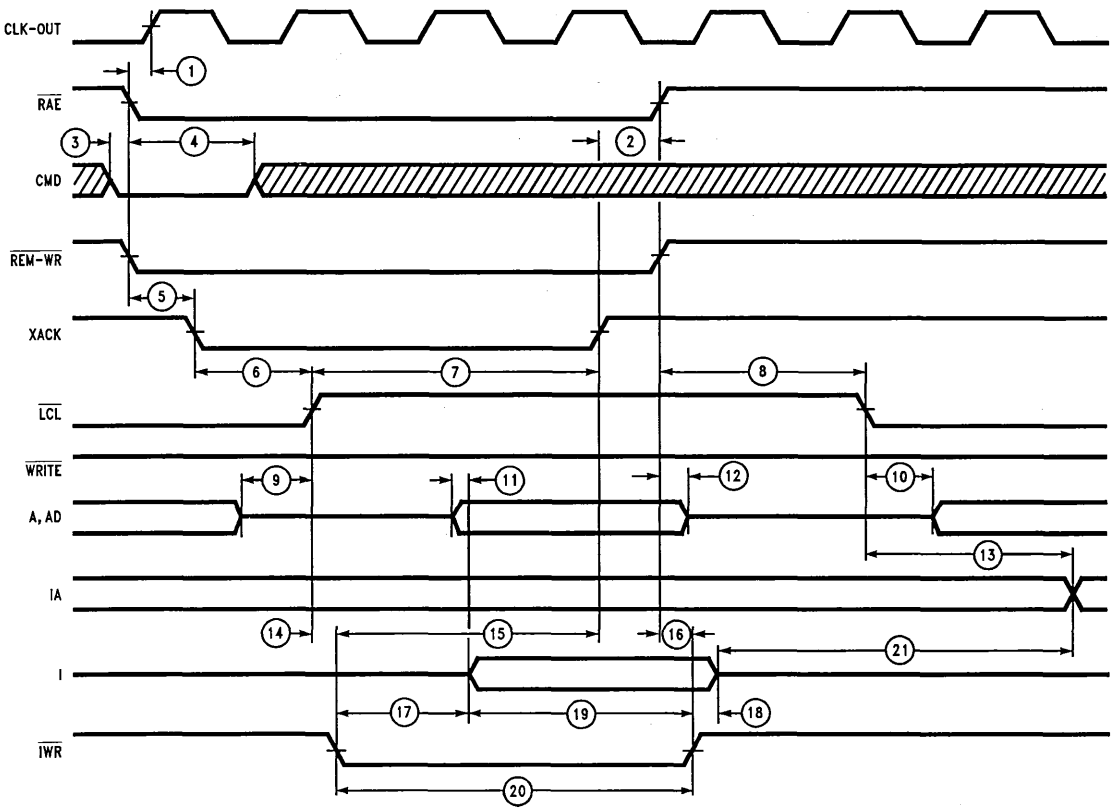
Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRW\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling before CLK-OUT Rising		24		ns
$t_{H\text{-}RRW\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after XACK Rising (Note 3)		0		ns
			T+		−34	ns
$t_{SU\text{-}CMD\text{-}RRW}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRW}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling	T+	26		ns
$t_{PD\text{-}RRW\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}X\text{-}LCL}$	6	XACK Falling to $\overline{LCL}$ Rising	T+	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	$(n_{IW}+2)T+$	−10	8	ns
$t_{PD\text{-}RRW\text{-}LCL}$	8	$\overline{RAE}$, $\overline{REM\text{-}WR}$ to $\overline{LCL}$ Falling	T+	5		ns
$t_{AZ\text{-}AAD\text{-}LCL}$	9	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{ZA\text{-}LCL\text{-}AAD}$	10	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{PD\text{-}RDAT\text{-}I}$	11	AD (Data) Valid to I Valid			30	ns
$t_{H\text{-}RDAT\text{-}RRW}$	12	AD (Data) Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising		14		ns
$t_{PD\text{-}LCL\text{-}IA}$	13	$\overline{LCL}$ Falling to next IA Valid	$T+T_H+$	−20	3	ns
$t_{PD\text{-}LCL\text{-}IWR}$	14	$\overline{LCL}$ Rising to $\overline{IWR}$ Falling		−3		ns
$t_{PD\text{-}IWR\text{-}X}$	15	$\overline{IWR}$ Falling before XACK Rising	$(n_{IW}+2)T+$	−19		ns
$t_{PD\text{-}RRW\text{-}IWR}$	16	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising to $\overline{IWR}$ Rising		5		ns
$t_{ZA\text{-}IWR\text{-}I}$	17	$\overline{IWR}$ Falling to I Enabled	T+	−2		ns
$t_{AZ\text{-}IWR\text{-}I}$	18	$\overline{IWR}$ Rising to I Disabled		22	52	ns
$t_{PD\text{-}I\text{-}IWR}$	19	I Valid before $\overline{IWR}$ Rising	$(n_{IW}+1)T+$	−10		ns
$t_{W\text{-}IWR}$	20	$\overline{IWR}$ Low	$(n_{IW}+2)T+$	−2		ns
$t_{PD\text{-}I\text{-}IA}$	21	I Disabled to IA Invalid	$2T+T_H+$	−64		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: Two remote writes to instruction memory are necessary to store a 16-bit instruction word to IMEM—low byte followed by high byte. The timing for the 2nd write is shown in the following diagram. The timing of the first write is the same as a write of the PC or RIC.

Note 3: The maximum value for this parameter is the latest $\overline{RAE}$, $\overline{REM\text{-}WR}$ can be removed without adding a T-state to the remote access.

5.0 Device Specifications (Continued)



TL/F/9336–73

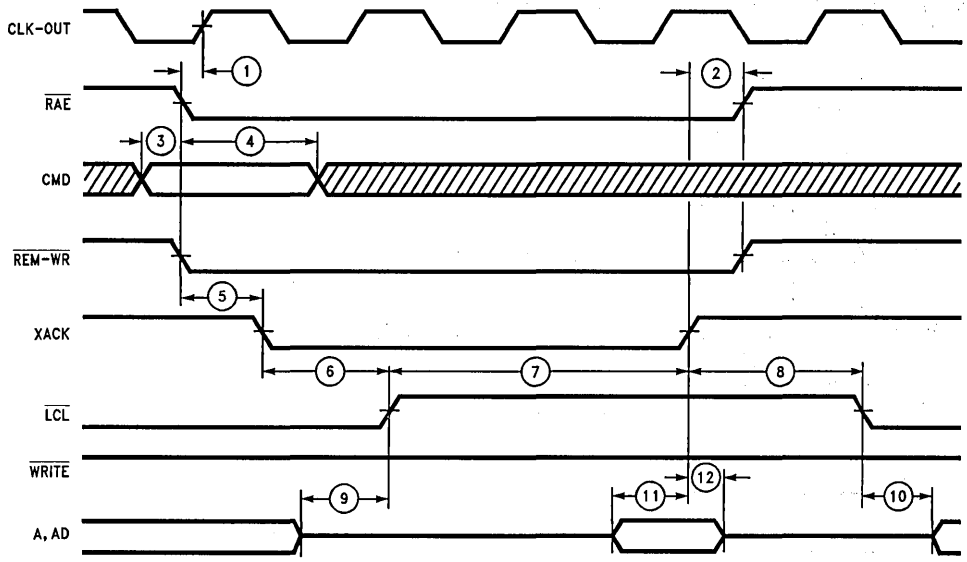
FIGURE 5-19. Slow Buffered Write of IMEM

5.0 Device Specifications (Continued)

TABLE 5-20. Fast Buffered Write of RIC, PC (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRW\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling before CLK-OUT Rising		24		ns
$t_{H\text{-}RRW\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after XACK Rising		0		ns
$t_{SU\text{-}CMD\text{-}RRW}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRW}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling	T+	26		ns
$t_{PD\text{-}RRW\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}X\text{-}LCL}$	6	XACK Falling to $\overline{LCL}$ Rising	$(n_{LW}+1)T+$	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	2T+	−10	8	ns
$t_{PD\text{-}Xr\text{-}LCLf}$	8	XACK Rising to $\overline{LCL}$ Falling	T+	−11	11	ns
$t_{AZ\text{-}AAD\text{-}LCL}$	9	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{ZA\text{-}LCL\text{-}AAD}$	10	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{SU\text{-}RDAT\text{-}X}$	11	AD (Data) Valid before XACK Rising		26		ns
$t_{H\text{-}RDAT\text{-}X}$	12	AD (Data) Invalid after XACK Rising		3		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.



TL/F/9336-74

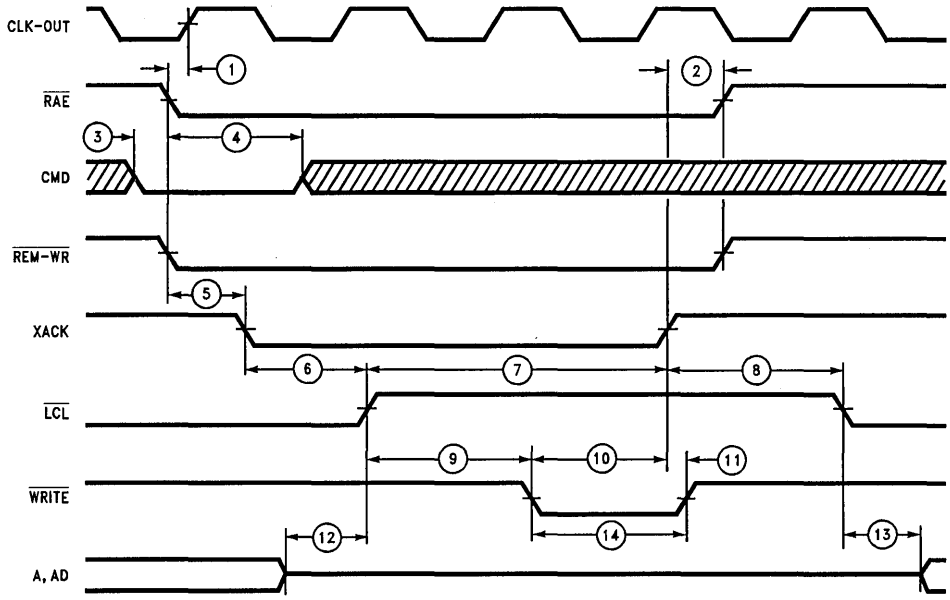
FIGURE 5-20. Fast Buffered Write of RIC, PC

5.0 Device Specifications (Continued)

TABLE 5-21. Fast Buffered Write of DMEM (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRW\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling before CLK-OUT Rising		24		ns
$t_{H\text{-}RRW\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after XACK Rising		0		ns
$t_{SU\text{-}CMD\text{-}RRW}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRW}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling	T+	26		ns
$t_{PD\text{-}RRW\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}Xf\text{-}LCLr}$	6	XACK Falling to $\overline{LCL}$ Rising	$(n_{LW}+1)T+$	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	$(n_{DW}+2)T+$	−10	8	ns
$t_{PD\text{-}Xr\text{-}LCLf}$	8	XACK Rising to $\overline{LCL}$ Falling	T+	−11	11	ns
$t_{PD\text{-}LCL\text{-}WR}$	9	$\overline{LCL}$ Rising to $\overline{WRITE}$ Falling	T+	−5		ns
$t_{PD\text{-}WR\text{-}X}$	10	$\overline{WRITE}$ Falling to XACK Rising	$(n_{DW}+1)T+$	−16		ns
$t_{PD\text{-}X\text{-}WR}$	11	XACK Rising to $\overline{WRITE}$ Rising		−4	13	ns
$t_{AZ\text{-}AAD\text{-}LCL}$	12	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{ZA\text{-}LCL\text{-}AAD}$	13	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{W\text{-}WR}$	14	$\overline{WRITE}$ Low	$(n_{DW}+1)T+$	−10		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.



TL/F/9336-75

FIGURE 5-21. Fast Buffered Write of DMEM

5.0 Device Specifications (Continued)

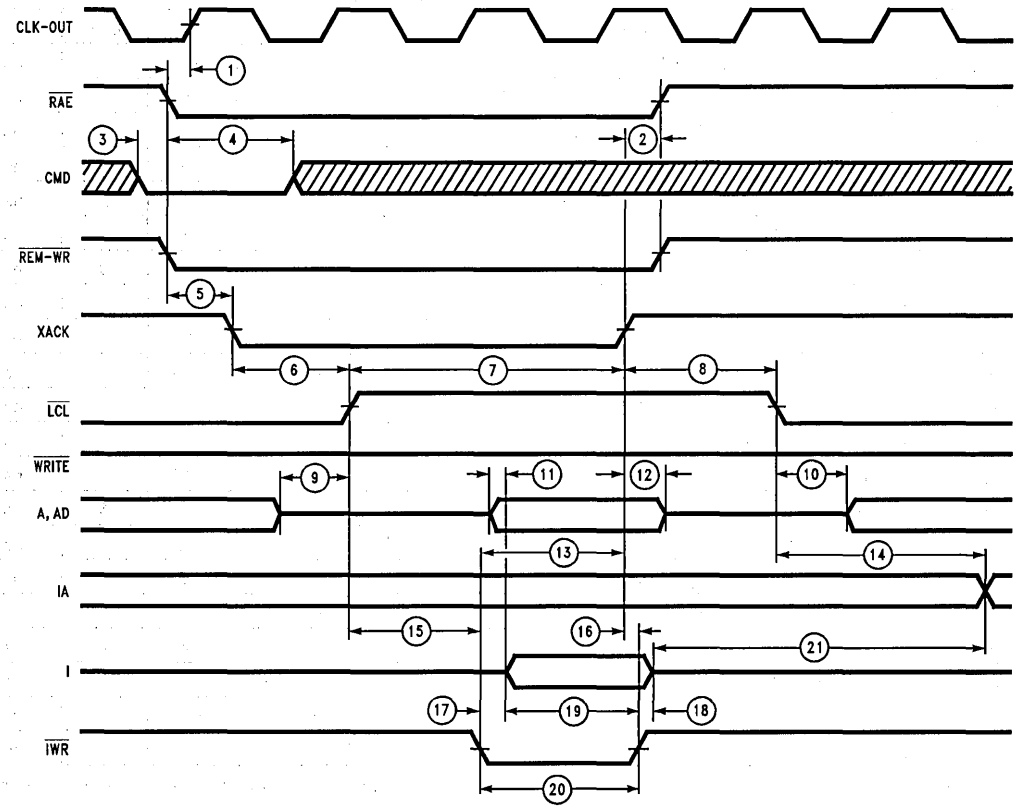
TABLE 5-22. Fast Buffered Write of IMEM (Notes 1, 2)

Symbol	ID #	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRW\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling before CLK-OUT Rising		24		ns
$t_{H\text{-}RRW\text{-}X}$	2	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after XACK Rising		0		ns
$t_{SU\text{-}CMD\text{-}RRW}$	3	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRW}$	4	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling	T+	26		ns
$t_{PD\text{-}RRW\text{-}X}$	5	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}Xf\text{-}LCLr}$	6	XACK Falling to $\overline{LCL}$ Rising	T+	−5		ns
$t_{PD\text{-}LCL\text{-}X}$	7	$\overline{LCL}$ Rising to XACK Rising	$(n_{IW}+2)T+$	−10	8	ns
$t_{PD\text{-}Xr\text{-}LCLf}$	8	XACK Rising to $\overline{LCL}$ Falling	T+	−11	11	ns
$t_{AZ\text{-}AAD\text{-}LCL}$	9	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{ZA\text{-}LCL\text{-}AAD}$	10	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{PD\text{-}RDAT\text{-}I}$	11	AD (Data) Valid to I Valid			30	ns
$t_{H\text{-}RDAT\text{-}X}$	12	AD (Data) Invalid after XACK Rising		3		ns
$t_{PD\text{-}IWR\text{-}X}$	13	$\overline{IWR}$ Falling before XACK Rising	$(n_{IW}+2)T+$	−19		ns
$t_{PD\text{-}LCL\text{-}IA}$	14	$\overline{LCL}$ Falling to next IA Valid	$T+T_H+$	−19	5	ns
$t_{PD\text{-}LCL\text{-}IWR}$	15	$\overline{LCL}$ Rising to $\overline{IWR}$ Falling		−3		ns
$t_{PD\text{-}X\text{-}IWR}$	16	XACK Rising to $\overline{IWR}$ Rising		−2		ns
$t_{ZA\text{-}IWR\text{-}I}$	17	$\overline{IWR}$ Falling to I Enabled	T+	−2		ns
$t_{AZ\text{-}IWR\text{-}I}$	18	$\overline{IWR}$ Rising to I Disabled		22	52	ns
$t_{PD\text{-}I\text{-}IWR}$	19	I Valid before $\overline{IWR}$ Rising	$(n_{IW}+1)T+$	−18		ns
$t_{W\text{-}IWR}$	20	$\overline{IWR}$ Low Time	$(n_{IW}+2)T+$	−10		ns
$t_{PD\text{-}I\text{-}IA}$	21	I Disabled to IA Invalid	$2T+T_H+$	−70		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: Two remote writes to instruction memory are necessary to store a 16-bit instruction word to IMEM—low byte followed by high byte. The timing of the 2nd write is shown in the following diagram. The timing of the first write is the same as a write of the PC or RIC as shown in *Figure 5-20*.

5.0 Device Specifications (Continued)



TL/F/9336-76

FIGURE 5-22. Fast Buffered Write of IMEM

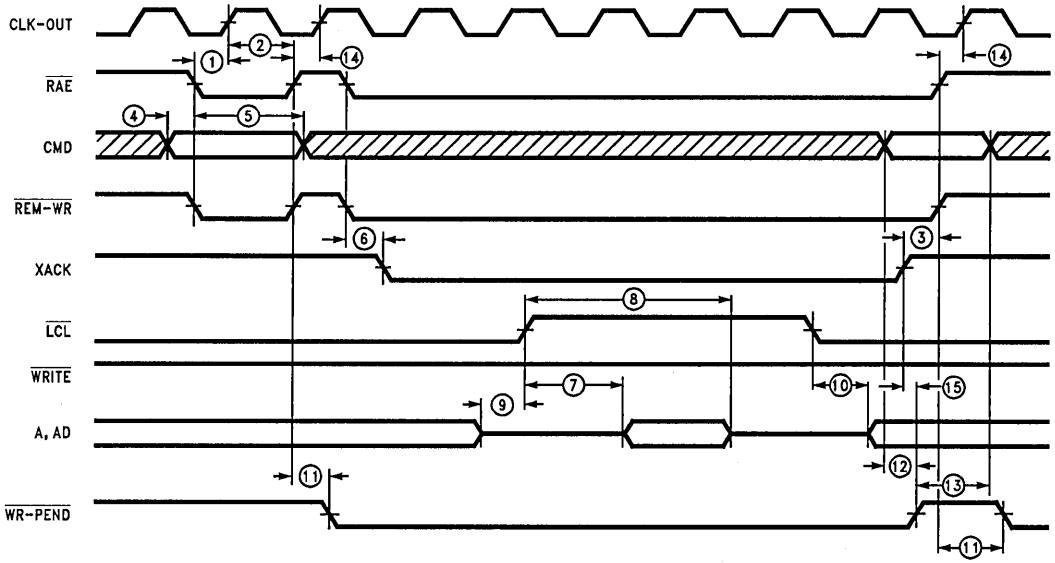
5.0 Device Specifications (Continued)

TABLE 5-23. Latched Write of PC, RIC (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRW\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling before CLK-OUT Rising		24		ns
$t_{H\text{-}RRW\text{-}CO}$	2	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after CLK-OUT Rising (Note 2)	T_H+	6		ns
			T+		−20	ns
$t_{H\text{-}RRW\text{-}X}$	3	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after XACK Rising		0		ns
$t_{SU\text{-}CMD\text{-}RRW}$	4	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRW}$	5	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling	T+	26		ns
$t_{PD\text{-}RRW\text{-}X}$	6	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling to XACK Falling			26	ns
$t_{SU\text{-}RDAT\text{-}LCL}$	7	AD (Data) Valid after $\overline{LCL}$ Rising	2T+		−30	ns
$t_{H\text{-}RDAT\text{-}LCL}$	8	AD (Data) Invalid after $\overline{LCL}$ Rising	2T+	2		ns
$t_{AZ\text{-}AAD\text{-}LCL}$	9	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{ZA\text{-}LCL\text{-}AAD}$	10	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{PD\text{-}RRW\text{-}WPND}$	11	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising to $\overline{WR\text{-}PEND}$ Falling		5		ns
			T+		34	ns
$t_{SU\text{-}CMD\text{-}WPND}$	12	CMD Valid before $\overline{WR\text{-}PEND}$ Rising		16		ns
$t_{H\text{-}CMD\text{-}WPND}$	13	CMD Invalid after $\overline{WR\text{-}PEND}$ Rising		4		ns
$t_{SU\text{-}RRWr\text{-}CO}$	14	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising before CLK-OUT Rising		20		ns
$t_{PD\text{-}X\text{-}WPND}$	15	XACK Rising to $\overline{WR\text{-}PEND}$ Rising			13	ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: The maximum value for this parameter is the latest $\overline{RAE}$, $\overline{REM\text{-}WR}$ can be removed without delaying the remote access by one T-state.



TL/F/9336-77

FIGURE 5-23. Latched Write of PC, RIC

5.0 Device Specifications (Continued)

TABLE 5-24. Latched Write of DMEM (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRW\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling before CLK-OUT Rising		24		ns
$t_{H\text{-}RRW\text{-}CO}$	2	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after CLK-OUT Rising (Note 2)	T_H+	6		ns
			$T+$		−20	ns
$t_{H\text{-}RRW\text{-}X}$	3	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after XACK Rising		0		ns
$t_{SU\text{-}CMD\text{-}RRW}$	4	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRW}$	5	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling	$T+$	26		ns
$t_{PD\text{-}RRW\text{-}X}$	6	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling to XACK Falling			26	ns
$t_{PD\text{-}LCL\text{-}WR}$	7	$\overline{LCL}$ Rising to $\overline{WRITE}$ Falling	$T+$	−5		ns
$t_{PD\text{-}WR\text{-}LCL}$	8	$\overline{WRITE}$ Rising to $\overline{LCL}$ Falling	$T+$	−11		ns
$t_{AZ\text{-}AAD\text{-}LCL}$	9	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{ZA\text{-}LCL\text{-}AAD}$	10	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{W\text{-}WR}$	11	$\overline{WRITE}$ Low Time	$(n_{DW}+1)T+$	−10		ns
$t_{PD\text{-}RRW\text{-}WPND}$	12	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising to $\overline{WR\text{-}PEND}$ Falling		5		ns
			$T+$		34	ns
$t_{SU\text{-}CMD\text{-}WPND}$	13	CMD Valid before $\overline{WR\text{-}PEND}$ Rising		16		ns
$t_{H\text{-}CMD\text{-}WPND}$	14	CMD Invalid after $\overline{WR\text{-}PEND}$ Rising		4		ns
$t_{SU\text{-}RRWr\text{-}CO}$	15	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising before CLK-OUT Rising		20		ns
$t_{PD\text{-}X\text{-}WPND}$	16	XACK Rising to $\overline{WR\text{-}PEND}$ Rising			13	ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: The maximum value for this parameter is the latest $\overline{RAE}$, $\overline{REM\text{-}WR}$ can be removed without delaying the remote access by one T-state.

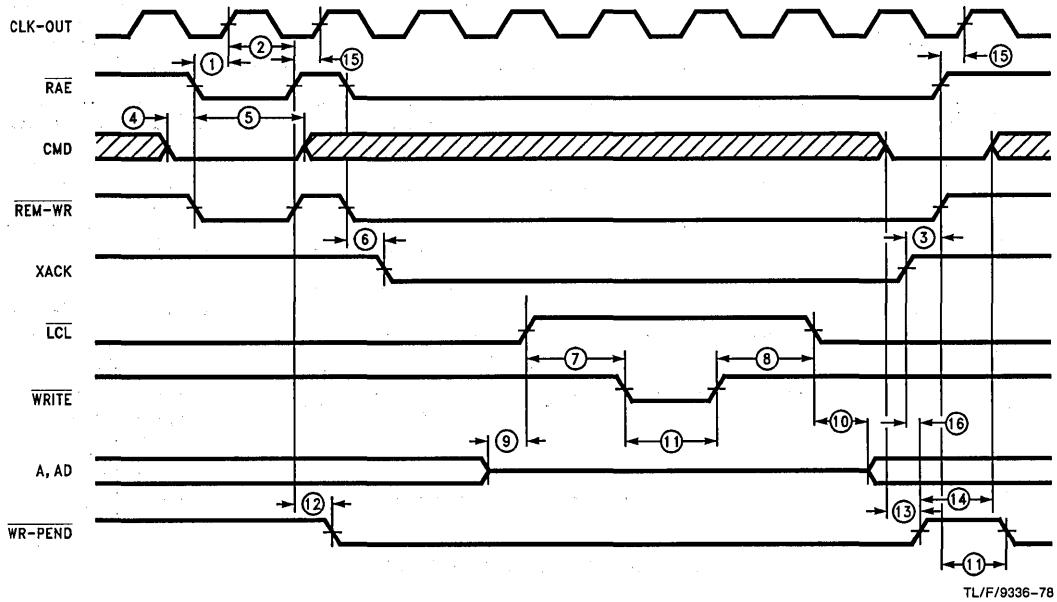


FIGURE 5-24. Latched Write of DMEM

5.0 Device Specifications (Continued)

TABLE 5-25. Latched Write of IMEM (Notes 1, 2)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}RRW\text{-}CO}$	1	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling before CLK-OUT Rising		24		ns
$t_{H\text{-}RRW\text{-}CO}$	2	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after CLK-OUT Rising (Note 3)	T_H+	6		ns
			$T+$		−20	ns
$t_{H\text{-}RRW\text{-}X}$	3	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising after XACK Rising		0		ns
$t_{SU\text{-}CMD\text{-}RRW}$	4	CMD Valid before $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling		0		ns
$t_{H\text{-}CMD\text{-}RRW}$	5	CMD Invalid after $\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling	$T+$	26		ns
$t_{PD\text{-}RRW\text{-}X}$	6	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Falling to XACK Falling			26	ns
$t_{AZ\text{-}AAD\text{-}LCL}$	7	A, AD Disabled before $\overline{LCL}$ Rising	T_L+	−20		ns
$t_{ZA\text{-}LCL\text{-}AAD}$	8	A, AD Enabled after $\overline{LCL}$ Falling	T_H+	−10		ns
$t_{PD\text{-}RDAT\text{-}I}$	9	AD (Data) Valid to I Valid			30	ns
$t_{H\text{-}RDAT\text{-}IWR}$	10	AD (Data) Invalid after $\overline{IWR}$ Rising		0		ns
$t_{PD\text{-}RRW\text{-}WPND}$	11	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising to $\overline{WR\text{-}PEND}$ Falling		5		
			$T+$		34	ns
$t_{PD\text{-}LCL\text{-}IA}$	12	$\overline{LCL}$ Falling to Next IA Valid	$T+T_H+$	−19	5	ns
$t_{ZA\text{-}IWR\text{-}I}$	13	$\overline{IWR}$ Falling to I Enabled	$T+$	−2		ns
$t_{AZ\text{-}IWR\text{-}I}$	14	$\overline{IWR}$ Rising to I Disabled		22	52	ns
$t_{PD\text{-}I\text{-}IWR}$	15	I Valid before $\overline{IWR}$ Rising	$(n_{IW}+1)T+$	−18		ns
$t_{PD\text{-}LCL\text{-}IWR}$	16	$\overline{LCL}$ Rising to $\overline{IWR}$ Falling		−3		ns
$t_{PD\text{-}IWR\text{-}LCL}$	17	$\overline{IWR}$ Rising to $\overline{LCL}$ Falling	$T+$	−17		ns
$t_{W\text{-}IWR}$	18	$\overline{IWR}$ Low Time	$(n_{IW}+2)T+$	−12		ns
$t_{SU\text{-}CMD\text{-}WPND}$	19	CMD Valid before $\overline{WR\text{-}PEND}$ Rising		16		ns
$t_{H\text{-}CMD\text{-}WPND}$	20	CMD Invalid after $\overline{WR\text{-}PEND}$ Rising		4		ns
$t_{PD\text{-}I\text{-}IA}$	21	I Disabled to IA Invalid	$2T+T_H+$	−70		ns
$t_{SU\text{-}RRWr\text{-}CO}$	22	$\overline{RAE}$, $\overline{REM\text{-}WR}$ Rising before CLK-OUT Rising		20		ns
$t_{PD\text{-}X\text{-}WPND}$	23	XACK Rising to $\overline{WR\text{-}PEND}$ Rising			13	ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: Two remote writes to instruction memory are necessary to store a 16-bit instruction word to IMEM—low byte followed by high byte. The timing of the 2nd write is shown in the following diagram. The first write is the same as a write of the PC or RIC as shown in *Figure 5-23.*

Note 3: The maximum value for this parameter is the latest $\overline{RAE}$, $\overline{REM\text{-}WR}$ can be removed without delaying the remote access by one T-state.

5.0 Device Specifications (Continued)

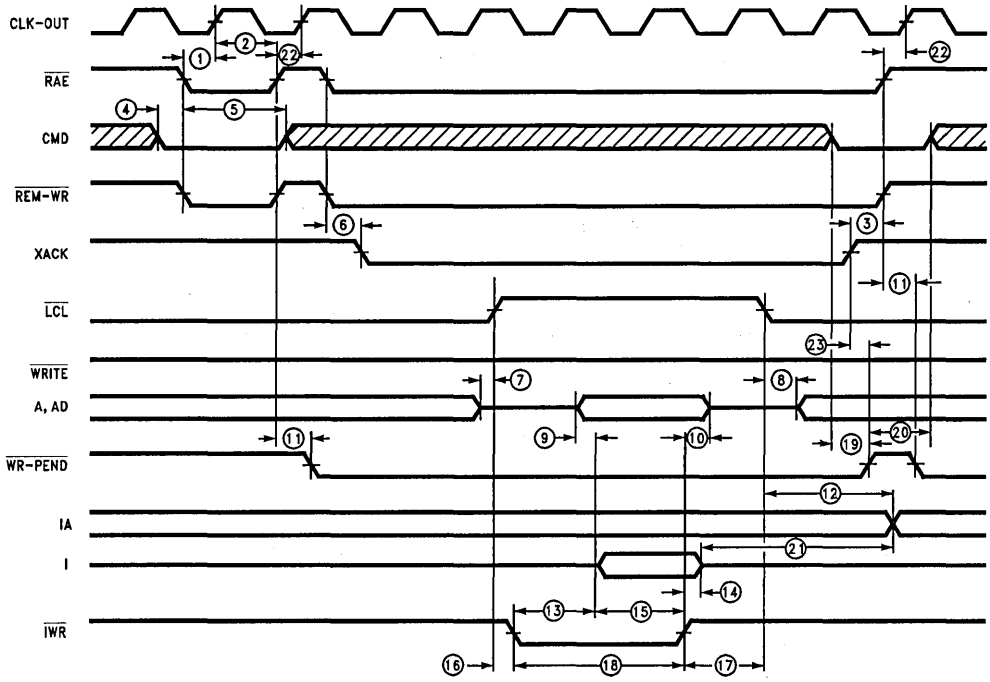


FIGURE 5-25. Latched Write of IMEM

5.0 Device Specifications (Continued)

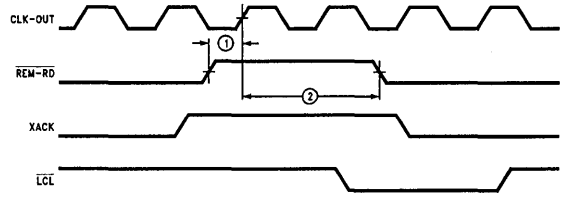
TABLE 5-26. Remote Rest Time (Note 1)

Symbol	ID #	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}BR\text{-}RR\text{-}CO}$	1	$\overline{\text{REM-RD}}$ Rising before CLK-OUT Rising (Buffered Read Mode)		19		ns
$t_{H\text{-}BR}$	2	CLK-OUT Rising after $\overline{\text{REM-RD}}$ Rising to $\overline{\text{REM-RD}}$ or $\overline{\text{REM-WR}}$ Falling (Buffered Read Mode)	$T+T_H+$	10		ns
$t_{SU\text{-}LR\text{-}RR\text{-}CO}$	3	$\overline{\text{REM-RD}}$ Rising before CLK-OUT Rising (Latched Read Mode)		16		ns
$t_{H\text{-}LR}$	4	CLK-OUT Rising after $\overline{\text{REM-RD}}$ Rising to $\overline{\text{REM-RD}}$ or $\overline{\text{REM-WR}}$ Falling (Latched Read Mode)	$T+T_H+$	10		ns
$t_{SU\text{-}SBW\text{-}RW\text{-}CO}$	5	$\overline{\text{REM-WR}}$ Rising before CLK-OUT Rising (Slow Buffered Write Mode)		22		ns
$t_{H\text{-}SBW}$	6	CLK-OUT Rising after $\overline{\text{REM-WR}}$ Rising to $\overline{\text{REM-RD}}$ or $\overline{\text{REM-WR}}$ Falling (Slow Buffered Write Mode)	$T+T_H+$	10		ns
$t_{SU\text{-}FBW\text{-}RW\text{-}CO}$	7	$\overline{\text{REM-WR}}$ Rising before CLK-OUT Rising (Fast Buffered Write Mode)		22		ns
$t_{H\text{-}FBW}$	8	CLK-OUT Rising after $\overline{\text{REM-WR}}$ Rising to $\overline{\text{REM-RD}}$ or $\overline{\text{REM-WR}}$ Falling (Fast Buffered Write Mode)	$T+T_H+$	10		ns
$t_{SU\text{-}LW\text{-}RW\text{-}CO}$	9	$\overline{\text{REM-WR}}$ Rising before CLK-OUT Rising (Latched Write Mode)		20		ns
$t_{H\text{-}LW}$	10	CLK-OUT Rising after $\overline{\text{REM-WR}}$ Rising to $\overline{\text{REM-RD}}$ or $\overline{\text{REM-WR}}$ Falling (Latched Write Mode)		10		ns
$t_{SU\text{-}LW\text{-}RWR\text{-}COa}$	11	$\overline{\text{REM-WR}}$ or $\overline{\text{REM-RD}}$ Falling to CLK-OUT Falling (Latched Write Mode) (Note 2)	T_H+	7		ns
$t_{SU\text{-}LW\text{-}RWR\text{-}COb}$	12	CLK-OUT Rising to $\overline{\text{REM-WR}}$ or $\overline{\text{REM-RD}}$ rising (Latched Write Mode) (Note 2)		8		ns
$t_{PD\text{-}CO\text{-}WP}$	13	CLK-OUT rising to $\overline{\text{WR-PEND}}$ Rising		−1	21	ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

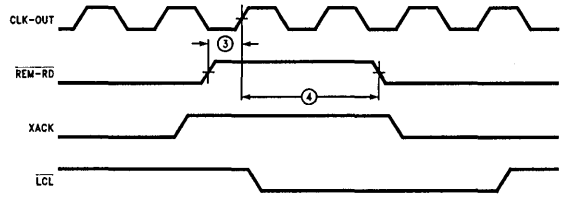
Note 2: Both specifications refer to the CLK-OUT falling edge after $\overline{\text{WR-PEND}}$ rising. See Section 4.2.6, RIAS remote rest time.

5.0 Device Specifications (Continued)



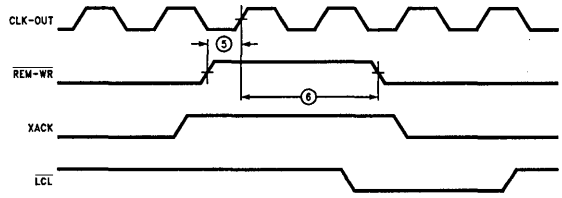
TL/F/9336-B0

(a) $\overline{\text{REM-RD}}$ Rest Time (Buffered Read Mode)



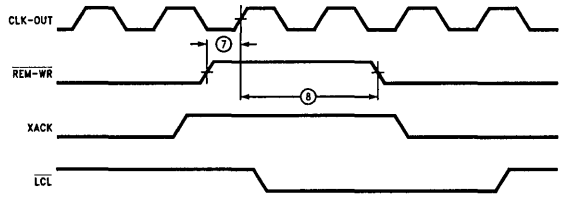
TL/F/9336-B1

(b) $\overline{\text{REM-RD}}$ Rest Time (Latched Read Mode)



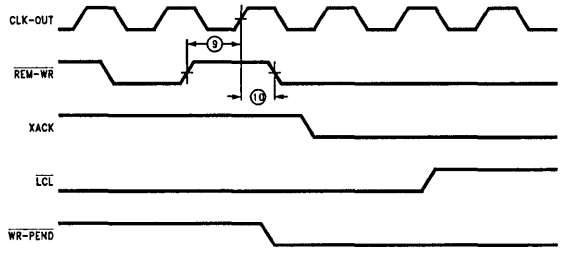
TL/F/9336-B2

(c) $\overline{\text{REM-WR}}$ Rest Time (Slow Buffered Write Mode)



TL/F/9336-B3

(d) $\overline{\text{REM-WR}}$ Rest Time (Fast Buffered Write Mode)

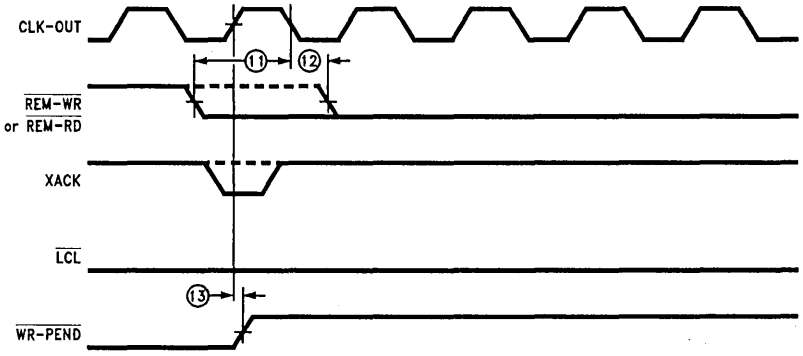


TL/F/9336-B4

(e) $\overline{\text{REM-WR}}$ Rest Time (Latched Write Mode)

FIGURE 5-26. Remote Rest Time

5.0 Device Specifications (Continued)



TL/F/9336-H9

(f) WR-PEND Rising (Latched Write Mode)

FIGURE 5-26. Remote Rest Time (Continued)

TABLE 5-27. Remote Interface WAIT Timing (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{SU\text{-}WT\text{-}LCL}$	1	WAIT Falling after LCL Rising to Extend Cycle (Buffered Read, Latched Read, Slow Buffered Write, Fast Buffered Write and Latched Write of PC, RIC)	$T+T_H+$		−28	ns
		WAIT Falling after LCL Rising to Extend Cycle (Buffered Read, Latched Read, Slow Buffered Write, Fast Buffered Write and Latched Write of DMEM)	$(n_{DW}+1)T+T_H+$		−28	ns
		WAIT Falling after LCL Rising to Extend Cycle (Buffered Read, Latched Read, Slow Buffered Write, Fast Buffered Write and Latched Write of IMEM)	$(n_{IW}+1)T+T_H+$		−28	ns
$t_{H\text{-}WT\text{-}LCL}$	2	WAIT Rising after LCL Rising (Buffered Read, Latched Read, Slow Buffered Write, Fast Buffered Write and Latched Write of PC, RIC) (Note 2)	$T+T_H+$	0		ns
			$2T+T_H+$		−27	ns
		WAIT Rising after LCL Rising (Buffered Read, Latched Read, Slow Buffered Write, Fast Buffered Write and Latched Write of DMEM) (Note 2)	$(n_{DW}+1)T+T_H+$	0		ns
			$(n_{DW}+2)T+T_H+$		−27	ns
		WAIT Rising after LCL Rising (Buffered Read, Latched Read, Slow Buffered Write, Fast Buffered Write and Latched Write of IMEM) (Note 2)	$(n_{IW}+1)T+T_H+$	0		ns
			$(n_{IW}+2)T+T_H+$		−27	ns
$t_{SU\text{-}WT\text{-}RD}$	3	WAIT Falling after READ Falling to Extend Cycle (Buffered Read and Latched Read)	$(n_{DW})T+T_H+$		−32	ns
$t_{SU\text{-}WT\text{-}WR}$	3	WAIT Falling after WRITE Falling to Extend Cycle (Slow Buffered Write, Fast Buffered Write and Latched Write)	$(n_{DW})T+T_H+$		−33	ns
$t_{SU\text{-}WT\text{-}IWR}$	3	WAIT Falling after IWR Falling to Extend Cycle (Slow Buffered Write, Fast Buffered Write and Latched Write)	$(n_{IW}+1)T+T_H+$		−38	ns

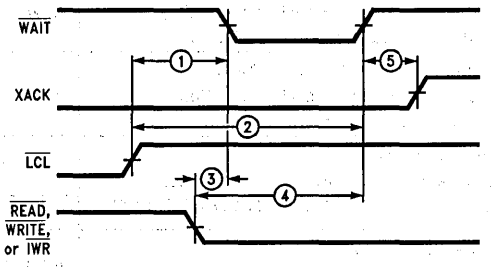
5.0 Device Specifications (Continued)

TABLE 5-27. Remote Interface $\overline{WAIT}$ Timing (Note 1) (Continued)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{H\text{-}WT\text{-}RD}$	4	$\overline{WAIT}$ Rising after $\overline{READ}$ Falling (Buffered Read and Latched Read) (Note 2)	$(n_{DW}T+T_H+$	−4		ns
			$(n_{DW}+1)T+T_H+$		−30	ns
$t_{H\text{-}WT\text{-}WR}$	4	$\overline{WAIT}$ Rising after $\overline{WRITE}$ Falling (Slow Buffered Write, Fast Buffered Write and Latched Write) (Note 2)	$(n_{DW})T+T_H+$	−5		ns
			$(n_{DW}+1)T+T_H+$		−34	ns
$t_{H\text{-}WT\text{-}IWR}$	4	$\overline{WAIT}$ Rising after $\overline{IWR}$ Falling (Slow Buffered Write, Fast Buffered Write and Latched Write) (Note 2)	$(n_{IW}+1)T+T_H+$	−5		ns
			$(n_{IW}+2)T+T_H+$		−38	ns
$t_{PD\text{-}WT\text{-}X}$	5	$\overline{WAIT}$ Rising to XACK Rising (Buffered Read, Latched Read, Slow Buffered Write and Fast Buffered Write)	T_L+	0		ns
			$T+T_L+$		24	ns
$t_{PD\text{-}WT\text{-}LCL}$	6	$\overline{WAIT}$ Rising to $\overline{LCL}$ Falling (Latched Write)	$T+T_L+$	1		ns
			$2T+T_L+$		26	ns
$t_{PD\text{-}WT\text{-}WR}$	7	$\overline{WAIT}$ Rising to $\overline{WRITE}$ Rising (Latched Write)	T_L+	2		ns
			$T+T_L+$		28	ns
$t_{PD\text{-}WT\text{-}IWR}$	7	$\overline{WAIT}$ Rising to $\overline{IWR}$ Rising (Latched Write)	T_L+	4		ns
			$T+T_L+$		38	ns

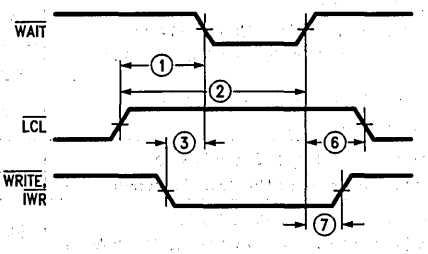
Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.

Note 2: The maximum value for this parameter is the latest $\overline{WAIT}$ can be removed without adding an additional T-state. The formula assumes a minimum external wait of one T-state.



TL/F/9336-B5

(a) Buffered Read, Latched Read, Slow Buffered Write and Fast Buffered Write



TL/F/9336-B6

(b) Latched Write

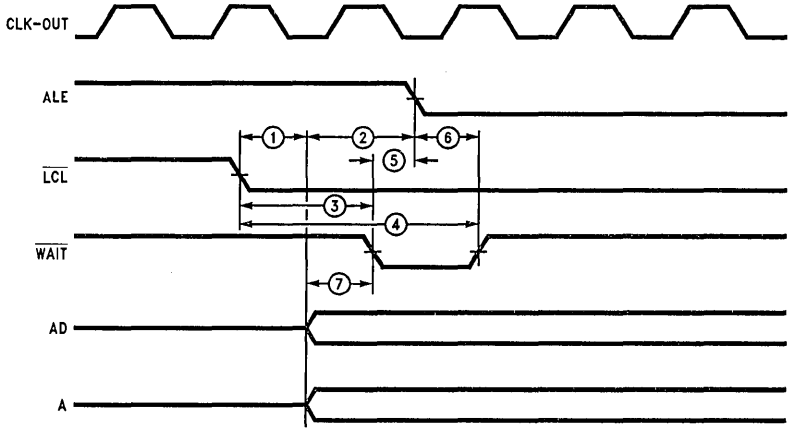
FIGURE 5-27. Remote Interface $\overline{WAIT}$ Timing

5.0 Device Specifications (Continued)

TABLE 5-28. Wait Timing After Remote Access (Note 1)

Symbol	ID#	Parameter	Formula	Min	Max	Units
$t_{PD\text{-}LCL\text{-}AAD}$	1	$\overline{LCL}$ Falling to A, AD (Data Address) Valid	T_H+		11	ns
$t_{PD\text{-}LCL\text{-}AAD\text{-}BR}$		$\overline{LCL}$ Falling to A, AD (Data Address) Valid for Buffered Read of RIC	$2T+$		29	ns
$t_{PD\text{-}AAD\text{-}ALE}$	2	A, AD (Data Address) Valid to ALE Falling	$T+$	−16		ns
$t_{SU\text{-}WT\text{-}LCL}$	3	$\overline{LCL}$ Falling to $\overline{WAIT}$ Falling to Extend Local Cycle	$(\max(n_{DW},n_{IW}-1)+1)T+T_H+$		−29	ns
$t_{H\text{-}WT\text{-}LCL}$	4	$\overline{WAIT}$ Rising after $\overline{LCL}$ Falling	$(\max(n_{DW},n_{IW}-1)+1)T+T_H+$	−3		ns
			$(\max(n_{DW},n_{IW}-1)+2)T+T_H+$		−28	ns
$t_{H\text{-}WT\text{-}LCL\text{-}BR}$		$\overline{WAIT}$ Rising after $\overline{LCL}$ Falling for Buffered read of RIC	$(\max(n_{DW},n_{IW}-1)+3)T+T_H+$	−3		ns
$t_{SU\text{-}WT\text{-}ALEf}$	5	$\overline{WAIT}$ Low Before ALE Falling to Extend Cycle		22		ns
$t_{H\text{-}WT\text{-}ALE}$	6	$\overline{WAIT}$ Rising After ALE Falling		0		ns
			$(\max(n_{DW},n_{IW}-1)+1)T+$		−28	ns
$t_{SU\text{-}WT\text{-}AAD}$	7	A, AD (Data Address) Valid to $\overline{WAIT}$ Falling to Extend Load Cycle	$T+$	−33		ns

Note 1: All parameters are individually tested and guaranteed. Interpreting this data by numerically adding two or more parameters to create a new timing specification may lead to invalid results.



TL/F/9336-10

FIGURE 5-28. Wait Timing After Remote Access

6.0 Reference Section

6.1 INSTRUCTION SET REFERENCE

The Instruction Set Reference section contains detailed information on the syntax and operation of each BCP instruction. The instructions are arranged in alphabetical order by mnemonic for easy access. Although this section is primarily intended as a reference for the assembly language programmer, previous assembly language experience is not a prerequisite. The intent of this instruction set reference is to include all the pertinent information regarding each instruction on the page(s) describing that instruction. The only exceptions to this rule concern the instruction addressing modes and the bus timing diagrams. The discussion of the instruction addressing modes occurs at the beginning of the BCP Instruction Set Overview section and, therefore, will not be repeated here. The figures for the bus timing diagrams are located at the end of this introduction rather than constantly repeating them under each instruction. The information that is contained under each instruction is divided into eight categories titled: Syntax, Affected Flags, Description, Example, Instruction Format, T-states, Bus timing, and Operation. The following paragraphs explain what information each category conveys and any special nomenclature that a category may use.

Syntax

This category illustrates the assembler syntax for each instruction. Multiple lines are used when a given instruction supports more than one type of addressing mode, or if it has an optional mnemonic. All capital letters, commas (,), math symbols (+, −), and brackets ([ ]) are entered into the assembler exactly as shown. Braces ({ }) surround an instruction's optional operands and their associated syntax. The text between the braces may either be entered in with or omitted from the instruction. The braces themselves should not be entered into the assembler because they are not part of the assembler syntax. Lower case characters and operands that begin with the capital R represent symbols. These must be replaced with actual register names, numbers, or equated registers and numbers. Table 6-1 lists all the symbols and their associated meanings.

Affected Flags

If an instruction sets or clears any of the ALU flags, (i.e., Negative [N], Zero [Z], Carry [C], and/or Overflow [V]), then those flags affected are listed under this category.

Description

The Description category contains a verbal discussion about the operation of an instruction, the operands it allows, and any notes highlighting special considerations the programmer should keep in mind when using the instruction.

Example

Each instruction has one or more coding examples designed to show its typical usage(s). For clarity, register name abbreviations are often used instead of the register numbers, (i.e., RTR is used in place of R4). Each example assumes that the ".EQU" assembler directive has been previously executed to establish these relationships. Information relating register abbreviations to register names, numbers, and purpose is located in the CPU Registers section.

Instruction Format

This category illustrates the formation of an instruction's machine code for each operand variation. Assembly or disassembly of any instruction can be accomplished using these figures.

T-states

The T-state category lists the number of CPU clock cycles required for each instruction, including operand variations and conditional considerations. Using this information, actual execution times may be calculated. For example, if the conditional relative jump instruction's condition is not met, the CPU's clock cycle is 18.867 MHz ([CCS] = 0), and no instruction wait states are requested ([IW1 − 0] = 00), then Jcc's execution time is calculated as shown below:

$$\begin{aligned} t_{execution} &= 1/(\text{CPU clock frequency}) \times \text{T-states} \\ &= 1/(18.867 \times 10^6\ \text{Hz}) \times 2 \\ &= (53 \times 10^{-9}\text{s}) \times 2 \\ &= 106\ \text{ns} \end{aligned}$$

See the section BCP Timing for more information on calculating instruction execution times.

Bus Timing

This category refers the user to the Bus Timing *Figures 6-1* to *6-6* on the following pages. These figures illustrate the relationship between software instruction execution and some of the BCP's hardware signals.

Operation

The operation category illustrates each instruction's operation in a symbolic coding format. Most of the operand names used in this format come directly from each instruction's syntax. The exceptions to this rule deal with implied operands. Instructions that imply the use of the accumulators use the name "accumulator" as an operand. Instructions that manipulate the Program Counter use the symbol "PC". Instructions that "push" onto or "pop" off of the internal Address Stack specify "Address Stack" as an operand. Instructions that save or restore the ALU flags and the register bank selections use those terms as operands. Two specialized operator symbols are used in the symbolic coding format, the arrow " → " and the concatenation operator "&". The arrow indicates the movement of data from one operand to another. For instance, after the operation "Rs → Rd" is performed the content of Rd has been replaced with the content of Rs. The concatenation operator "&" simply indicates that the operands surrounding an "&" are attached together forming one new operand. For example, "PC & [GIE] & ALU flags & register bank selections → Address Stack" means that the Program Counter, the Global Interrupt Enable bit, the ALU flags and the register bank selections are combined into one operand and pushed onto the internal Address Stack. Three conditional structures are utilized in the symbolic coding format: the "Two Line If" structure, the "Blocked If" structure, and the "Blocked Case" structure. In the "Two Line If" structure, if the *condition* is met then the *operation* is performed, otherwise the *operation* is not performed.

"Two Line If" structure:

If *condition*
then *operation*

6.0 Reference Section (Continued)

In the "Blocked If" structure, if the *condition* is met then all the *operations* between the "If" statement and the "End if" statement are performed.

"Blocked If" structure:

If *condition* then
operation
operation
etc . . .
End if

In the "Blocked Case" structure, the *operation* preceded by the equivalent numeric value of the *operand* is executed. For example, if the *operand's* value is equal to "1" then the *operation* preceded by "1:" is executed.

"Blocked Case" structure:

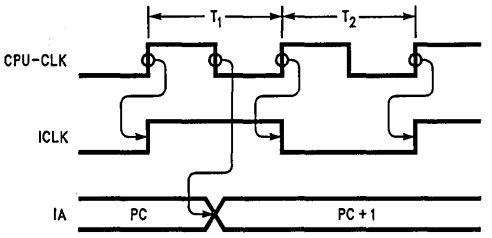
Case *operand* of
0: *operation*
1: *operation*
2: *etc* . . .
End case

Two reference tables have been added to the back of the Instruction Set Reference section. The first table, Table 6-2, lists all the instructions with their associated T-states, Affected Flags, and Bus Timing figure numbers in a compact format. The second table, Table 6-3, lists all the instructions in opcode order to facilitate disassembly.

TABLE 6-1. Notational Conventions for Instruction Set

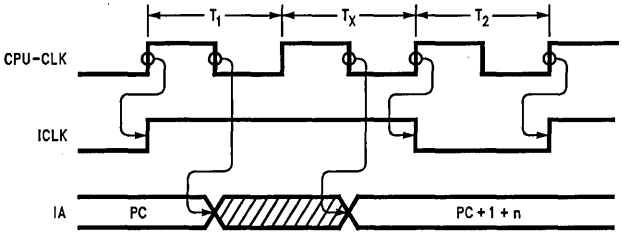
Symbol	Represents	Meaning	Length
n	0 to 255 +127 to −128	Unsigned Number Signed Number	8 Bits
nn	0 to 65535	Unsigned Number	16 Bits
Rs	R0–R31	Source Register	
Rd	R0–R31	Destination Register	
Rsd	R0–R31	Combination Source/Destination Register	
rs	R0–R15	Limited Source Register	
rd	R0–R15	Limited Destination Register	
rsd	R0–R15	Limited Combination Source/Destination Register	
Ir	IW, IX, IY, IZ	Index Register	
mIr		Index Register in One of the Following Address Modes:	
	Ir−	Post Decrement	
	Ir	No Change	
	Ir+	Post Increment	
	+Ir	Pre-Increment	
b	0–7	Shift Field	3 Bits
m	0–7	Mask Field	3 Bits
p	0–7	Position Field	3 Bits
s	0–1	State Field	1 Bit
f	0–7	Flag Reference Field	3 Bits
cc		Condition Code Instruction Extensions	
v	0–63	Vector Field	6 Bits
g	0–3	Global Interrupt Enable Flag [GIE] Status Control	2 Bits
g′	0–1	Global Interrupt Enable Flag [GIE] Limited Status Control	1 Bit
rf	0–1	Register Bank and ALU Flag Status Control	1 Bit
ba	0–1	Register Bank A Select	1 Bit
bb	0–1	Register Bank B Select	1 Bit

6.0 Reference Section (Continued)



TL/F/9336-21

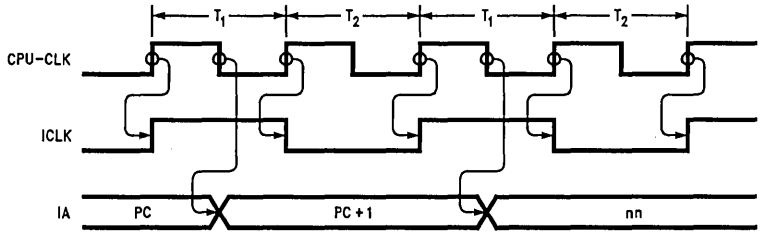
FIGURE 6-1. Instruction-Memory Bus Timing for 2 T-state Instructions (No Instruction Wait States [IW1-0] = 00, CPU Running at Full Speed [CCS] = 0)



TL/F/9336-22

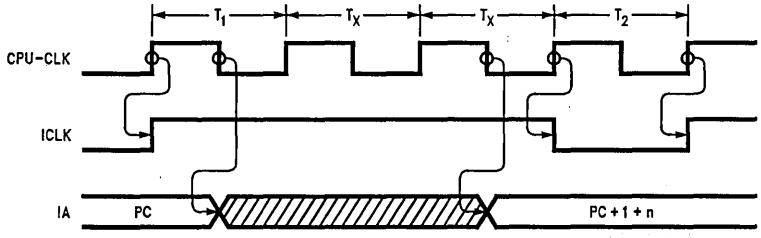
FIGURE 6-2. Instruction-Memory Bus Timing for 3 T-state Instructions (No Instruction Wait States [IW1-0] = 00, CPU Running at Full Speed [CCS] = 0)

6.0 Reference Section (Continued)



TL/F/9336-23

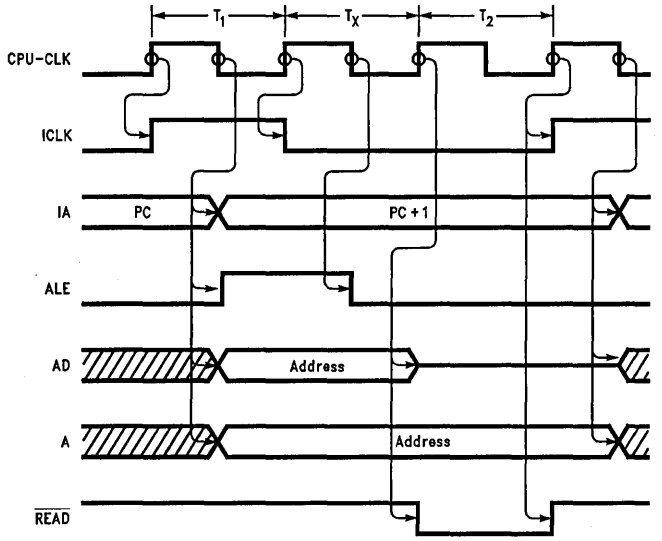
FIGURE 6-3. Instruction-Memory Bus Timing for (2 + 2) T-state Instructions (No Instruction Wait States [IW1-0] = 00, CPU Running at Full Speed [CCS] = 0)



TL/F/9336-24

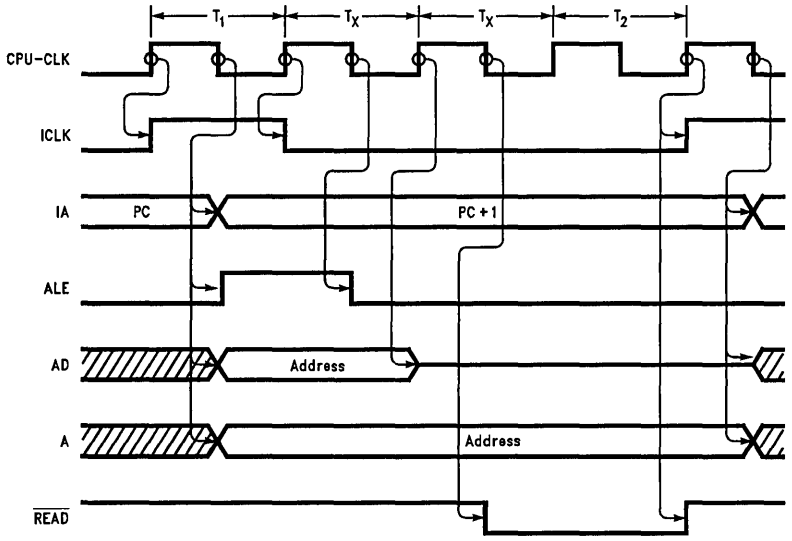
FIGURE 6-4. Instruction-Memory Bus Timing for 4 T-state Instructions (No Instruction Wait States [IW1-0] = 00, CPU Running at Full Speed [CCS] = 0)

6.0 Reference Section (Continued)



TL/F/9336-25

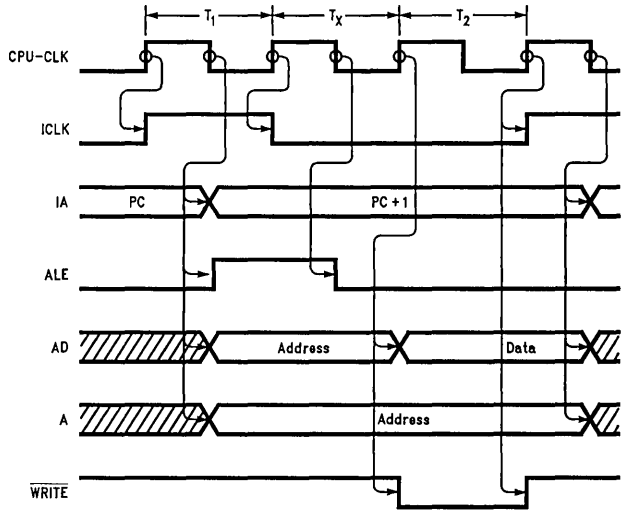
FIGURE 6-5. Instruction/Data Memory Bus Timing for Data Memory Read (No Instruction or Data Memory Wait States, CPU Running at Full Speed [CCS] = 0, [4TR] = 0)



TL/F/9336-I1

FIGURE 6-6. Instruction/Data Memory Bus Timing for Data Memory Read (No Instruction or Data Memory Wait States, CPU Running at Full Speed [CCS] = 0), [4TR] = 1)

6.0 Reference Section (Continued)



TL/F/9336-26

FIGURE 6-7. Instruction/Data Memory Bus Timing for Data Memory Write (No Instruction or Data Memory Wait States, CPU Running at Full Speed [CCS] = 0)

ADCA Add with Carry and Accumulator

Syntax

ADCA Rs, Rd —register, register
ADCA Rs, [mlr] —register, indexed

Affected Flags

N, Z, C, V

Description

Adds the source register Rs, the active accumulator, and the carry flag together, placing the result into the destination specified. The destination may be either a register, Rd, or data memory via an index register mode, [mlr]. Note that register bank selection determines which accumulator is active.

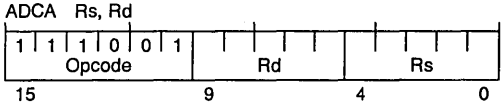
Example

Add the constant 109 to the index register IW, (which is 16 bits wide).

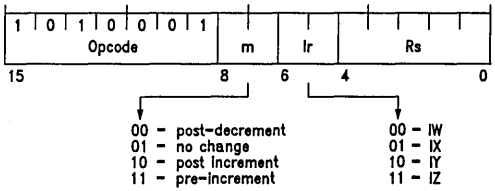
```
SUBA  A, A        ;Clear the accumulator
ADD   109, R12    ;Add 109 to low byte of IW
ADCA  R13, R13    ;Add carry to high byte of IW
```

Instruction Format

ADCA Rs, Rd



ADCA Rs, [mlr]



TL/F/9336-5

T-states

ADCA Rs, Rd —2
ADCA Rs, [mlr] —3

Bus Timing

ADCA Rs, Rd —*Figure 6-1*
ADCA Rs, [mlr] —*Figure 6-6*

Operation

ADCA Rs, Rd
Rs + accumulator + carry bit → Rd

ADCA Rs, [mlr]
Rs + accumulator + carry bit → data memory

6.0 Reference Section (Continued)

ADD Add Immediate

Syntax

ADD n, rsd —immediate, limited register

Affected Flags

N, Z, C, V

Description

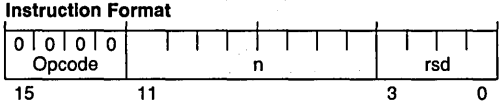
Adds the immediate value n to the register rsd and places the result back into the register rsd. Note that only the active registers R0–R15 may be specified for rsd. The value of n is limited to 8 bits; (unsigned range: 0 to 255, signed range: +127 to −128).

Example

Add the constant −3 to register 10.

```
ADD    −3, R10        ;R10 + (−3) → R10
```

Instruction Format



T-States

2

Bus Timing

Figure 6-1

Operation

rsd + n → rsd

ADDA Add with Accumulator

Syntax

ADDA Rs, Rd —register, register
ADDA Rs, [mlr] —register, indexed

Affected Flags

N, Z, C, V

Description

Adds the source register Rs to the active accumulator and places the result into the destination specified. The destination may be either a register, Rd, or data memory via an index register mode, [mlr]. Note that register bank selection determines which accumulator is active.

Example

In the first example, the value 4 is placed into the currently active accumulator, that accumulator is added to the contents of register 20, and then the result is placed into register 21.

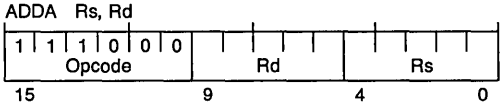
```
MOVE   4, A        ;Place constant into accum
ADDA   R20, R21    ;R20 + accum → R21
```

In the second example, the alternate accumulator of register bank B is selected and then added to register 20. The result is placed into the data memory pointed to by the index register IZ and then the value of IZ is incremented by one.

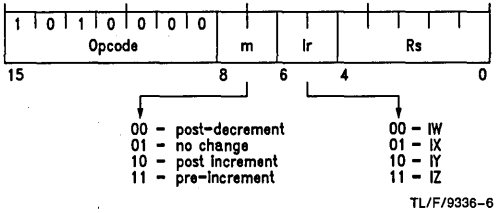
```
EXX    0, 1        ;Select alt accumulator
ADDA   R20, [IZ+]  ;R20 + accum → data mem
                   ;and increment data pointer
```

Instruction Format

ADDA Rs, Rd



ADDA Rs, [mlr]



TL/F/9336–6

T-states

ADDA Rs, Rd —2
ADDA Rs, [mlr] —3

Bus Timing

ADDA Rs, Rd —*Figure 6-1*
ADDA Rs, [mlr] —*Figure 6-6*

Operation

ADDA Rs, Rd
Rs + accumulator → Rd

ADDA Rs, [mlr]
Rs + accumulator → data memory

AND And Immediate

Syntax

AND n, rsd —immediate, limited register

Affected Flags

N, Z

Description

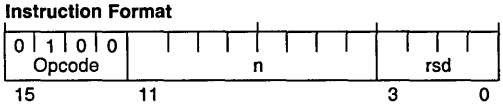
Logically ANDs the immediate value n to the register rsd and places the result back into the register rsd. Note that only the active registers R0–R15 may be specified for rsd. The value of n is 8 bits wide.

Example

Unmask both the Transmitter and Receiver interrupts via the Interrupt Control Register {ICR}, R2. Leave the other interrupts unaffected.

```
EXX    0,0            ;select main register banks
AND    11111100B,R2   ;unmask transmitter and
                      ; receiver interrupts
```

Instruction Format



T-states

2

Bus Timing

Figure 6-1

Operation

rsd AND n → rsd

6.0 Reference Section (Continued)

ANDA And with Accumulator

Syntax

ANDA Rs, Rd —register, register
ANDA Rs, [mlr] —register, indexed

Affected Flags

N, Z

Description

Logically ANDs the source register Rs to the active accumulator and places the result into the destination specified. The destination may be either a register, Rd, or data memory via an index register mode, [mlr]. Note that register bank selection determines which accumulator is active.

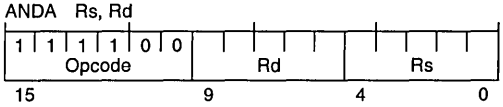
Example

This example demonstrates a way to quickly unload all 11 bits of the three words in the Receiver FIFO when the FIFO is full. The example assumes that the index register IZ points to the location in data memory where the information should be stored.

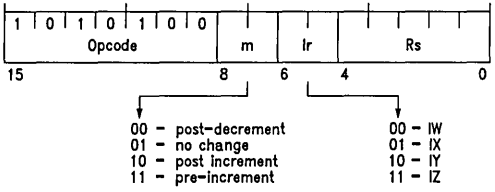
```
    EXX   1,1              ;select alternate banks
    MOVE  00000111B, A     ;place the {TSR} mask
                           ; into the accumulator
;   Pop the first word from the receiver FIFO
    ANDA  TSR, [IZ+]       ;read bits 8, 9, & 10
    MOVE  RTR, [IZ+]       ;pop bits 0-7
;   Pop the second word from the receiver FIFO
    ANDA  TSR, [IZ+]
    MOVE  RTR, [IZ+]
;   Pop the third word from the receiver FIFO
    ANDA  TSR, [IZ+]
    MOVE  RTR, [IZ+]
```

Instruction Format

ANDA Rs, Rd



ANDA Rs,[mlr]



TL/F/9336-7

T-states

ANDA Rs, Rd —2
ANDA Rs, [mlr] —3

Bus Timing

ANDA Rs, Rd —*Figure 6-1*
ANDA Rs, [mlr] —*Figure 6-7*

Operation

ANDA Rs, Rd
Rs AND accumulator → Rd

ANDA Rs, [mlr]
Rs AND accumulator → data memory

BIT Bit Test

Syntax

BIT rs, n —limited register, immediate

Affected Flags

N, Z

Description

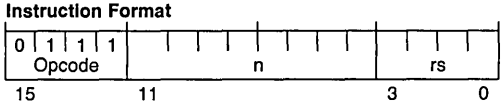
Performs a bit level test by logically ANDing the source register rs to the immediate value n. The affected flags are updated, but the result is not saved. Note that only the active registers R0–R15 may be specified for rs. The value n is 8 bits wide.

Example

Poll the Transmitter FIFO Empty flag [TFE] in the Network Command Flag register {NCF}, R1, waiting for the Transmitter to send the current FIFO data.

```
        EXX  0,1              ;select main A, alt B
Poll:   BIT  NCF,10000000B    ;All data sent yet?
        JZ   Poll             ; No, poll TFE
        ...                   ; Yes, send next byte(s)
```

Instruction Format



T-states

2

Bus Timing

Figure 6-1

Operation

rs AND n

6.0 Reference Section (Continued)

CALL Unconditional Relative Call

Syntax

CALL n —immediate

Affected Flags

None

Description

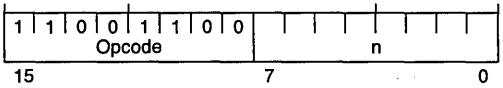
Pushes the Program Counter, the ALU flags, the Global Interrupt Enable bit [GIE], and the current register bank selections onto the internal Address Stack; then unconditionally transfers control to the instruction at the memory address calculated by adding the contents of the Program Counter to the immediate value n, (sign extended to 16 bits). Since the immediate value n is an 8-bit two's complement displacement, the unconditional relative call's range is from +127 to −128 relative to the Program Counter. Note that the Program Counter initially contains the memory address of the next instruction following the call.

Example

Transfer control to the subroutine "Send.it". Note that "Send.it" must be within +127/−128 words relative to the PC.

```
CALL    Send.it
```

Instruction Format



T-states

3

Bus Timing

Figure 6-2

Operation

PC & [GIE] & ALU flags & register bank selections
→ Address Stack
PC + n(sign extended) → PC

CMP Compare

Syntax

CMP rs, n —limited register, immediate

Affected Flags

N, Z, C, V

Description

Compares the immediate value n with the source register rs by subtracting n from rs. The affected flags are updated, but the result is not saved. Note that only the active registers R0–R15 may be specified for rs. The value of n is limited to 8 bits; (unsigned range: 0 to 255, signed range: +127 to −128).

Example

Compare the data byte in register 11 to the ASCII character "A".

```
CMP     R11,"A"          ;If:
JC      Less_than_A      ;    data<"A"
JEQ     Equal_to_A       ;    data="A"
...                      ;else data>"A"
```

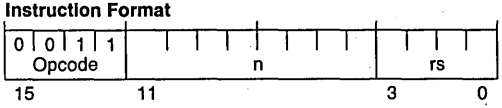
Compare the contents of register 8 to the value 25.

```
CMP     R8,25            ;if:
BIT     CCR,00000011B    ;    data > 25
JZ      Greater_than     ;    Goto Greater_than
```

Comparing of Unsigned Values	
Comparison	**Flag(s) to Test**
LT (<)	C
LEQ (<=)	C|Z
EQ (=)	Z
GEQ (>=)	$\overline{C}$
GT (>)	$\overline{C}$ & $\overline{Z}$

Note: & = logical AND
| = logical OR

Instruction Format



T-states

2

Bus Timing

Figure 6-1

Operation

rs − n

6.0 Reference Section (Continued)

CPL Complement

Syntax

CPL Rsd —register

Affected Flags

N, Z

Description

Logically complements the contents of the register Rsd, placing the result back into that register.

Example

Load the fill-bit count passed from the host into the Transmitter's Fill-Bit Register {FBR}, R3, and then perform the required one's complement of the fill-bit count. In this example, register 20 contains the fill-bit count.

```
EXX     1,1       ;select alternate banks
MOVE    R20, FBR  ;load {FBR}
CPL     FBR       ;complement fill-bit count
```

Instruction Format

1	0	1	0	1	1	1	0	0	0	0				
Opcode											Rsd			

15 4 0

T-states

2

Bus Timing

Figure 6-1

Operation

$\overline{\text{Rsd}} \rightarrow$ Rsd

EXX Exchange Register Banks

Syntax

EXX ba, bb {,g}

Affected Flags

None

Description

Selects which CPU register banks are active by exchanging between the main and alternate register sets for each bank. Bank A controls R0–R3 and Bank B controls R4–R11. The table below shows the four possible register bank configurations. Note that deactivated registers retain their current values. The Global Interrupt Enable bit [GIE] can be set or cleared, if desired.

Register Bank Configurations

ba	bb	Active Register Banks
0	0	Main A, Main B
0	1	Main A, Alternate B
1	0	Alternate A, Main B
1	1	Alternate A, Alternate B

Example

Activate the main register set of Bank A, the alternate register set of Bank B, and leave the Global Interrupt Enable bit [GIE] unchanged.

```
EXX     0,1       ;select main A, alt B reg banks
```

Instruction Format

1	0	1	0	1	1	1	0	1				0	0	0
Opcode									g	ba	bb			

15 6 4 3 2 0

00—GIE not affected
01—reserved
10—Set GIE
11—Clear GIE

T-states

2

Bus Timing

Figure 6-1

Operation

```
Case ba of
  0:  activate main Bank A
  1:  activate alternate Bank A
End case
Case bb of
  0:  activate main Bank B
  1:  activate alternate Bank B
End case
Case g of
  0:  leave [GIE] unaffected, (default)
  1:  (reserved)
  2:  set [GIE]
  3:  clear [GIE]
End case
```

6.0 Reference Section (Continued)

JMP Conditional Relative Jump

Jcc

Syntax

JMP f, s, n —immediate
Jcc n —immediate (optional syntax)

Affected Flags

None

Description

Conditionally transfers control to the instruction at the memory address calculated by adding the contents of the Program Counter to the immediate value n, (sign extended to 16 bits), if the state of the flag referenced by f is equal to the state of the bit s; or, optionally, if the condition cc is met. See the tables below for the flags that f can reference and the conditions that cc may specify. Since the immediate value n is an 8-bit two's complement displacement, the conditional relative jump's range is from +127 to −128 relative to the Program Counter. Note that the Program Counter initially contains the memory address of the next instruction following the jump.

Example

This example demonstrates both syntaxes of the conditional relative jump instruction testing for a non-zero result from a previous instruction; (i.e., [Z] = 0). If the condition is met then control transfers to the instruction labeled "Loop.back"; else the next instruction following the jump is executed.

```
JMP   000B,0,Loop.back   ;jump on not zero
...
JNZ   Loop.back          ;jump on not zero
```

Condition Specification Table for "cc"

cc	Meaning	Condition Tested for
Z	Zero	[Z] = 1
NZ	Not Zero	[Z] = 0
EQ	Equal	[Z] = 1
NEQ	Not Equal	[Z] = 0
C	Carry	[C] = 1
NC	No Carry	[C] = 0
V	Overflow	[V] = 1
NV	No Overflow	[V] = 0
N	Negative	[N] = 1
P	Positive	[N] = 0
RA	Receiver Active	[RA] = 1
NRA	Not Receiver Active	[RA] = 0
RE	Receiver Error	[RE] = 1
NRE	No Receiver Error	[RE] = 0
DA	Data Available	[DAV] = 1
NDA	No Data Available	[DAV] = 0
TFF	Transmitter FIFO Full	[TFF] = 1
NTFF	Transmitter FIFO Not Full	[TFF] = 0

Instruction Format

1 1 0 1 Opcode	s	f	n
15	11	10 7	0

T-states

2 if condition is not met
3 if condition is met

Bus Timing

Figure 6-1 if condition is not met
Figure 6-2 if condition is met

Operation

JMP f, s, n
If flag f is in state s
then PC + n(sign extended) → PC

Jcc n
If cc condition is true
then PC + n(sign extended) → PC

Flag Reference Table for "f"

f	(binary)	Flag Reference
0	(000)	[Z] in {CCR}
1	(001)	[C] in {CCR}
2	(010)	[V] in {CCR}
3	(011)	[N] in {CCR}
4	(100)	[RA] in {TSR}
5	(101)	[RE] in {TSR}
6*	(110)	[DAV] in {TSR}
7	(111)	[TFF] in {TSR}

***Note:** The value of f for [DAV] differs from the numeric value for the position of [DAV] in {TSR}.

6.0 Reference Section (Continued)

JMP Unconditional Relative Jump

Syntax

JMP n —immediate
JMP Rs —register

Affected Flags

None

Description

Unconditionally transfers control to the instruction at the memory address calculated by adding the contents of the Program Counter to either the immediate value n or the contents of the source register Rs, (both sign extended to 16 bits). Since the immediate value n and the contents of Rs are 8-bit two's complement displacements, the unconditional relative jump's range is from +127 to −128 relative to the Program Counter. Note that the Program Counter initially contains the memory address of the next instruction following the jump.

Example

Transfer control to the instruction labeled "Init_Xmit", which is within +127/−128 words relative to the PC.

```
JMP  Init_Xmit     ;go initialize Transmitter
```

Instruction Format

JMP n

1	1	0	0	1	0	1	1								
Opcode								n							

15 7 0

JMP Rs

1	1	0	0	1	1	0	1	1	0	0					
Opcode											Rs				

15 4 0

T-states

JMP n —3
JMP Rs —4

Bus Timing

JMP n —*Figure 6-2*
JMP Rs —*Figure 6-4*

Operation

JMP n
PC + n(sign extended) → PC

JMP Rs
PC + Rs(sign extended) → PC

6.0 Reference Section (Continued)

JRMK Relative Jump with Rotate and Mask on Register

Syntax

JRMK Rs, b, m —register

Affected Flags

None

Description

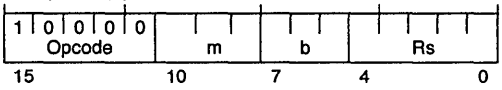
Transfers control to the instruction at the memory address calculated by adding the contents of the Program Counter to a specially formed displacement. The displacement is formed by rotating a copy of the source register Rs the value of b bits to the right, masking (setting to zero) the most significant m bits, masking the least significant bit, and then sign extending the result to 16 bits. Typically, the JRMK instruction transfers control into a jump table. The LSB of the displacement is always set to zero so that the jump table may contain two word instructions, (e.g., LJMP). The range of JRMK is from +126 to −128 relative to the Program Counter. Note that the Program Counter initially contains the memory address of the next instruction following JRMK. The source register Rs may specify any active CPU register. The rotate value b may be from 0 to 7, where 0 causes no bit rotation to occur. The mask value m may be from 0 to 7; where m=0 causes only the LSB of the displacement to be masked, m=1 causes the MSB and the LSB to be masked, m=2 causes bits 7–6 and the LSB to be masked, etc ...

Example

This example demonstrates the decoding of the address frame of the 3299 Terminal Multiplexer protocol. In the address frame, only the bits 4–2 contain the address of the Logical Unit.

```
EXX     0,1       ;select main A, alt B
JRMK    RTR,1,4   ;decode device address
LJMP    ADDR.0    ;jump to device handler #0
LJMP    ADDR.1    ;jump to device handler #1
LJMP    ADDR.2    ;jump to device handler #2
...
LJMP    ADDR.7    ;jump to device handler #7
```

Instruction Format



T-states

4

Bus Timing

Figure 6-4

Operation

Copy Rs to a temporary register:

Rs → register

Rotate the register b bits to the right:

TL/F/9336–8

Mask the most significant m bits and the LSB:

register AND $\overbrace{0 \ldots 0}^{m}\ 1 \ldots 1\ 0$ → register

Modify the Program Counter:

PC + register(sign extended) → PC

6.0 Reference Section (Continued)

LCALL Conditional Long Call

Syntax

LCALL Rs, p, s, nn —register, absolute

Affected Flags

None

Description

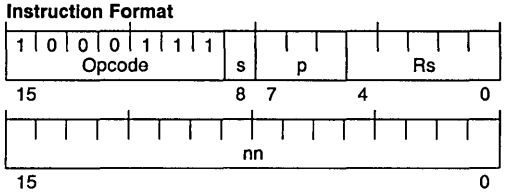
If the bit in position p of register Rs is equal to the bit s, then push the Program Counter, the ALU flags, the Global Interrupt Enable bit [GIE], and the current register bank selections onto the internal Address Stack. Following the push, transfer control to the instruction at the absolute memory address nn. The operand Rs may specify any active CPU register. The value of p may be from 0 to 7, where 0 corresponds to the LSB of Rs and 7 corresponds to the MSB of Rs. The absolute value nn is 16 bits long, (range: 0 to 64k), therefore, all of instruction memory can be addressed.

Example

Call the "Load.Xmit" subroutine when the Transmitter FIFO Empty flag, [TFE], of the Network Command Flag register {NCF} is "1".

```
EXX     0,0                     ;select main A, alt B
LCALL   NCF,7,1, Load.Xmit      ;If [TFE] = 1 call
```

Instruction Format



T-states

(2 + 2)

Bus Timing

Figure 6-3

Operation

If Rs[p] = s then
  PC & [GIE] & ALU flags & register bank selections
    → Address Stack
  nn → PC
End if

LCALL Unconditional Long Call

Syntax

LCALL nn —absolute

Affected Flags

None

Description

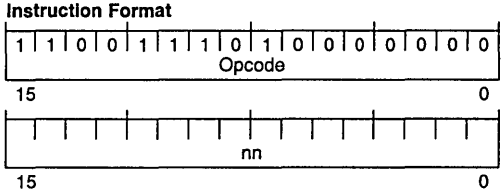
Pushes the Program Counter, the ALU flags, the Global Interrupt Enable bit [GIE], and the current register bank selections onto the internal Address Stack; then unconditionally transfers control to the instruction at the absolute memory address nn. The value of nn is 16 bits long, (range: 0 to 64k), therefore, all of instruction memory can be addressed.

Example

Transfer control to the subroutine "Send.it.all", which could be located anywhere in instruction memory.

```
LCALL    Send.it.all
```

Instruction Format



T-states

(2 + 2)

Bus Timing

Figure 6-3

Operation

PC & [GIE] & ALU flags & register bank selections
  → Address Stack
nn → PC

6.0 Reference Section (Continued)

LJMP Conditional Long Jump

Syntax

LJMP Rs, p, s, nn —register, absolute

Affected Flags

None

Description

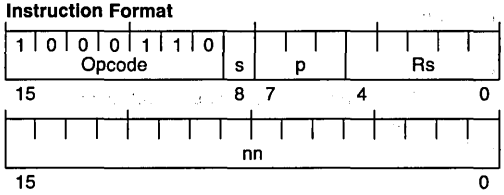
Conditionally transfers control to the instruction at the absolute memory address nn if the bit in position p of register Rs is equal to the state of the bit s. The operand Rs may specify any active CPU register. The value of p may be from 0 to 7, where 0 corresponds to the LSB of Rs and 7 corresponds to the MSB of Rs. The absolute value nn is 16 bits long, (range: 0 to 64k), therefore, all of instruction memory can be addressed.

Example

Long Jump to one of the receiver error handling routines based on the contents of the Error Code Register {ECR}.

```
  EXX     0,1,3              ;select main A, alt B
                             ; and clear [GIE]
  OR      01000000B,TSR      ;set [SEC] in {TSR}
  MOVE    ECR, R11           ;read {ECR}
; Determine error condition
  LJMP    R11, 0, 1, Software__error
  LJMP    R11, 1, 1, Loss__of__Midbit
  LJMP    R11, 2, 1, Invalid__Ending__Seq
  LJMP    R11, 3, 1, Parity__error
  LJMP    R11, 4, 1, Software__error
```

Instruction Format



T-states

(2 + 2)

Bus Timing

Figure 6-3

Operation

If Rs[p] = s
then nn → PC

LJMP Unconditional Long Jump

Syntax

LJMP nn —absolute
LJMP [Ir] —indexed

Affected Flags

None

Description

Unconditionally transfers control to the instruction at the memory address specified by the operand. The operand may either specify an absolute instruction address nn, (16 bits long), or an index register Ir, which contains an instruction address. Long Jump's addressing range is from 0 to 64k; (i.e., all of instruction memory can be addressed).

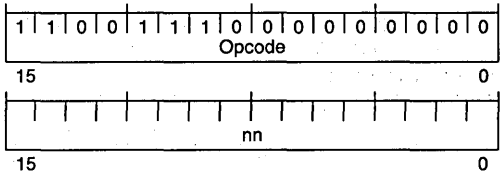
Example

Transfer control to the instruction labeled "Reset.System", which may be located anywhere in instruction memory.

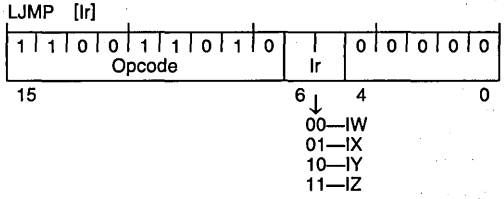
```
  LJMP    Reset.System       ;go reset the system
```

Instruction Format

LJMP nn



LJMP [Ir]



T-states

LJMP nn —(2 + 2)
LJMP [Ir] —2

Bus Timing

LJMP nn —*Figure 6-3*
LJMP [Ir] —*Figure 6-1*

Operation

LJMP nn
nn → PC

LJMP [Ir]
Ir → PC

6.0 Reference Section (Continued)

MOVE Move Data Memory

Syntax

MOVE [mIr], Rd —indexed, register
MOVE [Ir+A], Rd —register-relative, register
MOVE [IZ+n], rd —immediate-relative, limited register

Affected Flags

None

Description

Moves a data memory byte into the destination register specified. The data memory source operand may specify any one of the index register modes; [mIr], [Ir+A], [IZ+n]. The index register-relative mode, [Ir+A], forms its data memory address by adding the contents of the index register Ir to the unsigned 8-bit value contained in the currently active accumulator. The immediate-relative mode, [IZ+n], forms its data memory address by adding the contents of the index register IZ to the unsigned 8-bit immediate value n. The destination register operand Rd may specify any active CPU register; where as, the destination register operand rd is limited to the active registers R0–R15.

Example

The first example loads the current accumulator by "popping" an external data stack, which is pointed to by the index register IX.

```
MOVE    [+IX], A        ;pop accum from ext. stack
```

The second example demonstrates the random access of a data byte within a logical record contained in memory. The index register IY contains the base address of the logical record.

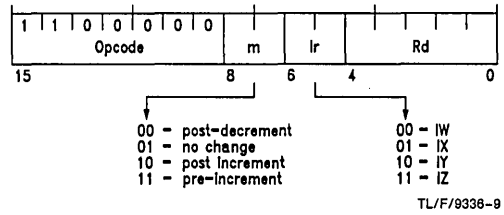
```
ADDA    R9, A           ;calculate offset into record
MOVE    [IY+A], R20     ;get data byte from record
```

In the final example, the 4th element of an Error Count table is transmitted to a host. The index register IZ points to the 1st entry of the table.

```
EXX     0,1             ;select main A, alt B
MOVE    [IZ+3], RTR     ;transmit 4th element
```

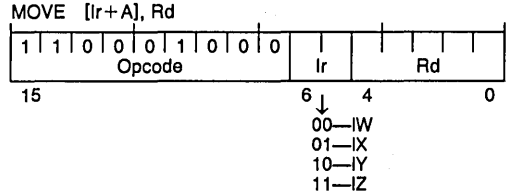
Instruction Format

MOVE [mIr], Rd

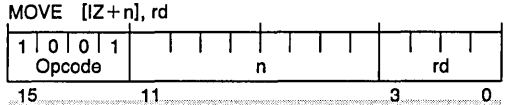


TL/F/9336-9

MOVE [Ir+A], Rd



MOVE [IZ+n], rd



T-states

3 [4TR] = 0
4 [4TR] = 1

Bus Timing

Figure 6-5 [4TR] = 0
Figure 6-6 [4TR] = 1

Operation

MOVE [mIr], Rd
data memory → Rd

MOVE [Ir+A], Rd
data memory → Rd

MOVE [IZ+n], rd
data memory → rd

6.0 Reference Section (Continued)

MOVE Move Immediate

Syntax

MOVE n, rd —immediate, limited register

MOVE n, [Ir] —immediate, indexed

Affected Flags

None

Description

Moves the immediate value n into the destination specified. The destination may be either a register, rd, (limited to the active registers R0–R15), or data memory via an index register, Ir. The value n is 8 bits wide.

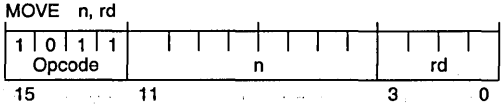
Example

Load the current accumulator with the value of 4.

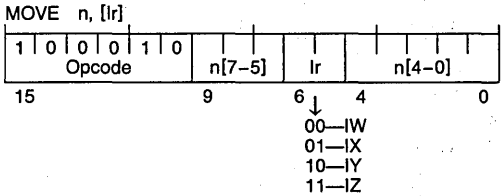
```
MOVE  4, A        ;Load accumulator
```

Instruction Format

MOVE n, rd



MOVE n, [Ir]



T-states

MOVE n, rd —2

MOVE n, [Ir] —3

Bus Timing

MOVE n, rd —*Figure 6-1*

MOVE n, [Ir] —*Figure 6-7*

Operation

MOVE n, rd

n → rd

MOVE n, [Ir]

n → data memory

6.0 Reference Section (Continued)

MOVE Move Register

Syntax

MOVE	Rs, Rd	—register, register
MOVE	Rs, [mIr]	—register, indexed
MOVE	Rs, [Ir+A]	—register, register-relative
MOVE	rs, [IZ+n]	—limited register, immediate-relative

Affected Flags

None

Description

Moves the contents of the source register into the destination specified. The source register operand Rs may specify any active CPU register; where as the source register operand rs is limited to the active registers R0–R15. The destination operand may specify either any active CPU register, Rd, or data memory via one of the index register modes; [mIr], [Ir+A], [IZ+n]. The index register-relative mode, [Ir+A], forms its data memory address by adding the contents of the index register Ir to the unsigned 8-bit value contained in the currently active accumulator. The immediate-relative mode, [IZ+n], forms its data memory address by adding the contents of the index register IZ to the unsigned 8-bit immediate value n.

Example

The first example loads the Transmitter FIFO with a data byte in register 20.

```
EXX     0,1          ;select main A, alt B
MOVE    R20, RTR     ;Load the Transmitter FIFO
```

The second example "pushes" the current accumulator's contents onto an external data stack, which is pointed to by the index register IX.

```
MOVE    A, [IX−]     ;push accum to ext. stack
```

The third example demonstrates the random access of a data byte within a logical record contained in memory. The index register IY contains the base address of the logical record.

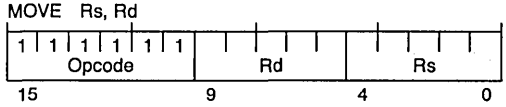
```
ADDA    R9, A        ;calculate offset into record
MOVE    R20, [IY+A]  ;update data byte in record
```

In the final example, the 4th element of an Error Count table is updated with a new value contained in the current accumulator. The index register IZ points to the 1st entry of the table.

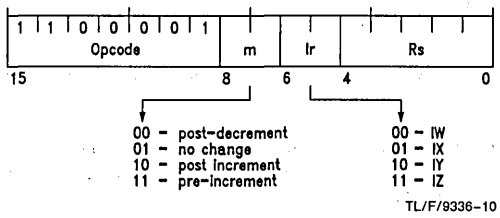
```
MOVE    A, [IZ+3]    ;update 4th element of table
```

Instruction Format

MOVE Rs, Rd

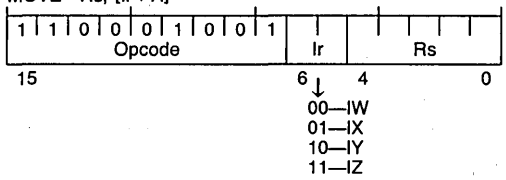


MOVE Rs, [mIr]

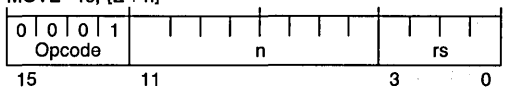


TL/F/9336–10

MOVE Rs, [Ir+A]



MOVE rs, [Z+n]



T-states

MOVE	Rs, Rd	—2
MOVE	Rs, [mIr]	—3
MOVE	Rs, [Ir+A]	—3
MOVE	rs, [IZ+n]	—3

Bus Timing

MOVE	Rs, Rd	—*Figure 6-1*
MOVE	Rs, [mIr]	—*Figure 6-6*
MOVE	Rs, [Ir+A]	—*Figure 6-6*
MOVE	rs, [IZ+n]	—*Figure 6-6*

Operation

MOVE	Rs, Rd	—Rs → Rd
MOVE	Rs, [mIr]	—Rs → data memory
MOVE	Rs, [Ir+A]	—Rs → data memory
MOVE	rs, [IZ+n]	—rs → data memory

6.0 Reference Section (Continued)

OR OR Immediate

Syntax

OR n, rsd —immediate, limited register

Affected Flags

N, Z

Description

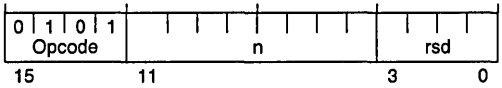
Logically ORs the immediate value n to the register rsd and places the result back into the register rsd. Note that only the active registers R0–R15 may be specified for rsd. The value of n is 8 bits wide.

Example

Mask both the Transmitter and Receiver interrupts via the Interrupt Control Register {ICR}, R2. Leave the other interrupts unaffected.

```
EXX  0,0              ;select main reg banks
OR   00000011B, ICR   ;mask transmitter and
                      ; receiver interrupts
```

Instruction Format



T-states

2

Bus Timing

Figure 6-1

Operation

rsd OR n → rsd

ORA OR with Accumulator

Syntax

ORA Rs, Rd —register, register

ORA Rs, [mlr] —register, indexed

Affected Flags

N, Z

Description

Logically ORs the source register Rs to the active accumulator and places the result into the destination specified. The destination may be either a register, Rd, or data memory via an index register mode, [mlr]. Note that register bank selection determines which accumulator is active.

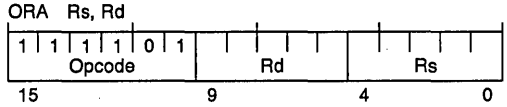
Example

Write an 11-bit word to the Transmitter's FIFO. This example assumes that the index register IZ points to the location of the data in memory.

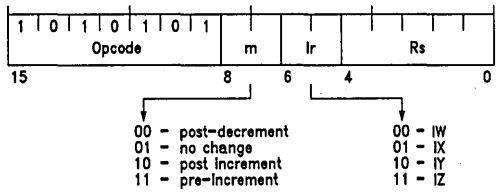
```
TCR.settings:  .EQU   00101000B
  . . .
EXX    1,1            ;select main A, alt B
MOVE   TCR.settings,A ;load accumulator w/mask
MOVE   [IZ+],R20      ;load bits 8, 9, & 10
ORA    R20,TCR        ;write bits 8, 9, 10 to {TCR}
MOVE   [IZ+],RTR      ;push 11-bit word to FIFO
```

Instruction Format

ORA Rs, Rd



ORA Rs, [mlr]



TL/F/9336–11

T-states

ORA Rs, Rd —2

ORA Rs, [mlr] —3

Bus Timing

ORA Rs, Rd —*Figure 6-1*

ORA Rs, [mlr] —*Figure 6-7*

Operation

ORA Rs, Rd

Rs OR accumulator → Rd

ORA Rs, [mlr]

Rs OR accumulator → data memory

6.0 Reference Section (Continued)

RETF Conditional Return

Rcc

Syntax

RETF f, s{,{g} {,rf}}
Rcc {g{,rf}} —(optional syntax)

Affected Flags

If rf = 1 then N, Z, C, and V

Description

Conditionally returns control to the last instruction address pushed onto the internal Address Stack by popping that address into the Program Counter, if the state of the flag referenced by f is equal to the state of the bit s; or, optionally, if the condition cc is met. See the tables on the following page for the flags that f can reference and the conditions that cc may specify. The conditional return instruction also has two optional operands, g and rf. The value of g determines if the Global Interrupt Enable bit [GIE] is left unchanged (g=0), restored from the Address Stack (g=1), set (g=2), or cleared (g=3). If the g operand is omitted then g=0 is assumed. The second optional operand, rf, determines if the ALU flags and register bank selections are left unchanged (rf=0), or restored from the Address Stack (rf=1). If the rf operand is omitted then rf=0 is assumed.

Example

This example demonstrates both syntaxes of the conditional return instruction testing for a carry result from a previous instruction; (i.e., [C]=1). If the condition is met then the return occurs, else the next instruction following the return is executed. The current environment is left unchanged.

```
RETF   001B,1        ; If [C]=1 then return
...
RC                   ; If [C]=1 then return
```

Instruction Format

15								6	4	3	2	0
1	0	1	0	1	1	1	1	0	g	rf	s	f
Opcode												

g:
00—GIE not affected
01—Restore GIE
10—Set GIE
11—Clear GIE

T-states

2 if condition is not met

3 if condition is met

Bus Timing

Figure 6-1 if condition is not met
Figure 6-2 if condition is met

Operation

```
If flag f is in state s then
  Case g of
    0: leave [GIE] unaffected, (default)
    1: restore [GIE] from Address Stack
    2: set [GIE]
    3: clear [GIE]
  End case
  If rf=1 then
    restore ALU flags from Address Stack
    restore register bank selection from Address Stack
  End if
  Address Stack → PC
End if
```

Condition Specification Table for "cc"

cc	Meaning	Condition Tested for
Z	Zero	[Z] = 1
NZ	Not Zero	[Z] = 0
EQ	Equal	[Z] = 1
NEQ	Not Equal	[Z] = 0
C	Carry	[C] = 1
NC	No Carry	[C] = 0
V	Overflow	[V] = 1
NV	No Overflow	[V] = 0
N	Negative	[N] = 1
P	Positive	[N] = 0
RA	Receiver Active	[RA] = 1
NRA	Not Receiver Active	[RA] = 0
RE	Receiver Error	[RE] = 1
NRE	No Receiver Error	[RE] = 0
DA	Data Available	[DAV] = 1
NDA	No Data Available	[DAV] = 0
TFF	Transmitter FIFO Full	[TFF] = 1
NTFF	Transmitter FIFO Not Full	[TFF] = 0

Flag Reference Table for "f"

f	(binary)	Flag Referenced
0	(000)	[Z] in {CCR}
1	(001)	[C] in {CCR}
2	(010)	[V] in {CCR}
3	(011)	[N] in {CCR}
4	(100)	[RA] in {TSR}
5	(101)	[RE] in {TSR}
6*	(110)	[DAV] in {TSR}
7	(111)	[TFF] in {TSR}

***Note:** The value of f for [DAV] differs from the numeric value for the position of [DAV] in {TSR}.

6.0 Reference Section (Continued)

RET Unconditional Return

Syntax

RET {g {,rf}}

Affected Flags

If rf=1 then N, Z, C, and V

Description

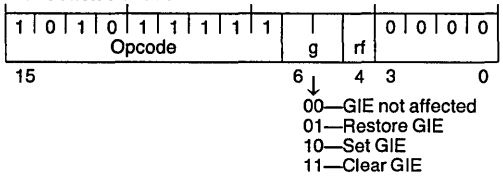
Unconditionally returns control to the last instruction address pushed onto the internal Address Stack by popping that address into the Program Counter. The unconditional return instruction also has two optional operands, g and rf. The value of g determines if the Global Interrupt Enable bit [GIE] is left unchanged (g=0), restored from the Address Stack (g=1), set (g=2), or cleared (g=3). If the g operand is omitted then g=0 is assumed. The second optional operand, rf, determines if the ALU flags and register bank selections are left unchanged (rf=0), or restored from the Address Stack (rf=1). If the rf operand is omitted then rf=0 is assumed.

Example

Return from an interrupt.

```
RET    1,1          ;Restore environment & return
```

Instruction Format



T-states

2

Bus Timing

Figure 6-1

Operation

```
Case g of
  0: leave [GIE] unaffected, (default)
  1: restore [GIE] from Address Stack
  2: set [GIE]
  3: clear [GIE]
End case
If rf=1 then
  restore ALU flags from Address Stack
  restore register bank selection from Address Stack
End if
Address Stack → PC
```

ROT Rotate

Syntax

ROT Rsd, b —register

Affected Flags

N, Z, C

Description

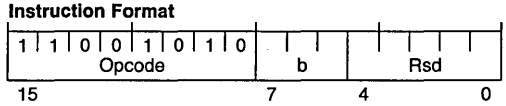
Rotates the contents of the register Rsd b bits to the right and places the result back into that register. The bits that are shifted out of the LSB are shifted back into the MSB, (and copied into the Carry flag). The value b may specify from 0 to 7 bit rotates.

Example

Add 3 to the Address Stack Pointer contained in the Internal Stack Pointer register {ISP}, R30.

```
MOVE   ISP, R8     ;get {ISP}
ROT    R8, 4       ;shift [ASP] to low order nibble
ADD    3, R8       ;add 3 to [ASP]
ROT    R8, 4       ;shift [ASP] to high order nibble
MOVE   R8, ISP     ;store new {ISP}
```

Instruction Format



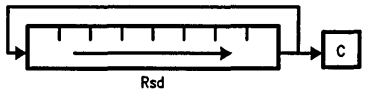
T-states

2

Bus Timing

Figure 6-1

Operation



TL/F9336-12

6.0 Reference Section (Continued)

SBCA Subtract with Carry and Accumulator

Syntax

SBCA Rs, Rd —register, register
SBCA Rs, [mlr] —register, indexed

Affected Flags

N, Z, C, V

Description

Subtracts the active accumulator and the carry flag from the source register Rs, placing the result into the destination specified. The destination may be either a register, Rd, or data memory via an index register mode, [mlr]. Negative results are represented using the two's complement format. Note that register bank selection determines which accumulator is active.

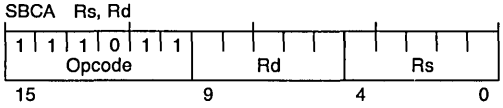
Example

Subtract the constant 109 from the index register IW, (which is 16 bits wide).

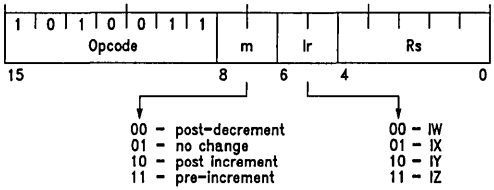
```
SUBA  A, A        ;Clear the accumulator
SUB   109, R12    ;low byte of IW—109
SBCA  R13, R13    ;high byte of IW—borrow
```

Instruction Format

SBCA Rs, Rd



SBCA Rs, [mlr]



TL/F9336-13

T-states

SBCA Rs, Rd —2
SBCA Rs, [mlr] —3

Bus Timing

SBCA Rs, Rd —*Figure 6-1*
SBCA Rs, [mlr] —*Figure 6-7*

Operation

SBCA Rs, Rd
Rs − accumulator − carry bit → Rd

SBCA Rs, [mlr]
Rs − accumulator − carry bit → data memory

SHL Shift Left

Syntax

SHL Rsd, b —register

Affected Flags

N, Z, C

Description

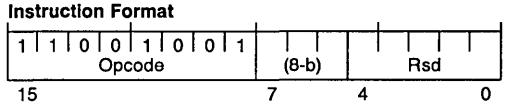
Shifts the contents of the register Rsd b bits to the left and places the result back into that register. Zeros are shifted in from the right, (i.e., from the LSB). The value b may specify from 0 to 7 bit shifts. The Carry flag contains the last bit shifted out.

Example

Place a new internal Address Stack Pointer into the Internal Stack Pointer register {ISP}, R30. Assume that the new [ASP] is located in register 20.

```
MOVE  ISP,R8        ;read {ISP} for [DSP]
AND   00001111B,R8  ;save [DSP] only
SHL   R20,4         ;left justify [ASP]
ORA   R20,ISP       ;combine [ASP] + [DSP],
                    ; then place into {ISP}
```

Instruction Format



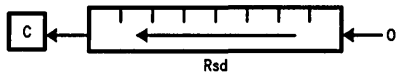
T-states

2

Bus Timing

Figure 6-1

Operation



TL/F/9336-14

6.0 Reference Section (Continued)

SHR Shift Right

Syntax

SHR Rsd, b —register

Affected Flags

N, Z, C

Description

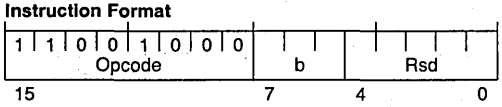
Shifts the contents of the register Rsd b bits to the right and places the result back into that register. Zeros are shifted in from the left, (i.e., from the MSB). The value b may specify from 0 to 7 bit shifts. The Carry flag contains the last bit shifted out.

Example

Right justify the Address Stack Pointer from the Internal Stack Pointer register {ISP}, R30.

```
MOVE  ISP, R20     ;Load [ASP] from {ISP}
SHR   R20,4        ;right justify [ASP]
```

Instruction Format



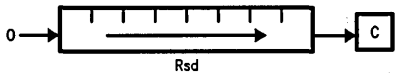
T-states

2

Bus Timing

Figure 6-1

Operation



TL/F/9336-15

SUB Subtract Immediate

Syntax

SUB n, rsd —immediate, limited register

Affected Flags

N, Z, C, V

Description

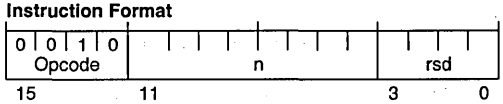
Subtracts the immediate value n from the register rsd and places the result back into the register rsd. Note that only the active registers R0–R15 may be specified for rsd. The value of n is limited to 8 bits; (signed range: +127 to −128). Negative numbers are represented using the two's complement format.

Example

Subtract the constant 3 from register 10.

```
SUB  3, R10    ; R10 − 3 → R10
```

Instruction Format



T-states

2

Bus Timing

Figure 6-1

Operation

rsd − n → rsd

6.0 Reference Section (Continued)

SUBA Subtract with Accumulator

Syntax

SUBA Rs, Rd —register, register
SUBA Rs, [mlr] —register, indexed

Affected Flags

N, Z, C, V

Description

Subtracts the active accumulator from the source register Rs and places the result into the destination specified. The destination may be either a register, Rd, or data memory via an index register mode, [mlr]. Negative numbers are represented using the two's complement format. Note that register bank selection determines which accumulator is active.

Example

In the first example, the value 4 is placed into the currently active accumulator, that accumulator is subtracted from the contents of register 20, and then the result is placed into register 21.

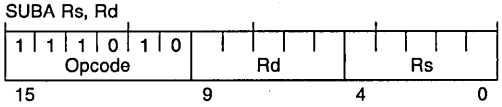
```
MOVE  4, A        ;Place constant into accum
SUBA  R20, R21    ;R20 − accum → R21
```

In the second example, the alternate accumulator of register bank B is selected and then subtracted from register 20. The result is placed into the data memory pointed to by the index register IZ and then the value of IZ is incremented by one.

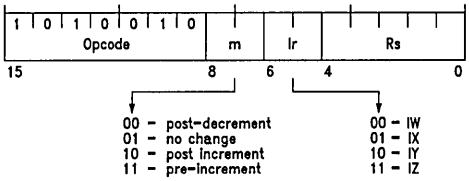
```
EXX   0, 1        ;Select alt accumulator
SUBA  R20, [IZ+]  ;R20 − accum → data mem
                  ;and increment data pointer
```

Instruction Format

SUBA Rs, Rd



SUBA Rs, [mlr]



TL/F/9336-16

T-states

SUBA Rs, Rd —2
SUBA Rs, [mlr] —3

Bus Timing

SUBA Rs, Rd —*Figure 6-1*
SUBA Rs, [mlr] —*Figure 6-7*

Operation

SUBA Rs, Rd
Rs − accumulator → Rd

SUBA Rs, [mlr]
Rs − accumulator → data memory

TRAP Software Interrupt

Syntax

TRAP v {,g′}

Affected Flags

None

Description

Pushes the Program Counter, the Global Interrupt Enable bit [GIE], the ALU flags, and the current register bank selections onto the internal Address Stack; then unconditionally transfers control to the instruction at the memory address created by concatenating the contents of the Interrupt Base Register {IBR} to the value of v extended with zeros to 8 bits. If the value of g′ is equal to "1" then the Global Interrupt Enable bit [GIE] will be cleared. If the g′ operand is omitted, then g′ = 0 is assumed. The vector number v points to one of 64 Interrupt Table entries; (range: 0 to 63). Since some of the Interrupt Table entries are used by the hardware interrupts, the TRAP instruction can simulate hardware interrupts. The following table lists the hardware interrupts and their associated vector numbers:

Hardware Interrupt Vector Table

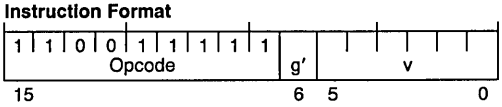
Interrupt	v	(Binary)
NMI	28	(011100)
RFF/DA/RA	4	(000100)
TFE	8	(001000)
LTA	12	(001100)
BIRQ	16	(010000)
TO	20	(010100)

Example

Simulate the Transmitter FIFO Empty interrupt.

```
TRAP   8, 1    ;TFE interrupt simulation
```

Instruction Format



T-states

2

Bus Timing

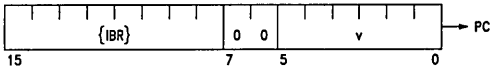
Figure 6-1

Operation

PC & [GIE] & ALU flags & register bank selections
→ Address Stack

if g′ = 1
then clear [GIE]

Create PC address by concatonating the {IBR} register to the vector number v as shown below:



TL/F/9336-17

6.0 Reference Section (Continued)

XOR Exclusive OR Immediate

Syntax
XOR n, rsd —immediate, limited register

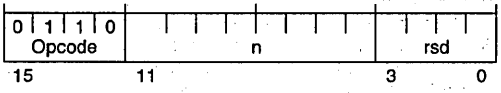
Affected Flags
N, Z

Description
Logically exclusive ORs the immediate value n to the register rsd and places the result back into the register rsd. Note that only the active registers R0–R15 may be specified for rsd. The value of n is 8 bits wide.

Example
Encode/decode a data byte in register 15.

```
XOR  code_pattern, R15     ;encode/decode
```

Instruction Format



T-states
2

Bus Timing
Figure 6-1

Operation
rsd XOR n → rsd

XORA Exclusive OR with Accumulator

Syntax
XORA Rs, Rd —register, register
XORA Rs, [mlr] —register, indexed

Affected Flags
N, Z

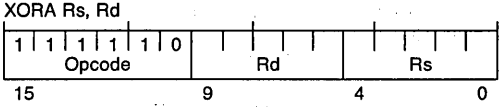
Description
Logically exclusive ORs the source register Rs to the active accumulator and places the result into the destination specified. The destination may be either a register, Rd, or data memory via an index register mode, [mlr]. Note that register bank selection determines which accumulator is active.

Example
Decode the data byte just received and place it into data memory. This example assumes that the accumulator contains the "key" and that the index register IY points to the location where the information should be stored.

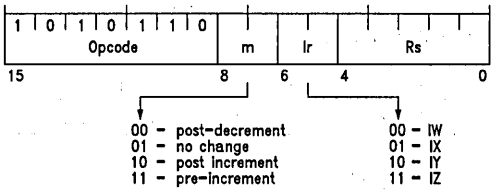
```
EXX    1,1          ;select alternate banks
XORA   RTR, [IY+]   ;decode received byte and
                    ; save it
```

Instruction Format

XORA Rs, Rd



XORA Rs, [mlr]



TL/F/9336–18

T-states
XORA Rs, Rd —2
XORA Rs, [mlr] —3

Bus Timing
XORA Rs, Rd —*Figure 6-1*
XORA Rs, [mlr] —*Figure 6-7*

Operation
XORA Rs, Rd
Rs XOR accumulator → Rd

XORA Rs, [mlr]
Rs XOR accumulator → data memory

6.0 Reference Section (Continued)

TABLE 6-2. Instructions Versus T-states, Affected Flags, and Bus Timing

Instruction		T-states	Affected Flags	Timing Figure
ADCA	Rs, Rd	2	N,Z,C,V	6-1
ADCA	Rs, [mlr]	3	N,Z,C,V	6-7
ADD	n, rsd	2	N,Z,C,V	6-1
ADDA	Rs, Rd	2	N,Z,C,V	6-1
ADDA	Rs, [mlr]	3	N,Z,C,V	6-7
AND	n, rsd	2	N,Z	6-1
ANDA	Rs, Rd	2	N,Z	6-1
ANDA	Rs, [mlr]	3	N,Z	6-7
BIT	rs, n	2	N,Z	6-1
CALL	n	3		6-2
CMP	rs, n	2	N,Z,C,V	6-1
CPL	Rsd	2	N,Z	6-1
EXX	ba, bb {,g}	2		6-1
Jcc	n	2 false 3 true		6-1 6-2
JMP	f, s, n	2 false 3 true		6-1 6-2
JMP	n	3		6-2
JMP	Rs	4		6-4
JRMK	Rs, b, m	4		6-4
LCALL	nn	(2+2)		6-3
LCALL	Rs, p, s, nn	(2+2)		6-3
LJMP	nn	(2+2)		6-3
LJMP	[lr]	2		6-1
LJMP	Rs, p, s, nn	(2+2)		6-3
MOVE	n, rd	2		6-1
MOVE	n, [lr]	3		6-7
MOVE	Rs, Rd	2		6-1
MOVE	Rs, [mlr]	3		6-7
MOVE	Rs, [lr + A]	3		6-7
MOVE	rs, [lZ + n]	3		6-7
MOVE	[mlr], Rd	3 [4TR] = 0 4 [4TR] = 1		6-5 6-6
MOVE	[lr + A], Rd	3 [4TR] = 0 4 [4TR] = 1		6-5 6-6
MOVE	[lZ + n], rd	3 [4TR] = 0 4 [4TR] = 1		6-5 6-6
OR	n, rsd	2	N,Z	6-1
ORA	Rs, Rd	2	N,Z	6-7
ORA	Rs, [mlr]	3	N,Z	6-7
Rcc	{g{,rf}}	2 false 3 true	 N,Z,C,V*	6-1 6-2
RET	{g{,rf}}	2	N,Z,C,V*	6-1
RETF	f, s {,{g} {,rf}}	2 false 3 true	 N,Z,C,V*	6-1 6-2
ROT	Rsd, b	2	N,Z,C	6-1
SBCA	Rs, Rd	2	N,Z,C,V	6-1
SBCA	Rs, [mlr]	3	N,Z,C,V	6-7
SHL	Rsd, b	2	N,Z,C	6-1
SHR	Rsd, b	2	N,Z,C	6-1
SUB	n, rsd	2	N,Z,C,V	6-1
SUBA	Rs, Rd	2	N,Z,C,V	6-1
SUBA	Rs, [mlr]	3	N,Z,C,V	6-7
TRAP	v {,g'}	2		6-1
XOR	n, rsd	2	N,Z	6-1
XORA	Rs, Rd	2	N,Z	6-1
XORA	Rs, [mlr]	3	N,Z	6-7

***Note:** If rf = 1 then N, Z, C, and V are affected.

6.0 Reference Section (Continued)

TABLE 6-3. Instruction Opcodes

Hex	Opcode	Instruction
0000–0FFF	0 0 0 0 Opcode (15) | n (11) | rsd (3–0)	ADD n, rsd
1000–1FFF	0 0 0 1 Opcode (15) | n (11) | rs (3–0)	MOVE rs, [IZ + n]
2000–2FFF	0 0 1 0 Opcode (15) | n (11) | rsd (3–0)	SUB n, rsd
3000–3FFF	0 0 1 1 Opcode (15) | n (11) | rs (3–0)	CMP rs, n
4000–4FFF	0 1 0 0 Opcode (15) | n (11) | rsd (3–0)	AND n, rsd
5000–5FFF	0 1 0 1 Opcode (15) | n (11) | rsd (3–0)	OR n, rsd
6000–6FFF	0 1 1 0 Opcode (15) | n (11) | rsd (3–0)	XOR n, rsd
7000–7FFF	0 1 1 1 Opcode (15) | n (11) | rs (3–0)	BIT rs, n
8000–87FF	1 0 0 0 0 Opcode (15) | m (10) | b (7) | Rs (4–0)	JRMK Rs, b, m

KEY

mlr

00	Ir−
01	Ir
10	Ir+
11	+Ir

Ir

00	IW
01	IX
10	IY
11	IZ

g

00	NCHG
01	RI
10	EI
11	DI

g′

0	NCHG
1	DI

ba/bb

0	MAIN
1	ALT

f

000	[Z]
001	[C]
010	[V]
011	[N]
100	[RA]
101	[RE]
110	[DAV]
111	[TFF]

6.0 Reference Section (Continued)

TABLE 6-3. Instruction Opcodes (Continued)

Hex	Opcode	Instruction
8800–8BFF	1 0 0 0 1 1 0 Opcode (15–9) | n[7–5] | Ir (6–4) | n[4–0] (0)	MOVE n, [Ir]
8C00–8DFF	1 0 0 0 1 1 1 0 Opcode (15–8) | s (7) | p | Rs (4–0)	LJMP Rs, p, s, nn
0000–FFFF	nn (15–0)	
8E00–8FFF	1 0 0 0 1 1 1 1 Opcode (15–8) | s (7) | p | Rs (4–0)	LCALL Rs, p, s, nn
0000–FFFF	nn (15–0)	
9000–9FFF	1 0 0 1 Opcode (15–11) | n | rd (3–0)	MOVE [IZ+n], rd
A000–A1FF	1 0 1 0 0 0 0 Opcode (15–8) | m | Ir (6) | Rs (4–0)	ADDA Rs, [mIr]
A200–A3FF	1 0 1 0 0 0 1 Opcode (15–8) | m | Ir (6) | Rs (4–0)	ADCA Rs, [mIr]
A400–A5FF	1 0 1 0 0 1 0 Opcode (15–8) | m | Ir (6) | Rs (4–0)	SUBA Rs, [mIr]

KEY

mIr

00	Ir−
01	Ir
10	Ir+
11	+Ir

Ir

00	IW
01	IX
10	IY
11	IZ

g

00	NCHG
01	RI
10	EI
11	DI

g′

0	NCHG
1	DI

ba/bb

0	MAIN
1	ALT

f

000	[Z]
001	[C]
010	[V]
011	[N]
100	[RA]
101	[RE]
110	[DAV]
111	[TFF]

6.0 Reference Section (Continued)

TABLE 6-3. Instruction Opcodes (Continued)

Hex	Opcode	Instruction
A600–A7FF	1 0 1 0 0 1 1 Opcode | m | Ir | Rs (15 8 6 4 0)	SBCA Rs, [mIr]
A800–A9FF	1 0 1 0 1 0 0 Opcode | m | Ir | Rs (15 8 6 4 0)	ANDA Rs, [mIr]
AA00–ABFF	1 0 1 0 1 0 1 Opcode | m | Ir | Rs (15 8 6 4 0)	ORA Rs, [mIr]
AC00–ADFF	1 0 1 0 1 1 0 Opcode | m | Ir | Rs (15 8 6 4 0)	XORA Rs, [mIr]
AE00–AE1F	1 0 1 0 1 1 1 0 0 0 0 Opcode | Rs (15 4 0)	CPL Rsd
AE80–AEF8	1 0 1 0 1 1 1 0 1 Opcode | g | ba | bb | 0 0 0 (15 6 4 3 2 0)	EXX ba, bb {,g}
AF00–AF7F	1 0 1 0 1 1 1 1 0 Opcode | g | rf | s | f (15 6 4 3 2 0)	RETF f,s{,{g}{,rf}} Rcc {g{,rf}}
AF80–AFF0	1 0 1 0 1 1 1 1 1 1 Opcode | g | rf | 0 0 0 0 (15 6 4 3 0)	RET {g{,rf}}
B000–BFFF	1 0 1 1 Opcode | n | rd (15 11 3 0)	MOVE n, rd

KEY

mIr

00	Ir−
01	Ir
10	Ir+
11	+Ir

Ir

00	IW
01	IX
10	IY
11	IZ

g

00	NCHG
01	RI
10	EI
11	DI

g′

0	NCHG
1	DI

ba/bb

0	MAIN
1	ALT

f

000	[Z]
001	[C]
010	[V]
011	[N]
100	[RA]
101	[RE]
110	[DAV]
111	[TFF]

6.0 Reference Section (Continued)

TABLE 6-3. Instruction Opcodes (Continued)

Hex	Opcode	Instruction
C000–C1FF	1 1 0 0 0 0 0 0 | Opcode | m | Ir | Rd (15 … 8 … 6 … 4 … 0)	MOVE [mIr], Rd
C200–C3FF	1 1 0 0 0 0 0 1 | Opcode | m | Ir | Rs (15 … 8 … 6 … 4 … 0)	MOVE Rs, [mIr]
C400–C47F	1 1 0 0 0 1 0 0 0 | Opcode | Ir | Rd (15 … 6 … 4 … 0)	MOVE [Ir+A], Rd
C480–C4FF	1 1 0 0 0 1 0 0 1 | Opcode | Ir | Rs (15 … 6 … 4 … 0)	MOVE Rs, [Ir+A]
C800–C8FF	1 1 0 0 1 0 0 0 | Opcode | b | Rsd (15 … 7 … 4 … 0)	SHR Rsd, b
C900–C9FF	1 1 0 0 1 0 0 1 | Opcode | (8-b) | Rsd (15 … 7 … 4 … 0)	SHL Rsd, b
CA00–CAFF	1 1 0 0 1 0 1 0 | Opcode | b | Rsd (15 … 7 … 4 … 0)	ROT Rsd, b
CB00–CBFF	1 1 0 0 1 0 1 1 | Opcode | n (15 … 7 … 0)	JMP n
CC00–CCFF	1 1 0 0 1 1 0 0 | Opcode | n (15 … 7 … 0)	CALL n

KEY

mIr

00	Ir−
01	Ir
10	Ir+
11	+Ir

Ir

00	IW
01	IX
10	IY
11	IZ

g

00	NCHG
01	RI
10	EI
11	DI

g′

0	NCHG
1	DI

ba/bb

0	MAIN
1	ALT

f

000	[Z]
001	[C]
010	[V]
011	[N]
100	[RA]
101	[RE]
110	[DAV]
111	[TFF]

6.0 Reference Section (Continued)

TABLE 6-3. Instruction Opcodes (Continued)

Hex	Opcode	Instruction
CD00–CD60	1 1 0 0 1 1 0 1 0 (Opcode) | Ir | 0 0 0 0 0 — 15 … 6 … 4 … 0	LJMP [Ir]
CD80–CD9F	1 1 0 0 1 1 0 1 1 0 0 (Opcode) | Rs — 15 … 4 … 0	JMP Rs
CE00 0000–FFFF	1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 (Opcode) — 15 … 0 nn — 15 … 0	LJMP nn
CE80 0000–FFFF	1 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 (Opcode) — 15 … 0 nn — 15 … 0	LCALL nn
CF80–CFFF	1 1 0 0 1 1 1 1 1 1 (Opcode) | g′ | v — 15 … 6 … 5 … 0	TRAP v{,g′}
D000–DFFF	1 1 0 1 (Opcode) | s | f | n — 15 … 11 … 10 … 7 … 0	JMP f, s, n Jcc n

KEY

mlr

00	Ir−
01	Ir
10	Ir+
11	+Ir

Ir

00	IW
01	IX
10	IY
11	IZ

g

00	NCHG
01	RI
10	EI
11	DI

g′

0	NCHG
1	DI

ba/bb

0	MAIN
1	ALT

f

000	[Z]
001	[C]
010	[V]
011	[N]
100	[RA]
101	[RE]
110	[DAV]
111	[TFF]

6.0 Reference Section (Continued)

TABLE 6-3. Instruction Opcodes (Continued)

Hex	Opcode	Instruction
E000–E3FF	1 1 1 0 0 0 Opcode (15–9) | Rd (9–4) | Rs (4–0)	ADDA Rs, Rd
E400–E7FF	1 1 1 0 0 1 Opcode (15–9) | Rd (9–4) | Rs (4–0)	ADCA Rs, Rd
E800–EBFF	1 1 1 0 1 0 Opcode (15–9) | Rd (9–4) | Rs (4–0)	SUBA Rs, Rd
EC00–EFFF	1 1 1 0 1 1 Opcode (15–9) | Rd (9–4) | Rs (4–0)	SBCA Rs, Rd
F000–F3FF	1 1 1 1 0 0 Opcode (15–9) | Rd (9–4) | Rs (4–0)	ANDA Rs, Rd
F400–F7FF	1 1 1 1 0 1 Opcode (15–9) | Rd (9–4) | Rs (4–0)	ORA Rs, Rd
F800–FBFF	1 1 1 1 1 0 Opcode (15–9) | Rd (9–4) | Rs (4–0)	XORA Rs, Rd
FC00–FFFF	1 1 1 1 1 1 Opcode (15–9) | Rd (9–4) | Rs (4–0)	MOVE Rs, Rd

KEY

mIr

00	Ir−
01	Ir
10	Ir+
11	+Ir

Ir

00	IW
01	IX
10	IY
11	IZ

g

00	NCHG
01	RI
10	EI
11	DI

g′

0	NCHG
1	DI

ba/bb

0	MAIN
1	ALT

f

000	[Z]
001	[C]
010	[V]
011	[N]
100	[RA]
101	[RE]
110	[DAV]
111	[TFF]

6.0 Reference Section (Continued)

6.2 REGISTER SET REFERENCE

The register set reference contains detailed information on the bit definitions of all special function registers that are addressable in the CPU. This reference section presents the information in three forms: a bit index, a register description and bit definition tables. The bit index is an alphabetical listing of all status/control bits in the CPU-addressable function registers, with a brief summary of the function. The register description is a list of all CPU-addressable special function registers in alphabetical order. The bit definition tables describe the location and function of all control and status bits in the various CPU-addressable special function registers. These tables are arranged by function.

6.2.1 Bit Index

An alphabetical listing of all status/control bits in the CPU-addressable special function registers, with a brief summary of function. Detailed definitions are provided in Section 6.2.3, Bit Definition Tables.

Bit	Name	Location	Function
4TR	Four T-State Read	ACR [3]	Timing Control
ACK	poll/**ACK**nowledge	NCF [1]	Receiver Status
ASP3–0	**A**ddress **S**tack **P**ointer	ISP [7–4]	Stacks
AT7–0	**A**uxilliary **T**ransceiver control	ATR [7–0]	Receiver Control
ATA	**A**dvance **T**ransmitter **A**ctive	TCR [4]	Transmitter Control
BIC	**B**i-directional **I**nterrupt **C**ontrol	ACR [4]	Interrupt Control
BIRQ	**B**i-directional **I**nterrupt **R**e**Q**uest	CCR [4]	Interrupt Control
C	**C**arry	CCR [1]	Arithmetic Flag
CCS	**C**PU **C**lock **S**elect	DCR [7]	Timing Control
COD	**C**lock **O**ut **D**isable	ACR [2]	Timing Control
DAV	**D**ata **AV**ailable	TSR [3]	Receiver Status
DEME	**D**ata **E**rror or **M**essage **E**nd	NCF [3]	Receiver Status
DS7–0	**D**ata **S**tack	DS [7–0]	Stacks
DSP3–0	**D**ata **S**tack **P**ointer	ISP [3–0]	Stacks
DW2–0	**D**ata memory **W**ait-state select	DCR [2–0]	Timing Control
FB7–0	**F**ill **B**its	FBR [7–0]	Transmitter Control
GIE	**G**lobal **I**nterrupt **E**nable	ACR [0]	Interrupt Control
IES	**I**nvalid **E**nding **S**equence	ECR [2]	Receiver Error Code
IM4–0	**I**nterrupt **M**ask select	ICR [4–0]	Interrupt Control
IV15–8	**I**nterrupt **V**ector	IBR [7–0]	Interrupt Control
IW1,0	**I**nstruction memory **W**ait-state select	DCR [4,3]	Timing Control
LA	**L**ine **A**ctive	NCF [5]	Receiver Status
LMBT	**L**oss of **M**id **B**it **T**ransition	ECR [1]	Receiver Error Code
LOR	**L**ock **O**ut **R**emote	ACR [1]	Remote Interface
LOOP	internal **LOOP**-back	TMR [6]	Transceiver Control
LTA	**L**ine **T**urn **A**round	NCF [4]	Receiver Status
N	**N**egative	CCR [3]	Arithmetic Flag
OVF	receiver **OV**er**F**low	ECR [4]	Receiver Error Code
OWP	**O**dd **W**ord **P**arity	TCR [3]	Transmitter Control
PAR	**PAR**ity error	ECR [3]	Receiver Error Code
POLL	**POLL**	NCF [0]	Receiver Status
PS2–0	**P**rotocol **S**elect	TMR [2–0]	Transceiver Control
RA	**R**eceiver **A**ctive	TSR [4]	Receiver Status
RAR	**R**eceived **A**uto-**R**esponse	NCF [2]	Receiver Status
RDIS	**R**eceiver **DIS**abled while active	ECR [0]	Receiver Error Code
RE	**R**eceiver **E**rror	TSR [5]	Receiver Status
RF10–8	**R**eceive **F**IFO	TSR [2–0]	Receiver Control
RFF	**R**eceive **F**IFO **F**ull	NCF [6]	Receiver Status
RIN	**R**eceiver **IN**vert	TMR [4]	Receiver Control
RIS1,0	**R**eceiver **I**nterrupt **S**elect	ICR [7,6]	Interrupt Control
RLQ	**R**eceive **L**ine **Q**uiesce	TCR [7]	Receiver Control
RPEN	**R**e**P**eat **EN**able	TMR [5]	Receiver Control
RR	**R**emote **R**ead	CCR [6]	Remote Interface
RTF7–0	**R**eceive/**T**ransmit **F**IFO	RTR [7–0]	Transceiver Control
RW	**R**emote **W**rite	CCR [5]	Remote Interface
SEC	**S**elect **E**rror **C**odes	TCR [6]	Receiver Control
SLR	**S**elect **L**ine **R**eceiver	TCR [5]	Receiver Control
TA	**T**ransmitter **A**ctive	TSR [6]	Transmitter Status
TCS1,0	**T**ransceiver **C**lock **S**elect	DCR [6,5]	Transceiver Control
TF10–8	**T**ransmit **F**IFO	TCR [2–0]	Transmitter Control

6.0 Reference Section (Continued)

6.2.1 Bit Index (Continued)

An alphabetical listing of all status/control bits in the CPU-addressable special function registers, with a brief summary of function. Detailed definitions are provided in Section 6.2.3, Bit Definition Tables.

Bit	Name	Location	Function
TFE	**T**ransmit **F**IFO **E**mpty	NCF [7]	Transmitter Status
TFF	**T**ransmit **F**IFO **F**ull	TSR [7]	Transmitter Status
TIN	**T**ransmitter **IN**vert	TMR [3]	Transmitter Control
TLD	**T**imer **LoaD**	ACR [6]	Timer
TM7–0	**TiMe**r	TRL [7–0]	Timer
TM15–8	**TiMe**r	TRH [7–0]	Timer
TMC	**TiMe**r **C**lock select	ACR [5]	Timer
TO	**T**ime **O**ut flag	CCR [7]	Timer
TRES	**T**ransceiver **RES**et	TMR [7]	Transceiver Control
TST	**T**imer **StarT**	ACR [7]	Timer
V	**oV**erflow	CCR [2]	Arithmetic Flag
Z	**Z**ero	CCR [0]	Arithmetic Flag

6.2.2 Register Description

A list of all CPU-addressable special function registers, in alphabetical order.

The Remote Interface Configuration register {RIC}, which is addressable only by the remote system, is not included. See Section 6.3, Remote Interface Reference for details of the function of this register.

Each register is listed together with its address, the type of access available, and a functional description of each bit. Further details on each bit can be found in Section 6.2.3, Bit Definition Tables.

ACR AUXILIARY CONTROL REGISTER
[Main R3; read/write]

7	6	5	4	3	2	1	0
TST	TLD	TMC	BIC	rsv	COD	LOR	GIE

rsv . . . state is undefined at all times.

TST — **Timer StarT . . .** When high, the timer is enabled and will count down from it's current value.

When low, timer is disabled. Timer is stopped by writing a 0 to [TST].

TLD — **Timer LoaD . . .** When high, generates timer load pulse. Cleared when load complete.

TMC — **TiMer Clock select . . .** Selects timer clock frequency. Should not be written when [TST] is high. Can be written at same time as [TST] and [TLD].

TMC	Timer Clock
0	(CPU-CLK)/16
1	(CPU-CLK)/2

BIC — **Bi-directional Interrupt Control . . .** Controls direction of BIRQ.

BIC	BIRQ
0	Input
1	Output

COD — **Clock Out Disable . . .** When high, CLK-OUT output is at TRI-STATE.

LOR — **Lock Out Remote . . .** When high, a remote system is prevented from accessing the BCP.

GIE — **Global Interrupt Enable . . .** When low, disables all maskable interrupts. When high, works with [IM4–0] to enable maskable interrupts.

4TR — **4 T-state Read . . .** When high, $\overline{\text{READ}}$ strobe timing is changed to allow more time between the TRI-STATE of the AD lines by the BCP and the falling of the $\overline{\text{READ}}$ strobe. All data memory reads take four T-states when this bit is set. See Section 2.2.2 for more information.

6.0 Reference Section (Continued)

ATR AUXILIARY TRANSCEIVER REGISTER
[Alternate R2; read/write]

7	6	5	4	3	2	1	0
AT7	AT6	AT5	AT4	AT3	AT2	AT1	AT0

AT7–0 — **Auxiliary Transceiver** ... In 5250 protocol modes, bits 2–0 define the receive station address, and bits 7–3 control the amount of time TX-ACT stays asserted after the last fill bit.

In 8-bit protocol modes, bits 7–0 define the receive station address.

For further information, see Section 3.0 Transceiver.

ATR 7–3	TX-ACT Hold Time (μs) (if TCLK = 8 MHz)
00000	0
00001	0.5
00010	1.0
00011	1.5
↓	↓
11111	15.5

CCR CONDITION CODE REGISTER
[Main R0; bits 0–3, 5–7 read/write, bit 4 read only]

7	6	5	4	3	2	1	0
TO	RR	RW	BIRQ	N	V	C	Z

TO — **Time Out flag** ... Set high when timer counts to zero. Cleared by writing a 1 to this location or by stopping timer (by writing a 0 to [TST]).

RR — **Remote Read** ... Set on the trailing edge of a $\overline{\text{REM-RD}}$ pulse, if $\overline{\text{RAE}}$ is asserted and {RIC} is pointing to Data Memory. Cleared by writing a 1 to this location.

RW — **Remote Write** ... Set on the trailing edge of a $\overline{\text{REM-WR}}$ pulse, if $\overline{\text{RAE}}$ is asserted and {RIC} is pointing to Data Memory. Cleared by writing a 1 to this location.

BIRQ — **Bi-directional Interrupt ReQuest** ... [Read only]. Reflects the logic level of the Bi-directional interrupt pin, $\overline{\text{BIRQ}}$. Updated at the beginning of each instruction cycle.

N — **Negative** ... A high level indicates a negative result generated by an arithmetic, logical or shift instruction.

V — **oVerflow** ... A high level indicates an overflow condition generated by an arithmetic instruction.

C — **Carry** ... A high level indicates a carry or borrow generated by an arithmetic instruction. During a shift/rotate operation the state of the last bit shifted out appears in this location.

Z — **Zero** ... A high level indicates a zero result generated by an arithmetic, logical or shift instruction. Further information: Section 2.2.1 ALU, Section 2.2.3 Interrupts.

6.0 Reference Section (Continued)

DCR DEVICE CONTROL REGISTER
[Alternate R0; read/write]

7	6	5	4	3	2	1	0
CCS	TCS1	TCS0	IW1	IW0	DW2	DW1	DW0

CCS — **CPU Clock Select** ... Selects CPU clock frequency. OCLK represents the frequency of the on-chip oscillator, or the externally applied clock on input X1.

CCS	CPU CLK
0	OCLK
1	OCLK/2

TCS1,0 — **Transceiver Clock Select** ... Selects transceiver clock, TCLK, frequency.

OCLK represents the frequency of the on-chip oscillator, or the externally applied clock on input X1. X-TCLK is the external transceiver clock input.

TCS1,0	TCLK
0 0	OCLK
0 1	OCLK/2
1 0	OCLK/4
1 1	X-TCLK

IW1,0 — **Instruction memory Wait-state select** ... Selects from 0 to 3 wait states for accessing instruction memory.

DW2–0 — **Data memory Wait-state select** ... Selects from 0 to 7 wait states for accessing data memory.

DS DATA STACK
[Main R31; read/write]

7	6	5	4	3	2	1	0
DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0

DS7–0 — **Data Stack** ... Data stack input/output port. Stack is 16 bytes deep. Further information: Section 2.1.1.8 Stack Registers.

rsv ... state is undefined at all times.

6.0 Reference Section (Continued)

ECR ERROR CODE REGISTER
[Alternate R4 with [SEC] high; read only]

7	6	5	4	3	2	1	0
rsv	rsv	rsv	OVF	PAR	IES	LMBT	RDIS

OVF — **Receiver oVerFlow** ... Set when the receiver has processed 3 words and another complete frame is received before the FIFO is read by the CPU. Cleared by reading {ECR} or by asserting [TRES].

PAR — **PARity error** ... Set when bad (odd) overall word parity is detected in any receive frame. Cleared by reading {ECR} or by asserting [TRES].

IES — **Invalid Ending Sequence** ... Set when the "mini-code violation" is not correct during a 3270, 3299, or 8-bit ending sequence. Cleared by reading {ECR} or by asserting [TRES].

LMBT — **Loss of Mid-Bit Transition** ... Set when the expected Manchester Code mid-bit transition does not occur within the allowed window. Cleared by reading {ECR} or by asserting [TRES].

RDIS — **Receiver DISabled while active** ... Set when transmitter is activated while receiver is active, without RPEN being asserted. Cleared by reading {ECR} or by asserting [TRES]. Further information: Section 3.2 Transceiver Functional Description.

FBR FILL-BIT REGISTER
[Alternate R3; read/write]

7	6	5	4	3	2	1	0
FB7	FB6	FB5	FB4	FB3	FB2	FB1	FB0

FB7–0 — **Fill Bits** ... 5250 fill-bit control. Further information: Section 3.0 Transceiver.

6.0 Reference Section (Continued)

IBR INTERRUPT BASE REGISTER
[Alternate R1; read/write]

7	6	5	4	3	2	1	0
IV15	IV14	IV13	IV12	IV11	IV10	IV9	IV8

IV15–8— **Interrupt Vector** . . . High byte of interrupt and trap vectors. Further information: Section 2.2.3, Interrupts.

Interrupt Vector

IBR	0 0	vector address

15 8 5 0

The interrupt vector is obtained by concatenating {IBR} with the vector address:

Interrupt	Vector Address	Priority
$\overline{\text{NMI}}$	011100	—
Receiver	000100	1 high
Transmitter	001000	2 ↑
Line Turn Around	001100	3
Bi-directional	010000	4 ↓
Timer	010100	5 low

ICR INTERRUPT CONTROL REGISTER
[Main R2; read/write]

7	6	5	4	3	2	1	0
RIS1	RIS0	rsv	IM4	IM3	IM2	IM1	IM0

rsv . . . state is undefined at all times

RIS1,0 — **Receiver Interrupt Select** . . . Defines the source of the Receiver Interrupt.

RIS1,0	Interrupt Source
00	RFF + RE
01	DAV + RE
10	(unused)
11	RA

"+" indicates logical "or"

Further information: Section 3.2.3 Transceiver Interrupts.

IM4–0 — **Interrupt Masks** . . . Each bit, when set high, masks an interrupt. IM3 functions as an interrupt mask only if BIRQ is defined as an input. When BIRQ is defined as an output, IM3 controls the state of BIRQ.

IM4–0	Interrupt
00000	No Mask
XXXX1	Receiver
XXX1X	Transmitter
XX1XX	Line Turn-Around
X1XXX	Bi-Directional
1XXXX	Timer

Further information: Section 2.2.3 Interrupts.

6.0 Reference Section (Continued)

ISP INTERNAL STACK POINTER
[Main R30; read/write]

7	6	5	4	3	2	1	0
ASP3	ASP2	ASP1	ASP0	DSP3	DSP2	DSP1	DSP0

ASP3–0 — **Address Stack Pointer** . . . Input/output port of the address stack pointer. Further information: Section 2.1.1.8 Stack Registers.

DSP3–0 — **Data Stack Pointer** . . . Input/output port of the data stack pointer. Further information: Section 2.1.1.8 Stack Registers.

NCF NETWORK COMMAND FLAG REGISTER
[Main R1; read only]

7	6	5	4	3	2	1	0
TFE	RFF	LA	LTA	DEME	RAR	ACK	POLL

TFE — **Transmit FIFO Empty** . . . Set high when the FIFO is empty. Cleared by writing to {RTR}.

RFF — **Receive FIFO Full** . . . Set high when the Receive FIFO contains 3 received words. Cleared by reading to {RTR}.

LA — **Line Active** . . . Indicates activity on the receiver input. Set high on any transition; cleared after detecting no input transitions for 16 TCLK periods.

LTA — **Line Turn Around** . . . Set high when end of message is received. Cleared by writing to {RTR}, writing a "1" to this location, or by asserting [TRES].

DEME — **Data Error or Message End** . . . In 3270 & 3299 modes, asserted when a byte parity error is detected. In 5250 modes, asserted when the [111] station address is decoded and [DAV] is asserted. Cleared by reading {RTR}. Undefined in 8-bit modes and in the first frame of 3299 modes.

RAR — **Received Auto-Response** . . . Set high when a 3270 Auto-Response message is decoded and [DAV] is asserted. Cleared by reading {RTR}. Undefined in 5250 and 8-bit modes and in the first frame of 3299 modes.

ACK — **Poll/ACKnowledge** . . . Set high when a 3270 poll/ack command is decoded and [DAV] is asserted. Cleared by reading {RTR}. Undefined in 5250 and 8-bit modes and in the first frame of 3299 modes.

POLL — **POLL** . . . Set high when a 3270 poll command is decoded and [DAV] is asserted. Cleared by reading {RTR}. Undefined in 5250 and 8-bit modes and in the first frame of 3299 modes. Further information: Section 3.0 Transceiver.

6.0 Reference Section (Continued)

RTR RECEIVE/TRANSMIT REGISTER
[Alternate R4; read/write]

7	6	5	4	3	2	1	0
RTF7	RTF6	RTF5	RTF4	RTF3	RTF2	RTF1	RTF0

RTF7–0 — **Receive Transmit FIFO's** ... Input/output port to the least significant eight bits of receive and transmit FIFO's. [OWP], [TF10–8] and [RTF7–0] are pushed onto the transmit FIFO on moves into {RTR}. [RF10–8] and [RTF7–0] are popped from receiver FIFO on moves out of {RTR}. Further information: Section 3.0 Transceiver.

TCR TRANSCEIVER COMMAND REGISTER
[Alternate R6; read/write]

7	6	5	4	3	2	1	0
RLQ	SEC	SLR	ATA	OWP	TF10	TF9	TF8

RLQ — **Receive Line Quiesce** ... Selects number of line quiesce bits the receiver looks for.

RLQ	Number of Quiesces
0	2
1	3

SEC — **Select Error Codes** ... When high {ECR} is switched into {RTR} location.

SLR — **Select Line Receiver** ... Selects the receiver input source.

SLR	Source
0	DATA-IN
1	On-chip analog line receiver

ATA — **Advance Transmitter Active** ... When high, TX-ACT is advanced one half bit time so that the transmitter can generate 5.5 line quiesce pulses.

OWP — **Odd Word Parity** ... Controls transmitter word parity.

OWP	Word Parity
0	Even
1	Odd

TF10–8 — **Transmit FIFO** ... [OWP], [TF10–8] and [RTF7–0] are pushed onto transmit FIFO on moves into {RTR}.

Further information: Section 3.0 Transceiver.

6.0 Reference Section (Continued)

TMR TRANSCEIVER MODE REGISTER
[Alternate R7; read/write]

7	6	5	4	3	2	1	0
TRES	LOOP	RPEN	RIN	TIN	PS2	PS1	PS0

TRES — **Transceiver RESet** . . . Resets transceiver when high. Transceiver can also be reset by $\overline{\text{RESET}}$, without affecting [TRES].

LOOP — **Internal LOOP-back** . . . When high, TX-ACT is disabled (held at 0) and transmitter serial data is internally directed to the receiver serial data input.

RPEN — **RePeat ENable** . . . When high, the receiver can be active at the same time as the transmitter.

RIN — **Receiver INvert** . . . When high, the receiver serial data is inverted.

TIN — **Transmitter INvert** . . . When high the transmitter serial data outputs are inverted.

PS2–0 — **Protocol Select** . . . Selects protocol for both transmitter and receiver.

PS2–0	Protocol
0 0 0	3270
0 0 1	3299 multiplexer
0 1 0	3299 controller
0 1 1	3299 repeater
1 0 0	5250
1 0 1	5250 promiscuous
1 1 0	8-bit
1 1 1	8-bit promiscuous

Further information: Section 3.0 Transceiver.

TRH TIMER REGISTER — HIGH
[Main R29; read/write]

7	6	5	4	3	2	1	0
TM15	TM14	TM13	TM12	TM11	TM10	TM9	TM8

TM15–8 — **TiMer** . . . Input/output port of high byte of timer. Further information: Section 2.1.1.4 Timer Registers.

6.0 Reference Section (Continued)

TRL TIMER REGISTER—LOW
[Main R28; read/write]

7	6	5	4	3	2	1	0
TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0

TM7–0— **TiMer** . . . Input/output port of low byte of timer. Further information: Section 2.1.1.4 Timer Registers.

TSR TRANSCEIVER STATUS REGISTER
[Alternate R5; read only]

7	6	5	4	3	2	1	0
TFF	TA	RE	RA	DAV	RF10	RF9	RF8

TFF — **Transmit FIFO Full** . . . Set high when the transmit FIFO is full. {RTR} must not be written to when [TFF] is high.

TA — **Transmitter Active** . . . Reflects the state of TX-ACT, indicating that data is being transmitted. Unlike TX-ACT, however, [TA] is not disabled by [LOOP].

RE — **Receiver Error** . . . Set high when a receiver error is detected. Cleared by reading {ECR} or by asserting [TRES].

RA — **Receiver Active** . . . Set high when a valid starting sequence is received. Cleared when either an end of message or an error is detected. In 5250 modes, [RA] is cleared at the same time as [LA].

DAV — **Data AVailable** . . . Set high when valid data is available in {RTR} and {TSR}. Cleared by reading {RTR}, or when an error is detected.

RF10–8— **Receive FIFO** . . . [RF10–8] and [RTF7–0] reflect the state of the top word of the receive FIFO.

Further information: Section 3.0 Transceiver.

6.0 Reference Section (Continued)

6.2.3 Bit Definition Tables

The following tables describe the location and function of all control and status bits in the various BCP addressable special function registers. The Remote Interface Configuration register, {RIC}, which is addressable only by a remote processor is not included.

6.2.3.1 Processor

	Bit	Name	Location	Reset State	Function
Timing/ Control	CCS	CPU Clock Select	DCR [7]	1	Selects CPU clock frequency. CCS: 0 = OCLK; 1 = OCLK/2 Where OCLK is the frequency of the on-chip oscillator, or the externally applied clock on input X1.
	DW2–0	Data memory Wait-state select	DCR [2–0]	111	Selects from 0 to 7 wait states for accessing data memory.
	IW1,0	Instruction memory Wait-state select	DCR [4,3]	11	Selects from 0 to 3 wait states for accessing instruction memory.
	COD	Clock Out Disable	ACR [2]	0	When high, CLK-OUT is at TRI-STATE.
	4TR	4 T-state Read	ACR[3]	0	When high, data memory reads take four T-states.
Remote Interface	LOR*	Lock Out Remote	ACR [1]	0	When high, a remote processor is prevented from accessing the BCP or its memory.
	RR*	Remote Read	CCR [6]	0	Set on the trailing edge of a $\overline{\text{REM-RD}}$ pulse, if $\overline{\text{RAE}}$ is asserted and {RIC} is pointing to Data Memory. Cleared by writing a 1 to [RR].
	RW*	Remote Write	CCR [5]	0	Set on the trailing edge of a $\overline{\text{REM-WR}}$ pulse, if $\overline{\text{RAE}}$ is asserted and {RIC} is pointing to Data Memory. Cleared by writing a 1 to [RW].
Interrupt Control	BIC	Bi-directional Interrupt Control	ACR [4]	0	Controls the direction of $\overline{\text{BIRQ}}$. BIC: 0 = Input; 1 = Output
	BIRQ	Bi-directional Interrupt ReQuest	CCR [4]	X	[Read Only]. Reflects the logic level of the $\overline{\text{BIRQ}}$ input. Updated at the beginning of each instruction cycle.
	GIE	Global Interrupt Enable	ACR [0]	0	When low, disables all maskable interrupts. When high, works with [IM4–0] to enable maskable interrupts.
	IM4–0	Interrupt Mask select	ICR [4–0]	11111	Each bit, when set high, masks an interrupt. 00000 No Mask — XXXX1 Receiver 1 High XXX1X Transmitter 2 ↑ XX1XX Line Turn-Around 3 X1XXX Bi-Directional 4 ↓ 1XXXX Timer 5 Low IM3 functions as an interrupt mask only when $\overline{\text{BIRQ}}$ is defined as an input. When $\overline{\text{BIRQ}}$ is defined as an output, IM3 controls the state of $\overline{\text{BIRQ}}$.

*These bits represent the only visibility and control that the processor has into the operation of the remote interface controller. The Remote Interface Configuration register, {RIC}, accessible only by a remote processor, provides further control functions. See Remote Interface section for more information.

6.0 Reference Section (Continued)

6.2.3 Bit Definition Tables (Continued)

The following tables describe the location and function of all control and status bits in the various BCP addressable special function registers. The Remote Interface Configuration register, {RIC}, which is addressable only by a remote processor is not included.

6.2.3.1 Processor (Continued)

	Bit	Name	Location	Reset State	Function
Interrupt Control (Continued)	IV15–8	Interrupt Vector	IBR [7–0]	0000 0000	High byte of interrupt and trap vectors. The interrupt vector is obtained by concatenating {IBR} with the vector address: **Interrupt** / **Vector Address** $\overline{\text{NMI}}$ — 0 1 1 1 0 0 Receiver — 0 0 0 1 0 0 Transmitter — 0 0 1 0 0 0 Line Turn Around — 0 0 1 1 0 0 Bi-Directional — 0 1 0 0 0 0 Timer — 0 1 0 1 0 0 **Interrupt Vector** IBR (15–8) | 0 0 | vector address (5–0)
	RIS1,0	Receiver Interrupt Select	ICR [7,6]	11	Defines the source of the receiver interrupt. **RIS1,0** / **Interrupt Source** 0 0 — RFF + RE 0 1 — DAV + RE 1 0 — (unused) 1 1 — RA
Address and Data Stacks	ASP3–0	Address Stack Pointer	ISP [7–4]	0000	Address stack pointer. Writing to this location changes the value of the pointer.
	DSP3–0	Data Stack Pointer	ISP [3–0]	0000	Data stack pointer. Writing to this location changes the value of the pointer.
	DS7–0	Data Stack	DS [7–0]	XXXX XXXX	Data Stack Input/Output port. Stack is 16 bytes deep.
Arithmetic Flags	C	Carry	CCR [1]	0	A high level indicates a carry or borrow, generated by an arithmetic instruction. During a shift/rotate operation the state of the last bit shifted out appears in this location.
	N	Negative	CCR [3]	0	A high level indicates a negative result generated by an arithmetic, logical, or shift instruction.
	V	oVerflow	CCR [2]	0	A high level indicates an overflow condition, generated by an arithmetic instruction.
	Z	Zero	CCR [0]	0	A high level indicates a zero result generated by an arithmetic, logical, or shift instruction.

6.0 Reference Section (Continued)

6.2.3. Bit Definition Tables (Continued)

The following tables describe the location and function of all control and status bits in the various BCP addressable special function registers. The Remote Interface Configuration register, {RIC}, which is addressable only by a remote processor is not included.

6.2.3.1 Processor (Continued)

	Bit	Name	Location	Reset State	Function
Timer	TLD	Timer LoaD	ACR [6]	0	Set high to load timer. Cleared automatically when load complete.
	TM15–8	TiMer	TRH [7–0]	XXXX XXXX	Input/output port of high byte of timer.
	TM7–0	TiMer	TRL [7–0]	XXXX XXXX	Input/output port of low byte of timer.
	TMC	Timer Clock select	ACR [5]	0	Selects timer clock frequency. Must not be written when [TST] high. Can be written at same time as [TST] and [TLD]. TMC: 0 = CPU-CLK/16; 1 = CPU-CLK/2
	TO	Time Out flag	CCR [7]	0	Set high when timer counts down to zero. Cleared by writing a 1 to [TO] or by stopping the timer (by writing a 0 to [TST]).
	TST	Timer StarT	ACR [7]	0	When high, timer is enabled and will count down from its current value. Timer is stopped by writing a 0 to this location.

6.2.3.2 Transceiver

Table includes control and status bits only. It does not include definitions of bit fields provided for the formatting (de-formatting) of data frames. For further information see the Transceiver section.

	Bit	Name	Location	Reset State	Function
Transceiver Control	LOOP	internal LOOP-back	TMR [6]	0	When high, TX-ACT is disabled (held at 0) and transmitter serial data is internally directed to the receiver serial data input.
	PS2–0	Protocol Select	TMR [2–0]	000	Selects protocol for both transmitter and receiver. PS2–0 / Protocol: 0 0 0 3270 0 0 1 3299 Multiplexer 0 1 0 3299 Controller 0 1 1 3299 Repeater 1 0 0 5250 1 0 1 5250 Promiscuous 1 1 0 8-bit 1 1 1 8-bit Promiscuous
	RTF7–0	Receive/Transmit FIFOs	RTR [7–0]	XXXX XXXX	Input/output port of the least significant 8 bits of receive and transmit FIFOs. [OWP], [TF10–8] and [RTF7–0] are pushed onto the transmit FIFO on moves to {RTR}. [RF10–8] and [RTF7–0] are popped from receive FIFO on moves from {RTR}.

6.0 Reference Section (Continued)

6.2.3 Bit Definition Tables (Continued)

6.2.3.2 Transceiver (Continued)

Table includes control and status bits only. It does not include definitions of bit fields provided for the formatting (de-formatting) data frames. For further information see the Transceiver section.

	Bit	Name	Location	Reset State	Function
Transceiver Control (Continued)	TCS1,0	**T**ransceiver **C**lock **S**elect	DCR [6,5]	10	Selects transceiver clock, TCLK, source. TCS1,0 / TCLK 00 / OCLK 01 / OCLK/2 10 / OCLK/4 11 / X-TCLK OCLK is the frequency of the on-chip oscillator, or the externally applied clock on input X1. X-TCLK is the external transceiver clock input.
	TRES	**T**ransceiver **RES**et	TMR [7]	0	Resets transceiver when high. Transceiver can also be reset by $\overline{\text{RESET}}$, without affecting [TRES].
Transmitter Control	ATA	**A**dvance **T**ransmitter **A**ctive	TCR [4]	0	When high, TX-ACT is advanced one half bit time so that the transmitter can generate 5.5 line quiesce pulses.
	AT7–3	**A**uxiliary **T**ransceiver control	ATR [7–3]	XXXXX	In 5250 modes. Controls the time TX-ACT is held after the last fill bit. AT7–3 / TX-ACT Hold Time (μs) (If TCLK = 8 MHz) 00000 / 0 00001 / 0.5 00010 / 1 ↓ / ↓ 11111 / 15.5
	FB7–0	**F**ill **B**it select	FBR [7–0]	XXXX XXXX	The value in this register contains the 1's complement of the number of additional 5250 fill bits selected.
	OWP	**O**dd **W**ord **P**arity	TCR [3]	0	Controls transmitter word parity. OWP / Word Parity 0 / Even 1 / Odd
	TF10–8	**T**ransmit **F**IFO	TCR [2–0]	000	[OWP], [TF10–8] and [RTF7–0] are pushed onto the transmit FIFO on moves to {RTR}.
	TIN	**T**ransmitter **IN**vert	TMR [3]	0	When high, the transmitter serial data outputs are inverted.
Receiver Control	AT7–0	**A**uxiliary **T**ransceiver control	ATR [7–0]	XXXX XXXX	In 5250 modes, [AT2–0] contains the station address. In 8-bit modes, [AT7–0] contains the station address.
	RF10–8	**R**eceive **F**IFO	TSR [2–0]	XXX	Reflects the state of the most significant 3 bits in the top location of the receive FIFO.
	RIN	**R**eceiver **IN**vert	TMR [4]	0	When high, the receiver serial data is inverted.
	RLQ	**R**eceive **L**ine **Q**uiesce	TCR [7]	1	Selects number of line quiesce bits the receiver requires before it will indicate receipt of a valid start sequence. RLQ / Number of Line Quiesce Pulses 0 / 2 1 / 3
	RPEN	**RePeat ENable**	TMR [5]	0	When high, the receiver can be active at the same time as the transmitter.
	SEC	**S**elect **E**rror **C**odes	TCR [6]	0	When high, {ECR} is switched into {RTR} location.

6.0 Reference Section (Continued)

6.2.3 Bit Definition Tables (Continued)

6.2.3.2 Transceiver (Continued)

Table includes control and status bits only. It does not include definitions of bit fields provided for the formatting (de-formatting) data frames. For further information see the Transceiver section.

	Bit	Name	Location	Reset State	Function
Receiver Control (Continued)	SLR	Select Line Receiver	TCR [5]	0	Selects the receiver input source. SLR: 0 = DATA-IN; 1 = On-Chip Analog Line Receiver
Transmitter Status	TA	Transmitter Active	TSR [6]	0	Reflects the state of TX-ACT, indicating that data is being transmitted. Is not disabled by [LOOP].
	TFE	Transmit FIFO Empty	NCF [7]	1	Set high when the FIFO is empty. Cleared by writing to {RTR}.
	TFF	Transmit FIFO Full	TSR [7]	0	Set high when the FIFO is full. {RTR} must not be written when [TFF] is high.
Receiver Status	ACK	poll/ ACKnowledge	NCF [1]	0	Set high when a 3270 poll/ack command is decoded and [DAV] is asserted. Cleared by reading {RTR}. Undefined in 5250 and 8-bit modes and in the first frame of 3299 modes.
	DAV	Data AVailable	TSR [3]	0	Set high when valid data is available in {RTR} and {TSR}. Cleared by reading {RTR}, or when an error is detected.
	DEME	Data Error or Message End	NCF [3]	0	In 3270 or 3299 modes, asserted when a byte parity error is detected. In 5250 modes, asserted when the [111] station address is decoded and [DAV] is asserted. Undefined in 8-bit modes and first frame of 3299 modes.
	LA	Line Active	NCF [5]	0	Indicates activity on the receiver input. Set high on any transition; cleared after no input transitions are detected for 16 TCLK periods.
	LTA	Line Turn Around	NCF [4]	0	Set high when an end of message is detected. Cleared by writing to {RTR}, writing a "1" to [LTA] or by asserting [TRES].
	POLL	POLL	NCF [0]	0	Set high when a 3270 Poll command is decoded and [DAV] is asserted. Cleared by reading {RTR}. Undefined in 5250 and 8-bit modes and in the first frame of 3299 modes.
	RA	Receiver Active	TSR [4]	0	Set high when a valid start sequence is received. Cleared when either an end of message or an error is detected.
	RAR	Received Auto-Response	NCF [2]	0	Set high when a 3270 Auto-Response message is decoded and [DAV] is asserted. Cleared by reading {RTR}. Undefined in 5250 and 8-bit modes and in the first frame of 3299 modes.
	RE	Receiver Error	TSR [5]	0	Set high when an error is detected. Cleared by reading {ECR} or by asserting [TRES].
	RFF	Receive FIFO Full	NCF [6]	0	Set high when the receive FIFO contains 3 received words. Cleared by reading {RTR}.

6.0 Reference Section (Continued)

6.2.3 Bit Definition Tables (Continued)

6.2.3.2 Transceiver (Continued)

Table includes control and status bits only. It does not include definitions of bit fields provided for the formatting (de-formatting) data frames. For further information see the Transceiver section.

	Bit	Name	Location	Reset State	Function
Receiver Error Codes	IES	**I**nvalid **E**nding **S**equence	ECR [2]	0	Set when the first mini-code violation is not correct during a 3270, 3299 or 8-bit ending sequence. Cleared by reading {ECR} or asserting [TRES].
	LMBT	**L**oss of **M**id-**B**it **T**ransition	ECR [1]	0	Set when the expected Manchester Code mid-bit transition does not occur within the allowed window. Cleared by reading {ECR} or by asserting [TRES].
	OVF	receiver **OV**erFlow	ECR [4]	0	Set when the receiver has processed 3 words and another complete frame is received before the FIFO is read by the CPU. Cleared by reading {ECR} or asserting [TRES].
	PAR	**PAR**ity error	ECR [3]	0	Set when bad (odd) overall word parity is detected in any receive frame. Cleared by reading {ECR} or asserting [TRES].
	RDIS	**R**eceiver **DIS**abled while active	ECR [0]	0	Set when transmitter is activated by writing to {RTR} while receiver is still active, without [RPEN] first being asserted. Cleared by reading {ECR} or asserting [TRES].

6.3 REMOTE INTERFACE CONFIGURATION REGISTER

This register can be accessed only by the remote system. To do this, CMD and $\overline{RAE}$ must be asserted and the [LOR] bit in the {ACR} register must be low.

7	6	5	4	3	2	1	0	
BIS	SS	FW	LR	LW	STRT	MS1	MS0	RIC

BIS **B**idirectional **I**nterrupt **S**tatus . . . Mirrors the state of IM3 ({ICR} bit 3), enabling the remote system to poll and determine the status of the $\overline{BIRQ}$ I/O. When $\overline{BIRQ}$ is an output, the remote system can change the state of this output by writing a one to BIS. This can be used as an interrupt acknowledge, whenever $\overline{BIRQ}$ is used as a remote interrupt. For complete information on the relationship between BIS, IM3 and $\overline{BIRQ}$, refer to Section 2.2.3 Interrupts.

SS **S**ingle-**S**tep . . . Writing a 1 with STRT low, the BCP will single-step by executing the current instruction and advancing the PC. On power up/reset this bit is low.

FW **F**ast **W**rite . . . When high, with LW low, selects fast write mode for the buffered interface. When low selects slow write mode. On power up/reset this bit is low (LW will also be low, so buffered write mode is selected).

LR **L**atched **R**ead . . . When high selects latched read mode, when low selects buffered read mode. On power up/reset this bit is low. (Buffered read mode is selected.)

LW **L**atched **W**rite . . . When high selects latched write mode, when low selects buffered write mode. On power up/reset this bit is low (FW will also be low, so slow buffered write mode is selected).

STRT **STaRT** . . . The remote system can start and stop the BCP using this bit. On power-up/reset this bit is low (BCP stopped). When set, the BCP begins executing at the current Program Counter address. When cleared, the BCP finishes executing the current instruction, then halts to an idle mode.

In some applications, where there is no remote system, or the remote system is not an intelligent device, it may be desirable to have the BCP power-up/reset running rather than stopped at address 0000H. This can be accomplished by asserting $\overline{REM\text{-}RD}$, $\overline{REM\text{-}WR}$ and $\overline{RESET}$, with $\overline{RAE}$ de-asserted. (Refer to Electrical Specification Section for the timing information needed to start the BCP in stand alone mode.)

MS1,0 **M**emory **S**elect **1,0** . . . These two bits determine what the remote system is accessing in the BCP system, according to the following table:

MS1	MS0	Selected Function
0	0	Data Memory
0	1	Instruction Memory
1	0	Program Counter (Low Byte)
1	1	Program Counter (High Byte)

The BCP must be idle for the remote system to read/write Instruction memory or the Program Counter.

All remote accesses are treated the same (independent of where the access is directed using MS0 and MS1), as defined by the configuration bits LW, LR, FW.

If the remote system and the BCP request data memory access simultaneously, the BCP will win first access. If the locks ([LOR], $\overline{LOCK}$) are not set, the remote system and BCP will alternate access cycles thereafter.

On power-up/reset, MS1,0 points to instruction memory.

Power-up/Reset state of {RIC[7-0]} is |000 000|.

6.0 Reference Section (Continued)

6.4 DEVELOPMENT TOOLS

National Semiconductor provides tools specifically created for the development of products that use the DP8344. These tools consist of the DP8344 BCP Assembler System, the DP8344 BCP Demonstration/Development Kit, and the DP8344 BCP Multi-Protocol Adapter (MPA) Design/Evaluation Kit.

6.4.1 Assembler System

The Assembler System is an MS-DOS compatible program used to translate the DP8344's instruction set into a directly executable machine language. The system contains a macro cross assembler, link editor and librarian. The macro cross assembler provides nested macro definitions and expansions, to automate common instruction sequences, and source file inclusion nested conditional assembly, which allows the assembler to make intelligent decisions concerning instruction sequence based on user directives. The linker allows relocatable object sections to be combined in any desired order. It can also generate a load map which details each section's contribution to the linked module. The librarian allows for the creation of libraries from frequently accessed object modules, which the linker can automatically include to resolve references.

6.4.2 Demonstration/Development Kit

The Demonstration/Development kit is a cost effective development tool that performs functions similar to an in-circuit emulator. The kit, developed by Capstone Technology, Inc., Fremont, California, consists of a DP8344 based development board, a monitor/debugger software package, National Semiconductor's DP8344 video training tapes, and all required documentation. The development board is a full size PC card that contains a 22 square inch area for logic prototype wiring. The monitor/debugger program displays internal register contents and status information. It also provides functions such as execution break points and single stepping.

6.4.3 Multi-Protocol Adapter (MPA) Design/Evaluation Kit

The Multi-Protocol Adapter (MPA) is a PC expansion card that emulates a 3270 or 5250 display terminal and supports industry standard PC emulation software. The MPA comes in a design/evaluation kit that includes the hardware, schematics and PAL equations, and software including all the DP8344 source code. This kit was produced to provide a blueprint for PC emulation products and a cornerstone for all 3270 and 5250 product development using the DP8344. The code was developed in a modular fashion so it can be adapted to any 3270 or 5250 application.

6.4.4 DP8344 BCP Inverse Assembler

The DP8344 BCP Inverse Assembler is a software package for use in an HP 1650A or HP1651A Logic Analyzer, or in an HP16500A Logic Analysis System with an HP 16510A State/Timing Card installed. The inverse Assembler was developed by National Semiconductor to allow disassembly of the DP8344 op-code mnemonics. This allows one to determine the actual execution flow that occurs in the system being developed with the DP8344.

6.5 THIRD PARTY SUPPLIERS

The following section is intended to make the DP8344 Customer aware of products, supplied by companies other than National Semiconductor, that are available for use in developing DP8344 systems. While National Semiconductor has supported these ventures and has become familiar with many of these products, we do not provide technical support, or in any way guarantee the functionality of these products.

6.5.1 Crystal Supplier

The recommended crystal parameters for operation with the DP8344 are given in Section 2.2.4. Any crystal meeting these specifications will work correctly with the DP8344. NEL Frequency Controls, Inc., Burlington, Wisconsin, has developed crystals, the NEL C2570N and NEL C2571N, specifically for the DP8344 which meet these specifications. The C2570N and C2571N are both 18.8696 MHz fundamental mode AT cut quartz crystals. The C2571N has a hold down pin for case ground and a third mechanical tie down. NEL Frequency Controls, Inc. is located at:

NEL Frequency Controls, Inc.
357 Beloit Street
Burlington, Wisconsin 53105
(414) 763-3591

6.5.2 System Development Tools

The DP8344, with its higher level of integration and processing power, has opened the IBM mainframe connectivity market to a wider range of product manufacturers, who until now found the initial cost and time to market prohibitive. This wider base of manufacturers created the opportunity for a more extensive line of development tools that dealt not only with the use of the DP8344 but also with the implementation of the 3270 and 5250 protocols. While National Semiconductor is dedicated to providing the Customer with the proper tools in both areas, we also have aided and encouraged a number of third party suppliers to offer additional development tools. This has further provided an avenue for faster and more reliable product development in this product area. The development tools discussed in this section are controller emulators and line monitors for the IBM 3270/3299 and 5250 protocols.

A controller emulator is a device that emulates an IBM 3x74 cluster controller or a System 3x controller. With the DP8344 both of these controllers can be emulated with the same piece of hardware. The controller emulator allows the designer to issue individual commands or sequences of commands to a peripheral. This is very useful in characterizing existing equipment and testing of products under development. Capstone Technology offers such a product. Their Extended Interactive Controller, part #CT-109, is a single PC expansion card that can emulate both 3270 and 5250 control devices (the 3x74 and System 3X, respectively). Newleaf Technologies, Ltd., Cobham, Surrey, England, and Azure Technology, Inc., Franklin, Mass., also supply products in this area. Newleaf Technology offers the COLT52, a twinax controller emulator, and Azure Technology offers a controller made with their CoaxScope and TwinaxScope line monitors.

A line monitor is a device that monitors all the activity on the coax or twinax cable. The activity includes both the commands from the controller and the responses from the peripheral. These devices typically decode the commands and present them in an easy to read format. The individual transmissions are time stamped to provide the designer with response time information. The line monitors are very useful in characterizing communications traffic and in determining the source of problems during development or in the field. Azure Technology offers both a 3270/3299 (Coax) and 5250 (Twinax) line monitor. Their Coax Scope and Twinax

6.0 Reference Section (Continued)

Scope are single PC expansion cards that can record, decode and display activity on the 3270 coax and 5250 twinax line respectively. These devices also allow the play back of the recorded controller information. Capstone Technology also supplies a line monitor. The CT101C, Network Analysis Monitor (NAM), is a coax line monitor.

These companies can be contacted at the following locations:

Azure Technology, Inc.
38 Pond Street
Franklin, Massachusettes 02038
(508) 520-3800

Capstone Technology
853 Brown Rd., Suite 207
Fremont, California 94539
(415) 438-3500

New Leaf Technology, Ltd.
24A High Street
Cobham
Surrey
KT113EB
ENGLAND
(0932) 66466

For technical assistance in using the DP8344B, contact the BCP Hot Line (817) 468-6676.

TABLE 6-4. DP8344 Application Notes

App Note No.	Title
AN-623	Interfacing Memory to the DP8344B
AN-624	A Combined Coax-Twisted Pair 3270 Line Interface for the DP8344 Biphase Communications Processor
AN-516	Interfacing the DP8344 to Twinax
AN-504	DP8344 BCP Stand-Alone Soft-Load System
AN-499	"Interrupts"-A Powerful Tool of the Biphase Communications Processor
AN-625	JRMK Speeds Command Decoding
AN-627	DP8344 Remote Processor Interfacing
AN-626	DP8344 Timer Application
AN-641	MPA - A Multi-Protocol Terminal Emulation Adapter Using the DP8344
AN-688	The DP8344 BCP Inverse Assembler

6.6 DP8344A AND DP8344B COMPATIBILITY GUIDE

The DP8344B is an enhanced version of the DP8344A, exhibiting improved switching performance and additional functionality. The device has been characterized in a number of applications and found to be a compatible replacement for the DP8344A. Differences between the DP8344A and DP8344B are detailed in this section.

6.6.1 Timing Changes to the CPU

Relative to the DP8344A, the DP8344B incorporates a number of timing changes designed to improve the system interface. These timing changes are improvements in the timing specifications and therefore should allow the DP8344B to drop into existing DP8344A designs without any hardware modifications.

The DP8344A exhibits a small amount of contention between certain bus signals as detailed in the Device Specifications section of this data sheet. The DP8344B interface timing improvements are designed to reduce and/or eliminate this bus contention.

- **70 ns Data Memory**

  At a 20 MHz CPU clock rate, the DP8344B can support 70 ns static RAM for data memory with no wait states. The DP8344A was limited to 55 ns static RAM for data memory with no wait states. (See Section 5.0 Device Specifications.)

- **$\overline{\text{READ}}$**

  The timing of the $\overline{\text{READ}}$ strobe has been improved to reduce bus contention during a data memory access. There is now more time between AD disabled and $\overline{\text{READ}}$ falling as well as one-half T-state between $\overline{\text{READ}}$ rising and AD enabled. In addition, a new 4 T-state read option has been provided to eliminate bus contention. (See Section 5.0 Device Specifications for timing changes, and **4 T-state Read** later in this document for more information on the 4 T-state Read option.)

  The user can therefore choose between a fast read mode (3 T-states) with a small amount of contention and a slower read mode (4 T-states) with no contention.

- **A/AD Bus Timing**

  The timing of the A and AD buses has been changed to eliminate bus contention during remote accesses of data memory. There is now a one-half T-state TRI-STATE zone during the bus transfer from local to remote control and vice versa. (See Section 5.0 Device Specifications.)

- **$\overline{\text{IWR}}$**

  The timing of $\overline{\text{IWR}}$ has been changed such that $\overline{\text{IWR}}$ now falls one T-state earlier. This eliminates bus contention during the start of soft loads. (See Section 5.0 Device Specifications.)

- **IA Bus Softload Timing**

  The auto-increment of the IA bus address during soft loads of instruction memory now occurs one T-state later to maintain in-phase data and thereby eliminate bus contention. (See Seection 5.0 Device Specifications.)

- **$\overline{\text{LCL}}$**

  $\overline{\text{LCL}}$ is now removed when $\overline{\text{REM-RD}}$ is taken high on buffered reads of {RIC}, the program counter, and instruction memory, to eliminate bus contention in this mode. (See Section 5.0 Device Specifications.)

- **RIC**

  The hold time on slow buffered writes to {RIC} and the program counter has been improved. (See Section 5.0 Device Specifications.)

- **"Kick-start"**

  The hold time on $\overline{\text{REM-WR}}$ and $\overline{\text{REM-RD}}$ to $\overline{\text{RESET}}$ to "kick-start" the CPU has been improved. (See Section 5.0 Device Specifications.)

6.6.2 Additional Functionality of the DP8344B

6.6.2.1 4 T-state Read

To eliminate bus contention during memory accesses, a new optional read mode has been created, controlled by

6.0 Reference Section (Continued)

[4TR] in {ACR}. When a one is written to this bit, all subsequent data memory read operations expand to 4 T-states with an extra one-half T-state between the falling edge of ALE and the falling edge of $\overline{\text{READ}}$. This eliminates bus contention on data memory read operations. After a BCP reset, or when a zero is written to this bit, the DP8344B data memory read operations operate in 3 T-states, as in the DP8344A, in which this bit was unused. (See Section 2.2.2 for more information.)

6.6.2.2 A/AD Reset State

After a BCP reset, the index registers and the A and AD buses will be zero. In the DP8344A, their states were undefined after a reset.

6.6.2.3 RIC

Each time instruction memory is selected via {RIC[1,0]} (i.e., {RIC} is set to XXXX XX01 binary), the next read (or write) of instruction memory by a remote processor will always return (or update) the low order 8 bits of the 16 bit instruction location pointed to by the program counter. In the DP8344A, setting {RIC} had no affect on which instruction memory byte would next be fetched and an algorithm had to be developed to determine this. (See Section 4.1.2 for more information.)

6.6.2.4 Transceiver

When the Transceiver is reset, $\overline{\text{DATA-OUT}}$ now goes into a state equal to [TIN] ⊕ [ATA], which eliminates coincident transitions on $\overline{\text{DATA-OUT}}$ and DATA-DLY with TX-ACT. (See Section 3.2 for more information.

6.7 REPORT BUGS

6.7.1 History

The DP8344 Data Sheet Reference, first published 10/29/87 (rev. 3.6), listed a total of 13 bugs. All these bugs were corrected in the DP8344A, released to production April 1989. Subsequent to this date, an additional bug has been reported. This bug is present in all versions of the BCP: DP8344, DP8344A and DP8344B.

For additional information regarding differences in functionality between the DP8344B and DP8344A, see Section 6.6.

6.7.2 LJMP, LCALL Address Decode

The LJMP and LCALL instructions to the address range $Af00_h$ through $AF7F_h$ do not function correctly. Both conditional and unconditional LCALL or LJMP instructions to this address range will not decode as LCALL or LJMP instructions. Instead the address field will be incorrectly decoded as the instruction. Thus a LJMP or LCALL to an instruction in the address range $AF00_h$ through $AF7F_h$ will be decoded as a RETF instruction.

Example: the instruction	`LJMP AF00`
will be decoded as	`AF00`
which is	`RETF 000, 00`

Note that LJMP and LCALL to all other addresses work correctly.

The LJMP or LCALL instruction should therefore not be used to transfer program control to an instruction in the range $AF00_h$ to $AF7F_h$.

6.7.2.1 Suggested Work-around

The simplest work-around is not to place any code necessary for system operation in the affected address range. This can be accomplished by creating a section of "filler" code that will occupy the instruction address range $AF00_h$ to $AF7F_h$. As an example, the "filler" section of code could be as follows:

```
FILLER: .SECT X     ; Start of ""filler'' code section
        .REPEAT 128 ; Repeat the following
                      instruction 128 times
        JMP $       ; Jump to self
        .ENDR       ; End of repeat block
        .END
```

The JMP $ instruction causes an infinite loop at that instruction. Thus one would be able to determine if the program inadvertently entered the "filler" section of code. The repeat 128 instruction causes the section to occupy 128 bytes of instruction memory which is the size of the affected address range.

Next, by using the Linker in the DP8344 BCP Assembler System, one can specify that this "filler" section of code must occupy instruction memory starting at address $AF00_h$ by using the -L option. For example, the following commands can be entered at the DOS command line to invoke the Assembler and Linker (this assumes that the "filler" section is located in the file FILLER.BCP):

```
NBCPASM FILLER.BCP
NLINK -LFILLER=AF00 FILLR.BCO
```

This will prevent any other section of code from occupying the range which the "filler" section of code is located in. Hence, one would not have to be concerned about using labels to specify the address in LJMP and LCALL instruction.

6.8 GLOSSARY

3270—An IBM **communication protocol** originally developed for the 370 class mainframe that implements a star topology using a single **coax** cable per slave device. In this master-slave protocol, all communication is initiated by the **controller** (master) and responses are returned by the terminal or other attached device (slave). The data is transmitted using **biphase encoding** at a bit rate of 2.3587 MHz.

3299—A **communications protocol** that is the **3270** protocol with an eight bit address frame added to the beginning of each controller transmission between the **start sequence** and the first coax word. Currently, IBM only uses three bits of the address field which allows up to eight devices to communicate with the **controller** through a **multiplexer.**

5250—An IBM **communications protocol** originally developed for the Series 3 that became widely used on the System 34/36/38 family of minicomputers and currently the AS/400. It uses a **multidrop** bus topology on **twin-ax** cable. This protocol is a master-slave type. The data is transmitted using **bi-phase encoding** at a bit rate of 1 MHz.

accumulator—The implied source register of one operand for some arithmetic operations. In the **BCP**, R8 in the currently enabled bank acts as the accumulator.

ALU—The Arithmetic Logic Unit, a component of the CPU that performs all arithmetic (addition and subtraction), logical (AND, OR, XOR, compare, bit test, and complement), rotational, and shifting operations.

ALU flags—Bits that indicate the result of certain ALU functions.

6.0 Reference Section (Continued)

banked registers—Two or more sets of CPU registers that occupy the same register space, but only one of which is accessible at a time.

barrel shifter—Dedicated hardware for shifting and rotating.

BCP—An abbreviation for Biphase Communications Processor, the National Semiconductor DP8344.

biphase—In this communications signal encoding technique, the data is divided into discrete bit time intervals denoted by a transition in the center of the bit time. This technique combines the clock and data information into one transmission. In **3270** and **3299** protocols, a **mid-bit** transition from low to high represents a bi-phase 1, and a **mid-bit** transition from high to low represents a bi-phase 0. For the **5250** protocol, the definition of biphase logic levels is reversed. Biphase encoding is also called **Manchester II encoding.**

BIRQ—The Bidirectional Interrupt ReQuest. Without any other notation, BIRQ will refer to the BIRQ interrupt itself. BIRQ with a bar on top of it ($\overline{\text{BIRQ}}$) is used where the pin is referenced. BIRQ in brackets ([BIRQ]) is bit 4 in the {CCR} register.

coax—(1) RG-62A/U 93Ω coaxial cable that is used in **3270** protocol systems. (2) Sometimes, this term is used to refer to the **3270** protocol itself.

code violation—A violation of the **bi-phase** encoding format that is part of the **start sequence**. In **3270, 3299,** and the **general purpose 8-bit mode**, the code violation is 1½ bit times low and then 1½ bit times high. In the **5250** protocol, the signal levels are reversed.

communications protocol—A set of rules which defines the physical, electrical, control, and formatting specifications required to successfully transfer data between two systems.

context switch—Switching between two theoretically independent functions that should not affect each other except under specified circumstances.

controller—The master device that initiates all communication to the slave device and controls the manner in which the slave presents the information. It acts as the interface, both physically and logically, between the slave terminals and printers and a host processor.

CPU-CLK—The clock that the operation of the **BCP**'s CPU is synchronized to. The period of this clock which defines **T-state** boundaries is either that of **OCLK** or one-half of **OCLK** depending on the configuration of the **BCP**. The timer clock is also derived from CPU-CLK.

CUT—Control Unit Terminal. A mode of the **controller** where attached devices have limited intelligence and are perceived to be hardware extensions of the **controller**. The **controller** directs all printer, screen, and keyboard activity.

DFT—Distributed Function Terminal. A **controller** mode that supports multiple logical terminals in the same device. The **controller** communicates in higher level commands via data placed in the buffer. The slave device has a greater amount of intelligence than the **CUT** mode device and is responsible for the terminal operation.

direct coupled—The connection of the **transceiver** to the transmission cable in a manner that does not isolate it from DC voltages. Contrast this with **transformer coupled.**

dual port memory—A memory architecture that allows two different processors to access the same memory range alternately.

ending sequence—A defined sequence of bits signifying the end of a transmission. In **3270** and **3299,** it consists of a **bi-phase** 0 followed by a low to high transition on the bit time boundary and two **mini-code violations.**

FIFO—A section of memory or, as in the case of the **BCP** transceiver, a set of registers that are accessed in a First-In First-Out method. In other words, the first data placed in the FIFO by a write will be the first data removed by a read.

fill bits—Fill bits are **bi-phase** 0's used only in the **5250** protocol. A minimum of three fill bits are required between each frame of a **multi-frame message.** This number may be increased by the controller to approximately 243 per the SetMode command. There are always only three fill bits after the last frame of the transmission.

general purpose 8-bit mode—A generic communications mode similar to **3270** and **5250** frame formatting using 8-bit serial data and **bi-phase** signal encoding. The **BCP** supports both **promiscuous** and **non-promiscuous** modes.

Harvard architecture—A computer architecture where the instruction and data memory are organized into two independent memory banks, each with their own address and data buses.

hold time—The amount of time the line is driven at the end of **5250** transmissions to suppress noise on the cabling system.

ICLK—The clock that identifies the start of each instruction when it rises and indicates when the next instruction address is valid when it falls.

immediate addressing mode—An addressing method where one operand, the data for Move instructions and the address for Jump instructions, is contained in the instruction itself.

immediate-relative addressing mode—An addressing method that adds an unsigned 8-bit immediate number to the index register IZ to form the data memory address of an operand.

indexed addressing mode—An addressing method that uses the contents of an index register as the data memory address for one of the operands in an instruction.

interrupt latency—The time from when an interrupt first occurs until it begins executing at its interrupt vector.

jitter—Timing variations for signals of different harmonic content that move the edges of a transmitted signal in time causing uncertainty in their decoding.

jitter tolerance—The total amount of time an edge of a transmitted bit may move and still have its data bit decoded correctly.

LIFO—A sequence of registers or memory locations that are accessed in a Last-In First-Out method; in other words, the last data written into the LIFO will be the first to be removed by a read. Also known as a **stack.**

limited register set—In the **BCP**, the first 16 register address locations (R0–R11 in both banks and R12–R15) that can be used in all instructions.

6.0 Reference Section (Continued)

line hold—The act of driving the transmission line during **5250** transmissions at the end of a message to allow the receivers to unsync. This insures that the receivers will not see line noise as the start of another frame when the line floats.

line interface—All the circuity between the **BCP** and the communications cable medium.

line reflection—Energy from a transmission that is not absorbed by a load impedance and can cause interference in that signal.

Manchester II encoding—See **bi-phase** encoding.

mask—(1) A mechanism that allows the program to specify whether interrupts will be accepted by the CPU. (2) To disable the accepting of an interrupt by the CPU.

mid-bit—In **bi-phase** encoding, the transition in the center of a bit time.

mini-code violation—A violation of the **bi-phase** encoding format that is part of the **ending sequence** in **3270, 3299,** and the **general purpose 8-bit mode.** The mini-code violation has no **mid-bit** transition being high for the entire bit time. There is no mini-code violation in **5250.**

multidrop—A communication method where all the slave devices are attached to the same cable and respond to **controller** commands and data only when their own address frame precedes the transmitted frame.

multi-frame message—Several bytes of data together in the same uninterrupted message that have only one **start sequence** and one **ending sequence.**

multiplexer—A device that receives **3299** protocol transmissions from a **controller,** strips off the address field, and determines over which of eight ports to transmit the message in **3270** format. The device then directs the response from the terminal back to the **controller.**

non-promiscuous—A receiver mode that only enables a data available interrupt when the address frame of the message matches that previously specified. The **5250** and **general purpose 8-bit modes** of the **BCP** support both **promiscuous** and non-promiscuous modes.

NRZ—Non Return to Zero. A data format that uses a high level to represent a data 1 and a low level to represent a data 0. The signal level does not return to a zero level in each bit time. See also **NRZI.**

NRZI—Non Return to Zero Inverted. A data format similar to **NRZ** but with the signal levels reversed.

OCLK—The external Oscillator CLocK connected to the **BCP.** This frequency, from a crystal or a clock, cannot be changed by the **BCP** itself. **CPU-CLK** is derived from OCLK; in addition, the **transceiver** can be configured so that **TCLK** is derived from OCLK.

parity—A one bit code, usually following data, that makes the total number of 1's in a data word odd or even, including the parity bit itself. It is included as an error checking mechanism.

POLL—A command issued by a **controller** to determine changes in terminal status, such as keyboard activity or keylock.

POLL/ACK (PACK)—A command issued by a **controller** to indicate to the terminal that the controller has recognized the non-zero status response of the terminal to its **POLL,** hence its full name poll/acknowledge.

pop—To remove data from a **stack.**

predistortion—The initial voltage step in a **Manchester encoded** bit used to change frequency components of the signal to limit introducing **jitter.**

promiscuous—A receiver mode that enables a data available interrupt regardless of the contents of the transmission address frame. The **5250** and **general purpose 8-bit** modes of the **BCP** support both promiscuous and **non-promiscuous** modes.

push—To place data onto a **stack.**

quiesce pulse—A **bi-phase** 1 bit that is placed at the beginning of a transmission to charge the cable in preparation for the transmission of data. In addition, the quiesce pulses are used as part of the identifying **start sequence.** Typically, five quiesce pulses are placed there.

register addressing mode—An addressing method that uses only operands contained in registers.

register-relative addressing mode—An instruction addressing mode that adds the unsigned 8-bit value in the current **accumulator** to any one of the index registers forming a data memory address for one of the instruction's operands.

remote access—An access to **dual port memory** by a device other than the **BCP.**

repeater—A device used to extend the communication distance between a **controller** and a slave device by receiving the message and re-transmitting it.

RIAS—The Remote Interface and Arbitration System that allows a remote processor and the **BCP** to share the same memory with arbitration of any conflict while the **BCP** is running. A remote processor may also stop and start the **BCP** as well as read and write the Program Counter.

soft-loadable—A feature of a processor system that allows another processor to provide it with instructions and data.

stack—See **LIFO.**

start sequence—A unique arrangement of bits that begin each transmission to ensure proper frame alignment and synchronization. Each transmission begins with five **bi-phase** encoded 1's **quiesce pulses,** a **code violation,** and the **sync bit** of the first frame.

station address—The identification number of a **5250** terminal or other slave device that will specify which device on a **multidrop** line a message is sent to.

sync bit—A **bi-phase** 1 that is placed as the first bit of a frame.

T-state—The period of **CPU-CLK.**

TCLK—The Transceiver CLocK that runs both the transmitter and receiver at a frequency equal to eight times the required serial data rate. The clock can be obtained from a scaled **OCLK** or from **X-TCLK.**

time-out—An interrupt that occurs when the timer reaches a count of zero.

transceiver—The TRANSmitter used for sending messages and the reCEIVER used for reading messages.

transformer coupled—The isolation of the **transceiver** from the transmission cable through the use of a transformer. Contrast this with **direct coupled.**

trap—A **BCP** instruction that forces a software interrupt.

6.0 Reference Section (Continued)

TT/AR—Transmission Turn-around / Auto Response. An acknowledgement by the terminal or other slave device that a write command has successfully been received or that a **POLL** command status response is all zero.

twin-ax—(1) The shielded pair cable that is used in a **5250** communications systems. (2) Sometimes used to refer to the IBM **5250** communications protocol itself.

unmask—Enable the accepting of an interrupt by the CPU.

wait state—Additional **T-states** that may be added to a memory access to increase the time from address generation to the beginning of either a memory read or write. The **BCP** may add as many as seven data wait states and three instruction wait states.

X-TCLK—The eXternal Transceiver CLocK. An independent clock source that the **BCP transceiver** operation may synchronize to rather than from **OCLK.**

Section 2

Application Notes

Section 2 Contents

MPA-II—A Multi-Protocol Terminal Emulation Adapter Using the DP8344

National Semiconductor
Application Note 641
Thomas Norcross
Paul J. Patchen
Thomas J. Quigley
Tim Short
Debra Worsley
Laura Johnson

Table of Contents

1.0 INTRODUCTION

About This System User Guide

The purpose of this document is to provide a complete description of the Multi-Protocol Adapter II (MPA®-II), a hardware and software design solution for emulating basic 3270 and 5250 terminal emulation products in an IBM® PC environment. This document discusses the system support hardware and complete link level firmware required to achieve 3270/3299 CUT, DFT, and 5250 emulation with the National Semiconductor Biphase Communications Processor, BCP®. The document is divided into the following chapters and appendices:

1.0 Introduction: provides a summary of each chapter and each appendix along with a checklist of items included in the MPA-II Design/Evaluation Kit. This chapter provides an MPA-II product description including a list of the new features in the MPA-II that were not present in the original MPA Evaluation Kit. Finally, a description of the DP8344 Biphase Communications Processor, and National Semiconductor's VLSI Products, is provided.

2.0 Operation: describes the system requirements, installation instructions, and steps for using the MPA-II to achieve 3270/3299 and 5250 emulation.

3.0 Development Environment: describes the environment under which the MPA-II has been developed, the tools used by the design team to characterize the products evaluated, and the tools used to test the MPA-II.

4.0 System Overview: describes the 3270/3299 environment, 5250 environment, and terminal emulation. This chapter also describes the DCA® and IBM emulator system architectures and discusses the MPA-II system organization.

5.0 Hardware Architecture: discusses the MPA-II hardware architecture including a description of the BCP core, PC interface, Front-end interface, and miscellaneous support circuitry.

6.0 Software Architecture: discusses the Kernel, coax task, twinax task, and interrupt structure.

Included in this chapter is an in depth discussion of the IRMA™, IBM and Smart Alec™ interfaces.

7.0 Loader and MPA-II Diagnostics: discusses soft-loading the BCP, configuring the MPA-II interface mode, and the diagnostics provided for testing the MPA-II hardware.

Appendix A. Hardware Reference: provides the complete MPA-II schematic, assembly drawing, board layout and PAL equations.

Appendix B. Timing Analysis: discusses the timing of the MPA-II system.

Appendix C. Filter Equations for the Combined Coax/Twisted Pair Interface: provides the derivation of the filter equations for the combined coax/twisted pair interface.

Appendix D. References: is a list of reference materials and company contacts.

MPA-II Description

The Multi-Protocol Adapter II (MPA-II) is a complete design solution for IBM 3270, 3299, and 5250 connectivity products. The MPA-II system is intended to be a design example for customers to use in developing their own products using the Biphase Communications Processor, BCP. The BCP is a "system on a chip" designed by National Semiconductor to specifically address the IBM connectivity market place. Built on the tradition of the DP8340/41 3270 receiver/transmitter pair, the BCP takes the state of the art in IBM communications a step further. The MPA-II provides the system support hardware and complete link level firmware to achieve 3270/3299 CUT, DFT, and 5250 emulation with the BCP and an appropriate PC emulator. The MPA-II Design/Evaluation Kit does not include the PC emulation software. Thus, the end user must purchase the PC emulation software to bring up a live terminal emulation session using the MPA-II. PC emulation software such as DCA's E78 for MPA-II IRMA mode, one of IBM's PC 3270 emulation programs for MPA-II IBM mode, DCA's EMU for MPA-II ALEC mode, or any of the third party vendors which support either the IRMA, IBM or ALEC emulation card interface modes, including SIMPC MASTER™ by SIMWARE, RELAY Gold® by RELAY Communications, and CrossTalk™ MK.4 by Digital Communications Associates, can be used with the MPA-II.

DP8344B BCP

The DP8344B BCP is a communications processor designed to efficiently process IBM 3270, 3299 and 5250 communications protocols. A general purpose 8-bit protocol is also supported.

The BCP integrates a 20 MHz, 8-bit, Harvard architecture, RISC processor and an intelligent, software-configurable transceiver on the same low power microCMOS chip. The transceiver is capable of operating without significant processor interaction, releasing processor power for other tasks. Fast, flexible interrupt and subroutine capabilities with on-chip stacks make the power readily available.

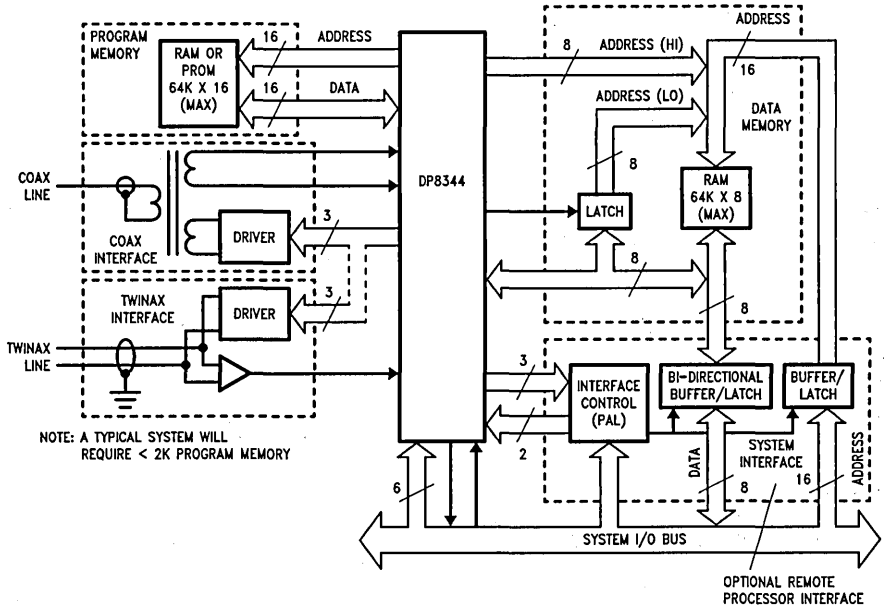
The transceiver is mapped into the processor's register space, communicating with the processor via an asynchronous interface which enables both sections of the chip to run from different clock sources. The transmitter and receiver run at the same basic clock frequency although the receiver extracts a clock from the incoming data stream to ensure timing accuracy.

The BCP is designed to stand alone and is capable of implementing a complete communications interface, using the processor's spare power to control the complete system. Alternatively, the BCP can be interfaced to another processor with an on-chip interface controller arbitrating access to data memory. Access to program memory is also possible, providing the ability to softload BCP code. The MPA-II implements these features.

A simple line interface connects the BCP to the communications line. The receiver includes an on-chip analog comparator suitable for use in a transformer-coupled environment, although a TTL-level serial input is also provided for applications where an external comparator is preferred.

A typical system is shown in *Figure 1-1.* Both coax and twinax line interfaces are shown, as well as an example of the (optional) remote processor interface.

For a detailed discussion on the BCP refer to the DP8344B Biphase Communications Processor data sheet.



TL/F/10488-1

FIGURE 1-1. Block Diagram of Typical BCP System

2.0 OPERATION

System Requirements

THE MPA-II system implements both 3270 and 5250 terminal emulation using the DCA and IBM industry standard interfaces. Note that the MPA-II system emulates the hardware and link-level firmware portion of the DCA and IBM interfaces. This allows the MPA-II system to run with a variety of emulators. For example, the DCA emulator system for the 3270 environment is called IRMA. IRMA consists of a full sized PC board along with its link-level firmware, and the PC emulator software "E78.EXE". The MPA-II system replaces the IRMA PC board and its link-level firmware. Therefore, the MPA-II system, when configured correctly, appears in every way to the emulator, E78, to be the actual IRMA hardware/link-level firmware portion of the DCA emulator system for the 3270 environment. Thus to operate the MPA-II system in a live communication system, a PC emulation program is required; for example DCA's E78.EXE. In DCA interface modes the emulators are: "E78", for the 3270 IRMA system; and "EMU", for the 5250 Smart Alec system. In the IBM interface mode the emulators are "PC3270" for the 3270/3299 CUT environment and "PSCPG" for the 3270/3299 DFT environment. Any emulator compatible with one of the emulators listed above can be used to achieve terminal emulation using the MPA-II system.

The system requirements for using the MPA-II are dependent upon which interface the MPA-II is emulating. In DCA interface modes, a PC interrupt is not used. However, in the IBM interface mode, a PC interrupt is required. The PC interrupt level is selected as follows: IRQ2 is selected with jumper JP6; IRQ3 by jumper JP4; and IRQ4 by jumper JP5. The factory configuration selects the PC interrupt level IRQ2.

To support the IBM interface mode, the MPA-II utilizes an 8k block of dual-port RAM. This RAM must be located somewhere in the PC's memory space. The default location in PC memory is CE000. This location can be relocated by writing the upper 8k byte boundary to I/O location 2D7h or by using the MPA-II Loader program (LD).

The I/O space requirements, for any interface mode, are the total of the I/O space requirements for the MPA-II.

This means that the I/O locations 220h–22Fh and 2D0h–2DFh are required for the MPA-II.

For execution space, the LD requirements are minimal (less than 64k). The amount of free RAM available for a PC emulator depends on the particular emulation package (i.e., E78, EMU, or IBM PC 3270, etc . . .). The MPA-II system does not use any resident software of its own accord.

In summary, the Multi-Protocol Adapter II Design/Evaluation Kit contains the hardware, software and the MPA-II System User Guide and Technical Reference to aid designers in development of peripheral devices and network interfaces based on the DP8344. The following items are not included in the MPA-II system and therefore MUST be provided by the user to use the MPA-II in a live terminal emulation session:

— IBM PC XT/AT or compatible

— PC-DOS version 3.0 or higher

— PC emulation software such as DCA's E78 for MPA-II IRMA interface mode, one of IBM's PC 3270 emulation programs for MPA-II IBM interface mode, DCA's EMU for MPA-II ALEC interface mode, or any of the third party vendors which support either the IRMA, IBM or ALEC emulation card interface modes, including SIMPC MASTER by SIMWARE, RELAY Gold by RELAY Communications, and CrossTalk MK.4 by DCA.

— Link to an IBM 370 class mainframe (for example, through the IBM 3174/3274 controllers) for 3270/3299 connectivity; or a link to a System 3X, or AS/400 for 5250 connectivity.

Requirements for Design Development

To create the software design environment for leveraging off the MPA-II source code, the following software must be purchased:

— National Semiconductor's DP8344 Assembler System, DP8344ASM1.2

— Microsoft's C 5.1 Optimizing Compiler for the IBM PC

— Microsoft's Macro Assembler 5.1 for the IBM PC

The minimum hardware requirements to set up a hardware evaluation and design environment for creating virtually any end product (terminal, printer, protocol converter, multiplexer, gateway, etc.) are an IBM PC/XT, IBM PC/AT or compatible and the MPA-II PC board.

Useful Tools

The tools listed in this section will greatly assist in the design process:

— Azure Technologies Coax Scope (or Twinax Scope) for monitoring and analyzing data transmitted on 3270 Coax Type "A" media (or on IBM System 3X or AS400 Twinax media).

— Capstone Technology CT-104 BCP Demonstration/Development Kit. This kit includes a development board with a 22 square inch logic prototype area and a 3 square inch line interface prototype area. Additionally, the kit supplies a Monitor/Debugger which features a simple operator interface, single step program execution and software break-points.

— CT-106 Enhanced Interactive Coax-A Controller, EICC, (or the CT-103 Interactive Twinax Controller, ITC) by Capstone Technology allows issuing specific 3270 (or 5250) instructions to a Device Under Test in place of the traditional mainframe and 3X74 controller operations (or the System 3X or AS400 controller operations).

— Logic Analyzer (National Semiconductor has an Inverse Assembler for the BCP which requires one of the following Hewlett Packard Logic Analyzer Models: HP1650A, HP1651A or an HP16500A with an HP16510 State/Timing Card).

See Section 3.0, Development Environment for a description of how these tools were used in developing the MPA-II system.

MPA-II Installation

The first step in using the MPA-II is installing the MPA-II circuit board in an IBM PC/XT, PC/AT or compatible. The MPA-II installs in the usual way: please be sure that the power is OFF, that the system unit is unplugged, and that proper grounding techniques are used.

- Remove the cover by following the directions supplied by the manufacturer.
- Remove the end plate from the system unit in the slot desired for the MPA-II.
- Remove the MPA-II from its anti-static bag, and hold it by the edges.
- If the MPA-II will be used for Twinax operation, determine if the MPA-II will operate in pass-through or terminate mode. If it is NOT the terminator, remove jumpers JP2 and JP3. The factory default is terminate.
- Install the MPA-II in an open PC bus slot.
- Replace the screw from the end plate previously removed to hold the MPA-II firmly in place. A good electrical connection here is important as it provides shield ground for the cables.
- Close the system unit and replace all screws, etc ... according to the manufacturers instructions.
- For 3270/3299 operation, install any 3270 coax type "A" port cable to the rear BNC/Twisted Pair connector.
- For 3270/3299 twisted pair operation, solder any 24 AWG unshielded twisted pair cable to the ADC Twisted Pair Plug provided with the MPA-II kit. Then, connect the Twisted Pair plug to the rear BNC/Twisted Pair connector on the MPA-II board. Make sure that the other end of the 24 AWG unshielded twisted pair cable is properly attached to the controller as a twisted pair cable.
- For twinax operation, install the Twinax Adapter cable to the MPA-II by inserting the 9 Pin D-Sub-miniature connector onto the mating connector on the rear panel, and connect the twinax cable(s) to the Tee connector.

Running Emulation—A Quick Start

To use the MPA-II immediately, follow these instructions. First, select a PC/XT, PC/AT, or compatible and make sure that the following I/O addresses, IRQ interrupt, and Memory addresses are unused in that PC:

I/O: 0220–022F and 02D0–02DF
IRQs: IRQ2
Memory: Segment CE00

Next, install the MPA-II hardware into the PC. Then, change the default DOS drive to A:, insert the distribution disk labeled DISK 1 into drive A:, and type at the DOS prompt:

```
SETUP C:
```

where c: is the target hard disk drive. This will install the MPA-II software onto the PC's hard disk. Next, change the default DOS drive to the hard disk and change the default DOS directory to \MPA. Execute the following program at the DOS command prompt to verify correct operation of the MPA-II hardware within the PC:

```
LD -LS
```

If the self test passed then the MPA-II board is operational within this PC. If it fails, check again for I/O, IRQ, or Memory address conflicts as each MPA-II is tested before it is shipped.

Now, install onto the hard disk the PC emulation software of your choice, such as DCA's E78 for MPA-II IRMA mode, one of IBM's PC 3270 emulation programs for MPA-II IBM mode, DCA's EMU for MPA-II ALEC mode, or any of the third party vendors which support either the IRMA, IBM or ALEC emulation card interface modes, such as SIMPC MASTER by SIMWARE, RELAY Gold by RELAY Communications, and CrossTalk MK.4 by DCA. Note that the PC emulation software must be supplied by the end user, it is not included as part of the MPA-II Evaluation Kit.

Finally, load the MPA-II emulation card with the DP8344AV microcode using the Loader and then start the PC emulation program. To use the listed emulator, or equivalent, type at the DOS prompt when in the \MPA directory:

```
LD MPA2 -M = IRMA ; to use the DCA IRMA
                    emulator "E78" or
                    equivalent

LD MPA2 -M = IBM  ; to use the IBM emulator
                    "PC3270" or equivalent

LD MPA2 -M = ALEC ; to use the DCA Smart
                    Alec emulator "EMU" or
                    equivalent
```

Then, change to the PC emulation program directory of the separately purchased and installed PC emulation software (see installation instructions of that PC emulation software for the name of that directory. In this example assume the directory name is \EMULATOR, and then type the name of the PC emulator program:

```
CD \EMULATOR
E78
```

Your emulator should now be operational.

Invoking the Loader program with no arguments will produce a short help screen. A detailed help for the Loader can be accessed using the -h option. Therefore, at the DOS command line enter:

```
LD -H
```

For more information on the Loader program, refer to the Loader documentation in Section 7.0.

3.0 DEVELOPMENT ENVIRONMENT

The environment used for development of the MPA-II consists of a few readily available, relatively inexpensive tools. The hardware was first prototyped with the Capstone Technology CT-104 BCP Demonstration/Development card. The software was developed with the National Semiconductor BCP Assembler. It was tested with Capstone's EICC (Enhanced Integrated Coax Controller), Capstone's ITC (Integral Twinax Controller), and Azure Technologies' Coax and Twinax scope products. Debugging was accomplished with BSID, Capstone's debugger/monitor which we modified for use with the MPA-II software model and the MPADB.EXE debugger included with the MPA-II (see Chapter 6). For particularly difficult interrupt problems a Hewlett Packard model 16500A Logic Analysis System with a State/Timing card installed was used to monitor instruction execution and PC accesses.

The CT-104 board was modified through the wire-wrap area to approximate the hardware design. This wire-wrap card allowed us to get a working version of the hardware design very quickly, since most of the circuitry was already there. In some development projects, it is often faster to go directly to pcbs as a prototype run. This process has advantages in speed when the device is large and complex, but often debugging is quite messy with multi-layer pcbs, not to mention expensive. Since the CT-104 has the major functional blocks already and the wire wrap area is large, the wire-wrap time was minimal, thus allowed us to easily debug the hardware.

A majority of the logic for the DCA and IBM interfaces is implemented in Programmable Array Logic. We used the abel program from DATA I/O to prepare the JEDEC files for programming the devices.

Software development was done on IBM PCs with the National Semiconductor DP8344 Assembler. The assembler allows relocatable code, equate files, macros, and many other "large CPU" features that make using it a pleasure. The modularity of the software design allowed us to use multiple coders and a single "system integrator" who linked the modules and handled system debugging. The assembler adapts well to large projects like this because of its relocation capability. The Microsoft MAKE utility was used to provide the system integrator with a automated way of keeping up with source modules' dependencies and changes. The BRIEF™ text editor from UnderWare™ was used for editing. This editor allowed us to invoke the National Semiconductor DP8344 Assembler from within the editor and to locate and correct bugs quickly. Finally, an ethernet LAN allowed the software development team to share files and update each other quickly and efficiently. These tools are not all necessary, but are common enough to be useful in illustrating a typical environment.

The BCP's sophistication and advanced development tools made the MPA-II development project proceed at a much greater rate than is possible with other comparable solutions.

Characterization of IBM 3270 and 5250 products was performed by using Capstone's EICC/ITC to drive the coax/twinax line and the Azure scopes to monitor the results. In this way we could stimulate the IBM terminal under controlled conditions, testing most every situation, and then stimulate the MPA-II under the same conditions to verify correct functionality.

The debuggers allow a developer to load and run code on the target system, set breakpoints, examine and modify instruction or data memory. Early configurations were accomplished using the standard DOS DEBUG tool, but once the MPA-II Loader program (LD) was operational, configuration and loading was accomplished through it.

The HP logic analyzer was attached to the target system to monitor the instruction accesses and data bus activity on the target card. This information is helpful in finding interrupt problems that the debugger cannot. Using ICLK from the BCP to sample the BCP instruction address and data busses allows one to monitor instruction execution. Symbolic disassembly can be done with the DP8344 BCP Inverse Assembler, which is a software package for use in an HP1650A or HP1651A Logic Analyzer, or in an HP 16500A Logic Analysis System with an HP 16510A State/Timing Card installed. The inverse assembler was developed by National Semiconductor to allow disassembly of the DP8344 op-code mnemonics. The inverse assembler provides the real time sequence of events by displaying on the HP Logic Analyzer's screen the actual execution flow that occurred in the system being developed with the DP8344.

4.0 SYSTEM OVERVIEW

The MPA-II addresses a systems market that is driven by the large installed base of IBM systems throughout the

world. The IBM plug compatible peripheral and terminal emulation markets are growing along with the success of IBM in the overall computer market place. The originally proprietary architecture of the IBM peripherals and the subsequent vague and confusing Product Attachment Information Manuals (PAIs) have kept the attachment technology elusive. The IBM communications system in general is not well understood. The desire of customers and systems vendors to achieve more attachment options, however, is significant.

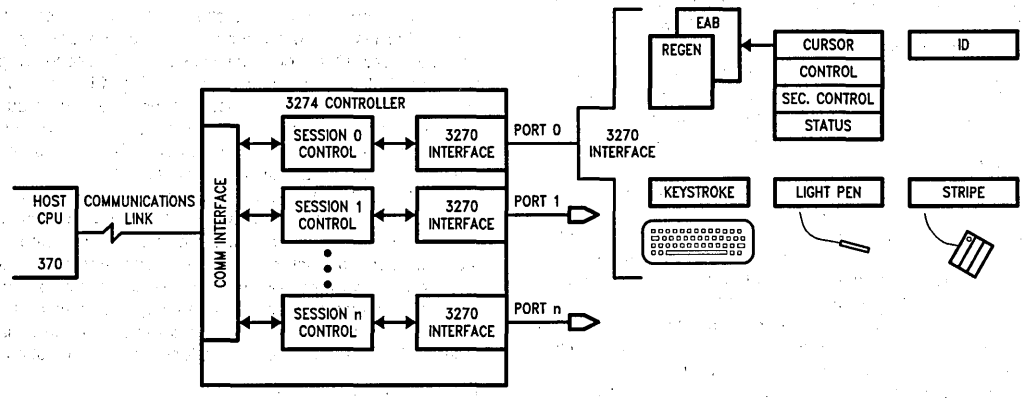
IBM 3270 and 5250 Environments

The study of IBM communications fills many volumes. The intent of this discussion is not to describe it fully, but to highlight the areas of IBM communications that the BCP and MPA-II address. Specifically, these areas are the controller/peripheral links that use the 3270/3299 and 5250 data streams. These links are found in 370 class mainframe networks and the smaller, mid-range systems such as the AS/400 and System/3x lines.

The 3270 communications sub-system was developed for 370 class mainframes as demand for terminal support began to outstrip batch job entry modes. These systems had large scale networking needs and often needed to support thousands of terminals and printers. The original systems were linked together through dedicated telephony lines using Binary Synchronous Communications (BSC) serial protocol. The 5250 communications system was developed originally for the Series 3 and became widely used on the System/34. The System/34 was a small, office environment processor with limited networking and terminal support capabilities. Typical System/34 installations supported up to 16 terminals and printers. The System/36 replaced the System/34 in 1984. Next, IBM introduced the System/38, a mid-range processor that could rival the 4300 series (small 370 class) mainframes in processing power. The System/36 and 38 machines now have greatly enhanced networking facilities, and can support up to 256 local terminals. In 1988 IBM released a new mid-range system line called the Advanced System 400, or AS/400, to replace the aging System/3x line. The Advanced System 400 series is highly modular and combines the best features of the System/36 and System/38 to produce IBM's most popular mid-range system to date. In addition, the AS/400 continues to expand the role and importance of the 5250 data stream, adding it to the definition of IBM's SAA. The 370 class and AS/400 machines have grown closer together through the advent of SNA (Systems Network Architecture). SNA allows both systems to function together in an integrated network.

The 3270 and 5250 communications systems evolved at a time when hardware design constraints were very different than today. Microprocessors and 1 Mb DRAMs were not available. Memory in general was very expensive. Telecommunications channel sharing between multiple peripherals was imperative. Even so, fast screen updates and keystroke handling were necessary. The 3270 and 5250 data stream architectures were developed to address specific design goals within IBM's overall network communications system. The controller sub-system where they were implemented has proved adaptable to new directions in SNA and the migration of processor power out into workstations.

The 3270 and 5250 controller sub-systems split the peripheral support tasks into two sections: screen with keyboard, and host communications interface. *Figure 4-1* shows the 3270 Communications System, 5250 is similar. The controller architectures can be thought of as having integral screen buffers and keyboards for each of their associated terminals with the caveat that screens and keyboards must be accessed through a secondary, high speed serial link. Since the controller views the terminal's screen buffer as its own, the controller does not maintain a copy of the information on that screen. The processing capability of some terminals is severely limited; the early terminals were state machines designed to handle the specific data stream. With the advent of SNA and APPC, (Advanced Peer to Peer Communications) the intelligence in some peripherals has become significant. The data streams have essentially remained the same, with hierarchically structured protocols built upon them. SNA and these higher protocols will be discussed later.



TL/F/10488-2

FIGURE 4-1. 3270 Communications System

Separating the screen buffer and keyboard from the intelligence to handle the terminal addressed several design goals. Since the terminal needs screen memory to regenerate its CRT, the "regen" buffer logically resides in the terminal. The controller need not duplicate expensive memory by maintaining another screen copy. The data stream architectures implemented with high speed serial links between the controller and terminal allow fast keystroke echoing. It also allows fast, single screen updates, giving the appearnace of good system performance. The terminal screen maintenance philosophy developed with these architectures lends itself well to the batch processing mode that traditionally was IBM's strong suit. The terminal system is optimized for single screen presentation with highly structured field oriented screens. Data entry applications common in business computing are well suited to this. Essentially, the architecture places field attributes and rudimentary error checking in the controller, so that most keystrokes can pass from the terminal to the controller and back to the screen very quickly without host CPU intervention. Only when particular keystrokes are sent (AID keys) does the controller read the contents of the screen fields and present the host with the screen data.

3270 Data Stream Architecture

The 3270 communications system, as discussed above, is a single logical function separated into two physical pieces of hardware connected by a protocol implemented on a high speed serial link. The terminal hardware has the screen buffers and keyboard, magnetic slot reader, light pen, etc., (i.e., all the user interface mechanisms). The controller has a communications link to the host CPU or network and the processing power to administrate the terminal functions. Controllers typically support multiple terminals and essentially concentrate the terminal traffic onto the host communications channel. The controller has a secondary communications system that implements the 3270 data stream protocol over coaxial cable at 2.3587 Mb/s. Each peripheral connected to the controller has its own coax port. The coax lengths may be up to 5000 feet. The protocol is controller initiated, poll/response type.

The serial protocol organizes data into discrete groups of 12 bits, called a frame. Biphase (Manchester II) encoding is used to impress the data frames onto the transmission medium. Biphase data have embedded clock information denoted as mid-bit transitions. Frames may be concatenated to form packets of commands and/or data. All transmissions begin with a line quiesce sequence of five biphase one bits followed by a three bit time line violation. The first bit of all frames is called the sync bit and is always a logic one. The sync bit follows the line violation and precedes all successive frames. Each frame includes a parity bit that establishes even parity over the 12-bit frame. Each transmission from the controller elicits a response of data or status from the device. The response time requirements are such that a device must begin its response within 5.5 ms after reception of the controller transmission. Simple reception of a correct packet is acknowledged by the device with a transmission of "TTAR", or transmission turn around/auto response. The controller initiated, poll/response format protocol addresses multiple logical devices inside the peripheral through a three or four bit command modifier. The different logical devices decode the remaining bits as their command sets. Commands to the base or keyboard are decoded as shown in Table 4-1.

TABLE 4-1. 3270 Data Stream Command Set

Command		Value	Description
READ TYPE:	TO BASE—Device Address 0 or 1		
	POLL	01h	Respond with Status
	POLL/ACK	11h	Special Status Acknowledgement Poll
	READ STATUS	0Dh	Respond with Special Status
	READ TERMINAL ID	09h	Respond with Terminal Type
	READ EXTENDED ID	07h	Respond with 4 Byte ID (Optional)
	READ ADDRESS COUNTER HI	05h	Respond with Address Counter High Byte
	READ ADDRESS COUNTER LO	15h	Respond with Address Counter Low Byte
	READ DATA	03h	Respond with Data at Address Counter
	READ MULTIPLE	0Bh	Respond with Up to 4 or 32 Bytes
WRITE TYPE**:	TO BASE—Device Address 0 or 1		
	RESET	02h	POR Device
	LOAD CONTROL REGISTER	0Ah	Load Control Byte
	LOAD SECONDARY CONTROL	1Ah	Load Additional Control Byte
	LOAD MASK	16h	Load Mask Used in Searches, CLEAR
	LOAD ADDRESS COUNTER HI	04h	Load Address Counter High Byte
	LOAD ADDRESS COUNTER LO	14h	Load Address Counter Low Byte
	WRITE DATA	0Ch	Load Regen Buffer with Data
*	CLEAR	06H	Clear Regen Buffer to Nulls
*	SEARCH FORWARD	10h	Search Forward in Buffer until Match
*	SEARCH BACKWARD	12h	Search Back in Buffer until Match
*	INSERT BYTE	0Eh	Insert Byte at Address Counter
	START OPERATION	08h	Begin Execution of Higher Level Command
	DIAGNOSTIC RESET	1Ch	Special DFD Reset

*Denotes foreground task

**All WRITE type commands elicit TTAR upon clean reception.

The 3299 variant on the 3270 data stream uses an additional eight bit address field to address up to 8 more 3270 devices with the same coax cable. Since coax installations are point-to-point between controller and peripheral, cabling costs motivated the introduction of 3299 multiplexer/demultiplexers. Using the extended address field, eight devices can be connected via one coax cable between the controller and the multiplexer. (The 3299 protocol can support up to 32 devices per line if IBM so chooses.)

Basic 3270 terminals have a structure as shown in *Figure 4-2*. The EAB (Extended Attribute Buffer) is a shadow of the regen buffer; each location in the regen has a corresponding location in the EAB. The EAB is a separately addressable device with an address modifier of 7h. The EAB bytes are used to provide extra screen control information. In the 3270 world, the screen and field attributes that the controller uses to format and restrict access to fields on the screen take up space in the screen. The attribute characters appear as blanks and cannot be used for displayable characters at the same time. Since the number of permutations of the 8-bit character byte is limited to 256, the number of attributes is limited by the size of the displayable character set. The EAB provides a method to enhance screen control, with color for instance, without losing character space. The EAB contains both character attributes, that correspond to characters in the regen buffer, and field attributes that correspond to attributes in the regen.

Status developed in the terminal, such as keystrokes or errors, are reported in the poll/response mechanism. A POLL command to the base device with keyboard status pending elicits a keystroke response in 5.5 μs. The controller then sends a POLL/ACK command to acknowledge the keyboard status and thus clear it. The terminal then responds with "clean" status, i.e., TTAR. Controllers poll frequently to assure that status updates are quick. Outstanding status is reported in the poll response and in some cases is handled directly by command modifiers in the POLL command. Keystrokes are the most command status and hence are acknowledged by the POLL/ACK command. Status reported in the status register can be read and acknowledged independently of the polling mechanism, if desired.

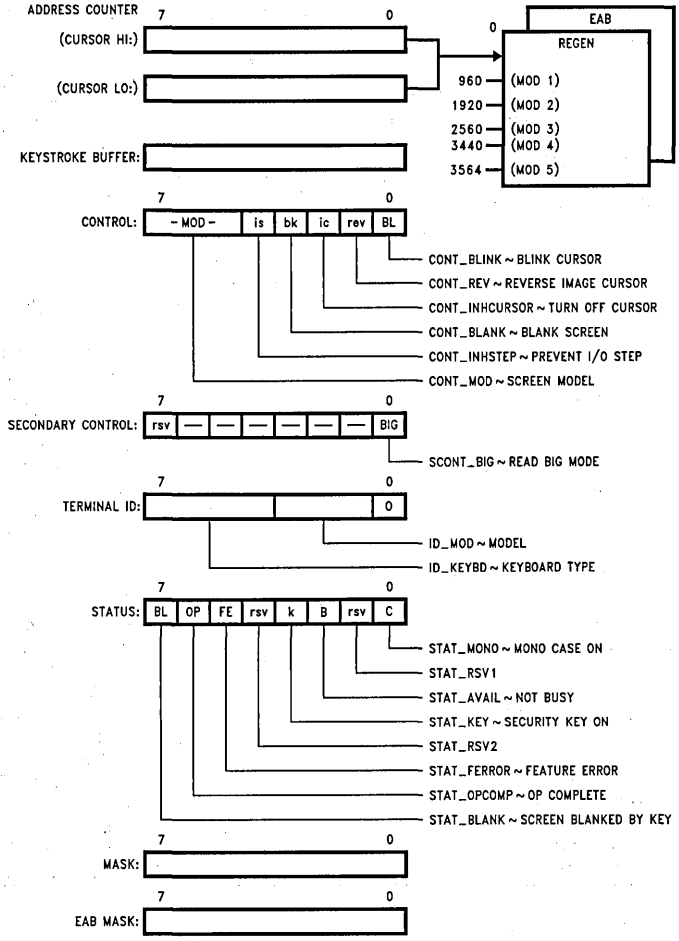


FIGURE 4-2. 3270 Internal Terminal Architecture

The SEARCH, INSERT and CLEAR commands require the terminal to process the command in the foreground while responding with "BUSY" status to the controller. (The foreground refers to non-interrupt driven routines. Foreground routines may be interrupted at any time.) Processing these commands requires substantially more time than the others, and hence are allowed to proceed without real-time response restrictions.

An interesting feature found in terminals and printers is the START OP command. Originally, this command was used only by controllers and printers to begin print jobs. Printers have specific areas within their buffers that are reserved for higher level commands from the controller. These higher level protocols started as formatting commands and extra printer feature control. With the advent of SNA and Distributed Function Devices, this concept is now used in terminals to pass SNA command blocks to multiple NAUs (Network Addressable Units) within the terminal. These NAUs are complete terminals, or peers, not just simple user interface devices.

As large mainframe systems proliferated, so did the need to off-load terminal support from the emerging 370 class mainframe. The need to "network" both remotely and locally was becoming apparent. In addition, the need to separate display and printer interface tasks from applications was sorely felt. The system developed by IBM eventually became Systems Network Architecture (SNA). The 370 class machines use secondary processors, or "front-ends" to handle the networking aspect of large scale systems and these "front ends" in turn use terminal and printer controlers to interface locally with the user interface devices. The controllers handle the device specific tasks associated with interfacing to different printers and displays. The front-ends handle connecting the routes from terminals or printers to applications on the mainframe. A session is a logical entity split into two halves; the application half and the terminal half, and connected by a virtual circuit. Virtual circuits can be set up and torn down by the system between applications and terminals easily, and the location of the specific terminal or printer is not important. NAUs are merely devices that can be addressed directly within the global network. Setting up multiple NAUs within a terminal allows all sorts of gateway opportunities, multi-display workstations, combination terminal/printers, and other things.

DFD devices can support up to five separate NAUs using a basic 3270 port. Using 3299 addressing allows eight sessions for each DFD device, or 40 possible NAUs per coax. By layering protocols over the basic 3270/3299 data stream, the controller can distribute more of the SNA processing to intelligent devices that replace terminals. APPC will allow more and more functions to be shared by NAUs that act as "peers" in the network.

5250 Data Stream Architecture

The 5250 data stream architecture has many similarities to 3270, although they are different in important ways. The primary difference is the multi-drop nature of 5250. Up to seven devices may be "daisy chained" together on the same twinax cable. Twinax is a very bulky, shielded, twisted pair as opposed to the RG/62U coax used in 3270.

The 5250 Bit stream used between the host control units and stations on the twinax line consists of three separate parts; a bit synchronization pattern, a frame synchronization pattern, and one or more command or data frames. The bit sync pattern is typically five one bit cells. This pattern serves to charge the distributed capacitance of the transmission line in preparation for data transmission and to synchronize receivers on the line to the bit stream. Following the bit sync or line quiesce pattern is the frame sync or line violation. This is a violation of the biphase, NRZI data mid-bit transition rule. A positive going half bit, 1.5 times normal duration, followed by a negative going signal, again 1.5 times normal width, allows the receiving circuitry to establish frame sync.

Frames are 16 bits in length and begin with a sync or start bit that is always a 1. The next 8 bits comprise the command or data frame, followed by the station address field of three bits, a parity bit establishing even parity over the start, data and address fields, and ending with a minimum of three fill bits (fill bits are always zero). A message consists of a bit sync, frame sync, and any number of frames. A variable amount of inter-frame fill bits may be used to control the pacing of the data flow. The SET MODE command from the host controller sets the number of bytes of zero fill sent by attached devices between data frames.

Message routing is accomplished through the use of the three bit address field and some basic protocol rules. There is a maximum of eight devices on a given twinax line. One device is designated the controller or host, the remaining seven are slave devices. All communication on the twinax line is host initiated and half duplex. Each of the seven devices is assigned a unique station address from zero to six; address seven is used for an End Of Message delimiter, or EOM. The first or only frame of a message from controller to device must contain the address of the device. Succeeding frames do not have to contain the same address for the original device to remain selected. The last frame must contain the EOM delimiter. For responses from the device to the controller, the responding device places its own address in the address field in all frames but the last one. It places the EOM delimiter in the address field of the last frame. However, if the response to the controller is only one frame, the EOM delimiter is used. The controller assumes that the responding device was the one addressed in the initiating command.

Responses to the host must begin within 60 ± 20 μs of receiving the transmission, although some specifications state a 45 ± 15 μs response time. In practice, controllers do not change their time out values per device type so that anywhere from 30 μs to 80 μs response times are appropriate.

The 5250 terminal organization is set up such that there are multiple logical devices within the terminal as in 3270. These devices are addressed through a command modifier field in the command frame. The command set for the base logical devices is shown in Table 4.2. Note that except for POLLs and ACTIVATE commands, all commands are executed in the foreground by the terminals, unlike the 3270 commands. In addition, 5250 terminals only respond after a POLL or ACTIVATE READ command. The remaining commands are loaded on a queue for passing to the foreground while the terminal responds with "busy" status to the host when Polled until all the commands on the queue have been processed. See *Figure 4-3* for the 5251 terminal architecture.

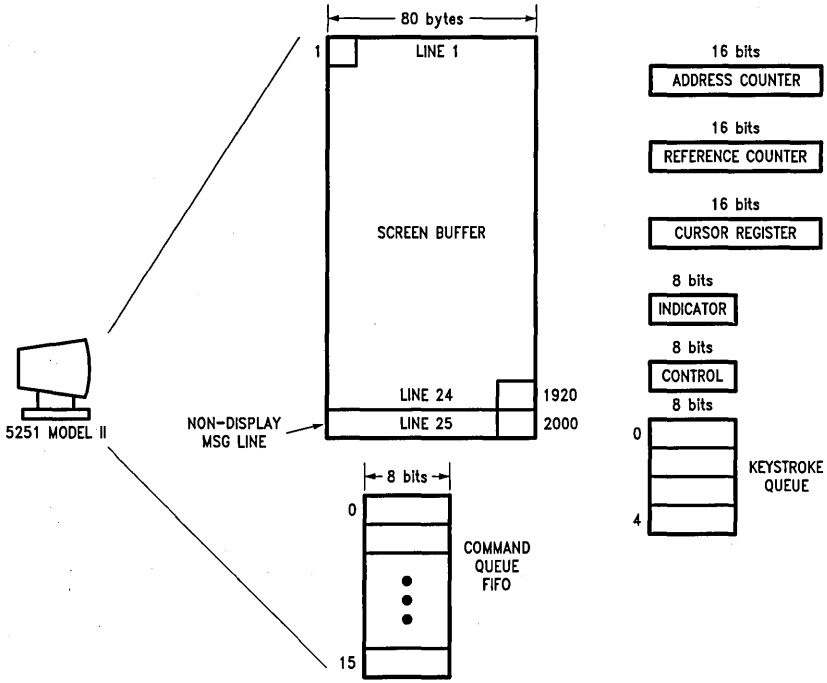
TABLE 4-2. 5250 Command Set

Queueable Commands			
Reads	**Writes**	**Control**	**Operators**
Read Data (Note 1)	Write Control Data	EOQ	Clear
Read Device ID (Note 1)	Write Data and Load Cursor	Load ADDR Counter	Insert Char.
Read Immediate (Note 1)	Write Immediate (Note 1)	Load Cursor Reg.	Move Data
Read Limits (Note 1)	Write Data (Note 1)	Load Ref. Counter	Search
Read Registers (Note 1)		Reset	
Read Line (Notes 1, 2)		Set Mode	

Non-Queueable Commands	
Responders	**Acceptors**
Poll	Activate Write
Activate Read	

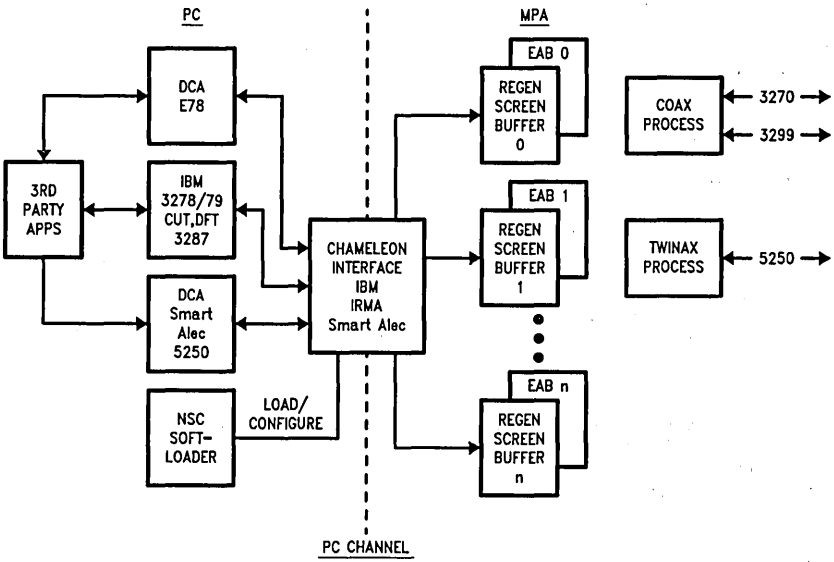
Note 1: Must be last command loaded onto queue, (EOQ may follow). When Terminal responds to POLL as not busy, then the appropriate ACTIVATE command must be sent.

Note 2: Not a documented command in the IBM PAI. (See MPA-II code for response.)



TL/F/10488-4

FIGURE 4-3. 5251 Terminal Architecture



TL/F/10488-5

FIGURE 4-4. MPA-II System Architecture

Terminal Emulation

Personal computers are often used to emulate 3270 and 5250 terminals, and in fact, have hastened the arrival of APPC functions in both the 3270 and 5250 arenas. Basic CUT (Control Unit Terminal) emulation is often accomplished by splitting the terminal functions into real-time chores and presentation services. The presentation services, such as video refresh and keyboard functions, are handled by the PC, and real-time response generation, etc., by an adapter card (see *Figure 4-4*). This is a somewhat expensive alternative to a "dumb" terminal. However, since PCs are becoming more and more powerful, their use as peers in SNA networks, as multiple NAUs, or multiple display sessions in 5250 is very promising. Although primitive in many ways, the 3270 and 5250 communications system's fast response times, unique serial protocols and processing overhead requirements have traditionally limited the confidence of third party developers in designing attachments. In addition, the high cost of many early solutions discouraged many would-be developers.

National Semiconductor opened the 3270 attachment market place to many third parties in 1980 with the release of the DP8340/41 protocol translation chip set. The chip set removed one of the major stumbling blocks to attachment designs, although formidable design challenges remained. Bit-slice or esoteric microcontrollers were still required to meet the fast response times specified by IBM. The difficulties and costs in designing interface circuitry for these solutions remained a problem. So in 1987 National Semiconductor introduced the DP8344 Biphase Communications Processor, BCP. By tightly coupling a sophisticated 3270/3299/5250 transceiver to a high speed RISC based CPU, National eliminated the last major stumbling block to IBM connectivity. National also made available for the first time a single hardware platform capable of supporting the 3270, 3299 and 5250 data streams.

The terminal emulation market opened with Technical Analysis Corporation's IRMA product in 1982. The 3278/79 terminal emulator quickly became the industry standard, even as IBM and many others entered the market place. Technical Analysis Corporation merged with Digital Communications Associates in 1983. The 3270 emulation market is now dominated by DCA and IBM. IBM produced the first 5250 terminal emulator in 1984, although it was a severely limited product. The market opened up in 1985 with the release of products by AST Research, IDE Associates, and DCA. DCA's Smart Alec was the first product to provide seven session support, address bidding, and a documented open architecture for third party interfacing. DCA's IRMA was released with a technical reference detailing their Decision Support Interface. This document along with the source code to E78 (their PC emulator software) allowed many companies to design micro to mainframe products using the DSI as the mainframe interface. IBM provides a technical reference for their 3278 Entry Level Emulator as well, (see Appendix C for a complete list of references).

The proliferation of the IBM and DCA interfaces coupled with the availability of detailed technical information about them made these interfaces good choices for the MPA-II. The MPA-II system was designed to do two major functions: one is emulation of the DCA and IBM emulation products; the second is to provide a powerful, multi-protocol interface that will afford greater utilization of the DP8344A. Specifically, the MPA-II emulates the hardware/firmware resident in PC add-in boards for 3270 and 5250 emulation products from DCA and IBM. To do this, we have constructed hardware and firmware that mimics the corresponding system components of the other emulators. The MPA-II system appears in every sense to be the board it is emulating, once it has been loaded and configured.

The DCA and IBM system organizations are similar. Each system is divided into two major functional groups: presentation services, and terminal emulation. The terminal emulation function resides entirely on the adapter hardware and maintains the screen buffers that belong to the host control unit. The terminal emulation function includes all real time responses and status generation necessary to appear as a true 5250 or 3270 device to the host controller. Presentation services carried out by the PC processor through the emulator software include fetching screen data from the adapter, translating it into displayable form, and providing the data to the PC's display adapter. In addition, the PC side presentation services collect keystrokes from the keyboard and present them to the adapter. The communication between the PC presentation handler and adapter emulation function consists mainly of status updates, keystrokes, and screen data.

DCA

The DCA products use an I/O mapped 4 byte mailbox to pass information between the PC's processor and the processor on the emulation card. The information is encoded in a <command>, [<argument>], [<argument>], [<argument>] and <status>, [response], [response], [response] format. Information flow is controlled through a Command/Attention semaphore implemented in hardware. Both the Smart Alec (5250) and IRMA (3270) interfaces have command sets that include reading and writing the screen buffers maintained on the adapter boards, sending keystrokes, and passing display information such as cursor position and general screen modes. The interfaces are both used in a polled manner, although both are capable of generating interrupts to the PC processor.

Both Smart Alec and IRMA have Signetics 8X305 processors that run the terminal emulation functions and interface to the PC presentation services. The PC function initiates commands and status requests by writing the appropriate value into the mailbox and setting the Command semaphore. The semaphore is then polled by the PC for a change in state that signals completion of the command and signals that valid response data is in the mailbox. The PC will poll for a specific amount of time before assuming a hardware malfunction has occurred. The 8X305 processors have no interrupt capabilities and handle all terminal emulation subtasks in a polled manner. The PC interface tasks are the lowest priority of all. The 8X305 may initiate information transfer to the PC by posting the Attention semaphore, and/or setting a PC interrupt, although this is not generally done. Both the Smart Alec and IRMA interfaces are implemented with 74LS670 dual-ported register files so that reads and writes from each processor are directed into separate register files.

DCA interfaces were designed for compatibility at the expense of interface through-put. The small I/O requirements and the fact that interrupts to the PC are not necessary allow the interfaces to install easily in most environments. The IRMA Decision Support Interface (DSI) utilizes eight I/O locations at 220h–227h. Smart Alec resides in I/O locations 228h–22Fh. All screen data and status information must pass through these mailboxes with the semaphore mechanism. This makes repainting the entire screen very slow. Both IRMA and Smart Alec utilize different schemes to reduce the necessity of reading entire screen buffers often. IRMA maintains a screen image in PC memory that is used in conjunction with a complex algorithm to determine which lines of the screen to update. Smart Alec maintains a 16 entry FIFO queue that contains screen modification information encoded in start/end addresses. This information is processed to decide which screen locations should be updated.

IBM

The IBM system organization, in general, is very similar to the DCA systems. The major differences lie in the interface implementations. The IBM system utilizes RAM dual-ported between the PC processor and the adapter board processor to transfer screen data from the adapter. In addition, IBM does not use an interpreted command/response I/O interface. The IBM interface uses 12 I/O locations with individual bits defined in each register for direct status availability. The status bits consumed by the PC presentation services are cleared through a "write under mask" mechanism. Consumable bits are read by the PC and, when written to, corresponding status bits are cleared by one bits in the value written. Reading a register of consumable bits and writing that value back out clears the bits set in that register. The interface can operate in a polled manner, although it typically is operated via interrupts. One register in the interface is dedicated to interrupt status (ISR—Interrupt Status Register, 2D0h) and when the PC is interrupted, the particular status change event is indicated in that register. Buffer modifications are indicated through a status change in the I/O interface which also provides an indication of the block modified. The actual screen data is in 8k of dual-port RAM and may be read by the PC when the "Buffer-Being-Modified" flag is cleared. This type of interface affords the IBM products great speed advantages, although limits compatibility with other add in PC boards.

Screen Presentation

Both the IBM and DCA systems present EBCDIC data to the PC presentation services for display. The presentation software must translate the EBCDIC codes into ASCII for PC display adapters. In addition, the screen attribute schemes for PCs and mainframe terminals differ greatly. The presentation services must provide the necessary display interface to emulate the "look" of the terminal that is being emulated. The PC keyboard scan codes are incompatible with mainframe scan codes, and must be translated for the keyboard type of the terminal being emulated. Both systems provide advanced PC functions such as residency, keyboard remapping, and multiple display support.

MPA-II

The MPA-II implements emulation of both the DCA and IBM interfaces. Therefore, an overall architecture similar to the DCA and IBM systems is employed (see *Figure 4-5*). The logical split in functionality between the PC and the adapter board processors is roughly analogous; the PC provides presentations services and the adapter hardware/firmware handles the host terminal emulation tasks (see *Figure 4-6*, *4-7* and *4-8*). The BCP on the adapter board is soft-loaded by the PC and configured to operate in one of the protocols and interface modes. The adapter board then assumes the hardware emulation tasks of the physical interfaces of the DCA or IBM products. At this time the DCA, IBM (or a DCA/IBM compatible) emulation program is executed on the PC. To this program the MPA-II appears to be a DCA or IBM emulation card.

The MPA-II hardware consists of a DP8344A running at 18.89 MHz with 8k × 16 bits zero wait state instruction memory, 32k × 8 bits one wait state data RAM, a coax/twisted pair 3270/3299 front end, a 5250 twinax front end, and a BCP software controlled PC interface that enables the MPA-II to appear as a variety of industry standards interfaces. The BCP Remote Interface Configuration register (RIC) is located in PC I/O space at 2DFh (see *Figure 4-9*). This register facilitates downloading of instructions and data memory from the PC, starting and stopping the processor, and configuring the low level interface mode. The MPA-II utilizes the low level fast buffered write/latched read interface mode.

The MPA-II Configuration register (see *Figure 4-10*) is located at I/O location 2DCh and controls which type of high level interface the MPA-II board is operating in (i.e., IRMA, Smart Alec, IBM, coax, etc.). Changing the value of this register while the MPA-II is operating will cause the MPA-II to change mode, resetting the emulation session in progress. In addition, a simple MPA-II command set can be issued through the MPA-II Configuration register and the MPA-II Parm/Response register (I/O location 2DBh) for use as a passive debugging aid.

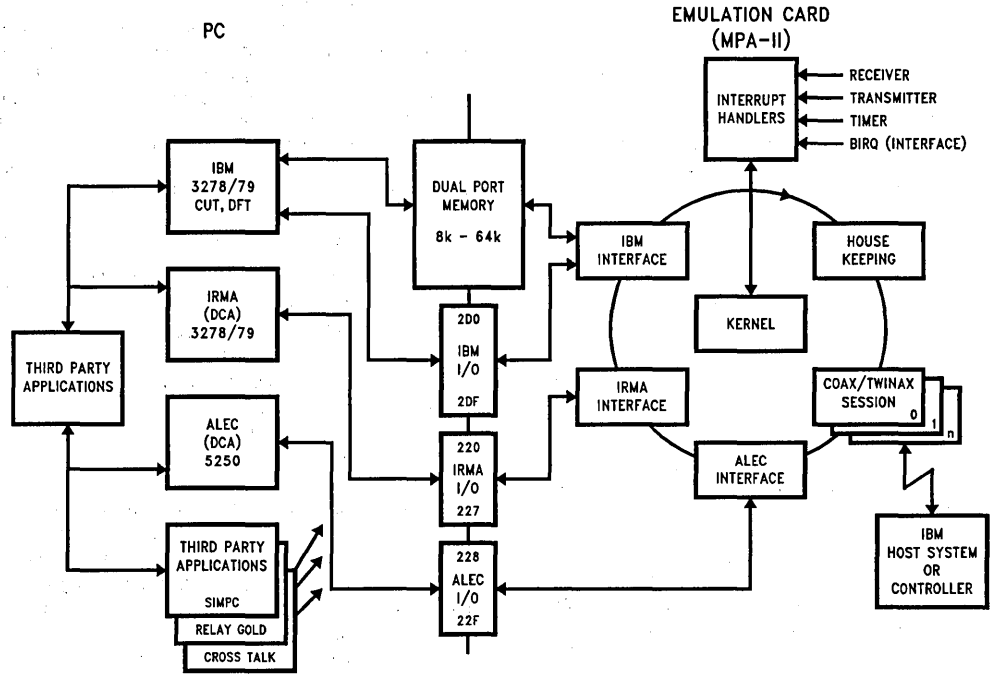
When either of the DCA modes are enabled, the I/O block 220h–22Fh is decoded, split into read and write banks, and mapped into the BCP's data memory. For the IBM mode, the I/O block from 2D0h–2DAh is decoded and the Write-Under-Mask function is enabled. In addition, the 8k of dual-port RAM is defined according to the IBM interface mode. For CUT emulation, only the lower 4k of the dual-port RAM is used. For DFT mode, the entire 8k block may be utilized. Neither DCA mode utilizes dual-port memory, but it is still available to the PC so the MPA-II firmware maps screen information there. Note that the MPA-II hardware always decodes I/O addresses 220h–22Fh and 2D0h–2DFh regardless of the PC interface selected.

The MPA-II interface mimics the DCA and IBM interfaces by interrupting the BCP when write accesses occur to the I/O space of interest (220h–22Fh, 2D0h–2D6h and 2D8h–2DEh) while holding off any other PC accesses to the MPA-II board, thus "locking out" the PC. The BCP monitors these I/O accesses through the use of the "MPA-II Access" register contained in a PAL. This register captures the location of the last PC I/O access. The BCP's I/O access interrupt routines then get control and emulate in software DCA's or IBM's I/O hardware functions (such as IBM's write under mask function). At the end of interrupt processing, the software "unlocks" the PC, allowing access once again to the MPA-II's memory and I/O registers by the PC. The extreme speed of interrupt processing by the BCP makes this feasible. Accesses of the dual-port RAM by the PC are regulated by the interface only in assuring that simultaneous accesses by the PC and BCP do not occur. The location of the dual-port RAM in the PC memory map is determined by a value written into the 2D7h I/O location. This "Segment" register is the upper 7 bits of the PC address field and is compared with the address presented during PC memory cycles for decoding. Writing different values to this register moves the decoded memory block anywhere within the PC memory space to avoid conflicts. The pacing of dual-port accesses is handled by provisions in the emulated interface definition.

The PC I/O map for the MPA-II adapter board is as follows:

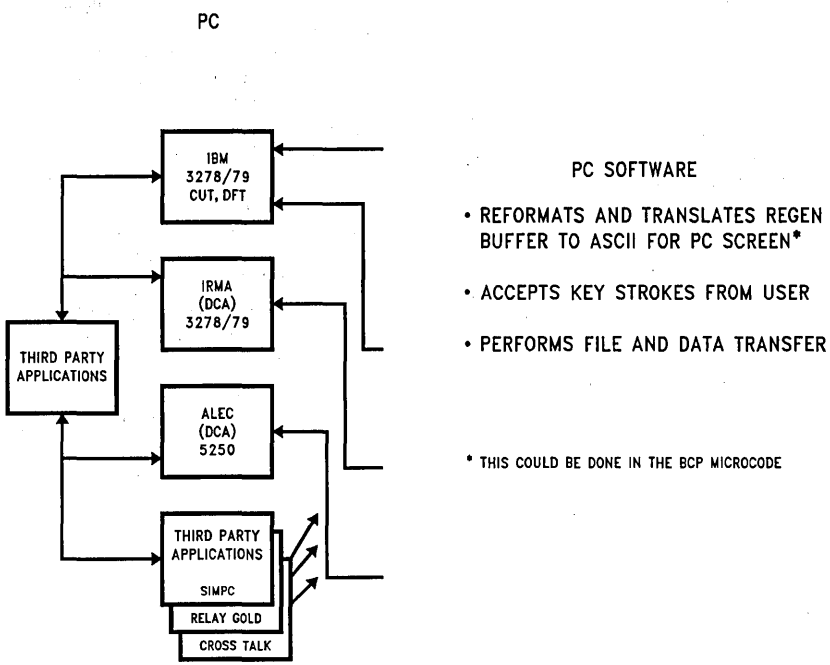
TABLE 4-3. MPA-II PC I/O Map

220h—	IRMA Command/Status Register
221h—	IRMA Argument/Response
222h—	IRMA Argument/Response
223h—	IRMA Argument/Response
224h—	Decoded, Unused
225h—	Decoded, Unused
226h—	IRMA Command/Attention Semaphore Control
227h—	IRMA Command/Attention Semaphore
228h—	Smart Alec Command/Status Register
229h—	Smart Alec Argument/Response Register
22Ah—	Smart Alec Argument/Response Register
22Bh—	Smart Alec Argument/Response Register
22Ch—	Decoded, Unused
22Dh—	Smart Alec Control Register
22Eh—	Smart Alec Control Register, Command/Attention Semaphore
22Fh—	Smart Alec Strobe
2D0h—	IBM Interrupt Status Register
2D1h—	IBM Visual/Sound
2D2h—	IBM Cursor Address Low
2D3h—	IBM Cursor Address High
2D4h—	IBM Connection Control
2D5h—	IBM Scan Code
2D6h—	IBM Terminal ID
2D7h—	IBM/MPA-II Dual-Port Segment Location Register
2D8h—	IBM Page Change Low
2D9h—	IBM Page Change High
2DAh—	IBM 87E Status
2DBh—	MPA-II Parm/Response Register
2DCh—	MPA-II Configuration/Command Register
2DDh—	Decoded, Unused
2DEh—	Decoded, Unused
2DFh—	MPA-II RIC Register



TL/F/10488-15

FIGURE 4-5. PC Terminal Emulation Architecture

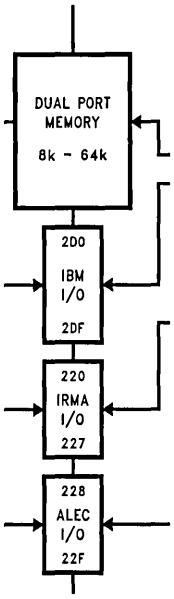


TL/F/10488-9

FIGURE 4-6. PC Software

PC INTERFACE

- ALLOWS PC TO COMMUNICATE WITH THE TERMINAL EMULATOR CARD.
- IBM
  -STATUS QUERY METHOD
- IRMA/ALEC
  -COMMAND/RESPONSE METHOD

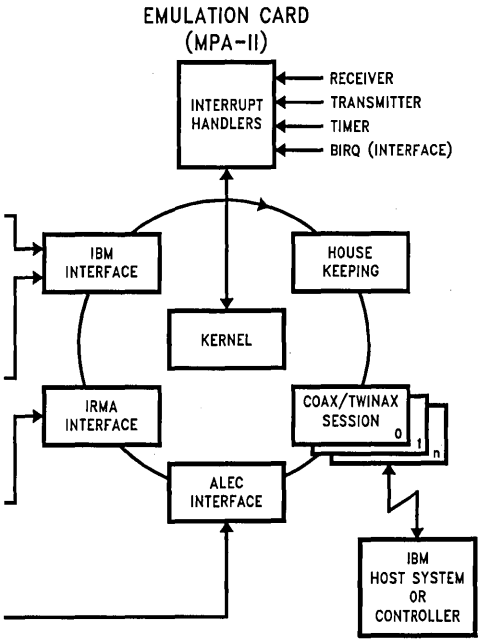


TL/F/10488-10

FIGURE 4-7. PC Interface

EMULATION CARD

- PROVIDES PHYSICAL AND ELECTRICAL CONNECTION
- PROCESSES ALL DATA LINK COMMANDS
- ISSUES ALL DATA LINK RESPONSES
- OPERATES INDEPENDENT OF PC CPU
  -REBOOTING PC HAS NO EFFECT ON ACTIVE SESSIONS



TL/F/10488-11

FIGURE 4-8. Emulation Card

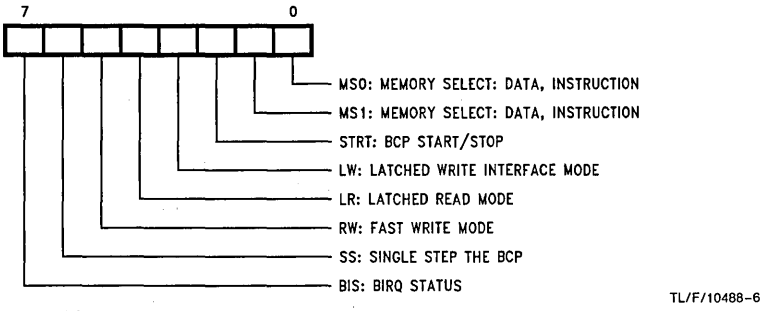


FIGURE 4-9. BCP Remote Interface Configuration Register

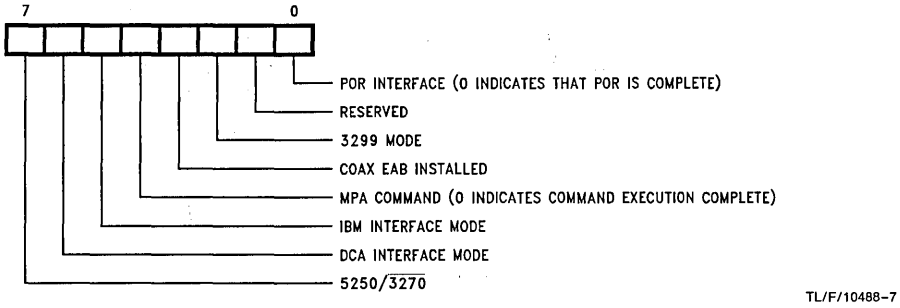


FIGURE 4-10. MPA-II Configuration Register

MPA-II Firmware Organization

The BCP firmware provides true 5250, 3270, and 3299 emulation support, as well as providing the intelligence behind the PC interface. To do this, a software architecture radically different than the DCA or IBM systems was developed. The real power of the BCP lies in its rich instruction set and full featured CPU. Taking advantage of that power, the BCP firmware is interrupt driven and task oriented. It is not truly multi-tasking, although the firmware logically handles multiple tasks at once. The firmware basically consists of a round robin task scheduler (called the Kernel) with real-time interrupt handlers to drive the system. Events that happen in real-time, such as accesses by the PC or host commands, schedule tasks to complete background processing. Real-time status and responses are developed and presented in real-time.

The BCP firmware uses a number of memory constructs known as templates to handle its data structures. The primary construct is the DCP, or Device Control Page. The DCP is a 256 byte block that contains all global system variables. The DCP contains a map of which SCPs, or Session Control Pages are active. Each SCP is 256 bytes and contains all variable storage for a particular session; 3270, 5250, or 3299. Each SCP has a corresponding screen buffer, and optionally an EAB buffer (there is no EAB in 5250 terminals).

MPA-II Performance

The BCP is running at 18.8696 MHz with no instruction memory wait states and one data memory wait state. This yields an average instruction cycle time of 160 ns, a maximum instruction cycle time of 212 ns and a maximum interrupt latency of 237 ns (excluding wait states due to PC accesses). Although such performance may seem excessive, remember that the 3270 protocol requires a 5.5 μs response time and that the newer controllers sometimes send commands less than 10 μs apart. These commands must be executed in real-time, so for short periods of time, extremely high performance is required. In the MPA-II, the BCP also has other real-time demands on it. For example, the MPA-II requires the BCP to perform DCA or IBM I/O hardware emulation real-time in firmware. Furthermore, both the controller and the PC are asychronous events which can (and do) occur at the same time.

Using Hewlet Packard's 16500A Logic Analyzer and 10390A System Performance Analysis Software, the MPA-II's worse case performance scenario was analyzed. This scenario consisted of the MPA-II running 3270 with EAB installed while performing IRMA file transfers using DCA's FTCMS software. A special NO-OP routine was added to the MPA-II software in order to achieve 100% utilization of the BCP. The breakdown of relative activity is shown in Table 4-4.

TABLE 4-4. MPA-II Performance

Coax Related Activity	9%
IRMA Related Activity	10%
Total Activity	19%

As is shown in Table 4-4, the BCP still has over 81% of its bandwidth free to do additional tasks.

Advanced Product Possibilities

With over 81% of the BCP's bandwidth unutilized, possibilities for advanced 3270/3299 and 5250 devices with exceptional overall system performance, advanced features, and compactness become both realizable and practical. For example, if a more efficient PC to MPA-II (BCP) interface was developed which eliminated the need for the BCP firmware to emulate I/O hardware, and additional tasks were off loaded to the BCP, such as Regen/EAB buffer to PC Screen buffer translation, then the overall system performance of a full featured MPA-II CUT mode terminal could rival that of the most advanced IBM CUT mode terminals. Yet, the PC memory requirements of such an emulator would be less than that of the simplest PC emulator on the market today because the PC software would only need to process keystrokes and copy the BCP's translated PC screen buffer directly into the PC's screen buffer memory. Furthermore, advanced features such as 3299 support could be included without additional hardware costs. All this is possible using the current MPA-II board without hardware modification because the MPA-II emulates DCA and IBM interface hardware using BCP software. Adding this new interface into the product requires only software changes.

5.0 HARDWARE ARCHITECTURE

This chapter focuses on the hardware employed to satisfy the goals of the MPA-II project. Designed to support both the coax (3270/3299) and twinax (5250) protocols, the hardware also allows emulation of the PC interfaces outlined in Chapter 2. By taking advantage of the BCP's power and integrating the extra logic requirements into programmable logic devices, this level of functionality was provided on a single half-height PC XT/AT card. In an effort to convey the reasons behind specific decisions made in the hardware design, the design methodology is presented from a "top-down" perspective.

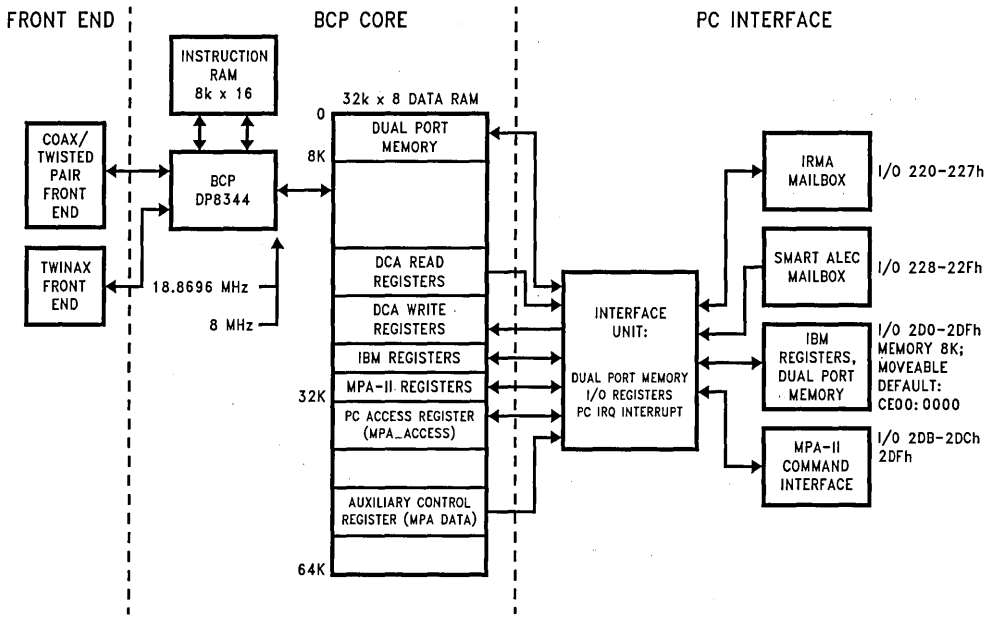
Architectural Overview

The MPA-II hardware should be viewed as three conceptual modules (see *Figure 5-1*), including:

1. BCP minimum system core, consisting of the BCP, instruction memory, data memory, clock, and reset logic.
2. PC interface including the PC and BCP memory decode and interrupts.
3. Coax/twisted pair and twinax front-end logic and connectors.

These module divisions are denoted by the dotted lines seen in *Figure 5-1*. The minimum system core is required, with some modifications, for any design using the BCP. The type of bus (PC, PS/2™ Micro Channel™, VME, etc.) and transfer rate requirements dictate the interface logic, which, for the MPA-II design, is optimized for the PC XT/AT I/O channel. The front-end logic meets the physical-layer requirements of the 3270 and 5250 protocols.

Since much of the logic external to the BCP is implemented in programmable logic devices (PALs), these conceptual partitions overlap at the device level. Although the design can be implemented in discrete logic, we chose to use programmable logic devices to shorten development time, decrease board real-estate requirements, and maintain maximum future adaptability. The schematic and the listings describing the logic embodied in the PALs are in the Hardware Reference in Appendix A.



TL/F/10488-8

FIGURE 5-1. MPA-II Hardware Architecture

BCP Minimum System Core

The BCP offers a high level of integration and many functions are provided on-chip; there is, however, a minimal amount of external logic required. This core is comprised of the BCP and the external logic require to support the clock requirements, reset control, Harvard memory architecture, and multiplexed AD bus (see *Figure 5-2*).

Clock Source

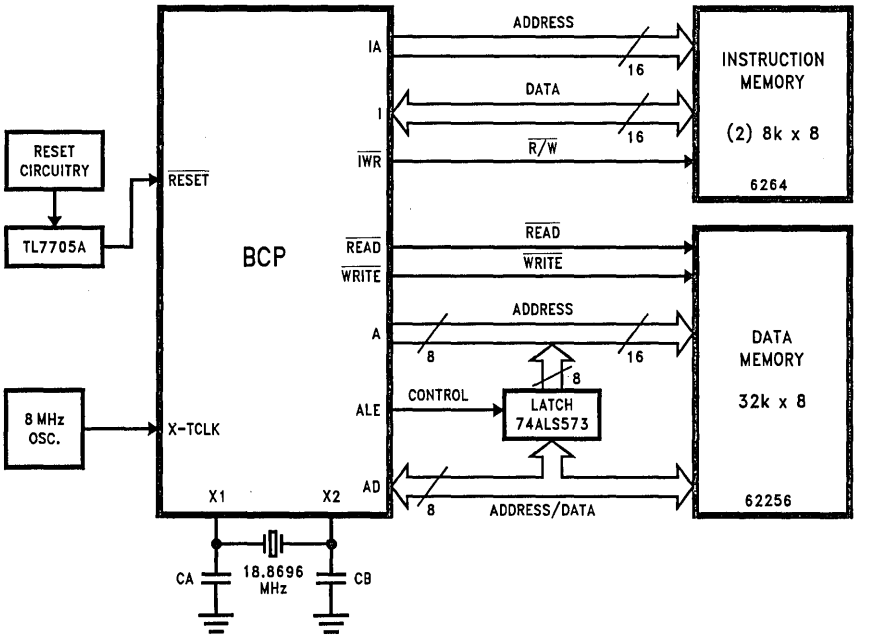
The coax and twinax protocols operate at substantially different clock frequencies (2.3587 MHz and 1 MHz, respectively), therefore two clock sources are required. The BCP has the software-programmable flexibility to drive both the CPU and transceiver in the following ways: the clock independently divided down to either or both sections, or by two separate asynchronous clocks (utilizing the external transceiver clock input, XTCLK). To provide sufficient waveform resolution, the transceiver must be clocked at a frequency equal to eight times the required serial bit rate. This means that an 18.8696 MHz (8 x 2.3587 MHz) clock source is required when operating in the 3270 coax environment and an 8 MHz clock (8 x 1 MHz) is needed for the 5250 twinax environment. An 18.8696 MHz clock is also a good choice for the BCP's CPU section.

Therefore, in the coax mode, the transceiver and the BCP's CPU share the same clock source. To maximize the available CPU bandwidth in the twinax mode, the 18.8696 MHz clock source drives the CPU while a TTL clock is used to drive the BCP's external transceiver clock input. Therefore, in the twinax mode, the BCP's CPU and transceiver sections operate completely asynchronously.

The 18.8696 MHz clock is provided by the BCP's on-chip clock circuitry and an external oscillator. This circuit, in conjunction with external series load capacitors, forms a "Pierce" parallel resonance crystal oscillator design. The oscillator is physically located as close as possible to the X1 and X2 pins of the BCP to minimize the effects of trace inductances. The traces (0.05") are wider than normal. NEL Industries makes a crystal specifically cut for the 18.8696 MHz frequency and is the recommended source for these devices. This crystal requires a 20 pF load capacitance which can be implemented as 40 pF on each lead to ground minus the BCP/socket capacitance and the trace capacitance. A typical value for the BCP/socket combination capacitance is 12 pF. The wide short traces contribute very little additional capacitance. We therefore chose a standard value of 27 pF for the discrete ceramic capacitors C24 and C25, placing them as close as possible to the crystal. The 5.6Ω pull up resistor tied to X1 is designed to improve oscillator start up under unusual power supply ramp conditions. This is normally not a problem for PC power supplies so that the resistor could be omitted. The twinax clock is provided by a standard 8 MHz TTL monolithic clock oscillator attached to the BCP's external clock input, XTCLK.

The MPA-II runs the BCP at full speed, 18.8696 MHz ({DCR[CCS]} = 0), with zero instruction (n_{IW}) and one data (n_{DW}) wait states, resulting in a T-state of 53 ns. For a system running the BCP at half speed, 9.45 MHz ({DCR[CCS]} = 1), with zero instruction (n_{IW}) and zero data (n_{DW}) wait states, the T-state would be 106 ns. The T-state can be calculated using the following equation:

$$\text{T-state} = 1/(\text{CPU Clock Frequency})$$



TL/F/10488-16

FIGURE 5-2. BCP Core

Reset Control

Power-up reset for the BCP consists of providing the debounced, active low, minimum pulse width specification of ten T-states. Since the BCP powers up in the slowest configuration, a T-state is the period of the oscillator divided by two, or 106 ns. The external logic must therefore provide a minimum 1.06 μs reset pulse to the BCP. The MPA-II design incorporated two reset sources in addition to power-up including: the PC I/O channel reset control signal (active high), and an automatic reset if the digital supply voltage drops by more than 10%.

We chose the Texas Instruments TL7705A supply voltage supervisor to monitor V_{CC} and provide the minimum pulse width requirement. This device will reset the system if the digital 5V supply drops by more than 0.5V, and keep the reset asserted until the voltage returns to an acceptable level. The TL7705A will also assure that the minimum time delay is met. The time delay is set by an external capacitor and an internal current source. Since this time delay is not guaranteed in the data sheet, we chose a 0.1 μF ceramic capacitor resulting in a typical 1.3 ms reset pulse width. A 0.1 μF ceramic capacitor is connected to the REF input of the chip to reduce the influence of fast transients in the supply voltage. The active high PC reset signal is inverted in the MPA-II_AC (MPA-II Auxiliary Control) PAL. The active low output of the bipolar TL7705A is the MPA-II system reset and is pulled up by a 10k resistor for greater noise immunity.

Memory Architecture

The BCP utilizes separate instruction and data memory sections to overcome the single bus bandwidth bottleneck often associated with more conventional architectures. Instruction memory is owned exclusively by the BCP (remote processor accesses to this memory occur through the BCP, and only when the BCP is stopped); therefore, the entire instruction memory/bus bandwidth is available to the BCP. This architecture allows the BCP to simultaneously fetch instructions and access data memory, thus load/store operations can be very quick. It is important to note, however, that the instruction bus bandwidth does have some dependency on data bus activity. If a remote processor, for instance, is currently the data bus master, execution of an instruction accessing data memory will be waited, degrading BCP CPU performance.

The speed of both instruction and data memory accesses is limited by memory access time. Since the BCP features programmable memory wait states, the designer has the flexibility of choosing memories strictly on a cost/performance trade-off. No external hardware is required to slow the BCP memory access down (unless the maximum number of programmable wait states is insufficient, in which case the WAIT input of the BCP can be utilized). Instruction memory access time has the biggest impact on system performance since every instruction executed involves an access of this memory. Each added instruction wait state degrades zero-wait state performance by approximately 40%. Load/store operations occur less frequently in normal code execution, therefore relatively slower data memory can often be utilized. Each additional data memory wait state degrades the performance of a zero-wait state data access by about 33%.

Instruction Memory

A design goal for the MPA-II project dictated our choice of static RAM for instruction memory, since the ability to soft-load code from the PC was necessary. Furthermore, to maximize CPU bandwidth we chose zero wait-state instruction memory operation. When the hardware was designed, instruction memory requirements were estimated at 4k to 8k words, therefore two 8k x 8-bit static RAMs were employed.

Instruction memory access time requirements can be calculated using Parameter 1, the Instruction Memory Read Time, Table 5-5, Instruction Memory Read Timing, of the Device Specifications section of the DP8344B Data Book.

$$(n_{IW} + 1.5)\,T + (-19)\ \text{ns}$$

Where: n_{IW} is the number of instruction wait-states, and T = 53 ns. Therefore the maximum access time is (0 + 1.5) 53 − 19 = 60.5 ns. For the MPA-II system running the BCP at half speed (T-state = 106 ns), the maximum access time is (0 + 1.5) 106 − 19 = 140 ns. Comparing both the half and full speed maximum instruction memory access time requirements, it is apparent that 55 ns RAMs are appropriate. A complete instruction memory timing analysis is provided in Appendix B.

Reads of instruction memory by the remote system occur through the BCP and look identical in timing to the local (BCP) reads on the instruction bus.

Soft-Load Operation

The BCP cannot modify instruction memory itself. Memory is only written through the BCP (while the BCP is stopped) from the remote system (PC), and is referred to as "soft-load" operation. Since the BCP has an 8-bit data path and a 16-bit instruction bus, instructions are read or written by the PC in two access cycles; the first cycle accessing the low byte of the instruction, the second cycle accessing the high byte of the instruction and automatically incrementing the Program Counter after the instruction has been accessed. See the Remote Interface section of the DP8344B Data Book for a complete description of instruction memory accesses.

The critical parameter for instruction writes is the minimum write strobe pulse width of the RAM, which is about 40 ns for most 8k x 8 55 ns static RAMs (55 ns RAM specifications are compared to the BCP minimum requirements since it represents the worst case). The $\overline{IWR}$ (BCP Instruction WRite output, active low) minimum pulse width is calculated from Parameter 20 ($\overline{IWR}$ Low Time) in Table 5-22, Fast Buffered Write of IMEM, of the Device Specifications section of the DP8344B Data Book:

$$(n_{IW} + 1)\,T - 10\ \text{ns}$$

For soft-loads that occur after reset, the CPU clock is in the POR half-speed state and the number of instruction and data memory wait states is a maximum; therefore a T-state is 106 ns and n_{IW} equals 3; thus, $\overline{IWR}$ minimum pulse width is (3 + 1) 106 − 10 = 414 ns. Soft-loads that occur after the BCP Device Control Register has been initialized to full speed operation with no instruction wait states represent the worst case timing of (0 + 1) 53 − 10 = 43 ns, which is still greater than the 55 ns RAM requirement of 40 ns.

Other parameters that must be considered are data setup and hold times for the RAM. The BCP must provide valid data on the Instruction bus before the minimum setup time of the RAM and hold the valid data on the bus at least as long as the minimum hold time. For the RAMs we considered, these times were 25 ns and 0 ns, respectively. Again, looking at Table 5-22 (Parameter 19, I valid before $\overline{IWR}$ rising), we see that if valid data for the high byte of the instruction is present on the AD bus in time, the BCP is guaranteed to present valid data on the Instruction bus a minimum of

$$(n_{IW} + 1)\,T - 18\text{ ns}$$

before the rising edge of $\overline{IWR}$. The BCP will hold that data on the bus for a minimum of 22 ns afterward (see Parameter 18, $\overline{IWR}$ rising to I Disabled). To see that the minimum set up time is met for both the half speed POR state and the full speed operation, note that both $(3 + 1)\ 106 - 18$ ns = 406 ns (half speed) and $(n_{IW} + 1)\ 53 - 18$ ns = 35 ns (full speed) are greater than the minimum set up time of the RAM which was 25 ns. Furthermore, the minimum hold time of 22 ns, for both half speed and full speed, is greater than the 0 ns required. Thus, successful operation is assured. See the MPA-II timing analysis in Appendix B and the PC interface section in this chapter for a discussion of AD bus timing.

Data Memory

A considerable amount of data memory was required for the MPA-II design since the system supports multiple sessions (see Chapter Six, MPA-II Software Architecture, for more information). For this reason we specified 32K of 8-bit data memory).

Data Memory Timing

Data RAM can be accessed by both the BCP and the remote system, part of the RAM appears to the remote system as dual-ported RAM via the Remote Interface logic of the BCP. This memory can be both read from and written to during BCP code execution. Designing in the data RAM is therefore a more complicated procedure than selecting instruction memory. Using 53 ns for the MPA-II T-state and one for n_{DW} (number of data wait-states) as defined earlier, we can verify the critical memory parameters by comparing the results of the calculations against the RAM requirements. The 32K x 8, 100 ns static CMOS RAM minimum requirements for the critical parameters are compared against the BCP's minimum specifications and are listed in Table 5-1. For a complete description of the BCP minimum specifications, see Appendix B.

TABLE 5-1. Data Memory Timing

Parameter	RAM	BCP*
Address Setup	0	47.5
Chip Select to Write End	90	122.5
Access Time	100	108.5
Write Strobe Width	60	96
Data Setup	40	86
Data Hold	0	31.5

—All units are in nanoseconds.

*53 ns T-state with one data wait state.

Again, the numbers reveal the validity of the hardware design for local (BCP) accesses of data memory. Please see the PC interface section for timing related to the remote access. Also, an MPA-II timing analysis of both 106 ns and 53 ns T-states is provided in Appendix B.

Multiplexed AD Bus

The BCP's 8-bit data bus is multiplexed with the lower 8-bits of the data memory address bus to lower pin count requirements. This necessitates de-multiplexing the Address/Data bus externally. The timing of the ALE (Address Latch Enable) control signal relative to the AD bus is optimized for use with a standard octal latch, therefore a 74ALS573 is employed to provide separate Address and Data buses for the system. The TRI-STATE buffers of the latch are enabled by the BCP output $\overline{LCL}$ (active low) such that if a remote access occurs this device will TRI-STATE.

PC Interface

As mentioned earlier, the MPA-II supports the industry-standard interfaces associated with coax and twinax terminal emulation. These include:

COAX:
- IBM 3270 Emulation Adapter Interface
- DCA Decision Support Interface (IRMA)

TWINAX:
- DCA Smart Alec Interface

These interfaces share some common elements, but have many differences as well. The IBM adapter employs an interrupt-driven interface, IRMA's PC interface is a polled implementation, and Smart Alec, while operating in a polled environment, has the capability of interrupting the PC as well. The IBM Emulation Adapter's control registers are mapped into the PC's I/O space; the screen buffer is mapped into the PC's memory space and is relocatable (see Table 5-2). The two DCA interface occupy a contiguous block of PC I/O space only; there screen buffer(s) are not directly visible to the PC. These architectures are explored in much greater detail in Chapter 6 of this manual. Note than the MPA-II utilizes some of the IBM reserved registers for MPA-II usage. These MPA-II registers may be easily relocated by changing the MPA-II PAL equations.

TABLE 5-2. PC Mapping of the MPA-II Board

Description	Address	
	I/O	Memory
IBM Interface:		
Remote Interface Control (RIC)	02DF*	
Decoded and Unused	02DD* – 02DE*	
MPA-II Configuration Register	02DC*	
MPA-II Parm/Response Register	2DB*	
IBM Control Registers	02D0–02DA	
IBM Screen Buffer		CE000 (Relocatable)
DCA DSI Interface:		
IRMA	0220–0227	
Smart Alec	0228–022F	

*Reserved IBM register spaces.

The MPA-II design had to encompass all of these implementations. This was accomplished by taking advantage of the underlying similarity of the interfaces as well as the speed and flexibility of the BCP. We minimized chip count and board space requirements through judicious partitioning of the PC address decode while emulating in BCP software the interface registers in data RAM. Refer to *Figure 5-3* for an overview of the hardware architecture employed in implementing the BCP/PC interface.

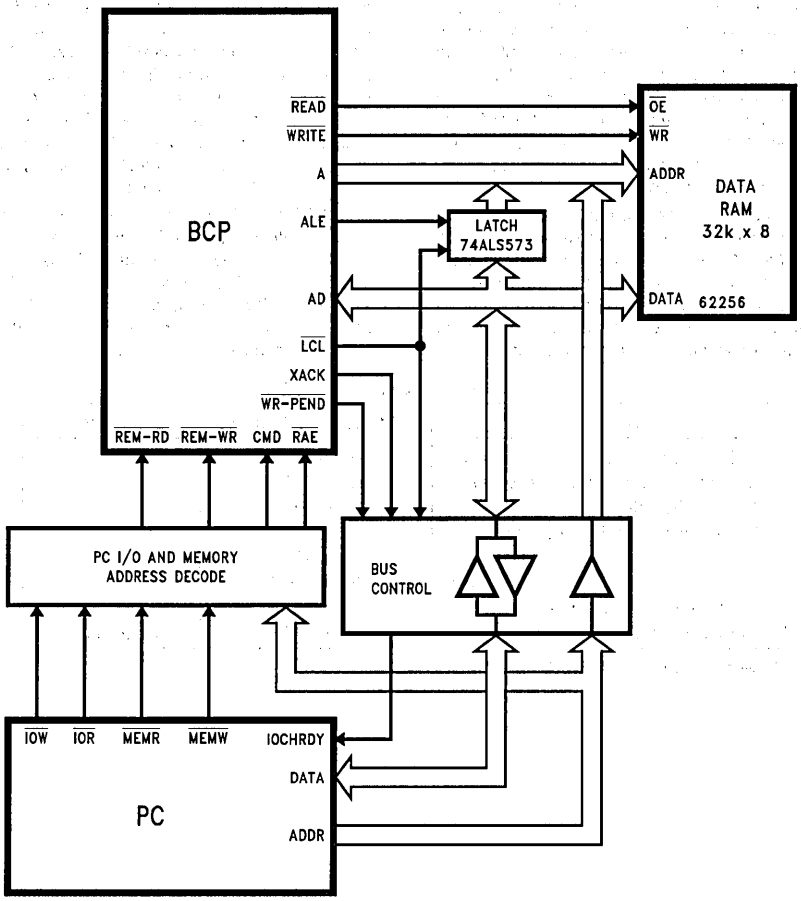
The PC address decoding is partitioned into sections that first check for accesses to the relocatable memory block and accesses to the I/O register addresses of the different interfaces. These addresses are then translated into the proper area of the BCP data memory. The BCP data memory map is divided in half, the lower 32k is contained in the single 32k x 8 RAM described earlier, and the upper 32k is decoded for several functions (see Table 5-3). The decoding sections feed into a control section that makes the final decision on whether (or not) the current PC bus cycle is an access of one of the emulated systems. It should be noted that the type of emulation is not selectable; the MPA-II board will respond to accesses of all of the PC addresses detailed in Table 5-2. The MPA-II will not run concurrently with any of the boards it emulates, or any other board that overlaps with these same addresses.

The BCP's RIC (Remote Interface Control) register is mapped into the PC's I/O space. The PC can always find this register by reading I/O hex address 02DFh. The DCA interfaces (IRMA and Smart Alex) occupy PC I/O addresses 220–22Fh. The IBM interface occupies PC I/O addresses 2D0–2DFh for register space, and a relocatable 8k block of memory for the screen buffer(s).

TABLE 5-3. BCP Data Memory Map

Description	BCP Address (A15–0)	PC I/O Address
Auxiliary Control Register (mpa_data)	A000–BFFF	
PC Access Register (mpa_access)	8000–9FFF	
*IBM API Registers	7FD0–7FDF	2D0–2DF
DCA API (IRMA and Smart Alec)		220–22F
PC Writes:	7F20–7F2F	
PC Reads:	7E20–7E2F	
BCP-Owned Memory Area	2000–7E1F	
*Screen Buffer Area	0000–1FFF	Relocatable

*Dual-Ported RAM (Visible to Both BCP and PC)



TL/F/10488-19

FIGURE 5-3. BCP/PC Interfaces

PC I/O and Memory Address Decode

The BCP CPU and Remote Interface units operate autonomously. Since the I/O registers are mapped into the BCP's data RAM and the CPU has to know which register was written to by the PC, external logic is provided that latches the low six bits of the address bus during remote accesses. The BCP can read this external register to identify which emulated register has been modified and take the appropriate action.

The relocatable memory segment location where the screen buffer of the IBM interface is located is decoded in discrete hardware consisting of the following components: U15, a 74ALS521 magnitude comparator that compares the PC memory address accessed against the stored value of the relocatable memory segment address and asserts the signal $\overline{\text{MMATCH}}$ (active low) when a match occurs; the Segment Register U16, a 74ALS574 containing the stored memory address used to identify the memory segment of the screen buffer block. The relocatable block of data memory defaults to base address CE000 on the IBM adapter. In the MPA-II System, the base address of the memory segment must be loaded into the segment register (PC I/O address 2D7h) before the PC can access the IBM screen buffer area in dual-port RAM. This Segment Register is not accessible by the BCP. It is only accessed by a PC write to I/O location 2D7h. A PC read of the I/O address 2D7h accesses a corresponding RAM location which is written in the same manner as all writes to the IBM I/O locations 2D0–2DAh, as described next.

Accesses to the I/O locations used by the IBM Interface (200h–2DFh) and the DCA DSI Interfaces (220–22Fh) are decoded as follows: PC address lines A12–A4 are brought into the MPA-II_PD (PC Address Decode) PAL-U9 for decode. PC address lines A14–A16 and A17–A19 are first decoded with three input NOR gates, U5B and U5C, which are in a 74ALS27. The outputs of both of these NOR gates are then brought into the MPA-II_PD PAL for further decode. Note that PC address lines A13 and A0–A3 are not decoded at this point. A preliminary decision is made by the MPA-II_PD Pal to indicate if the IBM or DCA interfaces are being accessed. The outputs $\overline{\text{DCA\_REG}}$ and $\overline{\text{IBM\_REG}}$

indicate which, if any, emulated interface is being accessed. These signals are used in conjunction with $\overline{\text{MMATCH}}$, the PC address lines A13 and A0–A4, and the read and write strobes of the PC in U7, the MPA-II_RD (MPA-II Register Decode) PAL to make the final determination on the validity of the access. If it is an emulated interface I/O register access, $\overline{\text{IO\_ACCESS}}$ will be asserted back to the MPA-II_PD PAL. This PAL will in turn translate the access to the top of the BCP data RAM where the I/O register page is located (see Table 5-3). Note the differentiation in Table 5-3 between PC reads and writes for the DCA translation. This is required to emulate the dual-ported register files used on the DCA boards.

If the PC access is to the IBM screen buffer, $\overline{\text{IO\_ACCESS}}$ will not be asserted out of the MPA-II_PD PAL. The MPA-II_PD PAL will, when $\overline{\text{LCL}}$ goes high on the remote access, force A15 low and pass the buffered address lines A12–8 onto the data RAM. Address lines A14 and A13 are implemented through U8, MPA-II_CT (MPA-II Control Timing) PAL. PC address lines A7–0 are buffered by U14, a 74ALS541 and passed onto the BCP data memory address lines AD7–0 when $\overline{\text{LCL}}$ switches high for the remote access. The data memory RAM's chip select, $\overline{\text{DMEM\_CS}}$, is asserted on any remote access. If the BCP's $\overline{\text{LCL}}$ output goes high, $\overline{\text{DMEM\_CS}}$ will be asserted low; on a local access, this signal will be asserted if the BCP's A15 signal is low (RAM occupies the lower half of the BCP's memory map).

This scenario for remote accesses works because RAM is the only element external to the BCP that is visible to the PC. If the PC is accessing the BCP (RIC, the Program Counter, or Instruction Memory), the BCP's READ/WRITE strobes will not be asserted to the data RAM. On a PC access of the BCP's RIC register, for example, data RAM will be selected and the CMD (CoMmanD) output of the MPA-II_RD PAL will be asserted to the BCP, selecting the BCP's RIC. No bus collision will occur on a read or data inadvertently destroyed on a write because the BCP will not assert the external strobes on an internal register access.

The MPA-II_RD PAL also combines the memory and I/O read/write strobes to form the $\overline{\text{REMRD}}$/$\overline{\text{REMWR}}$ strobes to the rest of the MPA-II system. Since PC bus cycles can only be validated by the assertion of one of these strobes, this PAL makes the final decision on the validity of the bus cycle. If the PC cycle is a valid access of the BCP system, this PAL will assert $\overline{\text{RAE}}$ (Remote Access Enable), the BCP's chip select. RIC, the output CMD, and the BCP's READ/WRITE strobes will determine which part of the system receives or provides data.

The PC IRQ interrupt for the IBM interface is set and cleared by the BCP through U3, the MPA-II_AC (Auxiliary Control) PAL. The interrupt is set from the BCP by pointing data memory to an address in the range A000–BFFF (see Table 5-3), and writing to this location with AD7 set high; it is likewise cleared by writing with AD7 low to this location. The interrupt powers up low (deasserted) and can be assigned to PC interrupts IRQ2, 3, or 4 by setting the appropriate jumper (JP4–6).

Remote accesses of the BCP are arbitrated and handled by the Remote Interface and Arbitration System (RIAS) control logic. The arbiter sequential state machine internal to the BCP shares the same clock with the CPU, but otherwise operates autonomously. This unis is very flexible and offers a number of configurations for different external interfaces (see the Remote Interface and Arbitration System chapter of the BCP data book). We chose to use the Fast Buffered Write/Latched Read interface configuration to maximize the possible data transfer rate and minimize the BCP performance degradation by the slower PC bus cycles. Data is buffered between the PC and BCP data buses with U18, a 74ALS646, giving us a monolithic, bidirectional transceiver with latches for PC reads and buffering for PC writes.

Rest Time Circuit

To support the newer high performance PC AT compatibles entering the market, a rest time circuit is implemented on the MPA-II. The purpose of this circuit is to prevent two remote accesses made by a high performance PC from being mistaken as one remote access. (For a detailed description of BCP remote rest time, refer to the Remote Interface and Arbitration System section of the DP8344A data sheet).

The rest time circuit is implemented in one PAL16RA8, MPA-II_RI, U4. This rest time circuit implements all modes except Latched Write and does not take advantage of the increase in speed possible when CMD does not change from one access to the next.

First, how the REM_ENABLE signal controls remote accesses will be discussed. Then, a description of the operation of the rest time state machine in the PAL16RA8 will be given.

The REM_ENABLE signal is produced in the rest time PALRA8 and is low during rest time. After rest time is over the REM_ENABLE signal goes high until the end of the next access, when it once again goes low during rest time. The signal REM_ENABLE is fed back into MPA-II_RD, U7.

Through the rest time circuit, both $\overline{\text{REMRD}}$ and $\overline{\text{REMWR}}$ are held high when REM_ENABLE = 0. This prevents all remote accesses during rest time. When rest time is over REM_ENABLE = 1 and then decodes of $\overline{\text{MEMW}}$, $\overline{\text{MEMR}}$, $\overline{\text{IOW}}$, and $\overline{\text{IOR}}$ control $\overline{\text{REMRD}}$ and $\overline{\text{REMWR}}$ respectively.

To describe the operation of the state machine, a state by state description follows. When reading through the states one should remember that the state machine can only change states on the rising edge of CLK-OUT.

STATE: IDLE

This state is entered when a system reset occurs. In this state REM_ENABLE = 1, and XACK controls the state of $\overline{\text{PC\_RDY}}$.

The state machine will stay in this state until a valid remote access starts (i.e. $\overline{\text{RAE}}$ = 0). Then the state machine moves to CYCLE_START.

NOTE: The signal $\overline{\text{RAE}}$ is a full decode of a valid access by MPA-II_RD, U7. If $\overline{\text{RAE}}$ is only an address decode, it alone would not indicate that a valid access has started.

STATE: CYCLE_START

In this state, REM_ENABLE = 1 and XACK controls the state of $\overline{\text{PC\_RDY}}$. The state machine will stay in this state until the remote access ends, indicated by $\overline{\text{RAE}}$ = 1. Then the state machine moves to WAIT1.

STATE: WAIT1

In this state, REM_ENABLE = 0 and, if a remote access starts, the $\overline{PC\_RDY}$ is driven low whenever $\overline{RAE}$ = 0. After one CLK-OUT cycle the state machine moves to WAIT2.

STATE: WAIT2

In this state, REM_ENABLE = 0 and $\overline{PC\_RDY}$ is driven low whenever $\overline{RAE}$ = 0. After another CLK-OUT cycle the state machine moves to WAIT3.

STATE: WAIT3

In this state, REM_ENABLE = 0 and $\overline{PC\_RDY}$ is driven low whenever $\overline{RAE}$ = 0. After another CLK-OUT cycle the state machine moves to WAIT4.

STATE: WAIT4

In this state, REM_ENABLE = 0 and $\overline{PC\_RDY}$ is driven low whenever $\overline{RAE}$ = 0. After another CLK-OUT cycle the state machine moves to WAIT5.

STATE: WAIT5

In this state, REM_ENABLE = 0 and $\overline{PC\_RDY}$ is driven low whenever $\overline{RAE}$ = 0. If the BIRQ signal is still active low, indicating that BIRQ has not been serviced yet by the BCP interrupt software, then the state machine will continue to loop in this state until BIRQ goes inactive high. This will prevent the PC from gaining access to the BCP's memory (Dual Port or I/O), thus "locking out" the PC if it attempts another access. A write to the MPA Access register, U17, which will toggle AREG_CLK~, will cause BIRQ to go inactive high, thus "unlocking" the PC. In this way the MPA-II hardware will lock out the PC until the BCP interface software has time to gain control and emulate the DCA or IBM register hardware. This feature allows the MPA-II to implement future IBM I/O register changes by simply updating the BCP software. If BIRQ was not active low or when it goes inactive high, the next state is WAIT6.

STATE: WAIT6

In this state, REM_ENABLE = 0 and $\overline{PC\_RDY}$ is driven low whenever $\overline{RAE}$ = 0. If a remote access has started (i.e., $\overline{RAE}$ = 1) the next state will be RESUME. Otherwise, the next state is WAIT7.

STATE: WAIT7

In this state, REM_ENABLE = 0 and $\overline{PC\_RDY}$ is driven low whenever $\overline{RAE}$ = 0. If a remote access has started (i.e., $\overline{RAE}$ = 1) the next state will be RESUME. Otherwise, the next state is WAIT8.

STATE: WAIT8

In this state, REM_ENABLE = 1 (allowing accesses) and $\overline{PC\_RDY}$ is driven low whenever $\overline{RAE}$ = 0. This state was included in the state machine to reduce the state machine's logic. Otherwise it would have been logical to return to the IDLE state from WAIT7 if $\overline{RAE}$ = 1 (no access in progress). If $\overline{RAE}$ = 0, then the next state will be RESUME. Otherwise, the state machine returns to IDLE.

STATE: RESUME

In this state, REM_ENABLE = 1 and $\overline{PC\_RDY}$ is driven low while $\overline{RAE}$ = 0. When the state machine moves to this state, it means that a remote access took place quickly after the previous access. The state machine allows the remote access to proceed since the PC-bus has been waited long enough by the previous states. However, the PC-bus must be waited until the XACK signal can take over control of driving $\overline{PC\_RDY}$. For the design of the MPA-II, once REM_ENABLE = 1, then the XACK signal would take over control within two CLK-OUT cycles. So the state machine will wait the PC-bus through this state and the next. On the next rising edge of CLK-OUT the state machine will move to the HOLD state.

STATE: HOLD

In this state, REM_ENABLE = 0, and $\overline{PC\_RDY}$ is driven low whenever $\overline{RAE}$ = 0. Again, this state is provided to wait the PC-bus for a second CLK-OUT cycle while still allowing the remote access. The next state is CYCLE_STATE. In CYCLE_START, XACK will take over control of $\overline{PC\_RDY}$.

The BCP BIRQ Interrupt

The BCP's bi-directional pin, BIRQ, is configured as an interrupt into the BCP, and is set on the trailing edge of a PC write of the BCP I/O register space (excluding RIC and the Segment Register, i.e., I/O addresses 2DFh and 2D7h, respectively). The BCP can identify which I/O register has been accessed by reading the Access Register, U17, a 74ALS574, mapped directly above the dual-ported RAM in the BCP's data memory map (see Table 5-3). The bits AD5–0 are the last 6 bits of the I/O register's address. A BCP write to this register will clear BIRQ, and therefore, the BCP interrupt. Timing for the clock enable of U17 is provided by the MPA-II_CT PAL, U8. U17 is clocked only on remote writes to the I/O register page (denoted by $\overline{IO\_ACCESS}$ being asserted from the MPA-II_RD PAL) and local BCP writes of U17. The BCP uses the BIRQ interrupt in order to service the PC in a timely manner since the PC is locked out until the BCP software unlocks the PC. After an MPA-II reset, or when the BCP writes a zero to AD5 of the Auxiliary Control Register (address A000h), called the BIRQ_EN line, then the BIRQ line is disabled. While BIRQ_EN is low (inactive) the PAL MPA-II_RI does not lock out the PC, nor does it assert the BIRQ line.

Front-End Interface

The line interface is divided into coax/twisted pair and twinax sections, each section being comprised of an interface connector, receiver, and driver logic. These sections are independent but are never operated concurrently. The coax medium requires a transformer-coupled interface while the multi-drop twinax medium is directly coupled to each device.

The transmitter interface on the DP8344A is sufficiently general to allow use in 3270, 5250, and 8-bit transmission systems. Because of this generality, some external hardware is needed to adapt the outputs to form the signals necessary to drive the twinax line. The chip provides three signls. $\overline{DATA\text{-}OUT}$, DATA-DLY, and TX-ACT. $\overline{DATA\text{-}OUT}$ is biphase serial data (inverted). DATA-DLY is the biphase serial data output (non-inverted) delayed one-quarter bit-time. TX-ACT, or transmitter active, signals that serial data is being transmitted when asserted. TX-ACT functions as an external transmitter enable. The BCP can invert the sense of the $\overline{DATA\text{-}OUT}$ and DATA-DLY signals by asserting TIN {TMR[3]}. This feature allows both 3270 and 5250 type biphase data to be generated, and/or utilization of inverting or non-inverting transmitter stages.

The line drivers are software selectable from the BCP via logic embedded in the MPA-II_AC and MPA-II_CT PALs.

Table 5-3 reveals that the Auxiliary Control Register is mapped into the A000–BFFF area of the BCP memory map. The coax/twisted pair module is selected by pointing to this address area and writing a "0" out on the AD6 data line. The twinax is selected by writing a "1" on this signal. The coax/twisted pair section is selected on power-up. The voltage supervisor described earlier in the Reset Control section also plays a role here, deactivating the line drivers of both sections if the +5V supply drops more than 10% at any time. The receivers are selected on-board the BCP by the SLR (Select Line Receiver) control in the Transceiver Control Register. Setting {TCR[5]} to a "1" selects the on-chip comparator and thus the coax input; a "0" on this control selects the TTL-IN receiver input for the twinax input.

Coax/Twisted Pair Interface

At this date, the largest installed base of terminals is the 3270 protocol terminal which primarily utilizes coax cabling. Because of phone wire's easy accessibility and lower cost, twisted pair cabling has become popular among end users for new terminal installations. In the past, baluns have been used to augment existing coax interfaces, but their poor performance and cost considerations leave designers seeking new solutions. In addition, the integration of coax and twisted pair on the same board has become a market requirement, but this is a considerable design challenge. A brief summary of the combined coax/twisted pair interface concepts, a discussion of the design, and a description of the results follows.

The concepts which must be addressed by the combined coax/twisted pair interface will be discussed at this time. These concepts are important to understand why the various design decisions are implemented in the interface. Coax cable is normally driven on the center conductor with the shield grounded. Conversely, unshielded twisted pair cable is driven on both lines. Because of the way that each is driven, coax operation is often called unbalanced and twisted pair operation balanced.

Transmission line characteristics of coax and twisted pair cables can be envisioned as essentially those of a low-pass filter with a length-dependent bandwidth. In 3270 systems, different data combinations generate dissimilar transmission frequencies because of the Manchester format. These two factors combine to produce data pulse widths that vary according to the data transmitted and the length and type of cable used. This pulse-width variation is often described as "data jitter".

In addition to line filtering, noise can cause jitter. Coax cable employs a shield to isolate the signal from external noise Electromagnetically balanced lines minimize differential noise in unshielded twisted pair cable. In other words, the twisted pair wires are theoretically equidistant from any noise source, and all noise super-imposed on the signal should be the common-mode type. Although these methods diminish most noise, they are not totally effective, and environmental interference from other nearby wiring and circuitry may still cause problems.

Besides the effects of jitter, reflections can produce undesirable signal characteristics that introduce errors. These reflections may be caused by cable discontinuities, connectors, or improper driver and receiver matching. Signal edge rates may aggravate reflection problems since faster edges tend to produce reflections that may dramatically distort the signal. Most reflection difficulties occur over short cable (less than 150 ft.) because at these distances reflections suffer little attenuation and can significantly distort the signal. Since the timing of the reflections is a function of cable length, it may be possible to operate at some short distance and not at some greater length.

An effective receiver design must address each of the above concerns. To counteract the effects of line filtering and noise, there must be a large amount of jitter tolerance. Some filtering is needed to reduce the effects of environmental noise caused by terminals, computers, and other proximate circuitry. At the same time, such filtering must not introduce transients that the receiver comparator translates into data jitter.

Like the receiver design, a successful driver design should compensate for the filtering effects of the cable. As cable length is increased, higher data frequencies become attenuated more than lower frequency signals, yielding greater disparity in the amplitudes of these signals. This effect generates greater jitter at the receiver. The 3270 signal format allows for a high voltage (predistorted) magnitude followed by a low voltage (nondistorted) magnitude within each data half-bit time. Increasing the predistorted-to-nondistorted signal level ratio counteracts the filtering phenomenon because the lower frequency signals contain less predistortion than do higher frequency signals. Thus, the amplitude of the higher frequency components are greater than the lower frequency components at the transmitter. Implementation of this compensation technique is limited because nondistorted signal levels are more susceptible to reflection-induced errors at short cable lengths. Consequently, proper impedance matching and slower edge rates must be utilized to eliminate as much reflection as possible at these lengths.

Besides improved performance, both unbalanced and balanced operation must be adequately supported. Electromagnetic isolation for coaxial cabling can be provided by a properly grounded shield. Electrically and geometrically symmetric lines must be maintained for twisted pair operation. For both cable types, proper termination should be employed, although terminations slightly greater than the characteristic impedance of the line may actually provide a larger received signal with insignificant reflection. In the board layout, the comparator traces should be as short as possible. Lines should be placed closely together along their entire path to avoid the introduction of differential noise. These traces should not pass near high frequency lines and should be isolated by a ground plane.

An extensive characterization of the BCP comparator was done to facilitate this interface design. The design enhances some of the BCP transceiver's characteristics and incorporates the aforementioned suggestions.

The interface design takes into account the common comparator attributes of power supply rejection, variable switching offset, finite voltage sensitivity, and fast edge rate sensitivity. V_{CC} noise can effect the comparator output when the inputs are biased to the same voltage.

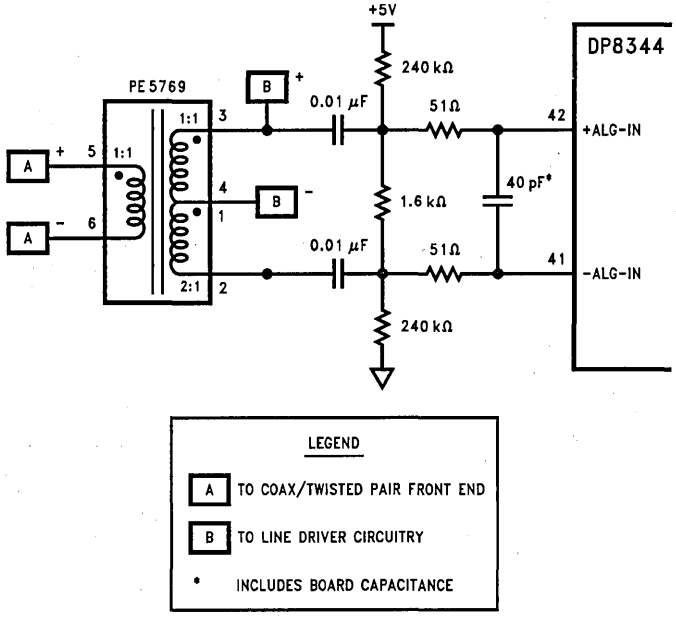
This particular type of biasing may render portions of the comparator susceptible to supply noise. Variable switching offset and finite voltage sensitivity cause the receiver de-

coding circuitry to see a substantial amount of data jitter when signal amplitudes approach the sensitivity limits of the comparator. At these signal magnitudes, considerable variation in the output of the comparator is observed. Finally, edge sensitivity may allow a fast edge to introduce errors as charge is coupled through the inputs during a rapid predistorted-to-nondistorted level transition, especially as the nondistorted level is reduced in magnitude.

The receiver interface design *(Figure 5-4)* addresses each of the BCP comparator's characteristics. A small offset (about 17 mV) separates the inputs to eliminate V_{CC}—coupled noise. This offset is relatively large compared to possible fabrication variations, resulting in a more consistent, device-independent operation. The offset has the added benefit of making the comparator more immune to ambient noise that may be present on the circuit board. A 2:1:1 transformer (arranged as a 3:1) restores any voltage sensitivity lost by introducing the offset. A bandpass filter is employed to reduce the edge rate of the signal at the comparator and to eliminate environmental noise. The bandwidth (30 kHz to 30 MHz) was chosen to provide sufficient noise attenuation while producing minimum data jitter. Refer to Appendix C for a derivation of the filter equations.

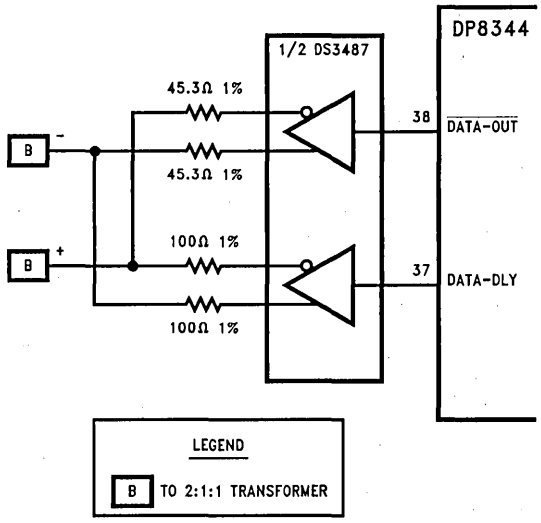
Like many present 3270 circuits, the driver design *(Figure 5-5)* utilizes a National Semiconductor DS3487 and a resistor network to generate the proper signal levels. The predistorted-to-nondistorted ratio was chosen to be about 3 to 1. This ratio was observed to offer good noise immunity at short cable lengths (less than 100 feet) and error-free transmission to an IBM 3174 controller at long cable lengths (greater than 5000 feet).

To allow for two interfaces in the same circuit design, the coax/twisted pair front end *(Figure 5-6)* includes an ADC Telecommunications brand TPC connector to switch between coax and twisted pair cable. This connector allows different male connectors for coax and twisted pair cable to switch in different interfaces for the particular cable type. The coax interface has only the shield capacitively coupled to ground. The 510Ω resistor and the filter loading produce a termination of about 95Ω. The twisted pair interface balances both lines and possesses an input impedance of about 100Ω. This termination is somewhat higher than the characteristic impedance (about 96Ω) of twisted pair. Terminations of this type produce reflections that do not tend to generate mid-bit errors, as well as having the benefit of creating a larger voltage at the receiver over longer cable lengths.



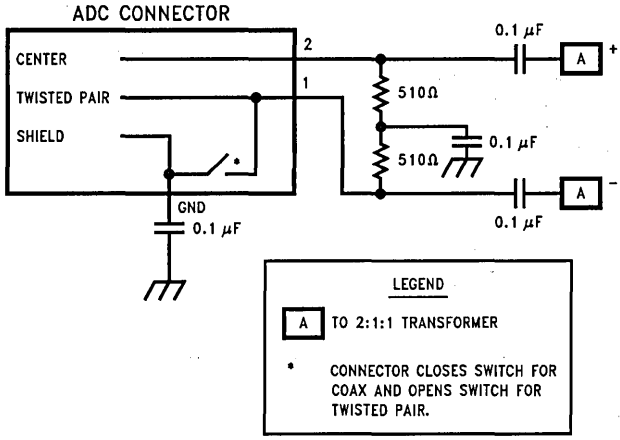
TL/F/10488-20

FIGURE 5-4. BCP Receiver Interface Design



TL/F/10488-21

FIGURE 5-5. BCP Driver Design



TL/F/10488-32

FIGURE 5-6. BCP Coax/Twisted Pair Front End

The performance of the combined coax/twisted pair interface is impressive. Performance of the BCP interface typically extended over 7000 feet of RG62A/U coax and 1700 feet of AT&T DIW 4 pair/24 AWG unshielded twisted pair. This operation met or exceeded many of the current 3270 solutions. The performance of other 3270 products was obtained from production stock of competitors' equipment and should be taken as typical operation. Although these long distances are possible, it is recommended that companies specify their products to IBM's PAI specifications of 5000 feet of coax cable. The extra long distance capability of the new interface will assure the designer a comfortable guardband of performance. Similarly, 50% margin on the unshielded twisted pair capability will approximately match the 900 foot specification.

On the MPA-II as much attention has been paid to the layout as to the interface design. The traces from the BNC/Twisted Pair ADC connector to the BCP's analog comparator were made as wide as possible, placed as close together as practical, and kept on the same side of the board. The ground plane has been placed directly under these traces. All digital lines have been kept as far away as practical. Finally, the ground plane has been partially split, keeping all the analog interface grounds on one part of the ground plane, including the BCP ground pin 43; and all of the digital logic ground pins on the other side. See Appendix A for the actual layout of the MPA-II.

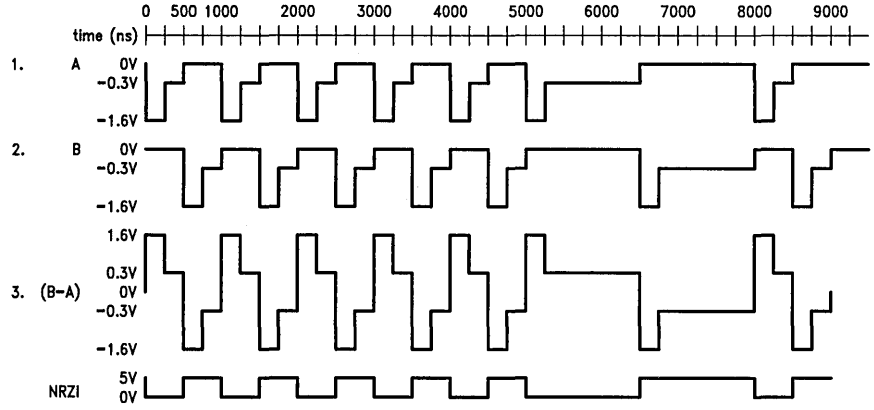
Twinax (5250) Interface

The 5250 transmission system is implemented in a balanced current mode; every receiver/transmitter pair is directly coupled to the twinax at all times. Data is impressed on the transmission line by unbalancing the line voltage with the driver current. The system requires passive termination at both ends of the transmission line. The termination resistance value is given by:

R_t: $= Z_0/2$; where
R_t: = Termination Resistance
Z_0: = Characteristic Impedance

In practice, termination is accomplished by connecting both conductors to the shield via 54.9Ω, 1% resistors; hence the characteristic impedance of the twinax cable of 107Ω ±5% at 1.0 MHz. Intermediate stations must not terminate the line; each is configured for "pass-through" instead of "terminate" mode. Stations do not have to be powered on to pass twinax signals on to other stations; all of the receiver/transmitter pairs are DC coupled. Consequently, devices must never output any signals on the twinax line during power-up or down that could be construed as data, or interfere with valid data transmission between other devices. The MPA-II board is factory set to "terminate" mode. To effect "pass-through" mode, jumpers JP2 and JP3 must be removed.

The bit rate utilized in the 5250 protocol is 1 MHz ±2% for most terminals, printers and controllers. The IBM 3196 display station has a bit rate of 1.0368 MHz ±0.01%. The data are encoded in biphase, NRZI (non-return to zero inverted) manner; a "1" bit is represented by a positive to negative transition, a "0" is a negative to positive transition in the center of a bit cell. This is opposite from the somewhat more familiar 3270 coax method. The biphase NRZI data is encoded in a pseudo-differential manner; i.e., the signal on the "A" conductor is subtracted from the signal on "B" to form the waveform shown in *Figure 5-7*. Signals A and B are not differentially driven; one phase lags the other in time by 180 degrees. *Figures 5-8* and *5-9* show actual signals taken at the driver and receiver after 5000 ft. of twinax, respectively.



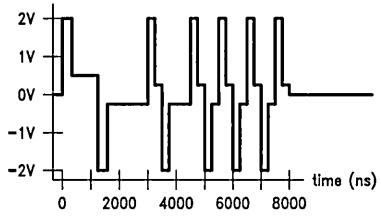
TL/F/10488-12

Note 1: The signal on phase A is shown at the initiation of the line quiesce/line violation sequence.

Note 2: Phase B is shown for that sequence, delay in time by 500 ns.

Note 3: The NRZI data recovered from the transmission.

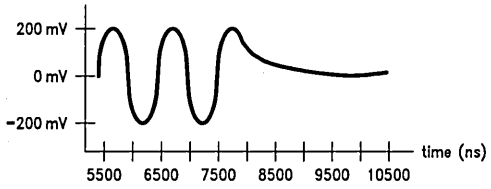
FIGURE 5-7. Twinax Waveforms



TL/F/10488–13

The signal shown was taken with channel 1 of an oscilloscope connected to phase B, channel 2 connected to A, and then channel 2 inverted and added to channel 1.

FIGURE 5-8. Signal at the Driver



TL/F/10488–14

The signal shown was viewed in the same manner as *Figure 5-8*. The severe attenuation is due to the filtering affect of 5000 ft. of twinax cable.

FIGURE 5-9. Signal at the Receiver

The signal on either the A or B phase is a negative going pulse with an amplitude of −0.32V ±20% and duration of 500 ±20 ns. During the first 250 ±20 ns, a pre-distortion or pre-emphasis pulse is added to the waveform yielding an amplitude of −1.6V ±20%. When a signal on the A phase is considered together with its' B phase counterpart, the resultant waveform represents a bit cell or bit time, comprised of two half-bit times. A bit cell is 1 μs ±20 ns in duraction and must have a mid bit transition. The mid bit transition is the synchronizing element of the waveform and is key to maintaining transmission integrity throughout the system. The maximum length of a twinax line is 5000 ft. and the maximum number of splices in the line is eleven. Devices count as splices, so that with eight devices on line, there can be three other splices. The signal 5000 ft. and eleven splices from the controller has a minimum amplitude of 100 mV and a slower edge rate. The bit cell transitions have a period of 1 μs ±30 ns.

The current mode drive method used by native twinax devices has both distinct advantages and disadvantages. Current mode drivers require less power to drive properly terminated, low-impedance lines than voltage mode drivers. Large output current surges associated with voltage mode drivers during pulse transition are also avoided. Unwanted current surges can contribute to both crosstalk and radiated emission problems. When data rate is increased, the surge time (representing the energy required to charge the distributed capacitance of the transmission line) represents a larger percentage of the driver's duty cycle and results in increased total power dissipation and performance degradation.

A disadvantage of current mode drive is that DC coupling is required. This implies that system grounds are tied together from station to station. Ground potential differences result in ground currents that can be significant. AC coupling removes the DC component and allows stations to float with respect to the host ground potential. AC coupling can also be more expensive to implement.

Twinax signals can be viewed as consisting of two distinct phases, phase A and phase B, each with three levels: off, high, and low. The off level corresponds with 0 mA current being driven, the high level is nominally 62.5 mA, +20% − 30%, and the low level is nominally 12.5 mA, +20% − 30%. When these currents are applied to a properly terminated transmission line the resultant voltages impressed at the driver are: off level is 0V, low level is 0.32V ±20%, high level is 1.6V ±20%. The interface must provide for switching of the A and B phases and the three levels. A bi-modal constant current source for each phase can be built that has a TTL level interface for the BCP.

The MPA-II's twinax line drivers are current mode driver parts available from National Semiconductor and Texas Instruments. The 75110A and 75112 can be combined to provide both the A and B phases and the bi-modal current drive required. The MPA-II_AC PAL adapts the BCP outputs to the twinax interface circuit and prevents spurious transmissions during power-up or down. The serial NRZ data is inverted prior to being output by the BCP by setting TIN, {TMR[3]}.

The pseudo-differential mode of the twinax signals make receiver design requirements somewhat different than that of the coax circuit. Hence, the analog receiver on the BCP is not used. The BCP provides both analog inputs to an on-board comparator circuit as well as a TTL level serial data input, TTL-IN. The sense of this serial data can be inverted in software by asserting RIN, {TMR[4]}.

The external receiver circuit must be designed with care to assure reliable decoding of the bit-stream in the worst environments. Signals as small as 100 mV must be detected. In order to receive the worst case signals, the input level switching threshold or hysteresis for the receiver should be nominally 29 mV ±20%. This value allows the steady state, worst case signal level of 100 mV, 66% of its amplitude before transitioning.

To achieve this, the National Semiconductor LM361 was chosen, a differential comparator with complementary outputs. The complementary outputs are useful in setting the hysteresis or switching threshold to the appropriate levels. The LM361 also provides excellent common mode noise rejection and a low input offset voltage. Low input leakage current allows the design of an extremely sensitive receiver without loading the transmission line excessively. In addition to good analog design techniques, a passive, single-pole, low pass filter with a roll-off of approximately 1 MHz was applied to both the A and B phases. This filter essentially conducts high frequency noise to the opposite phase, effectively making the noise common mode and easily rejectable.

Design equations for the LM361 in a 5250 application are shown here for example. The hysteresis voltage, V_h, can be expressed the following way:

$$V_H = V_{RIO} + ((R_{IN} / (R_{IN} + R_f) * V_{OH}) - (R_{IN} / (R_{IN} + R_f) * V_{OL}))$$

where:

V_H — Hysteresis Voltage, Volts
R_{IN} — Series Input Resistance, Ohms
R_F — Feedback Resistance, Ohms
C_{IN} — Input Capacitance, Farads
V_{RIO} Receiver Input Offset Voltage, Volts
V_{OH} — Output Voltage High, Volts
V_{OL} — Output Voltage Low, Volts

The input filter values can be found through this relationship:

$$V_{CIN} = V_{IN1} - V_{IN2/1} + jwC_{IN}(R_{IN1} + R_{IN2})$$

where $R_{IN1} = R_{IN2} = R_{IN}$:

$$F_{ro} = w/2c$$

$$F_{ro} = 1/(2c * R_{IN} * C_{IN})$$

$$C_{IN} = 1/(2c * R_{IN} * F_{ro})$$

where

V_{IN1}, V_{IN2} —Phase A and B Signal Voltages, Volts
V_{CIN} —Voltage Across C_{IN}, or the Output of the Filter, Volts
R_{IN1}, R_{IN2} —Input Resistor Values, $R_{IN1} = R_{IN2}$, Ohms
F_{ro} —Roll-Off Frequency, Hz
w —Frequency, Radians

The roll-off frequency, F_{ro}, should be set nominally to 1 MHz to allow for transitions at the transmission bit rate. The transition rate when both phases are taken together is 2 MHz, but then both R_{IN1} and R_{IN2} must be considered, so:

$$F_{ro2} = 1/(2c * (R_{IN1} + R_{IN2}) * C_{IN})$$

or,

$$F_{ro2} = 1/(2c * 2 * R_{IN} * C_{IN})$$

where $F_{ro2} = 2 * F_{ro}$, yielding the same results.

Table 5-4 shows the range of values expected.

Advanced Features of the BCP

The BCP has a number of advanced features that give designers flexibility to adapt products to a wide range of IBM environments. Besides the basic multi-protocol capability of the BCP, the designer may select the inbound and outbound serial data polarity, the number of received and transmitted line quiesces, and in 5250 modes, a programmable extension of the TX-ACT signal after transmission.

The polarity selection on the serial data stream is useful in building single products that handle both 3270 and 5250 protocols. The 3270 biphase data is inverted with respect to 5250.

Selecting the number of line quiesces on the inbound serial data changes the number of line quiesce bits that the receiver requires before a line violation to form a valid start sequence. This flexibility allows the BCP to operate in extremely noisy environments, allowing more time for the transmission line to charge at the beginning of a transmission. The selection of the transmitted line quiesce pattern is not generally used in the 5250 arena, but has applications in 3270. Changing the number of line quiesces at the start of a line quiesce pattern may be used by some equipment to implement additional repeater functions, or for certain inflexible receivers to sync up.

TABLE 5-4. Twinax Receiver Design Values

Value	Maximum	Minimum	Nominal	Units	Tolerance
R_{IN}	4.935E+03	4.465E+03	4.700E+03	Ohms	0.5
R_F	8.295E+05	7.505E+05	7.900E+05	Ohms	0.5
C_{IN}	4.4556E−11	2.6875E−11	3.3863E−11	Farads	
V_{OH}	5.250E+00	4.750E+00	5.000E+00	Volts	
V_{OL}	4.000E−01	2.000E−01	3.000E−01	Volts	
V_{IN}	+1.920E+00	1.000E−01		Volts	
V_{IN}	−1.920E+00	1.000E−01		Volts	
V_{RIO}	5.000E−03	0.000E+00	1.000E−03	Volts	
R	6.533E−03	5.354E−03	5.914E−03	Ohms	
F_{ro}	1.200E+06	8.000E+05	1.000E+06	Hz	0.2
V_H	3.368E−02	2.691E−02	2.880E−02	Volts	
X_c	7.4025E+03	2.9767E+03	4.7000E+03	Ohms	

The most important advanced feature of the BCP for 5250 applications is the programmable TX-ACT extension. This feature allows the designer to vary the length of time that the TX-ACT signal from the BCP is active after the end of a transmission. This can be used to drive one phase of the twinax line in the low state for up to 15.5 μs. Holding the line low is useful in certain environments where ringing and reflections are a problem, such as twisted pair applications. Driving the line after transmitting assures that receivers see no transitions on the twinax line for the specified duration. The transmitter circuit can be used to hold either the A or B phase by using the serial inversion capability of the BCP in addition to swapping the A and B phases. Choosing which phase to hold active is up to the designer, 5250 devices use both. Some products hold the A phase, which means that another transition is added after the last half bit time including the high and low states, with the low state held for the duration. Alternatively, some products hold the B phase. Holding the B phase does not require an extra transition and hence is inherently quieter.

To set the TX-ACT hold feature, the upper five bits of the Auxiliary Transceiver Register, {ATR[3–7]}, are loaded with one of thirty two possible values. The values loaded select a TX-ACT hold time between 0 μs and 15.5 μs in 500 ns increments.

The connectors called out in the IBM specifications for the twinax medium are too bulky to mount directly to a PC board, therefore a 9-pin D subminiature connector is provided. This connector is then attached to a cable assembly consisting of a 1 foot section of twin-axial cable with the opposite gender 9-pin on one end and a twinax "T" connector on the other. This is then spliced into the twinax multidrop trunk.

Miscellaneous Support

The remaining components of the MPA-II will be covered in the following section, including the board itself and decoupling capacitors.

The system is implemented on a four-layer substrate, using minimum 8 mil trace widths/spacing for all signals except the analog traces in the front-end. Here we specified minimal trace lengths and 55–80 mil trace widths. The traces from the BNC/Twisted Pair ADC to the BCP's analog comparator were made as wide as possible, placed as close together as practical, and kept on the same side of the board. The ground plane has been placed directly under these traces. All digital lines have been kept as far away as practical. Finally, the ground plane has been partially split, keeping all the analog interface grounds on one part of the ground plane, including the BCP ground pin 43; and all of the digital logic ground pins on the other side. See Appendix A for the actual layout of the MPA-II. These fairly common analog layout techniques are justified due to the complexity and power level of the analog waveforms present in the line interface.

Each device has one 0.1 μF decoupling capacitor located as close as possible to the chip. These are chip capacitors (0.3 spacing, DIP configuration) to minimize lead length inductance and facilitate placement. The +5V supply line has two 22 μF electrolytic capacitors, one at each end of the board. The other three supply lines (−5V, +12V, −12V) drive only the twinax analog circuitry, and are bypassed with 10 μF electrolytics where they come on to the board and 0.1 μF chip caps at the device(s). The BCP requires additional decoupling due to the large number of outputs, high frequency operation, and CMOS switching characteristics. We used a capacitor near each ground of the BCP. These decoupling capacitors, together with the ground and power planes of the multi-layer board, provide effective supply isolation from the switching noise of the circuitry.

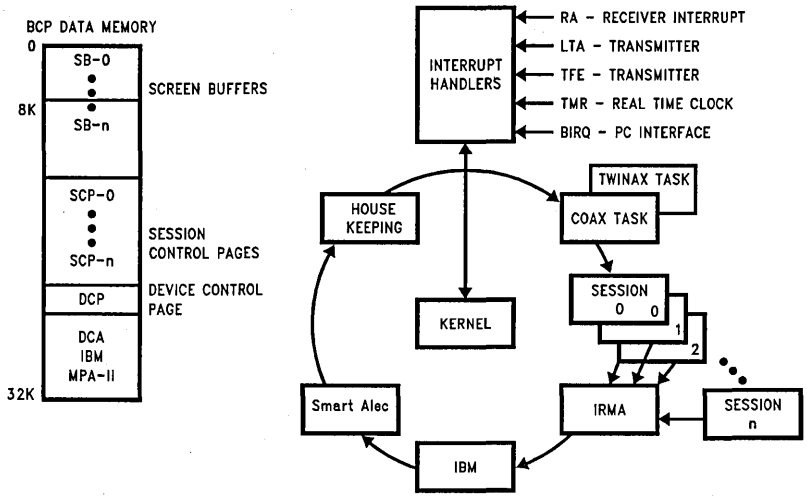
6.0 MPA-II SOFTWARE ARCHITECTURE

The primary goal of the MPA-II design was to accommodate multiple industry standard interfaces and protocol modes within a single, integrated structure (see *Figure 6-1*). The MPA-II software supports 3270, 3299, 5250, and all the PC interfaces in its 8k instruction memory bank, The system is configured at load time for the different options, and may be reconfigured "on the fly" by simply writing the new configuration byte into the MPA-II configuration I/O register (2DCh). New tasks may be added to and old tasks removed from the MPA-II system easily. The modular organization of the system allows for simple maintenance and enhancement.

The basic concepts employed in the software design are: modularity, comprehensive data structures, and round-robin task scheduling. The system has been designed to allow modules to be written and integrated into the system by different groups. In the case of the National Semiconductor team developing the MPA-II, different groups developed the 3270 and 5250 software modules. Some modules were set up in advance of any protocol development and have been the basis of the software development. The KERNEL.BCP module contains the task switching and scheduling routines. The header files MPA.HDR and DATARAM.HDR contain the basic global symbolic equates and data structures. DATARAM.HDR is organized such that the BCP's data RAM may be viewed through a number of templates, or maps. In other words, except for specific hardware devices mapped into memory, there are no hard coded RAM addresses. The 8k dual-port block is fixed at the top of RAM, and the PC I/O space is mapped into the upper page of installed RAM, but the locations of screen buffers and variable storage are all determined through the set of templates used. The templates serve only to cause the assembler to produce relative offsets. The software developer chooses which base physical address to reference the offset to in order to address RAM. Usually, a pointer to RAM is set up in the IZ register pair, and the data are referenced by the assembler mnemonics. For example:

```
MOVE[IZ+control_reg], rd
```

where: control_reg is a symbolic template offset.
rd is a destination register



TL/F/10488-17

FIGURE 6-1. MPA-II Software Architecture

This scheme allows the actual locations of data structures to move based on the system mode and current addressed device. This also allows the use of the dual-port RAM to change with the interface mode or protocol mode.

The MPA.HDR module is included (via the .INPUT assembler directive) in every module for use in the MPA-II system, regardless of protocol or interface mode. MPA. HDR defines specific hardware related constants such as RAM size, hardware I/O locations, etc . . . MPA.HDR in turn includes: MACRO.HDR, which contains commonly used macros; BCP.HDR, which defines specific bits and bit fields for BCP registers; STDEQU.HDR, which contains BCP and assembler specific declarations (it is included with the BCP Assembler System); and DATARAM.HDR, which contains the general RAM templates. Equate files for specific functions such as twinax, coax, and the different interfaces are included where needed. The Kernel module contains the basic software structures which support all system activities. System initialization, scheduling tasks, re-configuration and halting the system all fall under its jurisdiction. All tasks are called from the Kernel and return to it.

A number of rules have been adhered to during the MPA-II software development. These can best be discussed by referring to the BCP register allocation shown in *Figure 6-2*. The interrupt handlers are all considered background tasks. All 3270 "busy" type processing, 5250 command processing, and system functions are foreground tasks. The Main and Alternate banks are reserved for foreground and background functions, respectively. In addition, the index registers IW and IX are reserved for the background functions. The index registers IY and IZ are reserved for the foreground functions. "Reserved" means that the background routines promise to save and restore registers reserved for the foreground routines and that the foreground routines promise not to modify or rely upon registers reserved for the background routines. This system of reserving registers allows for extremely fast context switching since interrupt (background) routines only need to save and restore certain registers, (usually only IZ). The IZ pointer is generally used as the base pointer for all templates used by the tasks and interrupts. All foreground tasks are restricted to six levels of nesting to prevent the address stack from overflowing. Interrupt handlers are limited to three levels. Interrupts are generally not interruptable. Some special cases exist, and they are detailed later in this document.

The R20 and R21 registers are permanently reserved for the system. R20 is used as the R_CONFIG storage, or the current configuration state of the MPA-II (e.g., Coax/IRMA). R21 is the R_TASK register as defined by the Kernel. The Kernel uses this register as its task list, with scheduled tasks signified by their corresponding bits set and un-scheduled tasks' bits cleared.

Kernel

The major part of the Kernel module is a global routine called tasker. Tasker is a round robin task scheduler. Each major functional group in the MPA-II system has a corresponding task that is invoked in this way. All tasks run to completion, meaning that once a task is given control, the task must return to the tasker in order to relinquish control. Interrupt handlers are initialized and masked on and off by their corresponding tasks, although the tasker maintains ultimate control over all activity.

The Kernel consists of tasker, schedule_task, and desch_task routines. These three combine to allow tasks to be added or removed from the active task list, providing orderly execution of tasks. All tasks are scheduled by calling schedule_task with the task's identification byte in the selected accumulator. Schedule_task then adds the task to the active task list. The task list is implemented in R_Task (R21) as discussed above. The list of tasks in the MPA-II system is shown in Table 6-1.

TABLE 6-1. MPA Tasks

Task ID	Task Name	Description
0	cx_task	Coax Session Processor
1	tw_task	Twinax Session Processor
2	ibm_task	IBM Interface Emulation
3	irma_task	IRMA Interface Emulation
4	sa_task	Smart Alec Interface Emulation
7	house_task	System Initialization and Control

System Initialization

The file MPA2.BCX contains the microcode for the MPA-II system operation. The Loader (LD) softloads the BCP, single steps the BCP—which allows the BCP to disable GIE if any interrupts are pending from previously executing code, starts the BCP executing from address zero (0000h), and then writes the MPA-II Configuration register (2DCh) to establish the desired mode of operation, e.g., Coax-IRMA, Coax-IBM, etc. Note that the MPA-II Configuration register is written after the BCP is started. As discussed in the hardware section, the MPA-II is capable of performing a hardware "lock out" of the PC after the PC writes to the I/O locations 220h–22Fh, 2D0h–2D6h, and 2D8h–2DEh, if this feature has been enabled by the BCP microcode. This means that the next access (reading/writing dual-port memory as well as I/O memory) by the PC to the MPA-II board will be held off until the BCP's microcode signals the MPA-II hardware that the next PC access may complete. If the BCP is not running, its microcode cannot signal the hardware to unlock the PC and, therefore, the PC will stop processing. The user will then have to reset the PC in order for the PC's processor to regain control. When the MPA-II is reset (via the PC's reset bus line) this lock out capability is automatically disabled and the PC hs unlimited access to the MPA-II board. But, after the MPA-II has been running, and it is then arbitrarily stopped, the PC lock out capability may still be enabled. Therefore, never perform I/O writes to the above mentioned registers unless the MPA-II board has been reset, or until after starting the BCP with microcode that either disables the lock out capability or unlocks the PC after an access occurs.

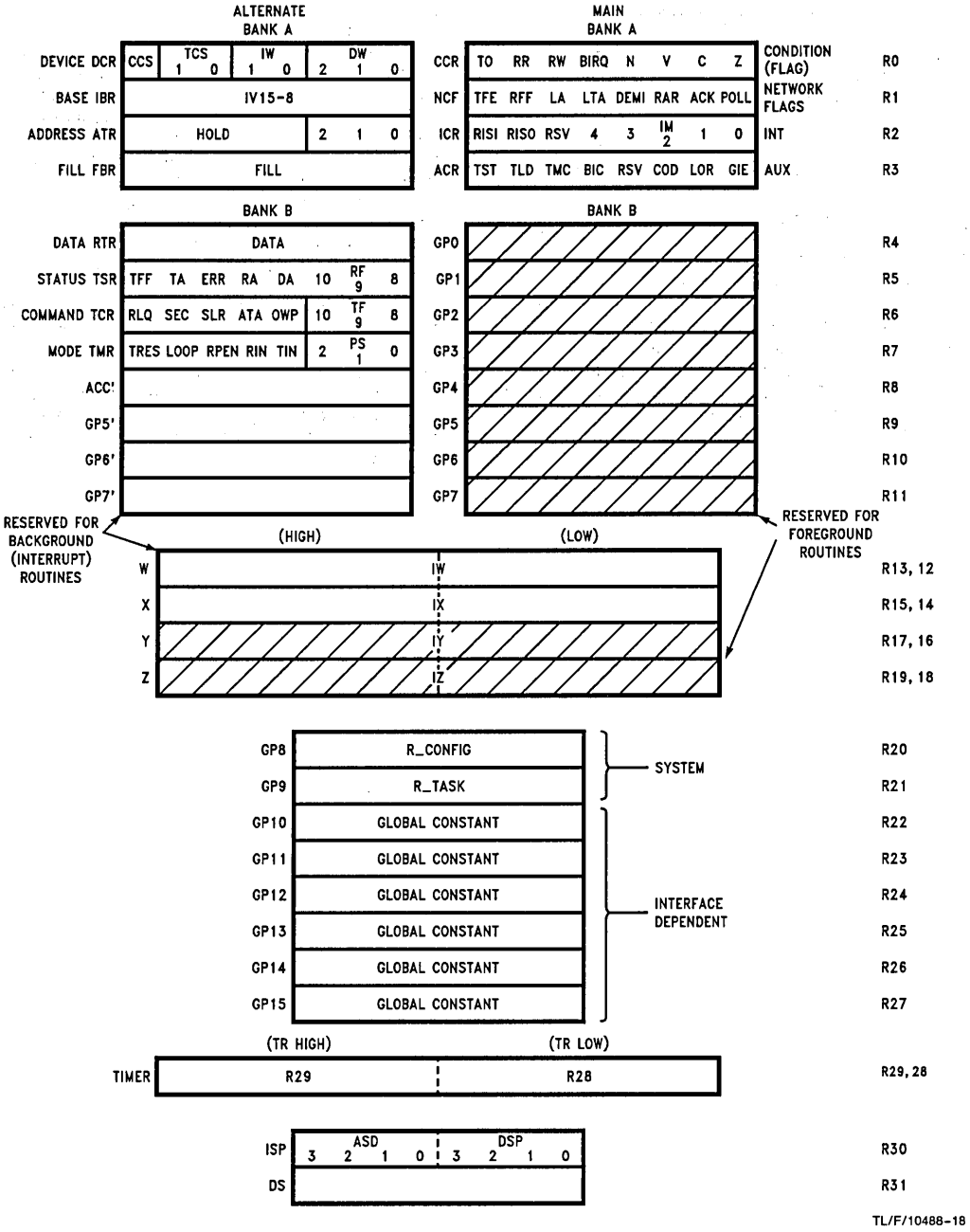
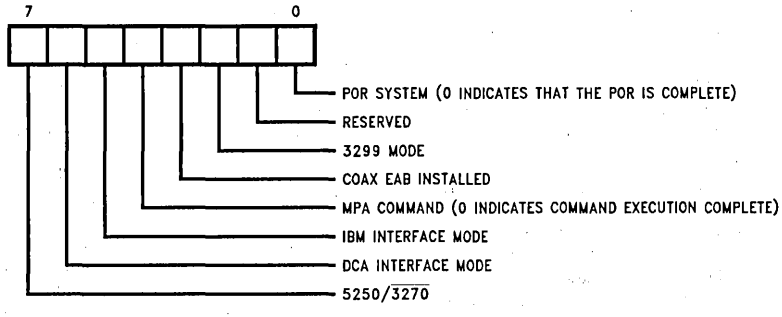


FIGURE 6-2. DP8344 MPA-II Register Map©



TL/F/10488-33

FIGURE 6-3. MPA-II Configuration Register

After the Loader has started the BCP executing MPA2.BCX microcode, the microcode proceeds by disabling interrupts and initializing certain BCP registers to set CPU speed, memory access wait states, BIRQ direction, etc. The IRQ PC interrupt line is deasserted, PC I/O write generated BIRQ interrupts are disabled, the PC lock out capability is disabled, and BCP data memory is cleared. Finally, initializations for the HOUSEKEEP task are performed and then control is permanently passed to the Tasker, which will retain control until the MPA-II is reset.

After the Tasker performs its own initialization, it begins calling any scheduled tasks. At this point, only the HOUSEKEEP task is scheduled. When HOUSEKEEP runs, the MPA_CONFIG register (I/O location 2DCh) is written into R20, the R_CONFIG register, and then its contents are used to call the appropriate task initialization routines, refer to *Figure 6-3*. These routines set up any variables needed for the task, initialize interrupt handlers associated with them, and schedule their tasks. For instance, if the MPA_CONFIG register has been loaded with 49h, the routine would call cx_init to initialize the 3270 coax task, set up the appropriate interrupt handlers, and schedule cx_task. Then the irma_init routine would be called which sets up the interface registers, the BIRQ interrupt, etc . . . Since the PC writes the MPA_CONFIG register, HOUSEKEEP must interpret the configuration value based on what it knows are valid configurations. In order to provide feedback to the PC, HOUSEKEEP builds a valid configuration value based on its interpretation. After all the initialization routines have completed execution and returned control to HOUSEKEEP, HOUSEKEEP places its value for the configuration back into the MPA_CONFIG register with the POR_SYSTEM bit of the configuration clear, thus signaling the PC that initialization has completed and has been interpreted as the HOUSEKEEP configuration value shows. The Loader polls the MPA_CONFIG register after writing it, waiting for the POR_SYSTEM bit to clear. When the Loader detects that the HOUSEKEEP mode initialization has completed, it compares its value for the configuration with that returned by HOUSEKEEP. The Loader then issues warning messages to the user if any mismatches are found. When HOUSEKEEP passes control back to the tasker, all applicable tasks are scheduled and interrupts have been unmasked. HOUSEKEEP remains scheduled so that upon subsequent executions the RAM value for MAP_CONFIG can be compared with R_CONFIG. If a difference is found or the POR_SYSTEM bit is set, then the initialization process takes place again. If no difference is detected, then HOUSEKEEP returns directly to the tasker.

Coax Task

Basic 3270 emulation is handled by the cx_task and its associated routines independent of the interface mode configured. The coax routines are set up to exploit the extremely quick interrupt latency of the BCP. Even so, the coax routines are fairly time critical. The basic structure used is divided into two distinct parts: the interrupt handler executes all real time tasks in the background and the cx_task routine handles the four "busy" type commands of the 3270 protocol. The vast majority of decisions and command executions must be carried out "on the fly", or under the auspices of the interrupt handlers. Primarily, the interrupt handlers do the bulk of command execution. See Table 4-1 in Chapter 4 for a list of some of the 3270 commands supported.

The scp_coax template, contained in CX_DATAR.HDR, is a reference to the RAM array that locates all the coax terminal variables, including relative pointers into the screen buffers. Both a Regen buffer and EAB is supported if the MPA_CONFIG register is set for EAB.

The cx_task module, CX_TASK.BCP, contains the task initialization routine as well as the task itself. Cx_init sets up the RA and LTA interrupts and initializes all scp_coax variables and inter-task communications, and initializes the transceiver. CX_TASK's functions are: processing inter-task mail, updating poll status, processing foreground commands, and resetting the coax terminal. The foreground commands include SEARCH forward, SEARCH backward, INSERT, and CLEAR.

The Session Control Page, SCP, for coax defines registers for each of the 3278 terminal registers, as well as additional ones for control of internal functions. Refer to *Figure 4-2* in Chapter 4 for the internal structure of a 3270 terminal. Initially, the primary and secondary control registers are cleared, [STAT_AVAIL] is loaded into status_reg, and the poll response is set to POR (Power On Reset). GP6 on Alternate Bank B is dedicated as the Coax_state register. It is used to provide fast access to protocol state information such as 3299 address, cursor change, and write in progress.

The MPA-II system uses a number of variables to maintain the coax session, including:

coax_stat —Emulation Mode

mpa_mainstat —Main Interface Control Bits, such as Clicker and Alarm Status

mpa_auxstat —Auxiliary Interface Control, such as Buffer being Modified and On-Line/Off-Line Control

mpa_control —Poll Status Control, such as POR, Key Pending, FERR, Operation Complete

mpa_auxcontrol —Additional Poll Status, such as EAB Status

The initial state of the mpa_mainstat register sets up flags to signal that a new cursor position is available and that the key buffer is empty. mpa_control is set up with POR state and the status_pending flag set. Status_pending signals the poll response routine that POR status is available. In addition to flags and registers, there are two mailboxes that are used: the sub-task mailbox, and sync_mailbox. The RA, or receiver active, interrupt uses the sub-task mailbox to communicate to cx_task which, if any, foreground coax command needs to be procesed. Initially this is cleared. The sync_mailbox is the PC interface routines' communication mechanism. Keystroke passing, alarm acknowledgement and resetting of the terminal by the PC are communicated via sync_mailbox.

In normal operation, the cx_task routine remains scheduled and the normal execution proceeds in the manner suggested in *Figure 6-1*. The update_poll response routine uses the values in mpa_control and mpa_auxcontrol to determine if the session should adjust its poll status to the controller. The new_status routine maintains the sync_mailbox and, therefore, communication with the various PC interface tasks. If there is mail, new_status reads and executes the PC interfaces' commands. Of chief importance, the state of the keystroke buffer is checked here. It is the mechanism through which keystrokes may be passed from the PC interfaces to the poll response for transmission to the host controller. A high MPA_MS_KEYEMPTY bit in mpa_mainstat signals that the interface may supply a keystroke. If MPA_MS_KEYEMPTY is low, the PC interface must wait. MPA_MS_KEYEMPTY is cleared by new_status when it infers from mpa_control that the previous key has been acknowledged by the coax controller.

The sub-task communication mailbox is checked by cx_task next. If the receiver interrupt handler has decoded a foreground coax command request from the host controller, the mailbox will be non-zero. The value in the mailbox indicates that either a forward or backward SEARCH, an INSERT, or CLEAR command, and its associated parameters are ready for execution. The appropriate foreground coax command routine is then run to completion. The status_reg is now updated, since completion of a foreground coax command requires an Operation Complete status to be returned to the host controller. The poll response is updated again, if necessary, and then the cx_task routine relinquishes control to the tasker.

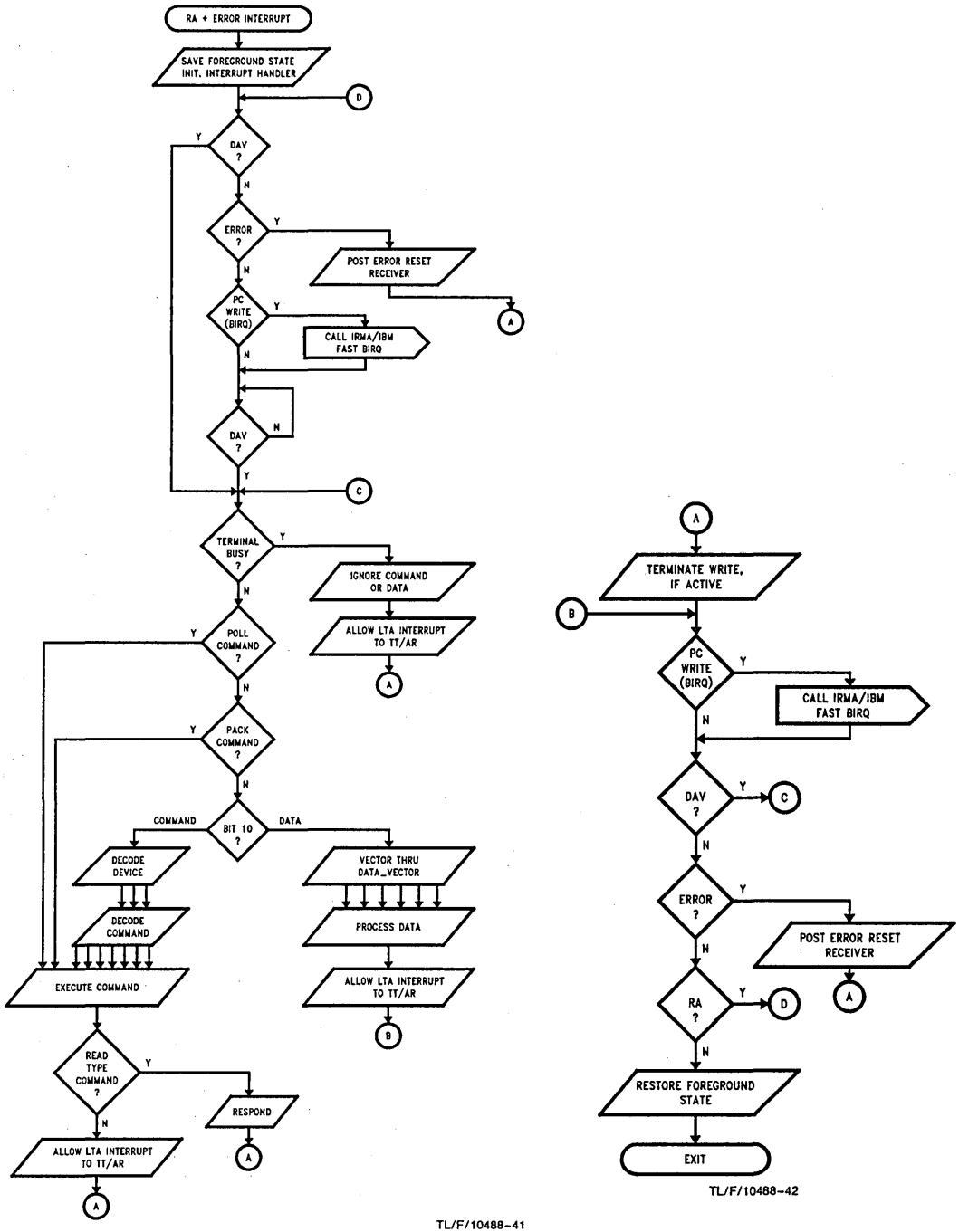
Coax Interrupt Handlers

The coax mode uses two interrupts to support coax activity: Receiver Active, RA, and Line Turn Around, LTA. There are two possible receiver interrupt handlers which can get control from the RA interrupt depending on whether 3270 or 3299 support has been selected in the MPA_CONFIG register. Two Interrupt Vector tables are used to determine which receiver interrupt handler will get control. One interrupt vector table, INT_PAGE, supports 3270 and 5250. The other interrupt vector table supports 3299. The active interrupt vector table is determined by the contents of the {IBR} register. The {IBR} register is set during configuration initialization by a coax initialization routine. HOUSEKEEP determines which coax initialization routine gets executed based on the MPA_CONFIG register, cx_init for 3270 and cx_3299init for 3299. cx_3299init actually calls on cx_init to perform most of the initialization, with cx_3299init performing only 3299 specific initializations.

The flow of the 3270 receiver interrupt handler is shown in *Figure 6-4*. The only difference between the 3270 and 3299 receiver interrupt handler is at the start. The 3299 receiver interrupt handler checks the first frame of the 3299 transmission for the terminal address. If the address does not match the user specified terminal address (usually specified via the Loader), the receiver is reset and that transmission is ignored. If the terminal address of the 3299 address frame does match, then control is passed to the 3270 interrupt handler for command processing and response transmission back to the coax controller.

The receiver interrupt handlers are background tasks to the Kernel and have been written to conform with the rules for all background tasks. These rules include the saving and restoring of any register used except those on the alternate B bank, IW and IX. Within the receiver interrupt handler, only the dedicated background register pair IX is used, IW is free for user enhancements. IX is used as the screen and EAB buffer pointer, and its is also used as the receiver software state machine variable DATA_VECTOR. More about the DATA_VECTOR will be discussed later.

When the 3270 receiver interrupt gets control, either directly from the RA interrupt vector or indirectly from the 3299 receiver interrupt handler, it retains control until all the frames sent from the controller have been processed by the interrupt handler or a transmission error is detected. We chose the Receiver Active interrupt and allowed the receiver interrupt routine to retain control until the transmission is complete because the MPA-II must support two asynchronous communications interfaces, the coax line and the PC interface. By using the RA interrupt the receiver interrupt handler has more time with which to get control before it must respond to the transmission sent. This extra time is needed when the receiver interrupt is held off while other interrupts are being processed or while the foreground routines have disabled interrupts. Note that care should be taken to insure that the receiver interrupt is never held off for more than 4.5 μs or the MPA-II may not be able to respond to coax commands with 5.5 μs.



TL/F/10488-42

TL/F/10488-41

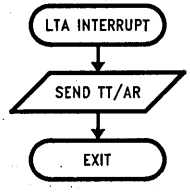
FIGURE 6-4. 3270 Coax Receiver Handler

Once the receiver interrupt handler gets control, it will check for Data AVailable, DAV, and receiver errors, handling them immediately. If neither condition mentioned above is true, which is the case unless the receiver interrupt has been held off, the receiver interrupt handler will check for PC interface activity and allow it to be serviced via one of the fast BIRQ routines, (i.e., either the IRMA or IBM PC interface fast BIRQ routine). As the coax transmission is processed, the receiver interrupt handler will check for PC interface activity in between the processing of coax data frames, when the receiver interrupt handler is idle anyway. Holding off the PC and its interface programs (i.e., IRMA's E78, IBM's PC3270, etc . . .) is possible because they are not as time critical as a coax controller in expecting responses from the MPA-II.

When data becomes available the receiver interrupt handler checks to see if the terminal is currently processing a coax foreground command and therefore "busy". If it is busy, then all data and commands are ignored, and the receiver interrupt handler enables just the LTA interrupt, allowing it to respond with TT/AR as soon as the coax line drops. Note that the LTA interrupt may now interrupt the receiver interrupt handler. If the terminal is not busy, then a quick check to see if the current data frame is either the POLL or PACK command is performed. If this is true, the POLL or PACK command is handled immediately. Otherwise, bit 10 of the coax data frame is checked. If it is high, the data frame is a command from the controller. First the terminal internal device address is decoded to determine which internal device the command is addressed to, for example EAB. Next, the command is decoded, its processing routine is called, and the command is processed. If it is a Read type command then the appropriate response is immediately sent. If the coax command processed is a Write type command that expects data frames to follow, either immediately or upon the next transmission, the DATA_VECTOR is loaded with the address of the part of the receiver interrupt handler routine which is responsible for processing the expected data frame(s). Next, the LTA interrupt is enabled to allow it to respond with TT/AR when the line drops. Again, note that the LTA interrupt handler may interrupt the receiver interrupt handler from this point on. Finally, control passes to the receiver interrupt handler exit routine which terminates write mode, if it has been active, checks for PC activity and, if any occurred, handles it, and then checks for receiver activity. If the receiver is still active or data is available, the receiver interrupt handler loops back to process the next data frame, else the interrupted foreground routine's state is restored and the receiver interrupt handler then exits.

If bit 10 of the coax data frame is low then the data frame contains data for a previously executed command. The DATA_VECTOR is used to pass control to the appropriate section of code which processes that data. After the expected data is processed, or a command is executed which does not require following data, or an error is detected, then the DATA_VECTOR is set to a receiver interrupt handler routine which accepts and trashes unexpected data frames. As with commands, after the data is processed, the LTA interrupt is enabled to allow it to respond with TT/AR when the line drops. Finally, control returns to the receiver interrupt handler exit routine, but note that write mode is not terminated, in most cases.

The other interrupt used by the coax mode, LTA, requires a very simple interrupt handler since its only task is to respond with TT/AR (see *Figure 6-5*). This is because all other responses are handled by the receiver interrupt handler, as stated above. Thanks to the dedicated registers of the BCP and the tight coupling of the CPU to the Transceiver, the LTA interrupt handler does not have to save or restore any registers. This feature allows it to easily interrupt both foreground and background tasks, as well as perform in a timely manner.



TL/F/10488-43

FIGURE 6-5. 3270 Coax LTA Handler

Due to the nature of the coax mode, most of the coax commands must be processed during the receiver interrupt. The commands can be broken up into three basic groups: Read type commands which respond with information requested by the controller. Write type commands which write following data frames into particular registers or screen buffers, and foreground commands which perform various time consuming tasks such as clearing screen buffer memory. Of the Read type commands there is a special case called the POLL command. This command will be discussed first.

Poll/Response Mechanism

The Poll and POLL/ACK commands are handled in the CX_BASRD.BCP module in routines cx_poll and cx_pack, respectively. The basic functions of the cz_poll routine are to decide if TT/AR or special status should be returned to the coax controller and to handle the POLL modifiers in the upper bits of the POLL command. These modifiers include the terminal alarm and key click control. The determination of which status to send is made after checking mpa_control for the MPA_STAT_PEND bit. If MPA_STAT_PEND is asserted, the poll response variables have new status to send. If no status is pending, TT/AR is sent. Next, the POLL command modifiers are applied to the alarm and clicker status bits in mpa_mainstat.

The POLL/ACK routine always responds with TT/AR. Next, mpa_control is checked to see if the pending status has been polled by the coax controller. If not, the POLL/ACK routine exits. Otherwise the pending status is cleared and both mpa_control and mpa_auxcontrol are updated. Then the poll response bytes, pollresp_lo and pollresp_hi, are cleared.

Update_poll in the CX_TASK.BCP module handles updating mpa_control and mpa_auxcontrol to reflect new status conditions. This routine updates the pollresp_lo and hi bytes based on the priority of the status in mpa_control and mpa_auxcontrol. POR is the highest priority condition and outstanding status from EAB is the lowest.

Read Commands

All read type commands to the base are found in the CX_BASRD.BCP module. Each read type command is decoded by the receiver interrupt handler and vectored to the appropriate cx_routine. The most basic read type command is cx_readata. This is invoked upon decoding the READ DATA data stream command. The character pointed to by the address counter is sent immediately. The addrcounter variable is incremented after the character is sent.

The cx_readmul routine is also found in the CX_BASRD.BCP module and is vectored to when a READ MULTIPLE command is decoded. READ MULTIPLE expects multiple bytes of screen data to be sent within 5.5 μs. The response is initiated inside cx_rdmul. The routine has two modes: 4 byte and 32 byte. The default mode is 4 byte and is determined by the state of the LSB in the secondary control register. Both modes use the variable addrcounter on the SCP to determine both where to find the data to send and how many bytes to send, up to the 4 or 32 byte limit. In other words, 4 and 32 bytes are the maximum that will be sent to the coax controller. The addrcounter is incremented after sending each byte and terminates the response when the two or five low order bits roll to zero. The transmit FIFO on the BCP will hold up to three bytes. The Transmitter FIFO Full flag, TFF, indicates when the transmitter's FIFO has been loaded with those three bytes. Using this flag, the read multiple routine begins by loading the transmitters FIFO. Once TFF is true, the read multiple routine then alternates between checking the TFF flag and checking for PC activity via the BIRQ flag. If PC activity is detected, then the appropriate fast BIRQ routine is called to handle the PC access. When all the requested bytes have been sent, the read multiple routine passes control to the receiver interrupt handler exit routine. The remaining read type commands are all handled similarly. Cx_rach and cx_racl respond with the high and low bytes of the addrcounter variable, respectively. Cx_rdid responds with the terminal ID byte. Cx_rxid responds with TT/AR since it is not implemented. Cx_rdstat responds with the stat_reg variable. All these commands check for LTA prior to responding. If LTA has not occurred, then a protocol error is posted since read type commands are required to be the last frame in a message from a coax controller. The cx_rdid routine does additional processing, however. The status conditions OPERATION COMPLETE and FEATURE ERROR are cleared by reception of the READ ID command.

Write Commands

All write type commands to the base are found in the CX_BASWR.BCP module. Commands are decoded by the receiver interrupt handler and vectored to this module at the cx_addresses. Each write command has an associated dv_stub for handling incoming data. The routines load the DATA_VECTOR with the appropriate stub before exiting.

Cx_write and its data vector stub dv_write are responsible for writing data into the screen buffer, setting the MPA-II's Buffer Being Modified semaphore and indicating the screen buffer update in the MPA page change word. When the next command is decoded, write mode will be terminated, the Buffer Being Modified bit will be cleared, and the Buffer Modified bit will be set. The dv_write stub is very critical in that very large blocks of data may be sent to the device through the routine and cumulative interrupt latency effects may become significant. To address this, the dv_write routine always empties the receiver FIFO.

Other write type commands found in the CW_BASWR.BCP module include the initial stubs for the foreground commands; SEARCH FORWARD, SEARCH BACKWARD, INSERT, and CLEAR. All these commands are initially decoded and vectored here in real-time. When their associated parameters are received, the foreground commands are scheduled through the sub-task communcation mailbox. All the foreground commands cause the terminal to set NOT_AVAIL status (busy) in the status register. All four respond with TT/AR to acknowledge reception of the command and parameters cleanly.

All the other write commands load variables on the SCP corresponding to registers in the emulated terminal, or cause some controlling action in the terminal. These include the low and high bytes of the address counter, the mask value for CLEAR and INSERT, the control registers and resetting the terminal. Cx_reset calls the host_reset routine that re-initializes the SCP variables to the POR state. The screen buffers are not cleared. The START OPERATION command causes a vector to the cx_start routine and returns TT/AR.

Foreground Commands

The foreground routines are all executed by cx_task when the sub-task communication mailbox is filled with the appropriate value. These are tk_insert, tk_clear, tk_sforward and tk_sback. The routines are found in the CX_COM.BCP module along with other local support routines.

EAB Commands

The EAB commands are found in the CX_EAB.BCP module. Read and write type commands addressed to the EAB feature are included here. The number of commands for the EAB feature are small enough that they are logically grouped together in one module, as opposed to the base commands. Some of the more complex commands from a performance standpoint are addressed to the EAB feature. WRITE ALTERNATE, WRITE UNDER MASK, and READ MULTIPLE EAB require the most real-time bandwidth of any coax function.

The READ MULTIPLE EAB command is the same as its base counterpart except for two features: it functions with the EAB exclusively and, if the Inhibit Feature I/O step bit in the Control register is set, then this command is ignored. WRITE ALTERNATE receives a variable length stream of data that is written with screen and EAB data alternately. The WRITE UNDER MASK command uses data associated with the command, the EAB byte pointed to by the cursor register, and the EAB mask to modify the contents of the EAB. The algorithm is quite strange and is best described by the code. Please refer to eab_wum and dv_wum for specifics on the command implementation.

IRMA Interface Overview

IRMA is a member of a family of micro-to-mainframe links produced by Digital Communications Associates. It provides the IBM PC, PC XT, or PC AT with a direct link to IBM 3270 networks via a coaxial cable connection to an IBM3174, 3274, or integral terminal controllers with type "A" adapters.

The IRMA product includes a printed circuit board that fits into any available slot in IBM PCs and a software package that consists of a 3278/79 Terminal Emulator program, called E78, and two file transfer utilities for TSO and CMS environments. Also included in the software are BASICA subroutines useful in developing other application programs for automatic data transfer.

The 3278/79 Terminal Emulator provides the user with all the features of a 3278 monochrome or 3279 color terminal. The IRMA file transfer program provides all the information required for the successful transfer of files under the TSO or CMS IBM mainframe software packages. Also included in the IRMA software package are many other features such as program customization, keyboard reconfiguration, independent and concurrent operation, ASYNC Character Support, and PC clone support.

As discussed in the introduction, the IRMA product was a forerunner in the 3270 emulation marketplace and quickly gained wide acceptance. DCA made a considerable effort in documenting the interface between IRMA and its PC host. As a result this interface has become one of the industry standards used today. So it is only natural that this interface be used on the DP8344 Multi-Protocol Adapter-II to highlight the power and versatility of the DP8344A. Biphase Communications Processor. The MPA-II hardware with the MPA-II soft-loadable DP8344A microcode is equivalent in function to the DCA IRMA board with its associated microcode. Both directly interface with the IRMA software that runs on the PC (E78, file transfer utilities, etc.) providing all functions and features of the IRMA product. The following sections describe the hardware interface and the BCP software in the Multi-Protocol Adapter II Design/Evaluation kit that is used to implement the IRMA interface. All of the following information corresponds to Rev 1.42 of the IRMA Application software. Later versions of the IRMA PC Application Software are downward compatible.

Hardware Considerations

The IRMA printed circuit board plugs into any normal expansion slot in the IBM PC System Unit. It provides a back-panel BNC connector for attachment by coaxial cable to a 3174, 3274, or integral controller. IRMA operates in a stand-alone mode, using an on-board microprocessor (the Signetics 8X305) to handle the 3270 protocol and screen buffer. Because of the timing requirements of the 3270 protocol, the on-board 8X305 operates independently of the PC microprocessor. The 8X305 provides the intelligence required for decoding the 3270 protocol, managing the coax interface, maintaining the screen buffer, and handling the data transfer and handshaking to the System Unit (PC microprocessor).

The IRMA card uses National Semiconductor's DP8340 and DP8341 3270 coax transmitter and receiver (respectively) to interface the 8X305 to the coaxial cable. The DP8340 takes data in a parallel format and converts it to a serial form while adding all the necessary 3270 protocol information. It then transmits the converted data over the coax in a biphase encoded format. The DP8341 receives the biphase transmissions from the control unit via the coaxial cable. It extracts the 3270 protocol specific information and converts the serial data to a parallel format for the 8X305 to read.

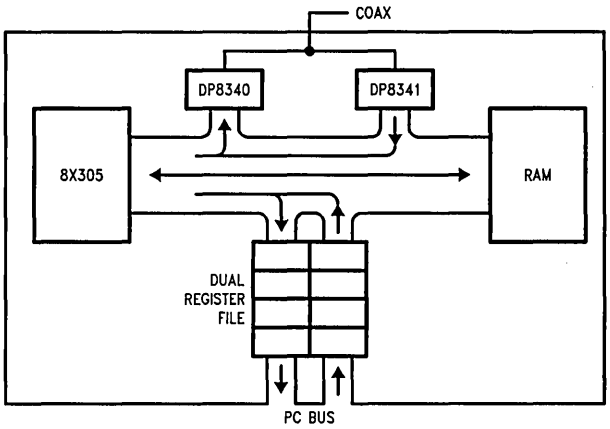
The IRMA card contains 8K of RAM memory for the screen buffers and temporary storage. The screen and extended attribute buffers use approximately 6K of this memory. The remaining memory space is used by the 8X305 for local storage. A block diagram of the IRMA hardware is shown in *Figure 6-6*.

The hardware used in enabling the 8X305 to communicate with the PC's 8088 processor is a dual four byte register array. The 8X305 writes data into one of the four byte register arrays which is read by the 8088. The 8088 writes data into the other four byte register array which is in turn read by the 8X305. The dual register array is mapped into the PC's I/O space at locations (addresses) 220h–223h.

A handshaking process is used between the two processors when transferring data. After the 8088 writes data into the array for the 8X305, it sets the "Command Request" flag by writing to I/O location 226h. The write to this location is decoded in hardware and sets a flip-flop whose output is read as bit 6 at location 227h. When the 8X305 has read the registers and responded with appropriate data for the 8088, it clears this flag by resetting the flip-flop. A similar function is provided in the same manner for transfers initiated by the 8X305. Here the flag is called the "Attention Request" flag and can be read as bit 7 at location 227h. This flag is cleared when the 8088 writes to I/O location 227h.

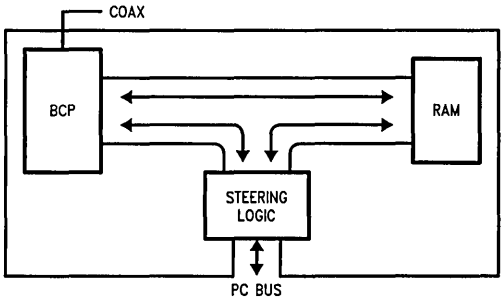
The Multi-Protocol Adapter-II printed circuit board also plugs into any expansion slot in the IBM PC System Unit. Like the IRMA card, it provides a back panel BNC/Twisted Pair connector for attachment by coaxial cable or unshielded twisted pair cable to a 3174, 3274, or integral controller. The MPA-II operates in a stand-alone mode, using the DP8344A Biphase Communications Processor to handle the 3270 protocol and screen buffer. Again, because of the timing requirements of the 3270 protocol, the BCP operates independently of the 8088 microprocessor of the System Unit. As with the 8X305, the BCP provides the intelligence required for decoding the 3270 protocol, managing the coax interface, maintaining the screen buffer, and handling the data transfer and handshaking to the System Unit. However, with the BCP's higher level of integration, it also directly interfaces with the coaxial cable. The BCP has an internal biphase transmitter and receiver that provides all the functions of the DP8340 and DP8341. However, unlike the 8X305, the DP8344's CPU can handle the 3270 communications interface very efficiently.

The MPA-II card contains a single 32K x 8 RAM memory device for the screen buffers and temporary storage. This memory size was chosen for the 5250 environment, where the BCP can handle up to seven sessions. In the IRMA mode, only a little over 4K of memory is required. The MPA-II hardware block is shown in *Figure 6-7*.



TL/F/10488–22

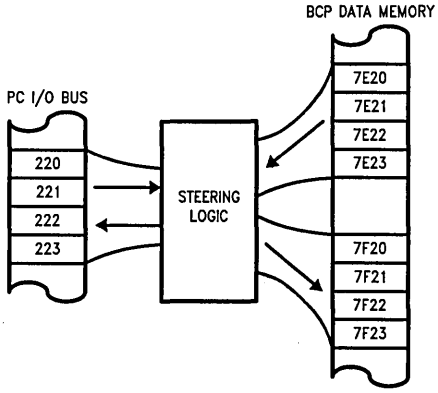
FIGURE 6-6. IRMA Hardware Block Diagram



TL/F/10488–23

FIGURE 6-7. MPA-II Hardware Block Diagram

The hardware used to enable the BCP to communicate with the PC's 8088 processor is steering logic (contained in PALs) and the BCP's data memory. In a typical application, the BCP communicates with a remote processor by sharing its data memory. This is true with the MPA-II, but because the MPA-II must run with the IRMA software, steering logic has been used to direct the 8088's I/O reads and writes of the IRMA dual register array locations (220h–227h) into the data memory on the MPA-II card. By using data memory instead of a discrete register file the component count has been reduced. The IRMA software requires that a "dual" register file be used (or in this case emulated). Therefore, the writes from the 8088 are directed to memory locations 7F20h–7F23h and the reads from the 8088 are sourced from memory locations 7E20h–7E23h. The MPA-II Register Array Implementation is shown in *Figure 6-8*.



TL/F/10488–24

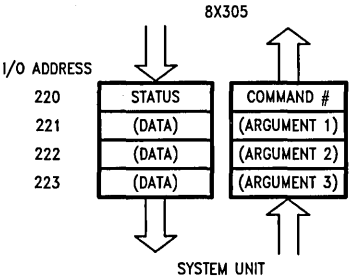
FIGURE 6-8. MPA-II Register Array Implementation

The handshaking process is still used when the BCP and the 8088 are transferring data. When the 8088 goes to set the command flag by writing to I/O location 226h, it actually does a write to 7F26h in the MPA-II's data memory via the steering logic. The steering logic locks out future accesses by the PC to the MPA-II and interrupts the BCP telling it that a write access has been made to the IRMA I/O space. This interrupt is signaled through the BIRQ I/O pin of the BCP, which is configured as an input interrupt. The MPA_CONFIG register determines which BIRQ interrupt handler will be called. In this case, assume that the DCA interface option is selected. Then the dca_int BIRQ interrupt handler located in the module DCA_INT.BCP is given control. The dca_int BIRQ interrupt handler determines if the PC wrote to 226h by reading the "MPA-II Access" register located in a PAL. This access register is located at BCP data memory address 8000h and it holds the lower 6 bits of the last I/O location written to on the MPA-II. If a write occurs to I/O location 226h, the BCP sets bit 6 in the MPA-II memory location that the PC's 8088 will read as its I/O location 227h. The BIRQ Interrupt handler will then write (any value) to the MPA-II Access register to unlock the PC. In the case of the "Attention Request" flag, the BCP will set this flag by simply setting bit 7 in the memory location which the 8088 reads as I/O 227h. The clearing of this flag by the 8088 is done in a similar fashion as the setting of the "Command Request" flag. Note that each time the 8088 writes to an I/O location between 220h and 22Fh the BCP is interrupted. However, specific action is taken only on writes to 226h or 227h. With all other locations the BCP simply returns from the interrupt service routine once it has determined the 8088 did not write to I/O 226h or 227h. This approach to the hardware has been chosen to minimize the discrete logic on the MPA-II card by taking advantage of the power of the BCP's CPU to handle some tasks in software that were typically done with hardware in the past. Another benefit of this "soft" approach is that changes to the IRMA interface definition by DCA will most likely only require a software change for the MPA-II board, thus protecting your hardware investment.

IRMA Microcode

The IRMA application software written for the personal computer (E78, file transfers, etc.) is designed around a defined interface between IRMA and the System Unit (the 8088 and its peripheral devices). The hardware portion of this interface is discussed above. The software portion of this interface is the microcode written for the 8X305 processor. When the software and hardware are viewed as one function, it is referred to as the Decision Support Interface (DSI). All of the IRMA application software has been written around this interface. When configured in the IRMA mode the MPA-II becomes the DSI. The method of communication between the DSI and the System Unit will be discussed briefly in the next section. A more exhaustive discussion on this interface is given in the IRMA Technical Reference.

The DSI and the System Unit communicate through the dual four byte register array just discussed. The System Unit issues commands to the DSI by writing to this array. This register array is viewed by the System Unit as four I/O locations (220h–223h). Each I/O location corresponds to one eight bit word. When the System Unit issues a command, the first byte, word 0, is defined as the command number. The next three bytes, word 1 through word 3, are defined as arguments for the command. The number of arguments associated with an individual command varies from zero to three. Sixteen commands have been defined for the DSI. These commands allow the System Unit program to read and write bytes in the screen buffer, send keystrokes, and access special features available on the DSI. To begin a command the System Unit program sets byte 0 equal to the command number and provides any necessary arguments in byte 1 through byte 3. It then sets the Command request flag. The Command Request flag is continually polled by the 8X305 processor when it is not busy managing the higher priority 3270 communications interface. When it detects the setting of this flag by the System Unit, it reads the data from the register array and executes the command. Once the command has been executed, the 8X305 will place the resulting data into the other side of the register array and clear the Command Request Flag (see *Figure 6-9*). The System Unit program has been continually polling this flag and after seeing it cleared reads the result from the register array. The Command Request flag can only be set by the System Unit. This is done by a write to I/O location 226h. The Command Request Flag can only be cleared by the DSI's 8X305.



TL/F/10488-25

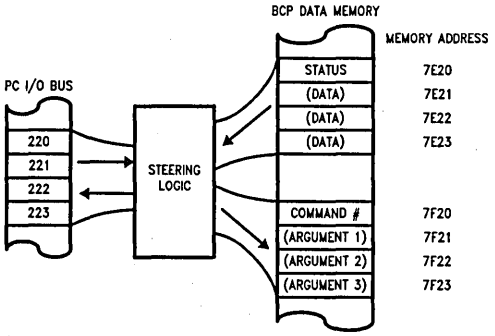
FIGURE 6-9. Command and Response Locations in the IRMA Register Array

The DSI can not issue commands to the System Unit but it can inform the System Unit of a status change. If a status change occurs in a status bit location when the corresponding attention mask bit is set, the 8X305 will set the Attention Request flag. This flag can be polled by the System Unit and is viewed as bit 7 in the I/O register at address 227h. The System Unit can clear this flag by executing a write to I/O location 227h. As is the case with both flags, the action of writing to the specific I/O location clears or sets the flags, the data written during the write have no affect. In typical operation the Attention Request flag is not used; however, it is implemented on the MPA-II. The current status of both flags can be read by both processors. The System Unit does this by reading I/O location 227h. The resulting eight bit number has the Attention flag as bit 7, the MSB, and the Command flag as bit 6. The other bits are not used.

MPA-II Implementation

The IRMA interface on the MPA-II board operates essentially in the same manner as described above. The System Unit I/O accesses to the IRMA register array space are transferred to two areas in the BCP's data memory (see *Figure 6-10*). One location is for System Unit reads of the

array (7E20h–7E23h), the other is for System Unit writes to the array (7F20h–7F23h). Different BCP memory locations are used because the register array on the IRMA card actually contains eight byte wide registers (four for System Unit reads and four for System Unit writes) in hardware. E78 was written to make the best use of this hardware design and in doing so it may write a new command and/or arguments before it reads the results of the old command. Therefore if just four memory locations were used, E78 would read back part of a new command it had just written and interpret this as data from the DSI from the previous command.



TL/F/10488–26

FIGURE 6-10. Command and Response Locations in the MPA-II Register Array

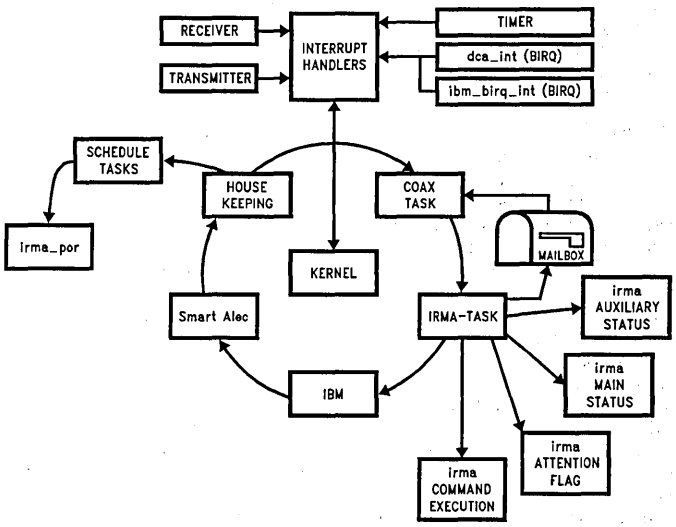
The Command Request and Attention Request flags are implemented using 74LS74's on the IRMA card, hence the setting and clearing by writing to 226h and 227h (this clocks or clears the associated flip-flop). This function is implemented on the MPA-II using an external PAL and the bi-directional interrupt pin, BIRQ. If there is a write to the IRMA I/O space 220h–227h, a PAL issues an interrupt to the BCP via the BIRQ input. The BCP reads the outputs of another PAL to determine which location has been written to. If the write is to I/O locations 226h or 227h then the appropriate bits are set or cleared in the "IRMA read location" (7E27h) in the BCP data memory. The BIRQ interrupt is generated only on System Unit I/O writes to 220h–22Fh but this also includes writes to the dual register array. If a write to 220h–223h occurred, the BCP irma BIRQ interrupt routine simply clears the interrupt and takes no further action.

The commands from the System Unit are executed in the irma task routine. This routine is a foreground, scheduled task in the MPA-II Kernel. The irma task routine first updates both the main and auxiliary status registers as defined by the DSI. Next the irma task sets the attention flag, if required. It then looks at the state of the command request flag in memory to determine if there is a command pending from the System Unit. If so, it reads the command number and the arguments from the BCP's data memory and executes the command. The task then places the results back in the data memory in the appropriate location (7E20h–7E23h). After this is complete the task clears the command request flag and returns program control to the Kernel.

There are three separate code modules used to allow the MPA-II to emulate the DSI.

1. Power-Up Initialization Routine
2. BIRQ Interrupt Routine
3. irma Task Routine

These three routines will be discussed in the following section. For clarity, the term "irma" is capitalized when referring to DCA products and lower case when referring to the MPA-II software that was written to emulate the IRMA DSI. *Figure 6-11* gives a graphical representation of where these routines fit into the software architecture of the MPA-II.



TL/F/10488–27

FIGURE 6-11. MPA-II Software Block Diagram in IRMA DSI Emulation Mode

MPA-II Power-Up Initialization Routine

The irma power up initialization routine is called by the housekeeping task if it detects that the DCA irma bit has just been set in the MPA-II configuration register (along with the 5252/$\overline{3270}$ bit clear). The irma initialization routine is titled irma_por in the MPA-II source code. This routine initializes the memory locations and BCP internal registers that are used by the irma emulation code. It also unmasks the BIRQ interrupt and schedules the irma_task in the MPA-II Kernel. The first memory location initialized is the Command Request and Attention Request flag byte, which is location 7E27h in the BCP's data memory. The data at location 7E27h is passed to the System Unit by the steering logic when the System Unit reads I/O location 227h. This byte is set to zero by the irma_por routine even though only bits 6 and 7, the command and attention request flags respectively, are used. The irma_por routine also initializes the memory locations that the irma—task routine uses to store the trigger variables and the attention mask.

The irma_por routine also initializes internal BCP registers. It does this because other routines, such as the dca_int interrupt routine, must access certain stored values very quickly to keep execution time short. The execution time in these routines is decreased if data needed in the routine are kept in internal registers rather than in data memory. For example, the value of the high byte of the address page of the "IRMA read registers" is stored in register GP14. In the BIRQ interrupt routine, the IZ index register needs to point to that address page. This is done in the routine with a single 2 T-state instruction which moves the contents of GP14 to the high byte of the IZ index register. If the value of the high byte of the address page was in memory, it would take a 4 T-state move to an immediate addressable register followed by a 2 T-state move to the IZ index register. The irma_por routine initializes the registers GP14 and GP12 with the "IRMA_read register" page memory address. The irma_por routine then signals the coax task, via sync_mailbox, to bring the MPA-II on line as a live terminal. The final function of the irma_por routine is to schedule the irma_task routine. This is done by loading the task number into the accumulator and calling the schedule_task routine. After this, program control is returned to the tasker.

DCA_INT BIRQ Interrupt Routine

The second code module required to emulate the IRMA DSI is the dca_int BIRQ routine. On the IRMA card, the Command Request and Attention Request flags are implemented in hardware. This implemention requires a number of discrete components to decode the System Unit I/O addresses 226h and 227h and to provide the set and clear function of these flags. The MPA-II board, on the other hand, uses extra CPU bandwidth to reduce the discrete components needed to provide the Command Request and Attention Request flag function. It does this by letting the CPU decode part of the System Unit I/O access address and provide the set and clear function of these flags. The BCP code necessary for this is the BIRQ interrupt routine whose source module is labeled DCA_INT.BCP. The BIRQ interrupt is generated when the System Unit writes to any I/O locations from 220h to 22Fh. It would have been more expedient in this case to only have interrupts generated on writes to I/O locations 226h and 227h. However, the MPA-II hardware also supports the DCA Smart Alec emulation program and the IBM emulation programs. The MPA-II implementation for the DCA Smart Alec and the IBM interfaces require interrupts to be generated from more System Unit I/O access locations, so to reduce external hardware, interrupts are generated for a sixteen byte I/O block. This flexibility of hardware design further illustrates the usefulness of the extra CPU bandwidth of the DP8344A.

When the BCP detects the BIRQ interrupt, it transfers program control to the dca_int routine. The function of this routine is to set the Command Request flag if the System Unit wrote to I/O location 226h or clear the Attention Request flag if the system unit wrote to I/O location 227h. The 3270 protocol timing requirements place another time constraint on this routine. Becuase this is an interrupt service routine, all other BCP interrupts are disabled upon entering. This means the coax interrupts will not be acknowledged until they are re-enabled by the program. To meet this critical timing constraint, the dca_int routine execution time must be as short as possible. The routine reads the MPA Access Register PAL to acquire the information needed to determine which register the System Unit actually wrote to. Keep in mind that at this point the PC is "locked out" from making any further accesses to the MPA-II. It then determines which I/O locations the System Unit wrote to by using the JRMK instruction and a jump table. If the write was to 226h then the Command Request flag is set. Next, the routine must "unlock" the PC by writing to the MPA Access Register. Now the routine only has to restore the environment (foreground registers used in interrupt routines are pushed on the data stack and must be restored before leaving the interrupt service routine) and return to the foreground program. If the write was to I/O location 227h, the routine clears the Attention Request flag. It then unlocks the PC, restores the environment and returns program control to the foreground program. If the write was to any other of the sixteen locations, the PC is unlocked, the environment is restored, and program control is returned to the foreground task.

There is a section of code in the dca_int routine that does the same function as that described above, but is called from the coax receiver interrupt routine and not by the external BIRQ interrupt. To increase performance, the transceiver interrupt handlers check the BIRQ flag in the CCR register before they return to the background task. If the flag is asserted (active low), they call the dca_fast_birq section of the dca_int routine. Here the same operations as described earlier are performed except for the saving and restoring of the environment. The dca_fast_birq routine does not have to provide this function because the coax receiver interrupt routine does it. This decreases the number of instructions executed, and therefore, improves the overall performance.

MPA-II Irma Task Routine

The majority of the DSI emulation takes place in the irma_task routine. This routine is run in the foreground as a scheduled task. Therefore the decision to execute this routine is dependent only on the MPA-II's task scheduler and is not impacted by the System Unit. In reality the task is run many times between System Unit accesses because the code execution speed of the BCP is greater than that of the PC. Therefore, the most current information and status is always available to the System Unit. The irma task routine,

appropriately labeled in the source code as "irma_task", contains two sections. These sections are the irma status update and the command execution routines.

The irma status update routine, called irma_status_update in the source code, gathers and formats the information required to produce the auxiliary status byte and main status byte as defined by the DSI (see Table 6-2). This routine is implemented in the irma_task routine as a subroutine. It gets the necessary status for the auxiliary status information from two predefined memory locations which contain general coax information placed there by the coax routine. These memory locations are labeled MPA_MAINSTAT and CONT_REG in the source code. The auxiliary status routine first moves the MPA_MAINSTAT byte from data memory into an internal register. It masks off the unwanted bits and combines the register with the contents of the CONT_REG memory location, which is also loaded into an internal register from data memory. The routine then loads the previous value of the auxiliary status byte from data memory. This value was saved from the previous time the task was executed and is required when determining the main status byte. The routine then stores the new value of the auxiliary status register in that same data memory location. The new auxiliary status byte is maintained in register GP6 for the remainder of the irma task.

The information required to determine the main status is gained partly from the pre-defined MPA_MAINSTAT byte, however, two of the status bits must be generated by this routine. These are the "Aux (auxiliary) Status change has occurred" bit and the "trigger occurred" bit. The "Aux Status change has occurred" bit is generated by comparing the old and new auxiliary status bytes from the calculation of the auxiliary status. If the values are different the bit is set. If the values are identical, the bit is left in its previous state. It is not cleared because this bit can only be cleared by a DSI command from the System Unit. The "trigger occurred" bit is set if a trigger data match occurs. The System Unit program can define an address location in the screen buffer and a corresponding data byte. If the data byte is found at that location in the actual screen buffer, the trigger occurs. The System Unit program can look for any number of bits in the data byte to match by applying a mask value. It can look for a change of state in the data byte by specifying a mask value of all zeroes. The trigger mask, address location and data byte values are stored in the BCP's data memory and are set by two of the defined DSI commands. The main status routine gets these values from memory and checks the screen buffer to see if the trigger bit should be set. Actually, this function is not used in the IRMA System Unit software. The remaining bits are generated by checking the MPA-II's main status byte for its status. As with the "Aux status change has occurred" bit, the "key buffer empty", "Unit reset by controller", and "buffer modified" bits in the main status register must be reset by the System Unit program. Therefore, the main status subroutine logically "ORs" these bits with their previous value. Two bits defined by the DSI in the main status register are always left cleared by the main status routine. These are the Fatal IRMA hardware error and the command interrupt request bits. After the main status byte has been generated, it is kept in register GP5 for the remainder of the irma task. The main status routine also loads the previous value of the main status from data memory and stores the new value in that same location.

The Attention Request flag section of the irma_status_update routine determines if the Attention Request flag should be set as defined by the DSI. This section compares the old main status value with the new main status value. If it detects that a bit in the old register was a zero and the corresponding bit in the new main status register is a one, it will compare this bit position to the attention mask. If the attention mask also has a "1" in that bit position the Attention Request flag will be set in the appropriate location in data memory. The attention mask is loaded from the BCP's data memory and its value is set by one of the sixteen defined DSI commands.

TABLE 6-2. IRMA Main and Auxiliary Status Byte Definition

Main Status Byte		Auxiliary Status Byte	
Bit	**Meaning**	**Bit**	**Meaning**
(MSB) 7	Aux Status Change has Occurred(*)	(MSB) 7	Unused
6	Trigger Occurred(*)	6	Unit Polled Since Last Status Read
5	Key Buffer Empty	5	Sound Alarm
4	Fatal IRMA Hardware Error(+)	4	Display Inhibited
3	Unit Reset by Controller	3	Cursor Inhibited
2	Command Interrupt Request(+)	2	Reverse Cursor Enabled
1	Buffer Modified(*)	1	Cursor Blink Enabled
0	Cursor Position Set(*)	0	Keyboard Click Enabled

(*) Bits which must be cleared by user program.

(+) Bits which will never be set in MPA implementation.

The final section of the irma task is the command execution routine which is called "irma_command_decode" with the source code located in module IRMA_COM.BCP. This routine, like the others, is implemented as a subroutine to the irma task routine. However, unlike the other routines, it is not executed every time the irma task is run. The System Unit program must have requested that a command be executed or the irma task will skip the command execution routine and return program control to the task scheduler. The irma task determines this by checking the Command Request flag in the IRMA status flag register at memory address 7E27h. If this bit is set the irma task calls the command execution routine.

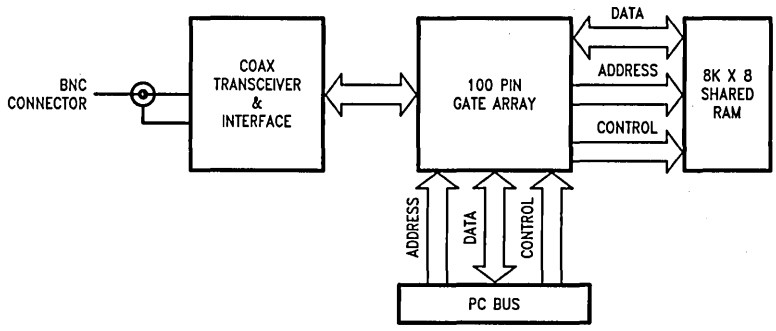
The command execution routine begins by determining which of the sixteen commands is to be executed. This is done by moving the command number data byte at memory address 7F20h into an internal register. It then uses the JRMK instruction and a jump table to transfer program control to the specific routine that corresponds to that command number. The individual command routine then loads any required command arguments from data memory locations 7F21h–7F23h and executes the command. The resulting data is placed in the data memory locations 7E20h–7E23h with the IRMA main status byte always in the first location (7E20h). The command execution routine then clears the Command Request flag in data memory. After this, it returns to the main body of the irma task routine.

The sixteen commands defined by the DSI are thoroughly decumented in the IRMA Technical Reference. The implementation of each command in the command execution routine is well documented in the corresponding section of BCP source code. For reference, the commands and the associated source code routine labels are given in Table 6-3.

As mentioned earlier, the MPA-II software uses a synchronous method of passing some status information between tasks. This is necessary because the status information can be updated on both foreground and interrupt routines. In this case the updating of such status information must be synchronized between the routines or the data could be corrupted. The synchronizing method is a "mailbox" in memory where the location of the status information and the change required is placed. The irma task uses the sync_mailbox to tell the coax task when to reset the "cursor change", "key buffer empty", "unit polled since last status read", and "unit reset by controller" status bits. The irma task also uses the mailbox to tell the coax routine that the System Unit has instructed the MPA-II to execute a Power On Reset sequence on the coax. The irma task accumulates the status change information in register GP2 throughout the routine (more specifically the cursor change reset from the main status routine and the others from the command execution routine). It then loads the mailbox just before returning to the task scheduler.

TABLE 6-3. IRMA DSI Commands and the Corresponding MPA-II Source Code Labels

	IRMA DSI Commands	MAP-II IRMA Command Source Labels
Code	**Command Definition**	**Source Code Label**
0	Read Buffer Data	irma_com_read_buffer
1	Write Buffer Data	irma_com_write_buffer
2	Read Status/Cursor Position	irma_com_status_cursor
3	Clear Main Status Bits	irma_com_clr_mstatus
4	Send Keystroke	irma_com_send_keystroke
5	Light Pen Transmit	irma_com_lpen_transmit
6	Execute Power-On-Reset	irma_com_por
7	Load Trigger Data and Mask	irma_com_trig_data_mask
8	Load Trigger Address	irma_com_trig_addr
9	Load Attention Mask	irma_com_attn_mask
10	Set Terminal Type	irma_com_set_term
11	Enable Auxiliary Relay	irma_com_aux_relay
12	Read Terminal Information	irma_com_read_term
13	Noop	irma_com_noop
14	Return Revision ID and OEM Number	irma_com_rev_oem
15	Reserved-Do Not Use	irma_com_reserved



TL/F/10488-28

FIGURE 6-12. IBM Hardware Implementation

IBM Interface Overview

The IBM Personal Computer 3270 Emulation Adapter Version A uses sixteen I/O mapped locations, PC interrupt level 2, and 8K of re-mappable shared RAM to provide the necessary hooks to do 3278/79 terminal emulation, 3287 printer, and DFT emulation. The PC emulation software reads and writes to the I/O locations to determine session status and reads the screen buffer maintained in the shared RAM when screen updates are made by the coax controller. The shared RAM concept and use of a PC interrupt make the speed of the terminal emulator very fast and efficient.

The IBM Adapter card uses a gate array, PALs and various logic chips to manage the interface and coax sessions. A block diagram of the IBM adapter hardware is shown in *Figure 6-12*. The sixteen I/O locations reserved for the interface are physically resident in the gate array located on the IBM Emulation Adapter card. The addresses of the sixteen I/O locations are 2D0h–2DFh. PC register addresses along with their corresponding read and write capabilities are defined in Table 6-4. The PC accesses the registers in four different modes of operation which are: 1) read only, 2) write only, 3) read/write, and 4) read/write with reset mask. The first three modes are self explanatory. The read/write with reset mask mode, also knows as "Write Under Mask" or WUM mode, means that the PC reads the value of the register as a normal I/O read to acquire the information. After reading the byte, the PC will write a mask with ones in the bit positions that the PC wishes to clear. This "write with reset mask" is usually used as an acknowledgement that the byte has been read by an earlier read. The resulting contents of the register will be cleared in bit positions that were written with corresponding ones. A brief description of each register and its function follows. For a detailed discussion on each register, refer to the *IBM 3270 Connection Technical Reference* (see References in Appendix D).

TABLE 6-4. IBM Emulation PC Register Address Locations and Read/Write Functionality

Address	PC Register	PC Read	PC Write
02D0	PC Adapter Interrupt Status	Data	Reset Mask
02D1	Visual Sound	Data	Reset Alarm
02D2	Cursor Address Lo	Data	—
02D3	Cursor Address Hi	Data	—
02D4	PC-Adapter Control	Data	Data
02D5	Scan Code	—	Data
02D6	Terminal ID	—	Data
02D7	Segment	—	Data
02D8	Page Change LO	Data	Reset Mask
02D9	Page Change HI	Data	Reset Mask
02DA	87E Status	Data	Reset Mask
02DB–02DF	Reserved		

PC Adapter Interrupt Status Register (2D0h)

The Interrupt Status register contains six interrupt flags and two status bits. The interrupts are set based on events occurring on the coax. If the interrupts are enabled in the Adapter Control register (2D4h), the PC interrupt level 2 (IRQ2) is set when one of the six interrupt conditions occur. The buffer-being modified status flag is set when the screen buffer is being modified by a WRITE DATA, a CLEAR, or INSERT command. The interrupt status flag is set whenever any interrupt has been set. The register is read/write with reset mask by the PC as defined above. To acknowledge an interrupt, the PC will write back to the register with a one in the corresponding bit location of that interrupt. That clears the interrupt. The wum scheme provides a clear handshake between the two asynchronous systems. This register is used by all three emulation modes (i.e., CUT, DFT and Printer mode). The definitions of some of the bits change depending on the currently active mode.

Visual/Sound Register (2D1h)

The Visual/Sound register contains control settings for the terminal that are affected by the load control register command, clicker status, and alarm status. This register is a PC wum with a different twist. Any value written to this register results in the clearing of the alarm bit only. Other bits are not affected by the PC write. This register is only used in CUT mode.

Cursor Address Low and High Registers (2D2h and 2D3h)

The Cursor Address registers contain the sixteen bit cursor value owned by the coax controller. These registers are read only by the PC and provide the location of the current cursor position. These registers are used in all three modes.

PC Adapter Control Register (2D4h)

The Adapter Control register determines the mode of operation of the adapter (i.e., 3278 terminal, 3287 printer, or DFT emulation), controls keystroke passing with a bit used as a handshake, and controls the masking of interrupts. The remaining bits control various operation situations (i.e., enabling/disabling the coax session, keystroke wrap testing etc.). This register is read/write by both the PC and the adapter. This function makes synchronization of reads and writes critical to ensure no data is lost. This register is used in all three modes. Some of the bit definitions change depending on the active emulation mode.

Scan Code Register (2D5h)

The Scan Code register, as the name implies, is where keyboard scan codes are written by the PC corresponding to the keystrokes struck on the keyboard. This register is PC write only and the byte written is the one's complement of the scan code to be sent to the host. This register is used in CUT mode only.

Terminal ID Register (2D6h)

The Terminal ID register is write only by the PC and should not be changed once the terminal has gone on line. The value written is the one's complement of the keyboard ID and model number of the terminal that will be requested by the coax controller when initializing the session. This register is used by all three modes.

Segment Register (2D7h)

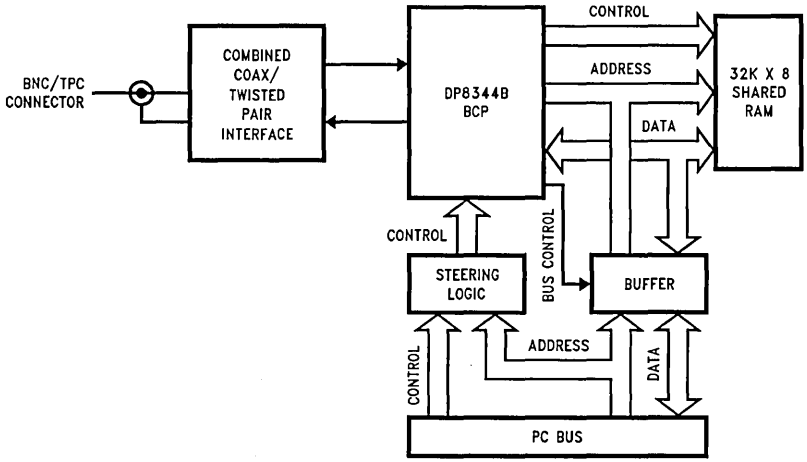
The Segment register is used for relocation of the dual port memory segment at which the adapter recognizes a memory read or write from the PC. The default value is CE. This register is write only by the PC.

Page Change Low and High Registers (2D8h and 2D9H)

The Page Change registers are used to communicate a change in the screen buffer. Each bit corresponds to a 256 byte block of the 4K screen buffer and is set by the adapter hardware when any screen modification occurs. The register is read/write with reset mask by the PC as described earlier. These registers are active for all three modes.

87E Status Register (2DAh)

The 87E status register contains status flags relevant to 3287 printer emulation. Included is a flag for the alarm and operation condition of the printer. The register is read/write with reset mask by the PC as described earlier.



TL/F/10488-29

FIGURE 6-13. MPA-II Implementation of IBM Emulation Card

The Multi-Protocol Adapter Solution

The Multi-Protocol Adapter (MPA-II) card has the ability to emulate the IBM Personal Computer 3270 Emulation Adapter allowing the IBM PC emulation programs to run using the MPA-II hardware in place of the adapter card while maintaining the same functionality. To emulate the adapter, the MPA-II utilizes the power of the DP8344A BCP to handle the coax session and interface maintenance in software. *Figure 6-13* gives a block diagram of the MPA-II hardware.

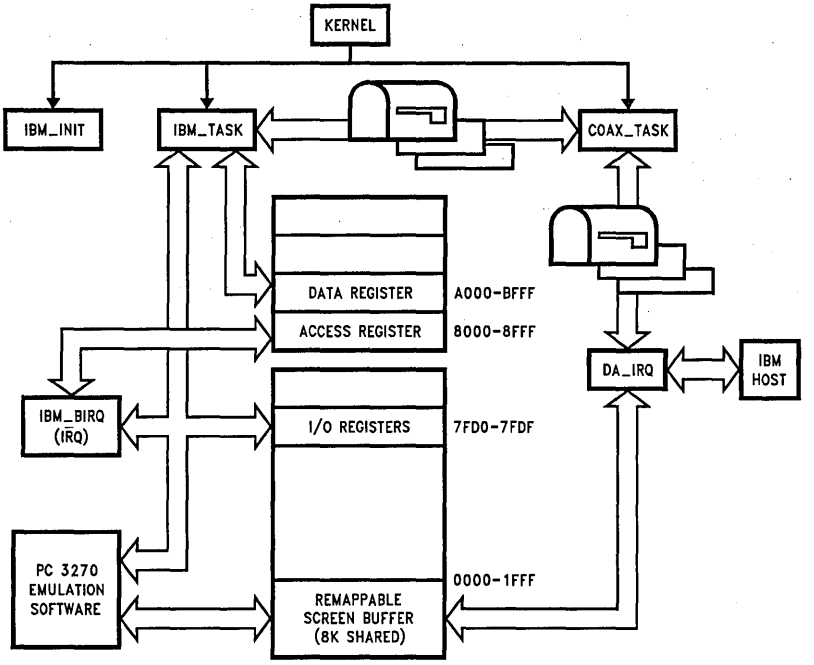
The I/O registers described above are maintained in a shared RAM located on the MPA-II board and the BCP software must "fake out" the PC software when any register update is made, leaving the correct value in the RAM for the next access. To emulate the function of the I/O registers, the MPA-II hardware sets the bi-directional interrupt pin (BIRQ) low on any PC write to the IBM I/O locations 2D0h–2D6h and 2D8h–2DEh. The write to the I/O location is routed into locations in the shared RAM. The mapping of the I/O registers in the shared RAM is shown in *Figure 6-14.* The BCP Code Variable Address column in *Figure 6-14* shows the variables used in the MPA-II source code to form the absolute RAM address of the I/O register contents. The PCIO value is a sixteen bit value and is the base pointer into the page of memory where the I/O registers reside. The variables listed are added to the PCIO base to form the absolute address pointer to the specified register in data memory. All registers that are cleared by the write under mask scheme have duplicate copies that are maintained solely under BCP control to allow software implementation of the write under mask handshake.

The BCP software, to handle the interface and coax routine, contains interrupt driven routines as well as foreground routines. A block diagram showing the code arrangement used to handle the IBM interface and coax session is shown in *Figure 6-15.* Four blocks run as tasks while the interrupt sources are used where immediate attention is required (i.e., the communication with the controller [receiver interrupt] and the PC interface maintenance [BIRQ interrupt]). The three sections of code that will be discussed below are responsible for initializing the I/O registers at power up, maintaining the I/O registers, and setting/clearing the PC level 2 interrupt. Each routine is described in the paragraphs that follow.

PC I/O Address		BCP CODE Variable Address (Absolute RAM address = PCIO value)
02D0	Interrupt Status / Local Copy	ibm-isr ibm-lisr
02D1	Visual/Sound / Local Copy	ibm-vsr ibm-lvsr
02D2	Cursor Address Low	ibm-cursorlo
02D3	Cursor Address Hi	ibm-cursorhi
02D4	Adaptor Control	ibm-control
02D5	Scan Code	ibm-scan
02D6	Terminal Id	ibm-id
02D7	Segment	ibm-segment
02D8	Page Change Low / Local Copy	ibm-pagelo ibm-lpagelo
02D9	Page Change High / Local Copy	ibm-pagehi ibm-lpagehi
02DA	87E Status / Local Copy	ibm-status ibm-lstatus

TL/F/10488-30

FIGURE 6-14. IBM I/O Register Mapping



TL/F/10488-31

FIGURE 6-15. IBM Interface Code Block Diagram

IBM_Initialization

The ibm_init routine initializes the I/O registers to the expected state at power up and initializes internal BCP variables in preparation for a new session. After clearing the screen buffer, the program schedules the ibm_task routine as a task to the Kernel routine and unmasks the BIRQ interrupt to enable the ibm_birq_int routine to run when the PC writes to the IBM I/O registers. This code is only executed when the card initially runs at power on time or when changing MPA-II modes via the MPA_CONFIG register. Upon completion of this and other initialization routines, the PC emulation software can be started to bring the PC emulator resident.

IBM_BIRQ Interrupt Routine

The BIRQ routine is unmasked by the ibm_init routine as mentioned above. The BIRQ input goes low (asserted) when the PC writes to the IBM I/O locations 2D0h–2D6h and 2D8h–2DEh. BIRQ is unaffected by PC reads of the I/O locations since no action is required by the MPA-II board. At the same time BIRQ is asserted, the MPA-II hardware "locks out" the PC from performing any further memory or I/O accesses to the MPA-II board until the BCP software "unlocks" the PC. When the BIRQ interrupt handler, ibm_birq_int, gets control, it first reads the Access register (mpa_access) to determine which IBM I/O register has been written to. If the I/O register written to is a read only or write only register then no action is required by the interrupt routine so the routine unlocks the PC by writing any value to the Access register, and then exits. If the I/O register written to is a WUM type register then the BIRQ interrupt routine complements the value currently in the I/O register location (for it is the mask value written by the PC) and ANDs it to the local copy of that I/O register. The result is then placed into the I/O register location as well as into the local copy memory location. The PC is then unlocked by the interrupt routine and the routine exits. A write to the Visual/Sound IBM register of any value causes the local copy to be retrieved, its alarm bit cleared, and both the I/O register and its local copy to be updated. The Interrupt Status IBM register will not only have the WUM performed, the interrupt routine will also de-assert the IRQ PC interrupt line by writing a zero in bit position 7 to the Data register (mpa_data). Bit 7 of the Data register controls the state of the PC's IRQ interrupt line. The PC interrupt is set in the ibm_task routine (IBM_TASK.BCP) if interrupts are pending and not disabled.

There is a simplified version of the ibm_birq_int BIRQ interrupt handler called ibm_fast_birq. The ibm_fast_birq routine is directly called by the receiver interrupt handler in between the processing of coax data frames in order to handle PC activity without impacting the coax command 5.5 μs response timing, which is so critical. The ibm_fast_birq routine is identical to the ibm_birq_int routine except that it does not perform any saving or restoring of BCP registers since this is handled by the receiver interrupt handler.

IBM_TASK Foreground Routine

The ibm_task routine runs in the foreground and is called by the Kernel. The ibm_task is enabled to run by the ibm_init routine. Once it has been scheduled by the initialization routine, the ibm_task runs any time it is called by the Kernel.

The primary purpose of the ibm_task routine is to keep the I/O registers current as to the state of the emulated terminal session so that the PC software can update the screen in a timely manner. The ibm_task routine maintains communication with the coax task routine via a two byte mailbox in data memory. The ibm_task routine monitors coax activity through bit settings in the MPA-II status variables (mpa_mainstat and mpa_auxstat) and updates the I/O Interrupt Status register, Visual Sound register, PC Adapter Control register, and PC interrupt level, IRQ2, accordingly. The task is non-interrupt driven and uses both main banks of the CPU for processing.

The ibm_task routine first checks the MPA-II status variables, mpa_mainstat and mpa_auxstat, clearing certain status bits (such as Buffer Modified) to acknowledge receipt of that status. Next, the ibm_task updates the IBM Page Change registers and the IBM Cursor registers since they are common to all three interface modes, (CUT, DFT, and Printer). The ibm_task routine then determines the current interface mode and calls that interface mode's routine to update the remaining IBM register specific to that mode.

For CUT mode, ibm_task calls the ibm_3278 routine. This routine updates the Visual/Sound register (2D1h), the Adapter Control register (2D4h), and the Interrupt Status register (2D0h). The ibm_3278 routine will also interrupt the PC via its IRQ interrupt line if PC interrupts have not been suppressed by the Adapter Control register.

For DFT mode, ibm_task calls the ibm_dft routine. This routine updates the Adapter Control register (2D4h) and the Interrupt Status register (2D0h). As with the ibm_3278 routine, this routine will also interrupt the PC via its IRQ interrupt line if PC interrupts have not been suppressed by the Adapter Control register.

The 3287 Printer mode is not supported in this version of the MPA-II microcode, but may easily be added. In fact, Revision B of the IBM Emulation Adapter can also be supported through simple microcode enhancements if the MPA_CONFIG register (2DCh), MPA_PARM register (2DBh), and BCP RIC register (2DFh) are relocated. (Relocating these registers only requires some simple PAL equation changes for the existing hardware.) That is one of the advantages of the soft architecture concept that the BCP allows. Not only is your product protected against changes on the Coax side of the interface, but your product is also protected against changes on the PC side of the interface!

After the above routines return to the ibm_task routine, the ibm_task routine sends mail via sync_mailbox back to the cx_task routine, if anything needs to be communicated to the coax side, such as keystrokes. Then ibm_task returns to the kernel.

Twinax Task

The twinax task tw_task (located in module TW_TASK.BCP) is responsible for directing twinax terminal emulation. It monitors all seven internal twinax sessions for current polling status, for 2 second Auto-POR time-outs, and for 5 second POR OFFLINE timeouts. In addition, tw_task invokes the twinax command processor, tw_session (located in module TW_SESS.BCP), for each twinax session that requires attention.

When the MPA_CONFIG register is set (or changed) to select twinax emulation, the task housekeep calls tw_init (located in module TW_TASK.BCP) to initialize the twinax routines, and then calls tw_sa_init (located in module SA_INIT.BCP) to initialize the smart alec interface routines. The routine tw_init initializes the hardware interface for twinax, initializes and unmasks the twinax receiver interrupt, initializes and unmasks the transmitter interrupt, initializes and unmasks the timer interrupt, initializes the twinax dependent Device Control Page (DCP) variables, and initializes all seven Session Control Pages (SCPs) for twinax emulation. The initialization of everything except the SCPs is straight forward; the appropriate bits and bytes are simply set to their required values. The initialization of the SCPs are a bit more complicated, however, with the following steps performed for each SCP. First, the SCP is filled with "55" hex (as a debugging aid). Second, tw_por (located in module TW_CNTL.BCP) is called, which initializes the twinax dependent SCP variables, except for these set by the Smart Alec interface routines (i.e., Model ID, Keyboard ID, etc . . .). Third, tw_init takes each session out of POR since a true POR has not been requested yet. (A true POR can only be performed on an active session). After the SCPs are initialized, tw_init schedules the twinax task tw_task to run under the Kernel. It is tw_task's job to direct twinax emulation in the foreground. Tw_init then returns control to house-keep, which in turn calls tw_sa_init. The tw_sa_init routine initializes the memory locations and internal registers that are used by the Smart Alec emulation code. This is discussed in detail in the Smart Alec Interface Overview section later in this chapter. House-keep then enables interrupts and returns control to the Kernel's tasker with the twinax emulation and interface tasks now scheduled to execute.

The monitoring functions performed by tw_task break down into two groups: ONLINE sessions, those sessions which are configured by the Smart Alec emulator (attached) and seen by the host 3x or AS/400 system; and OFFLINE sessions, whose sessions are not configured by the Smart Alec emulator (unattached) and therefore not seen by the host 3x or AS/400 system. ONLINE (configured) sessions are monitored for current polling status, Auto-POR time-outs, and POR OFFLINE time-outs. Current polling status simply indicates whether the physical address for a session is being polled at least once every 2 seconds. When this is false, tw_task clears the line active indicator for that session. (The System Available indicator status is monitored by the smart alec interface task). An Auto-POR time-out occurs when tw_task determines that 2 seconds have elapsed since the last poll to a physical address. The task tw_task request that the session attached to that physical address perform a POR. It then schedules the session in question so that the request will be processed. (Scheduling sessions is discussed in the following paragraph.) POR OFFLINE time-outs occur when tw_task determines that 5 seconds have elapsed since a given session initiated a POR. It is tw_task's responsibility to bring the session ONLINE by signaling the receiver interrupt handler to start responding to and accepting commands from the host 3x or AS/400 system. OFFLINE (non-configured) sessions are only monitored for current polling status.

After every internal session has been checked by the monitor, tw_task invokes the twinax session command processor, tw_session for each scheduled session. (This action is similar to the Kernel's tasker.) Both background and foreground tasks schedule sessions when they require a session to perform some sort of action. For example, a session is scheduled when a new command is placed onto the internal command queue, or when another task, such as the smart alec interface task, requires a session to POR. The task tw_task calls the twinax command processor, tw_session, and passes a pointer to the SCP of the scheduled session.

The command processor then performs the requested action and/or executes the command(s) in the internal command queue.

When all the sessions have been checked and all the scheduled sessions have been processed by the command processor once, tw_task returns control to the Kernel's tasker.

Twinax Interrupt Handlers

The twinax mode uses four interrupts: DAV, Data Available, for handling receiver data; TFE, Transmitter FIFO Empty, for all responses; TIMER for handling response window timing and as a real time clock for 5250 protocol requirements; and BIRQ for host interface accesses. All interrupts except BIRQ are unmasked in the tw_init routine after initialization requirements for each have been executed. The BIRQ interrupt is unmasked in the sa_init routine. As with the coax interrupt routines, the twinax interrupt routines can use the alternate B bank registers without having to save and restore them. The twinax DAV and TFE interrupt routines are set up as state machines whose current state is stored in the "DATA_VECTOR" and "TX_VECTOR" memory locations. IW and IX are reserved for the TX_VECTOR and DATA_VECTOR addresses that point to the appropriate state in the TFE interrupt and DAV interrupt routines, respectively. The TFE routine always expects TX_VECTOR to be set appropriately upon entry. DAV loads the DATA_VECTOR from memory upon reception of the first frame of a message and uses IX directly for frames 2-n. Also, GP5 on alternate B bank has been reserved for DAV, TFE, and TIMER interrupt routine usage. The name of this register is "R_STATE" since it is used primarily by the receiver for station address information and protocol control.

Twinax Receiver Interrupt Routine

The DAV interrupt routine is responsible for decoding the commands sent by the controller, loading commands on the internal processing queue, stuffing data in to the regen buffer, "OFFLINE" address activity determination, maintaining protocol related real time status bits, and supporting all seven station addresses if necessary. A flow diagram of the DAV interrupt routine is shown in *Figure 6-16*.

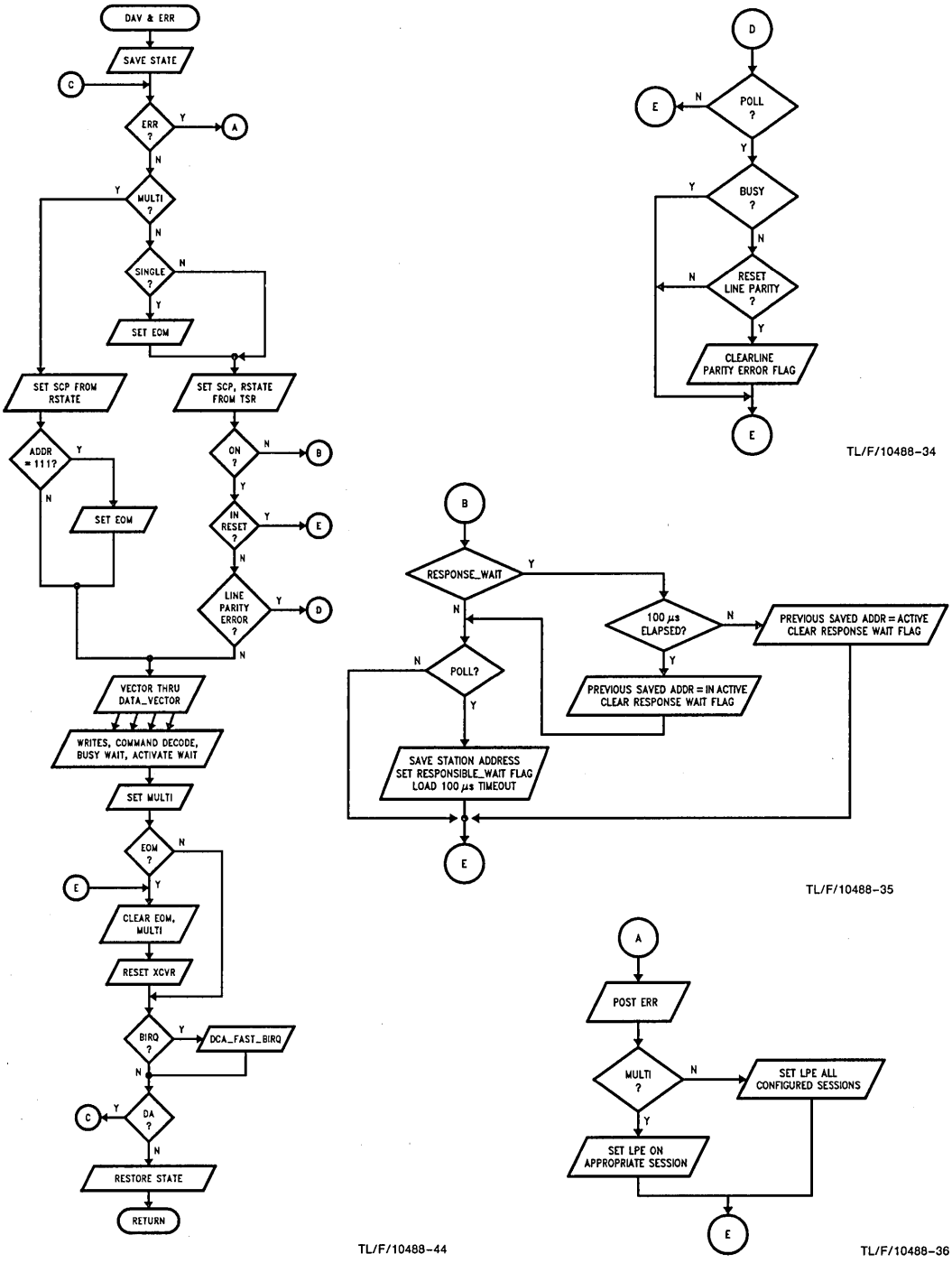


FIGURE 6-16.Twinax DAV Interrupt Routine

Initialization requirements of the DAV interrupt are:

1. R_STATE (GP5 on alternate B) set to TW_RSTATE_INIT;
2. tw_level_cnt set to TW_LEVEL_INIT;
3. tw_busy_cnt set to TW_BUSY_MAX.

The Main A Alternate B bank of registers are first selected and IZ is saved so that it can be restored upon exiting the interrupt. Since the DAV interrupt source is an "OR" of both the reception of a valid data frame and the flagging of an error by the receiver, a check for an error is done first to make this destination. (Error handling will be discussed later in this section.)

A key pivotal point in the routine is controlled by a flag set in R_STATE called RX_MULTI which is set after processing the first frame of a multiframe message. The purpose of RX_MULTI is to ensure that the received station address is only sampled on the first frame of each message from the controller and causes the DAV interrupt routine to search for the "111" end of message delimiter on all subsequent frames received. The station address saved in R_STATE[2-0] will be used by the receiver for setting the SCP pointer on all subsequent frames for setting the SCP pointer on all subsequent frames of the multiframe message. When the end of message is detected, the flag RX_EOM is set in R_STATE. If RX_EOM is set at exit time, then RX_MULTI and RX_EOM will be reset along with the transceiver to ensure that any errors flagged by the receiver logic of the BCP resulting from a noisy line after the transmission of the fill bits will be ignored. If RX_MULTI is not set, the data received is either the first frame of a multi-frame message or a single frame command. In this condition, the station address is placed in R_STATE[2-0] and IZ is set to point to the SCP page of memory corresponding to the station address. RX_EOM will get set here only if the data is a single frame command, which is determined by the state of RTR[0] (bit 14, see 5250 PAI). The station address received is the "physical station address" and should not be confused with the "logical station address" which is used solely by Smart Alec for aesthetics. The physical station address is loaded into bit 8-10 of the sixteen bit SCP pointer. This scheme provides 256 bytes of data memory for emulating each station address.

Once the SCP pointer has been established, the receiver interrupt must know if the station address of the data received is currently being emulated ("ONLINE") or is not being emulated ("OFFLINE"). Addresses that are offline have to be monitored for activity to inform Smart Alec whether or not the address can be attached as an online session in the future (see OFFLINE section for line activity determination).

When the session in ONLINE, checks are made upon reception of the first frame of the message to see if the session is currently in a reset state or if a line parity error is pending. For subsequent frames of the mesasge, no checks are made for reset or pending line parity errors, although each frame is still parity checked. The reset state is determined by the RX_RESET flag stored in tw_rxtx_status on each SCP page. When the reset flag is set, all data is ignored. The line parity error state is needed since once a line parity error is detected, only POLL commands are processed by the terminal until the error condition is cleared. The error is cleared when a POLL is received with the Reset Line Parity Error bit set in conjunction with the terminal being in the non-busy state. (See POLL discussion in 5250 PAI).

If the terminal is not in a reset condition and no line parity error is pending, the DATA_VECTOR is loaded to determine what state to branch to. The DATA_VECTOR must be stored on the SCP page due to the multi-session nature of twinax. When the first frame of a message is received, the IX index register is loaded from the SCP tw_data_vectorhi and tw_data_vectorlo locations prior to the indexed jump to the appropriate processing state. For frames 2-n of a message, IX is used in its current state for processing speed since it is reserved for the interrupt and is already set accordingly.

Command/Data Processing Routines

There are basically four states used in the DAV interrupt routine: 1) command decode, 2) writes, 3) busy_wait, and 4) activate wait. Each state is vectored to via an indexed jump using the DATA_VECTOR as discussed above. However, when exceptions are detected by the foreground command processing routines, the DATA_VECTOR is modified.

The command decode state, as the name implies, is where the received byte is decoded and pushed onto the 16 byte internal processing queue as specified in the 5250 protocol. Commands are decoded first by checking to see if the command is a POLL. Next, two jump tables are used to further decode the command. One table is used for commands addressed to features (i.e., RTR[7] = 1) and only the lower four bits of the command are decoded. The other jump table processes all commands in base format so the lower five bits of the command are decoded. No destinction is made as to what internal device is addressed since this is done by the foreground tw_session routine when the command is unloaded from the queue. The only commands that can have duplicate meanings in this scenerio are the END OF QUEUE and RESET BASE since they are identical in the lower five bits of the commands. They are further processed before being loaded onto the queue to handle this overlap.

Once the command is decoded, it is loaded onto the queue by the QUE_LOADER routine which will be discussed later. Since commands may or may not have associated operands with them, the DAV interrupt modifies DATA_VECTOR appropriately for the command just decoded. Single frame commands do not change the DATA_VECTOR from command decode since there are no operands associated with them. This is not true for the end of queue command as it results in the DAV routine moving into the busy_wait state which will be discussed later. Commands that have associated operands with them, for example LOAD ADDRESS COUNTER, set the DATA_VECTOR to the rx_operands routine and a frame count value is maintained on the SCP (tw_frame_cnt) to control how many additional frames stay in the rx_operands state for processing the entire command packet. Some commands require special routines to process them. The READ and WRITE IMMEDIATE commands set DATA_VECTOR to rx_imm_operands so that it will be set to activate_wait upon completion of the commands operands. WRITE CONTROL DATA requires a special stub since it can be a +2 operand command or +3 for the 3180 emulation (see 5250 PAI). WRITE DATA AND LOAD CURSOR also requires a special routine since the number of associated operands expected is embedded in the first operand of the command.

After a complete command packet (i.e., the command plus any associated operands) has been loaded into the queue, the DAV interrupt schedules the twinax command processor, tw_session, to process the command. The appropriate session task is scheduled by moving TW_SESS_SCHED into tw_sess_state on the SCP corresponding to this command's physical address. This scheme provides the communication to the foreground task to tell it which of the seven sessions to process.

The QUE_LOADER routine is called upon reception of all commands and operands that are queable and handles stuffing the command in the queue with some exception detection. (Commands that are not queable are POLLS and ACTIVATES.) The QUE_LOADER maintains the position of commands on the queue and status of the queue with a byte on the SCP called tw_que_ptr. The lower five bits of the byte form a pointer to the next available position to stuff a byte on the queue. Each time a byte is loaded, the pointer is incremented making bit 5 correspond to the queue being full (TW_QUE_FULL) since it will be set upon loading the sixteenth entry into the queue. Another flag, TW_QUE_NOT_RDY, in tw_que_ptr is used to tell tw_session if a complete command packet (i.e., a command and associated operands) is ready for processing. This flag uses tw_frame_cnt to determine packet boundaries and allows tw_session to process packets as soon as they are available, instead of waiting for a complete queue load before processing the queue. If QUE_LOADER detects that the queue is full, flag TW_QUE_COMPLETE in tw_que_ptr is set and DATA_VECTOR is set to busy_wait for handling busy. TW_QUE_COMPLETE is used as a handshake between the background DAV interupt and foreground command processor to communicate when the terminal can go unbusy. Exceptions that are set by QUE_LOADER are invalid command and queue overrun exceptions. When an exception is deteted, it will not be set if there is already a pending exception. Also, when the exception is detected, the DATA_VECTOR is set to busy_wait to ensure that the terminal will go unbusy to allow the controller to handle the posted exception. The invalid command exception is posted by the queue loader and the tw_session command processor. QUE_LOADER will post an invalid command when a command with associated operands is loaded in the last queue position but operands are still expected. The queue overrun exception is posted when the sixteenth frame received completes a queue load but the RX_EOM flag is still set meaning more frames are still being received.

The busy_wait state of the DAV interrupt has a number of functions. The DATA_VECTOR is set to busy_wait when exceptions are detected in both foreground and background routines. Also, DATA_VECTOR is set to busy_wait upon receiving a complete queue load of sixteen frames or the reception of an End Of Queue command. The major role of the busy_wait state is to handle the transition of busy (i.e., having commands on the queue) to unbusy (queue empty waiting for more commands). To go unbusy the foreground command processor must have finished processing all the commands from the prior queue load. Once the last command of the queue load is received, TW_QUE_COMPLETE is set by DAV in tw_que_ptr to mark the completion of the queue load. Then, in busy_wait, the DAV routine uses the clearing of TW_QUE_COMPLETE as an indication to clear the POLL response busy bit. In conjunction with TW_QUE_COMPLETE, the DAV interrupt maintains a POLL counter called tw_busy_cnt to provide maximum flexibility in going unbusy. In has been observed that some IBM controllers require that after a complete queue load is received, the terminal must be busy for some finite amount of time before being unbusy. To accomplish this task, the value of tw_busy_cnt is decremented with each POLL received while in the busy_wait state. Upon reaching a count of zero with TW_QUE_COMPLETE low, busy will go low in tw_presp_stat and tw_busy_cnt will be reinitialized to TW_BUSY_MAX in preparation for the next queue load. The TW_BUSY_MAX equate is set up in TWINAX.HDR and should be set accordingly. We recommend that TW_BUSY_MAX be set to one since older versions of the 5294 remote controller require at least one "busy" POLL response after a queue load. If a command other than a POLL is received prior to signaling unbusy, the DAV will process the command and set DATA_VECTOR to command decode if TW_QUE_COMPLETE is low. In this case, the tw_busy_cnt value is ignored to ensure that commands are not discarded.

When a preactivate READ or WRITE command packet is completely received, the DATA_VECTOR is set to the activate_wait state. The role of activate_wait is to handle the transition of busy to unbusy (as with busy_wait), to flag an invalid ACTIVATE exception if the controller sends the ACTIVATE before the terminal is unbusy, set up the write_both state for reception of ACTIVATE WRITEs, and schedule the response for an ACTIVATE READ reception. As with busy_wait, TW_QUE_COMPLETE hass been set high before entering this state and the interrupt routine uses both TW_QUE_COMPLETE low and tw_busy_cnt equal to zero as criteria for going unbusy. Once the terminal is unbusy, a flag stored in tw_rx_act_flags called RX_PREAC_WR determines whether or not to look for an ACTIVATE WRITE or an ACTIVATE READ command. When an ACTIVATE WRITE is received and expected, the busy flag is set in tw_presp_stat to ensure that the terminal is busy upon completion of the write and the DATA_VECTOR is set to write_both since the WRITE IMMEDIATE command and WRITE DATA command are similar enough to be handled by one state. When an ACTIVATE READ is received or expected, a response is scheduled by loading a timeout into the timer and setting TW_TIMER_RESP in R_STATE. Also, busy is set so that at the end of the read the terminal is busy, and DATA_VECTOR is set to command decode in preparation for the next queue load. Commands other than ACTIVATEs are simply discarded in this state. An invalid ACTIVATE exception is posted if the expected ACTIVATE arrives before the terminal is unbusy. TW_QUE_COMPLETE is set in conjunction with TW_QUE_CORRUPT to tell tw_session to flush the queue. DATA_VECTOR is set to busy wait to handle going unbusy. As with QUE_LOADER, the exception is only posted if there is no pending exception.

As mentioned above, DATA_VECTOR is set to the write_both state to handle stuffing data in the regen buffer following reception of the ACTIVATE WRITE command. The data is always concatenated with the ACTIVATE WRITE command. The write_both state is responsible for detect-

ing the storage overrun exception when the controller attempts to send data beyond the size of the regen buffer. The only difference at this point between the WRITE IMMEDIATE and WRITE DATA commands is that the address counter remains unchanged with the WRITE DATA command while the address counter is set to one greater than the address of the last byte stuffed in the WRITE IMMEDIATE comand. To determine whether a WRITE IMMEDIATE or WRITE DATA command is being processed, a flag in tw_rx_act_flags called RX_WR_DATA is set upon reception of the WRITE DATA command. To minimize time on the DAV interrupt, the WRITE DATA or WRITE IMMEDIATE command routines set up the starting location of the write in tw_act_beginhi/lo on the appropriate SCP. Tw_act_beginhi/lo are then used as a pseudo address counter as each byte is received, incrementing upon stuffing the byte in the regen buffer. Upon completion of the write, which is determined by reception of an end of message indicator (RX_EOM set), the pseudo address counter is placed into tw_act_endhi and lo locations with the most significant bit of tw_act_endhi set to inform tw_session that the write is complete. tw_session can then make an action stack entry for Smart Alec screen updates.

POLL

POLL commands are processed completely by the background interrupt routines. The POLL command is decoded in several states since polls play a part in all states mentioned above. The key decisions that are made in the DAV interrupt when a POLL is received and the associated station address is configured by Smart Alec are, what is the state of level and what "type" of POLL response to make. The 5250 PAI states that after a Power On Reset, the 5251-11 will respond with a single frame POLL response that is simply a status byte. After the SET MODE command is received, the next reception of a POLL/ACK command causes the terminal to respond with a two frame poll response; the first frame being the former mentioned status byte and the second a keystroke. Also, the PAI states that the first two frame response after receiving the SET MODE will be from level 1. To function in this manner, a flag called TW_PACK_SM is maintained by the DAV interrupt in location tw_level_cnt on the SCP. This bit is set when TX_SET_MODE_RCVD (a SET MODE command has been processed) located in tw_rxtx_status is set and a POLL/ACK is received. Level is used to indicate to the controller that new status is available from the terminal and toggles each time a new keystroke is presented. The reception of a POLL/ACK after the terminal has been put in the two byte response mode results in the POLL response with level toggle from its prior state. Each toggle of level also contains a new keystroke, if available. The section of code in the DAV routine that handles level transition is rx_level_hndlr.

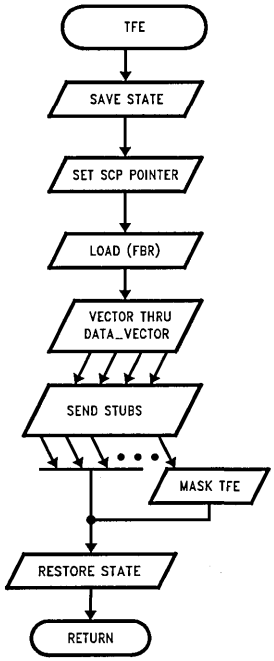
POLLs to nonconfigured station addresses do not result in a response but are used in monitoring activity on station addresses for Smart Alec address bidding purposes. When a frame to an OFFLINE address (i.e., not configured by Smart Alec) is received, the OFFLINE activity monitoring routine is responsible for setting or clearing bits corresponding to each OFFLINE address in tw_line_act on the DCP. Each bit in this location corresponds to a physical address on the network (therefore bit7 is unused), and is set when another terminal or printer is active on that particular address. If the address is available for attachment, the corresponding bit is cleared. Smart Alec monitors this status regularly to communicate to the user whether or not he can attach to addresses via seven locations on the screen. To determine if the address is active, the DAV interrupt looks for POLLS on all OFFLINE addresses. Once a POLL is received, RX_RESPONSE_WAIT and TW_TIMER_RESP flags are set in R_COUNT into the timer to set a time limit for a response to be received. Also, R_STATE is saved at tw_off_save addr on the DCP to store the address and response flag. The next time the DAV interrupt hits with a frame to this address, tw_off_save_addr is fetched to see whether we are waiting for a response or not. If we are waiting for a response, RX_RESPONSE_WAIT is checked. If the timer interrupt routine has already run, RX_RESPONSE_WAIT will be cleared which means that a response was not received and the saved address is marked inactive. If RX_RESPONSE_WAIT is still set, this means that the frame just received was a response and the saved address is marked active. When an address is marked active, the save address and response flag are cleared in preparation for the next OFFLINE reception. When an address is marked inactive, the saved address and response flag are cleared only if the frame received is not a POLL. A reception of a POLL results in the new address being saved with a timeout scheduled just as before mentioned.

Errors detected by the receiver are handled on the DAV interrupt and can result in two different actions. All error types flagged by the receiver are treated as equal importance and are logged by maintaining error counters on the DCP for each error type. The appropriate error counter is fetched and incremented upon reception of an error. Once the error is handled, a check to see if the error occurred in the frist frame of a message or frames 2−n is checked. First frame errors result in the setting of the line parity error detected bit, TW_LP, and TW_BUSY in tw_presp_stat on each of the current ONLINE sessions. Also, the TW_QUE_COMPLETE flag is set in tw_que_ptr marking the End of Queue load to ensure we can eventually go unbusy. The 5250 PAI states that all active addresses will report line errors on the first frame since the error could have occurred in the address portion of the frame. If the error is encountered in frames 2−n of a message, the station's address is known so only that station sets TW_LP in tw_presp_stat. Also, TW_QUE_COMPLETE and TW_QUE_CORRUPT are set since the validity of the queue load is in question. The task tw_session will flush the queue in this case, allowing the terminal to go unbusy. This allows the controller to handle the line error.

All receiver states exit through a common exit point. Upon exit, if RX__EOM has not been set, RX__MULTI is set to indicate that a multi-frame is in progress. If RX__EOM is set, this means that no more frames are expected and results in the transceiver being reset with RX__EOM and RX__MULTI being cleared. Many subroutines in the DAV interrupt branch directly to rx__eom__rcvd which results in the reset just mentioned. Using the transceiver reset capability of the BCP avoids spending unnecessary time on the DAV interrupt processing information of no concern. For example, the OFFLINE activity monitoring routine only looks for POLLS and flushes any other frames. What this means is that the DAV interrupt has to process the first frame of each message but by issuing a reset, subsequent frames of a multi-frame message can be entirely ignored for they will not be recognized by the BCP. After the reset, the receiver hardware looks for a starting sequence and will not extract data until seeing it. Therefore, the remainder of the message is ignored and the next message will be recognized. Before returning, the state of BIRQ is checked to see if a PC I/O access needs service. If BIRQ is low, a call to dca__fast__birq handles the access and returns control back to the DAV interrupt routine. At this point, a check to see if more data is ready for processing is done to avoid unnecessary overhead of exiting the DAV interrupt only to be interrupted again. If no more data is available, IZ, banks and flags are restored on the return back to the foreground routine.

Twinax Transmitter Interrupt Routine

The TFE interrupt routine is responsible for loading the transmit FIFO and making the correct response to the controller. The TFE interrupt is normally masked and is unmasked by the timer interrupt when a response timeout count is encountered. A flow diagram of the TFE interrupt routine is shown in *Figure 6-17*.



TL/F/10488-37

FIGURE 6-17. Twinax TFE Interrupt

Upon entering the TFE interrupt, the contents of the IZ pointer are saved and the pointer is loaded with the appropriate SCP address. The appropriate SCP address corresponds to the physical address of the session that is responding to the controller. The address is stored in R__STATE bits 2-0 and these bits are loaded into IZHI bits 2-0 with IZLO cleared forming the pointer to the first location of the appropriate SCP. Finally, (FBR) is loaded with the value at the tw__mode offset on the SCP to determine the number of fill bits to insert between frames.

Commands that require a response back to the controller are POLLs and ACTIVATE READs. All PREACTIVATE READ commands are processed in the foreground by various command processing routines branched to from tw__ session. The various routines do exception checking and are responsible for setting up TX__VECTOR to the correct address corresponding to the command type decoded. When the ACTIVATE READ is received in the DAV interrupt, a response is scheduled by setting the TW__TIMER__RESP flag in R__STATE and loading a response timeout value into the timer. When the TIMER interrupt hits and it determines that this is a response timeout by checking for TW__ TIMER__RESP set, TW__TIMER__RESP is cleared and the TFE interrupt routine is called to make the response.

POLL commands are handled entirely on the background interrupts due to the real time nature of the status response associated with the command. The DAV interrupt schedules the response just as described above for ACTIVATE READS and sets TX__VECTOR to one of three addresses to cover the various POLL responses that can be made. The first frame of all responses must be sent to the controller in a 45 ±15 μs window as defined in the 5250 product attachment information. The response timing is controlled by loading a timeout value (TW__RESPONSE__CNT) into the timer when reception of a POLL or PREACTIVATE READ command is processed in the DAV interrupt routine. For responses that are less than or equal to four bytes, only one entry into the TFE interrupt is required to send the entire frame back to the controller. To load the fourth byte successfully, a test of TFF is made prior to loading the fourth byte to ensure that the first byte has propagated through the transmit FIFO and is being transmitted out the serial shift register. When responses are greater than four bytes in length, the TX__VECTOR is modified prior to exiting so that the next time TFE hits, the correct state will gain control to continue or complete the remainder of the message. Upon determining that the last frame of the response is ready for load, [TCR2-0] are set to 111 for the end of message delimiter as required by the protocol.

Keystroke passing in the 5250 protocol is different than in 3270. After a POR, 5250 terminals respond with a single status response. For the 5251-11, a SET MODE followed by a POLL/ACK causes the terminal to go into a two byte poll response mode where the second byte is a keystroke. If no keystroke is pending, the keystroke value is a null (00h). New keystrokes can only be presented following a POLL/ACK from the controller. When a new keystroke is made available to the controller, the LEVEL bit in the first frame status byte of the response toggles from the prior value to inform the controller that new status is now available. The DAV routine controls the poll responses by setting the TX__VECTOR to one of three possible locations for POLL or

POLL/ACK responses. For single frame status responses to polls, TX_VECTOR is set to tx_presp_one. As soon as the criteria to go into two frame poll response mode is met, the DAV interrupt sets TX_VECTOR to either tx_presp_crnt or tx_presp_new. In tx_presp_crnt, the keystroke sent back to the controller is the value stored in tw_presp_key_crnt and LEVEL remains unchanged. In tw_presp_key_new, LEVEL is toggled in the first frame status byte response, and tw_presp_key_new is cleared after moving its value to tw_presp_key_crnt. With this approach, keystroke passing with the terminal emulation is simple since by simply checking to see if tw_presp_key_new = 00h determines whether a new keystroke can be loaded for passing back to the controller. In other words, if tw_presp_key_new is nonzero, a keystroke is pending and the emulation program must wait before loading a new keystroke into tw_presp_key_new.

All TFE "states" exit through a common exit point that handles masking the TFE interrupt if no more frames are to be sent, checking to see if a pending BIRQ interrupt is present, restoring foreground registers and restoring banks and flags upon returning. If a BIRQ interrupt is pending, DCA_FAST_BIRQ is called to handle the remote access (see Smart Alec Interface discussion). When more frames need to be sent, all of the above occur except masking the TFE interrupt. Also, TX_VECTOR may be modified to ensure that the correct state is entered upon re-entering TFE when it hits again.

TW-TIMER

The timer the BCP serves dual purposes in the twinax emulation program: as a real time clock counter and as an interval timer.

A 5251 terminal will turn off the System Available flag if no POLL is received for more than 200 ms. It will initiate an automatic power on reset if no POLL is received for more than 2 seconds. Furthermore, the terminal will return to ONLINE from reset mode in approximately 5 seconds. The emulation program uses seven 8-bit counters (tw_sysa_por_cntX, where X is from 0 to 6) to keep track of these real time events (one for each session). These counters are incremented by one every 21 ms. This 21 ms clock tick is generated by the TIMER interrupt. The value of 21 ms gives a maximum counting time (around 5.4 second) and a reasonable counting resolution (±10% for a count of 200 ms). The timer of the BCP is configured to use 1/16 CPU clock as input clock.

In addition, the DAV and TFE interrupts utilize the timer to provide a 45 μs time-out signal. When the receiver routine receives a POLL or ACTIVATE READ command and decides to respond to the host, as per IBM's requirement, it has to do it in 45 μs ±15 μs after the reception of the command. The receiver interrupt will setup the timer to generate a 45 μs time-out signal which in turn activates the transmitter routine. The receiver interrupt first stops the 21 ms counting of the timer, it saves the current counting value, it loads the timer to a count of 45 μs (minus some offset to compensate for program execution time), it then starts the timer and reloads the previous counting value to the timer registers. When time-out occurs, the previous counting value will be loaded into the timer automatically to resume the 21 ms counting. In addition, the program will set a flag to indicate that the timer has counted 45 μs. In this way, the timer is occasionally interrupted from the normal 21 ms counting and "borrowed" to provide a 45 μs time-out. Since 45 μs is much shorter than 21 ms and the interruption is not too frequent, the error introduced is negligible.

When either the 21 ms or 45 μs time-out occurs, program execution will be transferred to the timer interrupt service routine (tw_timer_int). At the beginning of the routine, the timer routine checks the source of the interrupt. If it is due to the 45 μs time-out, the program reloads the 21 ms count value into the timer registers and calls the TFE interrupt. The TFE interrupt will return to the timer routine after the response has been started. If the interrupt is due to the 21 ms time-out, the program increments all real time clock counters by one unless the counter has already reached "FF". It is necessary to keep these counters from overflowing because the foreground program has no way to distinguish counter overflow. In order to keep the execution time of the interrupt service routine as short as possible, the timer routine does not perform any other checking to these counters. However, the routine still has to check pending host accesses and call dca_fast_birq if needed. The foreground program (tw_session) is responsible for checking these counters and invoking real time events at the right moment.

The Command Stubs

The twinax part of the MPA-II program emulates the IBM's 5251 model 11 display terminal. The following discussion will be based on the commands for 5251 model 11. The command set of 5251 model 11 is shown in Table 4-2, 5250 Command Set, located in Chapter 4. The commands are divided into two main groups: the queueable commands and non-queueable commands. The three non-queueable commands POLL, ACTIVATE READ, and ACTIVATE WRITE are not handled by the foreground programs as they are not queueable. Instead they are handled in real time by the background interrupt service routines as discussed above.

All other commands are queueable, namely, they are pushed into the command queue when received by the receiver interrupt routine. They are processed by the foreground task, tw_task, when it is invoked by the Kernel. In order to divide the program into properly grouped modules and make documentation easier, the queueable commands are further divided into four groups according to their functions: Reads, Writes, Control and Operators. This grouping is not a definition by IBM's PAI document. The commands shall be discussed according to this grouping.

One may observe that in addition to the 5251 model 11 command set documented in the IBM's PAI, there is an extra command in Table 4-2 of Chapter 4. The READ LINE command is an undocumented read command that is recognizable by the IBM 5251 emulation card. In addition, the READ DATA command has some undocumented variations. To allow the MPA-II board to work with IBM's System Units properly, the BCP program must be able to handle these commands. Responses to these commands will be discussed under the READS section.

Commands to the display terminal can be addressed to different logical devices and feature devices. This is specified in the modifier/device address field of the command. The device address or feature address should not be confused with the station address. Station address appears in another

field and is handled by the receiver and transmitter interrupt routines. In the MPA-II twinax emulation program, Base and regeneration buffer, Keyboard, Indicators and Model ID are implemented. The Magnetic Stripe Reader feature is not implemented and commands to this feature will return a "not installed" response.

As described earlier, tw_session is responsible for decoding the commands and directing the execution of the program to the proper command processing routines. There are some common practices or "rules" in coding command processing routines so that they can interface with the session task properly. On entering a command routine, GP0 contains the command word and IZ contains the current SCP pointer, plus Main Bank A & B are selected. On leaving from a command routine, IZ and GP7 must not be trashed and register bank selection should not be changed. The common point of exit is to LJMP to tw_cmd_ret (twinax command return). For most commands, all 8 bits of the device address and command fields have been fully decoded upon entry and, therefore, require no additional decode in the command routine. However, for the RESET, READ DEVICE ID and READ DATA commands, the device/feature address field must be decoded in the command routines. This is because these three commands can be addressed to a number of device/features or can be addressed to uninstalled device/features. A number of commands are associated with one or more data frames. Therefore, the command routines must pop those frames off the command queue with LCALL(s) to tw_que_popper. The command routines should check the queue empty flag to prevent catastrophic errors when popping frames off the command queue. In normal operation, the queue will never be empty when it is popped by the command routines. Should the empty flag be true after a call to the tw_que_popper, it suggests that a programming error has been encountered. At this time a LCALL to tw_bugs is performed followed by a graceful error recovery (The tw_bugs routine is discussed in the Software Debugging Aid section). Most commands require the command routines to check for the validity of the operands which are held by the address counter, reference counter or cursor register prior to, or in the course of the operation of the command. If any invalid operand is detected, it must be reported back to the System Unit through the exception status. The command processing routines should set the exception type, LCALL to tw_post_exception and then pass control back to tw_session via tw_cmd_ret if an exception is detected. The tw_clear_exception routine should be called if a command is going to clear exception status. In addition, command routines should never flush the command queue directly.

The 5250-11 regeneration buffer size is 2000 bytes. The valid values of the address counter, reference counter and cursor register ranges from 0 to 1999. However, within the BCP twinax emulation program, these counters contain an offset which corresponds to their starting address within the BCP's data memory. For example, if the address counter sent by the System Unit is 20h and the regen buffer of that session starts at the BCP's data memory address of 2048h, then the address counter value stored in the SCP is 2068h. We refer to the original values of the counters as relative addresses and the stored values as absolute addresses. The reason for storing these counters in absolute address form is that the command processing routines can use them directly as data pointers without adding an offset value. This can speed up the time-critical interrupt service routines. However, whenever these counter values are passed to or from the System Unit via the Smart Alec interface, a conversion procedure is needed. Furthermore, as these values no longer start from zero, one has to check whether they are less than the lower boundry of the regen buffer address when performing the validity check. Another point is that for some commands, the final values of the counters may be rolled to 2000 if the last affected location is 1999 (in forward operation) or 65535 if the last affected location is 0 (in backward operation). Exception status should not be reported in these cases.

As mentioned in Chapter 4, Smart Alec utilizes a 31 entry FIFO queue that contains screen modification information. The FIFO queue contains starting and ending addresses of the screen area that has been modified. In the Smart Alec documentation this queue is referred to as the action stack. In order to emulate the Smart Alec interface, an action stack was implemented on the MPA-II. Every command processing routine that will modify the screen is therefore responsible for loading the action stack with the proper address values. In the tw_util module, there is an action stack loader, tw_act_ldr, and an action stack popper, tw_act_popper, dedicated to maintaining the action stack. The action stack is actually a circular FIFO queue with a length of 124 bytes located in the SCP of every session. It can hold up to 31 entries as defined by the Smart Alec document. To load the action stack, the command processing routines must first load the appropriate memory locations and registers with the starting and ending address of the modified buffer area. Second, they must determine the type of modification as defined by the Smart Alec interface. Finally, the routines should call the action stack loader.

READ C

All read type commands are grouped in the TW_READ.BCP module. The entry names of the command routines are shown in Table 6-5. The read command routines are in general, quite straightforward. This is because the actual response of all read commands is controlled by the transmitter interrupt routine. The foreground read command routines are only responsible for setting up the proper response routine addresses for the transmitter interrupt and for performing some regen buffer address checking, if needed.

TABLE 6-5. Entry Names of Module tw_read

Command Name	Command Routine Entry Name
READ REGISTER	tw_read_regs_cmd
READ LINE	tw_read_line_cmd
READ DEVICE ID	tw_read_dev_id_cmd
READ DATA	tw_read_data_cmd
READ LIMITS	te_read_limits_cmd
READ IMMEDIATE DATA	tw_read_imm_cmd

The tw_read_regs_cmd command routine sets up the READ REGISTERS routine tx_read_registers for the transmitter and then jumps back to tw_cmd_ret. The transmitter will in turn respond to the System Unit with six bytes containing the values of the address counter, cursor register, and reference counter.

The READ LINE command is an undocumented command the IBM 5250 terminal emulation card responds to. The READ LINE command reads the screen buffer starting at

the address counter until it comes to the end of the current screen line. The tw_read_line_cmd routine first checks whether the address counter value lies within the visual screen buffer range. Note that this range is different from the other reads. If it does not, then an invalid register value exception is posted and the tw_read_line_cmd routine returns to tw_session. Otherwise, the starting address of the response is placed into tw_act_beginhi/lo, the address counter is modified to point to the end of the screen line, and then the tx_read_line vector is set up for the transmitter interrupt. The transmitter will in turn respond to the System Unit with the contents of the regen buffer line.

The tw_read_dev_id_cmd command routine first decodes the device/feature address by comparing the field to all defined logical devices and feature addresses. If there is a match, it will jump to the appropriate command routine to set up routines to respond with the device or feature ID. Otherwise it will jump to the tw_read_fid_not_install routine which will direct the transmitter to respond with zero data.

There are three different flavors of the READ DATA command. The READ DATA command addressed to the Magnetic Strip Reader is documented in the 5250 PAI. Since the MSR is not installed, the tw_read_data_cmd command routine sets up the tx_read_data routine address for the transmitter interrupt and them jumps back to tw_cmd_ret. The transmitter will in turn respond to the System Unit with sixteen bytes of zero data, per the 5250 PAI. The other two flavors of the READ DATA command are undocumented, but supported by the IBM 5250 terminal emulation card. The READ DATA command 08h directed to the Base device simply returns the regen buffer byte that the address counter currently points to. An invalid register exception is posted if the address counter value lies outside the regen buffer area. Then the tx_data_vector is set to the tx_rd_data_base 08 routine address for the transmitter interrupt by the tw_rd_data_base08_cmd command routine. The READ DATA command 18h is the other undocumented read command. It is very similar to the read immediate command discussed below except that the address counter points to the start of the response, the address counter is set to the last byte of the response plus one, and that if no attribute is found when the end of the regen buffer is reached, then an attribute exception is posted. The tw_rd_data_base18_cmd sets up the tx_rd_data_base18 routine address for the transmitter interrupt, as well as the starting address for the response. Note that the tw_rd_data_base18_cmd command routine actually determines the ending address and then simply passes a count to the transmitter interrupt as to how many bytes of the regen buffer to return. This keeps the transmitter interrupt very simple.

The tw_read_limits_cmd transfers a display field of data to the controller. The area of transfer is delimited by the address counter and reference counter; therefore, tw_read_limits_cmd first checks whether they lie within the regen buffer and whether the reference counter is greater than or equal to the address counter. If any one of these tests fail, the program will post an invalid register value exception and return to the session task. Otherwise, it will pass the address counter and the byte count (reference minus address) to the transmitter interrrupt through four memory storage locations: tw_act_beginlo, tw_act_beginhi, tw_act_endlo and tw_act_endhi, and then set up the READ LIMITS routine. The transmitter will then fetch the data from the regen buffer and send it to the System Unit. Before returning to session task, this command routine will update the address counter to the value of reference counter plus one so that the transmitter interrupt will not have to.

The tw_read_imm_cmd command pops out the starting address from the command queue and determines whether it is valid. If it is valid, it will be converted into an absolute address, as we discussed in the introduction, and passed it to the transmitter. The tw_read_imm command will then determine the ending point of the read and pass a count of the number of regen bytes to send to the transmitter. Finally, the tw_read_imm stub will be set up for the transmitter interrupt.

WRITE Commands

All write type commands are grouped in the TW_WRITE.BCP module. The entry names of the command routines are shown in Table 6-6. The PREACTIVATE WRITE command routines, tw_write_imm_cmd and tw_write_data_cmd, are relatively simple. They just set the beginning address of the operation to tw_act_beginhi and tw_act_beginlo. When the receiver interrupt gets an ACTIVATE WRITE command, the receiver interrupt will put the data into the regen buffer and determine the end of operation. Processing of other write commands is done completely in the foreground. We shall discuss each command in more detail.

TABLE 6-6. Entry Names of Module tw_write

Command Name	Command Routine Entry Name
WRITE CONTROL DATA	tw_write_cntl_cmd
WRITE DATA and LOAD CURSOR—base	tw_write_data_ld_cur_cmd
WRITE DATA and LOAD CURSOR—indicate	tw_write_data_lo_ind_cmd
WRITE IMMEDIATE DATA	tw_write_imm_cmd
WRITE DATA	tw_write_data_cmd

The tw_write_cntl_cmd command pops the data byte following the command from the queue and puts it into the control register location (tw_ctrl1) in the SCP. It also checks the Reset Exception Status bit (bit 12) of the data word. If the bit is set; the tw_clear_exception subroutine is called. On the 3180-2 model terminal, the command may have a second data byte. This routine checks bit 8 of the first data byte, if it is set, one more byte will be popped out and saved into tw_ctrl2 in the SCP.

The tw_write_data_ld_cur_cmd command may also have one or more data bytes associated with it. This routine checks the first data byte to determine if it is in the range of 01 to 0Eh. If the data byte is not in this range, it is the only data byte associated with the command and the routine just writes it to the location pointed to by the address counter. If the data byte is in this range, the routine will take the first byte as the byte count and will pop that number of data bytes from the queue and write them into the regen buffer. During the write operation, the address counter will be incremented and checked for overflow. Storage exception status will be posted if an overflow occurs. At the end of the operation, the program updates the cursor register to the value of the address counter and loads up the action stack by calling the tw_act_ldr routine.

The tw_write_data_to_ind_cmd command routine handles the WRITE DATA AND LOAD CURSOR command ad-

dressed to the indicators. It simply pops out the data byte following the command and saves it in the memory location tw_idctr_data in the appropriate SCP. It also notes the transition direction of certain indicators and saves this information in the memory location tw_sa_trans_ident for Smart Alec.

The tw_write_imm_cmd routine first pops the starting address from the queue, then checks to see if it is valid. If it is valid, it will be converted into absolute form and passed to the receiver interrupt. The starting address entry of the action stack is also set up. The receiver will then pick up the rest of the operation when the ACTIVATE WRITE command is received.

The tw_write_data_cmd routine checks the address counter and passes it to the receiver interrupt as the starting address of the operation. The subsequent operation is identical to the WRITE IMMEDIATE command.

Operators

The module TW_OPER.BCP contains command routines for all operator commands. Entry names of these routines are shown in Table 6-7.

The CLEAR command routine is actually a subroutine that returns to its caller. Therefore, the command routine tw_clear_cmd simply calls the actual clear routine, tw_clear_routine, and upon return from that routine, tw_clear_cmd LJMP's back to tw_session as required by all command routines. The subroutine tw_clear_routine checks the address and reference counters to see if they point at valid screen addresses and that the address counter is less than or equal to the reference counter. If any of these are false an invalid register exception is posted and no clearing takes place. Otherwise, the bytes starting with the byte pointed to by the address counter are zeroed up to and including the byte pointed to by the reference counter. Then an action stack entry is made to notify the Smart Alec interface of the screen update. The address counter and reference counter's contents are not modified.

TABLE 6-7. Entry Names of Module tw_oper

Command Name	Command Routine Entry Name
INSERT CHARACTER	tw_insert_cmd
CLEAR	tw_clear_cmd
MOVE DATA	tw_move_cmd
SEARCH NEXT ATTRIBUTE	tw_search_attr_cmd
SEARCH NEXT NULL	tw_search_null_cmd

The tw_insert_cmd command routine first examines the regen buffer location pointed to by the reference counter. If it is not a null, a Null or Attribute error exception will be posted and operation terminates. If it is a null, the program proceeds to check the address counter and reference counter to see whether they are valid. If the counter values are valid, the insert operation will be carried out. At the end of the operation, the address counter and cursor register will be updated and the action stack will be loaded by calling the tw_act_ldr routine.

Although the operation of the tw_move_cmd command is quite complex, the IBM PAI gives a fairly clear description of it. This routine checks the address counter, reference counter and cursor register to determine whether the move is forward or backward. The program then carries out the move operation as per the description of the PAI. The action stack load for the move command consists of two entries or four values. The first entry is the starting address and ending address of the destination area of the move. The second entry is the starting address of the source area and the direction of operation. Details of these entries can be found in the Smart Alec user manual.

The tw_search_attr_cmd command routine first checks the address counter to make sure it is within the valid range. Next, starting from the current address counter value, the routine searches the regen buffer to find an attribute. If an attribute is located, the reference counter will be set to the address of the attribute minus one. The routine will post a null or attribute error exception if no atribute is found when the end of buffer is reached.

At the beginning of the tw_search_null_cmd routine, it checks both the address counter and reference counter to make sure they are within valid range and that the reference counter is equal to or greater than the address counter. If the checks are successful, the program proceeds to search for a null character starting from the current value of the address counter. If a null is found, the reference counter will be set to the address of the null minus one. Otherwise the operation will terminate when the reference counter is reached and a null or attribute error exception will be posted.

Control

The module TW_CNTL.BCP contains all the routines that handle the control commands. The entry names of all routines are shown in Table 6-8.

TABLE 6-8. Entry Names of Module tw_cntl

Command Name	Command Routine Entry Name
LOAD ADDRESS COUNTER	tw_load_addr_cmd
LOAD CURSOR REGISTER	tw_load_cursor_cmd
LOAD REFERENCE COUNTER	tw_load_ref_cmd
SET MODE	tw_set_mode_cmd
RESET	tw_reset_cmd
EOQ	—

The tw_load_addr_cmd command routine pops the address counter value from the command queue and saves it on the SCP after changing it to absolute form. However, as per IBM's PAI, there is no need to check the validity of the value before loading. As a remark to clarify the ambiguity of the PAI, the address counter value consists of two bytes, the upper byte is the first data byte following the command while the lower byte is in the second byte.

The tw_load_cursor_cmd command routine loads the cursor register in the SCP with a new value. The operation is similar to tw_load_addr_cmd routine.

The tw_load_ref_cmd command routine loads the reference counter in the SCP with a new value. The operation is similar to tw_load_addr_cmd routine.

The tw_set_mode_cmd routine pops the fill bit count from the command queue, converts it to the BCP's Fill Bit Register format, and saves it on the SCP. Next, the set mode received bit is set in the SCP. This signals the background receiver interrupt that it may start responding to polls using the two byte response format, (after a PACK is received). Finally, if the current exception state indicates POR then the exception state is cleared.

Like the tw_clear_cmd routine, tw_reset_cmd actually calls the subroutine tw_por which performs a POR on the current session. The routine tw_por first places the current session OFFLINE by signaling to the background receiver interrupt (via the RX_RESET bit) that it is not to respond to the host until further notice for this station address. Once this is done, the tw_por routine can begin changing memory locations normally updated by the background receiver interrupt without disabling interrupts because the RX_RESET bit effectively disables the receiver interrupt when working with this physical session. Next the exception status is changed, notifying other tasks that this session is in POR. The time count for this session is cleared and a bit is set (in the tw_por_waited_session byte on the DCP) informing the other tasks that the 5 second POR timeout has commenced. The tw_task routine will use this time count and this session's POR wait bit to determine when to bring the session back on line. Other tasks use the POR wait bit when interpreting the meaning of the time count for the current session. The action stack is cleared next, along with the smart alec task handshake bits. Then, the screen buffer for this session is cleared via a call to tw_clear_routine, which issues an action stack entry reflecting the cleared screen. (This allows the PC to accurately reflect the POR state.) Finally, the remaining SCP variables are set to their appropriate values, except for the variables controlled by the smart alec task, (i.e., Model ID, Keyboard ID, etc . . .), which are left unchanged.

The End Of Queue command does not actually have a command routine, for at this point in the command decoding process of the MPA-II it does not provide any additional information. As far as the command processor is concerned, the queue load complete flag, set by the background receiver interrupt, indicates the actual end of queue. So the act of popping the EOQ command off the queue completed this command's execution, no call to a command routine is required.

The Twinax Session Command Processor

The twinax session command processor, tw_session, is located in module TW_SESS.BCP. Its job is to perform all non time-critical functions related to sustaining an active twinax session. This includes processing the internal command queue, error recovery, and performing a POR. In addition, tw_session and its subordinate routines are responsible for communicating important events (like screen updates) to the emulation interface routine (i.e., the smart alec task), which operates asynchronously to twinax session activity.

The command processor, tw_session, and its subordinate routines are written with "reusable" code. That is, all the information regarding a given twinax session's state is kept in the SCP (the data memory Session Control Page) attached to that physical session. There is no dependency between tw_session and an active session's state from one call to the next. At any time, any SCP may be passed to tw_session. In other words, the current state of a given physical twinax session exists only in its SCP, not in the command processor. This gives one set of routines (tw_session and its subordinates) the ability to process all the active twinax sessions concurrently. The twinax task tw_task simply passes the pointer of the scheduled session's SCP (via the IZ register) to tw_session and tw_session then determines the current state of that session and what action(s) need to be performed.

The program flow of tw_session proceeds as follows. First, tw_session checks for the ACTIVATE WRITE command for the current session completed in the background. If one has occurred, tw_session performs an action stack push, which notifies the Smart Alec interface of the screen update. Next, the command processor checks for actions requested by other tasks. Currently, two actions are defined: the "forced" POR and the "requested" POR. The "forced" POR is usually issued by the smart alec interface task and it forces a POR regardless of the current session status. After the POR is initiated control returns to the calling routine (tw_task). The "requested" POR is usually issued by tw_task when an Auto-POR is desired. A POR is only performed if the current session is not already in the POR exception state or if an error condition does not exist. Otherwise, this request is ignored. In this way, the twinax session will not unnecessarily POR. Again, after a POR is initiated control returns to the calling routine.

Once all the requested actions from other tasks have been handled, the command processor attempts to process the internal command queue of the current session. Rather then holding off the command processor from processing commands on the queue until a queue load is complete, we opted to exploit the power of the BCP by using a parallel processing approach where both the background receiver interrupt and the foreground command processor have access to the command queue simultaneously. This enables the command processor to execute commands even while the queue is still being loaded by the host. To avoid conflicts, the command processor tw_session takes a "snap shot" of the current internal command queue and current exception status (in the poll response byte). The command processor then works from the "snap shot" while the background receiver task updates in real time.

The "snap shot" involves the following steps. Interrupts are disabled to prevent background tasks from updating the command queue. The command queue is then checked to see if another task has marked it as "corrupt". When a background task determines that the command queue may contain invalid data (for example, due to a line parity error or the detection of an exception) it marks the queue as corrupt and schedules that session. The tw_session routine then flushes the queue when it gets control. Flushing the command queue resets all the queue pointers and flags. This marks the command queue as empty. It also signals the background tasks that tw_ session has acknowledged the error and cleaned up the command queue. This handshake is required since background tasks are only allowed to push onto the internal command queue, never flush it. (At the next poll to this session, the background receiver interrupt will indicate "not busy" to signal the host that this device has completed error recovery.) After the command queue is flushed, tw_session will deschedule this twinax session and return to the calling routine (tw_task). If the internal command queue is not corrupt, tw_session checks to see if it is "ready" for processing. The command queue is marked as "not ready" while the background receiver interrupt is in the middle of pushing a multi-byte command (for example the LOAD ADDRESS COUNTER command) onto the

queue. While the queue is marked as "not ready", tw_session will not attempt to process any commands on the queue. Instead, tw_session leaves this session scheduled and returns to tw_task. This keeps the command processor and its subordinate routines from attempting to pop incomplete commands off the internal command queue. On the next Kernel cycle, tw_ session will once again be called upon (by tw_task) to process this session's command queue. If the internal command queue is marked "ready" for processing then tw_session copies the current queue pointer, the current exception status (located in the poll response byte), and then deschedules this session. This completes the "snap shot". Interrupts are enabled so that other tasks may continue to update the command queue.

Now that the "snap shot" of the command queue has been taken, tw_session can begin popping commands off the queue and decoding them. The command queue is processed based on tw_sessions' current verion of the exception status, initially recorded during the "snap shot". This exception status is checked before the decode of each command to determine the current exception state of this session, since command decode depends on this state and previous command execution may change the state. (Note that this copy of the poll response's exception status may not match the actual exception status after the "snap shot" has been taken. This is simply a consequence of background/foreground parallel processing and is not a problem. The next time a queue "snap shot" is taken the tasks are brought back into sync.) While in POR exception state, only the SET MODE and RESET commands are considered valid. While in any other exception state, only the SET MODE, RESET, and WRITE CONTROL DATA commands are considered valid. In normal mode (no exception state,) all commands are considered valid. If an invalid command for the current exception state is decoded, the command queue is flushed and tw_session will attempt to post an exception. A valid command decode causes tw_session to pass control to that command's routine (called a command routine) for processing. Most of the commands have been fully decoded by tw_session before their command routine is executed, but a few commands require the command routines to further decode the feature address field. Each command routine is responsible for popping its associated data off the command queue. Each command stub is responsible for carrying out complete command execution, including posting exceptions, making action stack entries, etc ... (Many of these tasks are actually carried out by calls to support subroutines.) All command routines return to the same entry point in tw_session. (See the comments in tw_session, at the command decode section, for a complete set of rules regarding command stub coding.)

When all the commands have been popped off the current command queue snap shot, the queue load complete flag (TW_QUE_COMPLETE) is checked. This flag is set by the background receiver interrupt when an EOQ designator has been received. (An EOQ designator can be an EOQ command, a PREACTIVATE command, or a full command queue.) If the queue load complete flag is set then tw_session flushes the command queue, clearing this flag and resetting the command queue pointer. The clearing of the queue load complete flag by tw_session signals the receiver task that it may clear the poll response busy status flag at its discretion. This in turn signals the host that the queue load has been completely processed and a new queue load may be initiated.

Finally, tw_session returns control to the calling routine, tw_ task, not to be called again for the current session until another task schedules this session to perform additional work.

Handling Exceptions

Exceptions are posted by the subroutine tw_post_exception (located in module TW_UTIL.BCP). This is the only reliable way for foreground tasks to post exceptions since both foreground and background tasks must be made aware of the exception. The tw_post_exception routine first disables interrupts to hold off background processing. It then updates tw_session's exception status. Next, it updates the poll response exception status, but only when no exception is currently pending. The tw_post_exception routine then places the background receiver interrupt into its busy wait state. This prepares the receiver interrupt to respond "not busy" on subsequent polls from the host. Following that, tw_post_exception flushes the command queue per the PAI. Finally, after a quick check of BIRQ, interrupts are enabled and tw_post_ exception returns to the calling command stub.

Exception status is cleared by tw_clear_exception, located in module TW_UTIL.BCP, for the same reason as stated above. This routine sets both tw_session's exception status and the poll response exception status to zero while interrupts are disabled. Again, BIRQ is checked before interrupts are enabled and then control returns to the calling command routine.

Twinax Software Debugging Aids

The subroutine tw_bugs, located in the module TW_TASK.BCP, is used for a debugging aid. Routines call tw_bugs when they detect invalid states; for example, the Smart Alec read command addressed to physical session 7 (the seven physical sessions are numbered 0–6). During initial debug, the SCPs and DCP are usually relocated into dual port memory by trading them with screen buffer 3 (sbp 3). The tw_bugs routine is then set to disable interrupts, unlock the PC, and jump to itself so that when called, the current state of the MPA-II is frozen and can then be viewed using the Capstone Technology debugger. After initial debug is complete, tw_bugs is set to simply log the occurence of a bug by incrementing a counter in the DCP and return to the caller. The caller should then attempt a graceful recovery. A check of the tw_bugs counter will reveal if routines are detecting unexpected conditions when in the field.

Smart Alec Interface Overview

Smart Alec is a micro-to-System 3x or AS/400 link produced by Digital Communications Associates. It provides the IBM PC, PC XT, or PC AT with a direct link to IBM System 34, System 36, System 38, or AS/400 midrange computers. The Smart Alec product includes a printed circuit board that installs in any full length slot in the PC, and a software package that consists of a 5250 terminal emulation program, called EMU, and a bi-directional file transfer utility. A splice box to facilitate connection to the twinaxial cable is also included.

The terminal emulation program provides the user with all the features of 5251 model 2, 5291, or 5292 model 1 termi-

nal. It also allows a PC printer to emulate the IBM 5256, 5219, 5224, 5225, and 4214 system printers. The file transfer utility provides bi-directional data transfer between the PC and the System 3x. Additional features include the ability to support up to seven host sessions, the capability to bid for unused addresses, compatibility with software written to comply with the IBM Application Program Interface, "hot key" access, and 3270 pass through support.

As mentioned earlier, IBM was the first to enter the marketplace with a 5250 terminal emulator. This was soon followed by the release of similar products including DCA's Smart Alec. Smart Alec was however, the first product to offer seven session support, address bidding, and a documented architecture for third party interfacing. As with IRMA, Smart Alec and its associated interface gained acceptance in its respective market place. As a result of this the Smart Alec interface was chosen for the Multi-Protocol Adapter-II to further show the power and versatility of the DP8344A Biphase Communications Processor. The MPA-II hardware with the MPA-II soft-loadable microcode is equivalent in function to the DCA Smart Alec board and its associated microcode with respect to terminal emulation and file transfer capabilities (the printer emulation and non-vol RAM configuration storage were not implemented on this version of the MPA-II). Both directly interface with the Smart Alec terminal emulation software that runs on the PC (EMU, file transfer utilities, etc . . .) providing the same terminal emulation functions and features of the Smart Alec product. The following sections describe the hardware interface and the BCP software in the Multi-Protocol Adapter-II Design and Evaluation kit that is used to implement the Smart Alec interface. All of the following information corresponds to Rev 1.51 of the Smart Alec product.

Hardware Considerations

The Smart Alec printed circuit board plugs into any full size expansion slot in the IBM PC System Unit. It provides a cable and splice box that allows the bulky twinaxial cable from the System 3x or AS/400 to be connected to the back panel of the Smart Alec board. The splice box also contains termination resistors that can be switched in to terminate the line if it is the last device. Smart Alec operates in a stand-alone mode, using an on-board microprocessor (the Signetics 8X305) to handle the 5250 protocol and multiple session screen buffers. Because of the timing requirements of the 5250 protocol, the on-board 8X305 operates independently of the 8088 or the System Unit. The 8X305 provides the intelligence required for decoding the 5250 protocol, maintaining the multiple screen buffers, and handling the data transfer and handshaking to the System Unit.

The Smart Alec card uses a custom integrated circuit to interface the 8X305 to the twinaxial cable. This custom device is essentially a transmitter and receiver built for the 5250 environment. It can take parallel data from the 8X305 and convert it to a serial format while adding the necessary 5250 protocol information and transmit this to the twinaxial cable through additional interface circuitry. It also accepts a serial TTL level signal in the 5250 word format and extracts the 5250 protocol specific information and converts it to a parallel format for the 8X305 to read.

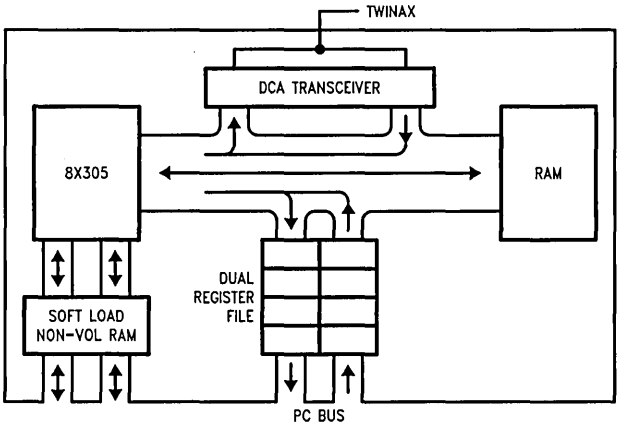
The card contains 16K of data memory for the screen buffers and temporary storage. Each session can require up to 2K of data memory for its associated screen buffer, accounting for a total of 14K. The remaining memory space is used by the 8X305 for local storage.

The hardware used in enabling the 8X305 to communicate with the PC's 8088 processor is a dual four byte register array. The 8X305 writes into one side of the four byte dual register array which is read by the 8088. The 8088 writes into the other side of the dual array which is in turn read by the 8X305. The dual register array is mapped into the PC's I/O space at locations (addresses) 228h–22Bh. This interface is identical to that found on the IRMA board except for the I/O addresses.

A handshaking process is used between the two processors when transferring data. After the 8088 writes data into the array for the 8X305, it sets the "Command" flag by toggling bit 0 (writing a "1" then writing a "0") in I/O location 22Eh. This is decoded in hardware and sets a flip-flop whose output is read as bit 7 (the msb) at location 22Eh. When the 8X305 has read the registers and responded with appropriate data for the 8088, it clears this flag by resetting the flip-flop. A similar function is provided in like manner for transfers initiated by the 8X305. Here the flag is called the "Attention" flag and can be read as bit 6 at location 22Eh. This flag is cleared when the 8088 toggles an active low bit in bit position 0 at location 22Dh. Even though the attention flag function is documented, it is not used on this revision of Smart Alec.

Two additional features not found on rev. 1.42 of the IRMA card were implemented on the Smart Alec board. These are the ability to softload the 8X305's instruction memory and the ability to save configuration information in a non-volatile RAM on the board. The control signals needed for these tasks are transferred to the Smart Alec Board from the 8088 in bits 1–5 at location 22Dh and 22Eh, and in bits 6 and 7 at I/O location 22Fh. When the terminal emulation program, EMU, is invoked for the first time after each power up the 8X305 microcode is downloaded into RAM on the Smart Alec board. Information generated through the configuration program EMUCON is loaded into a 9306 serial non-vol RAM on the Smart Alec board. This is accessed at power up thus eliminating the need for the user to configure the board every time the PC is turned on. A block diagram of the Smart Alec hardware is shown in *Figure 6-18*.

The Multi Protocol Adapter-II printed circuit board also plugs into any expansion slot in the IBM PC System Unit. Like Smart Alec, it provides an adapter to allow the bulky twinaxial cable from the System 3x or AS/400 to be connected to the back panel of the card. The MPA-II board contains the termination resistors on the PC card and not in a splice box. These resistors can be "switched in" via two jumpers. The MPA-II operates in a stand-alone mode, using the DP8344A Biphase Communications Processor to handle the 5250 protocol and multiple screen buffers. Again. because of the timing requirements of the 5250 protocol, the BCP operates independently of the 8088 microprocessor of the System Unit. As with the 8X305, the BCP provides the intelligence required for decoding the 5250 protocol, maintaining the



TL/F/10488–38

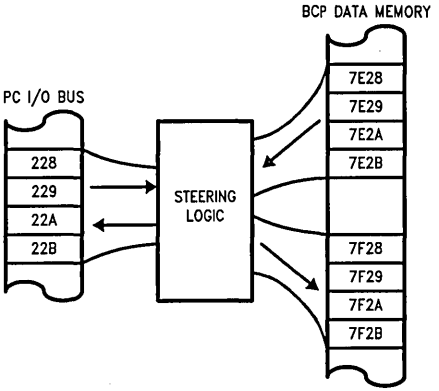
FIGURE 6-18. Smart Alec Hardware Block Diagram

multiple screen buffers, and handling the data transfer and handshaking to the System Unit. However, with the BCP's higher level of integration, it also interfaces with the twinaxial cable. The BCP has an internal biphase transmitter and receiver that provides functions similar to the custom transceiver on the Smart Alec board. As is the case in 3270, the BCP's CPU can handle the 5250 communications interface very efficiently. It also has the extra bandwidth to allow the MPA-II to easily handle the multiple sessions.

The MPA-II card contains a single 32K x 8 RAM memory device for the screen buffers and temporary storage. This memory size was chosen to handle all seven twinax sessions in a single RAM.

The hardware used to enable the MPA-II's BCP to communicate with the PC's 8088 processor is steering logic (contained in PALs) and the data RAM. In a typical application, the BCP communicates with a remote processor by sharing its data memory. This is true with the MPA-II, but because the MPA-II must run with the Smart Alec software, steering logic has been used to direct the 8088's I/O reads and writes done by the Smart Alec software into data memory locations on the MPA-II card. The I/O accesses performed by the Smart Alec software can be fit into three groups; accesses to the dual register array, accesses to the handshaking flags, and accesses to configure the card. All of these are directed into the BCP's data memory, however each are handled differently by the MPA-II. By using data memory and the extra processing power of the BCP's CPU instead of discrete components the number of integrated circuits on the board was reduced.

The Smart Alec dual register array is implemented on the MPA-II card in the same fashion as the IRMA dual register array. The I/O accesses from the System Unit are "steered" to two different BCP data memory locations depending on if they are reads or writes. The writes from the 8088 are directed to memory locations 7F28h–7F2Bh, and the reads from the 8088 are sourced from memory locations 7E28h–7E2Bh. The MPA-II Register Array Implementation is shown in *Figure 6-19*.



TL/F/10488–39

FIGURE 6-19. MPA-II Register Array Implementation for Smart Alec

The handshaking process on the Smart Alec card differs from the IRMA implementation. To set the command flag, bit 0 in the register at I/O location 22Eh must be toggled (a write of a "1", followed by a write of a "0"). In the IRMA interface, just writing to an I/O location would set the command flag. This is not the case with Smart Alec because the additional softload and configuration capabilities of the Smart Alec card required that each of the bits in these registers have different functions. The MPA-II hardware used to emulate the handshaking function for Smart Alec is similar to its IRMA implementation. When the 8088 goes to set the command flag by toggling bit 0 at I/O location 22Eh, it actually does a write to 7F2Eh in the MPA-II's data memory via the steering logic. The steering logic also interrupts the BCP telling it an access has been made to the Smart Alec I/O space. The BCP then determines if it was a write to the PC I/O location 22Eh by reading the access register from the steering logic. If a write occurs to I/O location 22Eh, the BCP reads the memory location 7E2Eh and determines if

the "set command flag" bit has been toggled. It does this by checking to see if bit 0 and bit 4 (the non-vol RAM enable bits) are low. If this is the case, it then knows the Smart Alec software intended to set the command flag. The attention flag is not implemented on this version of Smart Alec and is therefore not implemented on the MPA-II. However, if one chooses to do so it can easily be done in the same manner.

The System Unit accesses used to configure the Smart Alec Board consist of a method to softload the 8X305 and to read and write set-up information into a non-vol RAM. Because the MPA-II uses the DP8344B, there is no need to emulate the 8X305 softload function. The DP8344B is itself softloaded using the MPA-II Loader before the Smart Alec software is invoked. The reading and writing of the non-vol RAM is done using additional bits in the control and strobe registers at I/O locations 22Dh, 22Eh and 22Fh. In the Smart Alec implementation the System Unit must provide all the control, data and clock signals to the non-vol RAM via the above mentioned I/O locations. The non-vol RAM is not implemented on the MPA-II card but because the Smart Alec emulator, EMU, reads this information on power-up the MPA-II does emulate the non-vol RAM when it is being read.

This is done in the same manner as the handshaking flags and further illustrates the flexibility a designer is given with the additional bandwidth of the BCP's CPU.

Smart Alec Microcode

The Smart Alec application software written for the personal computer (EMU, file transfers, etc . . .) is architected around a defined interface between Smart Alec and the System Unit (the 8088 and its peripheral devices). The hardware portion of this interface was discussed in a previous section. The software portion of this interface is the microcode written for the 8X305 processor. For the following discussion, the software and hardware are viewed as a single interface function. All of the Smart Alec application software has been written around this interface. When configured in the Smart Alec mode the MPA-II becomes this interface. The method of communication between Smart Alec and the System Unit will be discussed briefly in the next section. A more exhaustive discussion on this interface is given in the Smart Alec manual.

Smart Alec and the System Unit communicate through the dual four byte register array. The System Unit issues commands to Smart Alec by writing to this array. This register array is viewed by the System Unit as four I/O locations (228h–22Bh). Each I/O location corresponds to one eight bit word. When the System Unit issues a command, the first byte, word 0, is defined as the command number and logical device. The next three bytes, word 1 through word 3, are defined as arguments for the command. The number of arguments associated with an individual command varies from zero to three. Twenty-three commands are used in the communication between the System unit and Smart Alec. The upper three bits of each command specify the logical device to be referenced by the command. To begin a command the System Unit program sets word 0 equal to the logical device and the command number. It also provides any necessary arguments in word 1 through word 3, and sets the command flag. The command flag is continually being polled by the 8X305 processor when it is not busy managing the higher priority 5250 communications interface. When it detects the setting of this flag by the System Unit, it will read the data from the register array and execute the command. Once the command has been executed, the 8X305 will place the resulting data into the other side of the register array and clear the command flag. The System Unit program has been continually polling this flag and, after seeing it cleared, reads the result from the register array. The command flag can only be set by the System Unit. This is done by toggling bit 0 at I/O location 22Eh. The command flag can only be cleared by the Smart Alec's 8X305.

The Smart Alec board was designed at DCA after the IRMA product. It is obvious from the additional commands that steps were taken to improve the performance of the interface with the System Unit. An action stack was generated to hold address pairs that denoted where the screen buffer had been modified and with what type of modification. Also read multiple commands were added to speed up data transfer through the interface. While this did improve the performance of the interface it still contains the inherent limitations of not dual porting data memory.

MPA-II Implementation

The smart alec interface on the MPA-II board operates essentially in the same manner as described above. The System Unit I/O accesses to the Smart Alec register array space are transferred to two locations in the BCP's data memory. One location is for System Unit reads of the array (7E28h-7E2Bh), the other is for System Unit writes to the array (7F28h-7F2Bh). Different BCP memory locations were used because the register array on the Smart Alec card actually contains eight byte wide registers (four for System Unit and four for System Unit writes) in hardware.

The command flag is implemented using a 74LS74 on the Smart Alec board, hence the setting and clearing by toggling a bit in the control register at 22Eh (this clocks the flip-flop). This function has been implemented on the MPA-II using an external PAL and the bi-directional interrupt pin, BIRQ. Also, the MPA-II takes advantage of the fact that the Smart Alec software accesses the I/O locations in exactly the same fashion for each command. This is done because the Smart Alec emulation program, EMU, was written in the C programming language. It accesses the Smart Alec I/O registers by calling an assembly language subroutine to perform the command/data and handshaking flag communications. This assembly routine writes to the I/O locations 228h through 22Bh, toggles the command flag, and then reads the state of the command flag (bit 7 at location 22Eh) until it returns low. If there is a write to the Smart Alec I/O space 228h-22Fh, then a PAL issues an interrupt to the BCP via the BIRQ input. The BCP then reads other outputs of that PAL to determine to which of those I/O locations the write occurred. If it is to 228h-22Bh then the MPA-II will assert the bit which tells the System Unit that the command flag is set. The MPA-II then waits for a System Unit write to I/O location 22Eh with the set command flag bit (bit 0 at 22Eh) low. The MPA-II then sets an internal command flag. It is this internal command flag that tells the MPA-II's smart alec task routine that an actual command has been issued by the System Unit.

The commands from the System Unit are executed in the smart alec task routine. This routine is a foreground scheduled task in the MPA-II Kernel. The smart alec task routine first checks to see if the non-vol RAM is being read. If so it

supplies the necessary data in bit position 6 at I/O location 22Fh. If the non-vol RAM is not being read, the smart alec task routine then determines if a command is present. If so the command is decoded and executed by the appropriate command routine. In most cases, the appropriate physical device will have to be determined in order to access the correct session control page, or SCP, and the correct screen buffer for each active session. The SCP contains status and control information for each of the seven possible sessions. During the command execution the required status is calculated by calling the status update subroutine. The command's result and the calculated status are then placed in the appropriate memory locations (7E28h-7E2Bh). After this is complete, the task clears the command flags and returns program control to the Kernel.

There are three separate code modules used to allow the MPA-II to emulate the Smart Alec Interface.

1) power-up initialization routine
2) BIRQ interrupt routine
3) smart alec task routine

These three routines will be discussed in the following section. For clarity, the term "smart alec" is capitalized when referring to DCA products and lower case when referring to the MPA-II software that has been written to emulate the interface. *Figure 6-20* gives a graphical representation of where these routines fit into the software architecture of the MPA-II.

MPA-II Smart Alec Initialization Routine

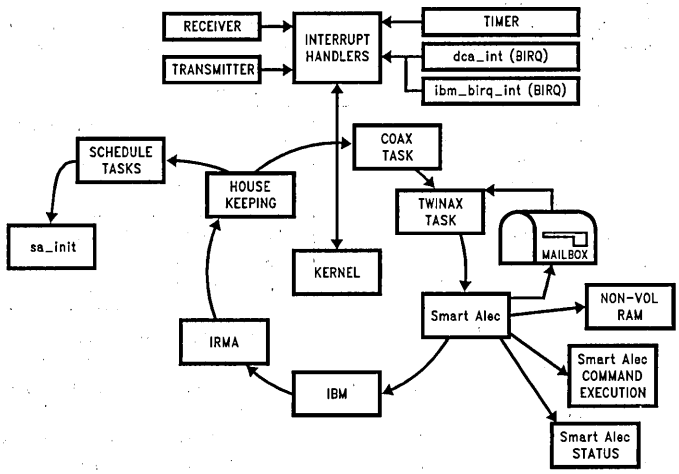
The smart alec power up initialization routine is called by the housekeeping task if it detects that the smart alec bit has just been set in the MPA-II configuration register. The smart alec initialization routine is titled sa_init in the MPA-II source code. This routine initializes the memory locations and BCP internal registers that are used by the smart alec emulation code. It also unmasks the BIRQ interrupt and schedules the smart_alec_task in the MPA-II Kernel. The memory locations that are initialized in this routine are the blocks of memory that correspond to the contents of the emulated non-vol RAM, the memory locations used to emulate the dual register array and the flag registers, the locations on the seven session control pages that EMU controls, and the device control page memory locations that control the logical to physical mapping for the multiple sessions.

The sa_init routine also initializes some internal BCP registers. It does this because other routines, such as the dca BIRQ interrupt routine, must access certain stored values very quickly to keep their execution time quick. The final function of the sa_init routine is to schedule the sa_task routine. This is done by loading the task number into the accumulator and calling the schedule_task routine. After this, program control is returned to the Kernel.

MPA-II DCA Interrupt Routine

The second code module required to emulate the Smart Alec Interface is the dca BIRQ interrupt routine. The MPA-II board uses the extra CPU bandwidth of the BCP to reduce the discrete components needed to provide the command and flag function. It does this by letting the CPU decode part of the System Unit I/O access address and by letting it provide the set function of this flag. The BCP code necessary for this is the BIRQ interrupt routine whose source module is DCA_INT.BCP. The BIRQ interrupt is generated when the System Unit writes to any I/O locations from 220h to 22Fh. It would have been more expedient in this case to only have interrupts generated on writes to I/O location 22Eh. However, the MPA-II hardware also supports the IBM and IRMA emulation programs. The MPA-II implementation for the IBM interface requires interrupts to be generated from more System Unit I/O access locations, so to reduce external hardware, interrupts are generated for a sixteen byte I/O block. This flexibility of hardware design further illustrates the usefulness of the extra CPU bandwidth of the DP8344B.

When the BCP detects the BIRQ interrupt, it transfers program control to the dca_int routine. The function of this routine is to set the command flag or provide the appropriate serial non-vol RAM data. There is a section of code in the dca_int routine that does the same function as that described above, but is called from the other routines and not by the external BIRQ interrupt. To increase performance, the interrupt routines check the BIRQ flag in the CCRregister before they return. If the flag is set, it calls the dca_fast_birq section of the dca_int routine. Here the same operations as described earlier are performed except for the saving and restoring of the environment. The



TL/F/10488-40

FIGURE 6-20. MPA-II Software Block Diagram in Smart Alec Mode

dca_fast_birq routine does not have to provide this function because the other routines do so. This decreases the number of instructions executed and therefore improves the overall performance.

MPA-II Smart Alec Task Routine

The majority of the Smart Alec emulation takes place in the smart alec task routine. This routine is run in the foreground as a scheduled task. Therefore the decision to execute this routine is dependent only on the MPA-II's task scheduler and is not impacted by the System Unit. In reality, the task is run many times between System Unit accesses because the code execution speed of the BCP is much greater than that of the 8088. The smart alec task routine, appropriately labeled in the source code as "sa_task", contains four major sections. These sections are the non-vol RAM routine, the command execution routines, the physical session determination routine, and the status update routine.

When the smart alec task is called, it first checks to see if EMU has tried to read the non-vol RAM. If so, it determines how many times it was read (the non-vol RAM is read serially) so it can adjust the serial bit stream it is providing to EMU. If no accesses have been made to the non-vol RAM, the smart alec task checks to see if a command is present.

If there is no command present (the internal command flag is not set), the task returns to the Kernel. If a command is present, the lower five bits of the command word is decoded to determine which of the twenty three commands has been issued by the System Unit. Program control is then transferred to the specific routine that executes the command. In most cases, the first thing done in the specific command routine is to determine which session the command was issued to. This is done by decoding the top three bits in the command word to determine what logical session the command was issued for. After that, the corresponding physical session is determined and pointers are set up in internal registers to point to the appropriate session control page and screen buffer. Both of these functions are performed in the tw_sa_spset subroutine. Using this information the command is executed and the required status is calculated. The status is calculated in the tw_sa_all_ status routine if full status is required. If only common status is required, the tw_sa_common_status routine is called instead. After this, the resulting data is placed in the section of memory that is accessed by the System Unit when it reads the I/O locations 228h-22Bh. The smart alec task then clears both the internal and Smart Alec command flags and returns program control to the Kernel.

MPA-II Command Set

New to the MPA-II is the support of an MPA-II command set. The primary purpose of this command set is to allow any part of the MPA-II'S data memory to be accessed by the PC without having to stop the BCP or depend on the current interface mode running, (i.e., IRMA, IBM, ALEC). As almost always happens, the usefulness of this interface caused the MPA-II command set to expand. Another benefit of the MPA-II command set is that it demonstrates a better way to communicate with the BCP than that of the IRMA, IBM or ALEC interfaces. By taking advantage of the fact that the BCP directly supports dual port memory, one bit semaphores can be used to handshake with the PC and, therefore, no BIRQ interrupt routine nor lock out of the PC is required.

The MPA-II commands are listed in Table 6-9. The routine housekeep in KERNEL.BCP is responsible for the execution of these commands. The commands allow the PC to read and write any part of the BCP's data memory (including non-dual port memory), determine what version of MPA-II code is actually executing, and read or clear the receiver's error counters.

The MPA-II commands consist of a command byte written to the MPA-II configuration register (2DCh) and an optional parm written to the MPA-II parm/response register (2DBh). If the command returns a response, it is read by the PC from the MPA-II parm/response register (2DBh).

A command is identified by setting the CF_MPA_CMD bit to one. This bit is part of the command's value listed in Table 6-9. The completion of a command's execution is indicated by the restoration of the current MPA-II configuration in the MPA-II configuration register (which clears the CF_MPA_CMD bit).

Use the following steps to issue a command via the PC:

1) Write the command's parm (if any) into the parm/response register (I/O location 2DBh).
2) Write the command into the MPA-II configuration register (2DCh).
3) Read the MPA-II configuration register (2DCh) until the CF_MPA_CMD bit is cleared. This indicates completion of command execution by the MPA-II microcode. (Note, the current MPA-II configuration has been restored).

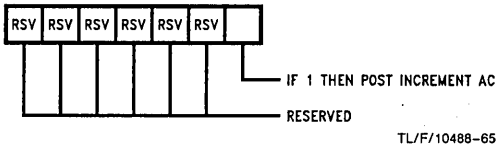
TABLE 6-9. MPA-II Command Set

Name	(Value)	Parm	Response	Comment
LACL	(10)	AC low byte	none	Load new MPA-II AC, low byte
LACH	(11)	AC high byte	none	Load new MPA-II AC, high byte
WRITE	(12)	data byte	none	WRITE "data byte" into memory at MPA-II AC's location
LCR	(13)	control byte	none	Load "control byte" into MPA-II Control Register
RACL	(18)	none	AC low byte	Read current MPA-II AC, low byte
RACH	(19)	none	AC high byte	Read current MPA-II AC, high byte
READ	(1A)	none	data byte	READ "data byte" from memory at MPA-II AC's location
REV	(1B)	none	rev byte	read REVision number of the MPA-II software
CLRE	(30)	none	none	CLear REceiver error counters
RDIS	(31)	none	error count	Read receiver's DISable error count
RLMBT	(32)	none	error count	Read receiver's Loss of Mid-Bit error count
RIES	(33)	none	error count	Read receiver's Invalid Ending Sequence error count
RPAR	(34)	none	error count	Read receiver's PARity error count
ROVF	(35)	none	error count	Read receiver's OVerFlow error count
RPRO	(36)	none	error count	Read PROtocol detected error count

4) Read the parm/response register (I/O location 2DBh) for the command's response (if any).

A PC program called MPADB.EXE has been included with the MPA-II which communicates with the MPA-II using this interface. MPADB is written in C and has some additional debugging capabilities, such as reading blocks of BCP memory using one command. After starting MPADB, type "help" at the prompt, −>, for information on the commands supported by MPADB. All the source code for MPADB is included, see MPADB.C under the directory DEBUG.

The read and write data commands use an internal MPA-II register called the MPA-II address counter, AC. This address counter works much like the Coax and Twinax address counters. The read command returns the byte pointed to by the MPA-II address counter. The write command places its data at the location pointed to by the MPA-II address counter. Whether or not the MPA-II address counter auto-increments depends on the contents of the MPA-II control register, see *Figure 6-21*. If the LSB is a one (1) then the MPA-II address counter auto-increments, otherwise it does not change.



TL/F/10488–65

FIGURE 6-21. MPA-II Control Register

The Load Address Counter High, Load Address Counter Low, Read Address Counter High, and Read Address Counter Low commands simply provide access to the 16-bit MPA-II address counter. The Load Control Register allows one to write to the 8-bit MPA-II control register.

The receiver error counter commands provide an easy, reliable way to read the MPA-II receiver error counters located in the MPA-II Device Control Page, DCP. PC software that uses these commands does not have to be updated if the receiver errors are relocated in BCP data memory because the BCP assembler will automatically update all references to those error counters when the MPA-II microcode is re-assembled.

Finally, the Revision Number command allows the PC to determine a) if the MPA-II running and b) what version of the MPA-II microcode is the MPA-II running. This MPA-II command is used by the Loader when the Loader performs an autoload (-a option). For the PC to read the revision number, the REV command must be executed three (3) times. Each returned byte's bits are defined as "xxcc dddd", where:

dddd = a revision digit coded in Binary Coded Decimal, BCD

cc = a count showing the position of the revision digit

xx = reserved

For example, if calling REV three times returned (in hex) 20, 34, and 13, then the revision number is 3.04.

Last notes, unused commands and invalid parms are ignored. In addition, commands with values less than 3F(hex) are reserved for National Semiconductor. Feel free to define commands with values greater than this if compatibility with future MPA-II releases is desired.

7.0 LOADER AND MPA-II DIAGNOSTICS

The Loader is a PC program designed to load the MPA-II with BCP microcode, start the BCP, and configure the MPA-II interface mode. A number of user selectable options are available with the Loader which provide maximum flexibility in loading, running, and configuring the MPA-II system. The Loader can also be used to run diagnostics by specifying the "selftest" option. This will test the functionality of the MPA-II hardware. The Loader syntax is:

LD [Config_options . . .] [Options . . .] <Filename>

where the following notation applies:

[ ] Items enclosed in square brackets are optional.

< > Items enclosed in triangular brackets are required.

ALL CAPS Items in all capital letters must be entered exactly as shown.

lower case Items in lower case letters indicate that desired values should be substituted.

The Loader Options that apply to the "soft-loading" of instruction memory will be discussed in the section titled "Soft-Loading Instruction Memory". The Loader Config_options will be discussed in the section titled "Configuring the MPA-II". The Loader options that apply to the selftest facility will be discussed in the section titled "MPA-II Diagnostics". Examples demonstrating the Loader options as well as the Loader defaults will also be provided in this chapter.

The Loader is primarily written in Microsoft "C"5.1. The portion of the Loader code which performs the MPA-II Diagnostics has been written using National Semiconductor's DP8344 BCP Assembler System as well as Microsoft's Macro Assembler 5.1. All of the source code required for the Loader is included on the distribution disks and is well documented. For complete details of the implementation of the Loader functions described in this section, refer to the source code.

The Loader provides two levels of help. The first level of help is provided in a brief, single screen and is accessed by typing LD with no options at the DOS system prompt. A multi-screened, comprehensive help, is accessed by specifying the -h option of the Loader as shown below:

LD −h

The Loader provides the following return values which are useful when using the Loader in a batch file:

0 PASSED: Loader ran to completion as requested by the user.

8 WARNING: Loader ran to completion, but not exactly as requested by the user.

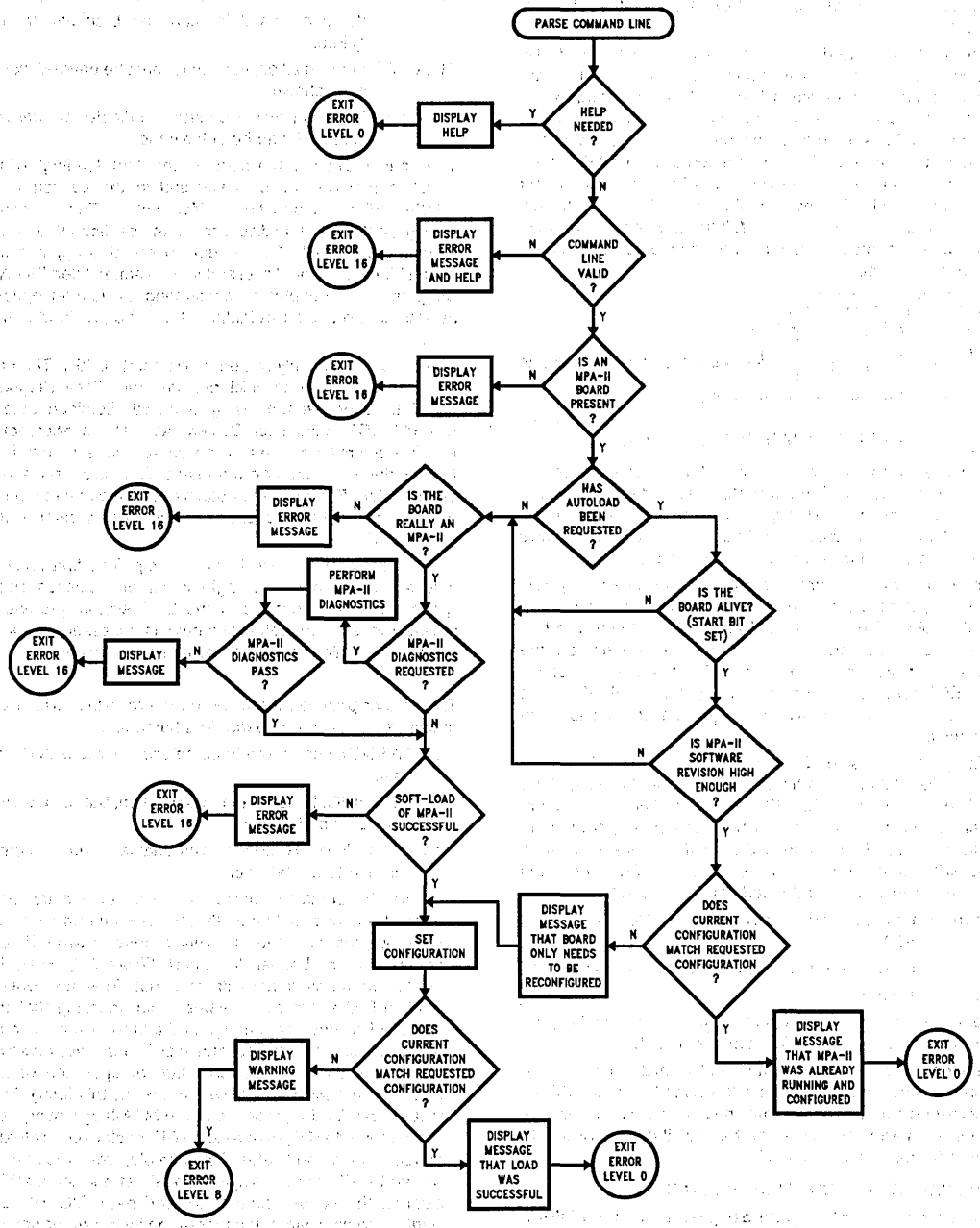
16 FATAL ERROR: Loader was unable to run to completion due to a fatal error.

Before the Loader implements any of its primary functions, the Loader will verify that the MPA-II printed circuit board is present in the PC. This is done in two different stages (see the Loader flow chart, *Figure 7-1*). First, the Loader performs a non-intrusive test. This test entails reading RIC a number of times while checking that the value of RIC does not change and that the single step bit of RIC is not set. The second test is intrusive, meaning that it will affect the current state of operation of the MPA-II, if the MPA-II is "alive" (more on this later). This test checks for the presence of the MPA-II by writing various patterns to RIC, then reading RIC back to check that it contains the correct value. For example, when the pattern written to RIC has the single-step bit set and the start bit cleared, the Loader expects to read back RIC with the single-step bit cleared. If either of the instrusive or non-in-

trusive tests fail, the Loader indicates the failure and exits with an error level of 16. The failure mechanism could be either of the following: an MPA-II printed circuit board is not present or an I/O conflict is occurring.

Soft-Loading Instruction Memory

The Loader uses the "soft-load" feature of the BCP to load files in either a binary format, referred to as "BCX" format; or in a simple ASCII PROM format, referred to as "FMT" format, into instruction memory. Files in these formats can be produced with National Semiconductor's DP8344 BCP Assembler System. The Loader can be used to load any file in one of these formats using the −B option to specify that the file format is "BCX" or the −F option to specify that the file format is "FMT". These options are useful when using the MPA-II Design/Evaluation kit to develop BCP code. The



TL/F/10488-45

FIGURE 7-1. Loader Flow Chart

MPA-II system can be soft-loaded immediately upon power-up, or from any state after power-up. Thus, the system may be reloaded without powering down or resetting.

Dual-port memory must be enabled prior to soft-loading the MPA-II because the Loader uses dual port memory to pass information, such as the instructions to be loaded, to the BCP. The Loader enables dual port memory by writing the upper byte of the address for the relocatable memory segment to the MPA-II's segment register. The MPA-II's segment register is mapped into the PC I/O space 2D7h. The Loader defaults to map dual port memory to the PC memory space CE000, But the user can move the location of dual port memory using the −U option.

The soft-load procedure begins by stopping the BCP's CPU. The BCP must be stopped when writing to either the program counter or instruction memory. The Loader then verifies that the BCP is set to access the low byte of instruction memory. This is accomplished in the following manner: the program counter is set to 0000h; RIC is then pointed to instruction memory, and a byte is read from instruction memory. At this time, the program counter is read to determine if it incremented. If it did, the BCP is now set to access the low byte of instruction memory. If the program counter did not increment, then the BCP is set to access the high byte of instruction memory, so the Loader reads another byte of instruction memory. Next, the Loader initializes the program counter to the starting address where instruction memory is to be loaded. The starting address of the program counter defaults to 0000h, but it is user selectable with either the −N or −R options. The program counter is written by pointing {RIC} to the low byte of the program counter, and then writing the low byte of the Instruction Address to dual port data memory. Next, {RIC} is set to point to the high byte of the program counter, and the high byte of the Instruction Address is written to dual port data memory.

Once the program counter has been initialized, the first instruction to be loaded into instruction memory is fetched from the BCX or FMT format file specified by the user. The instruction is then split into high and low bytes. This is necessary because the instructions are 16 bits wide, but they must be latched into instruction memory through the BCP's 8-bit Data bus. The instructions are then loaded into the MPA-II's instruction memory by pointing RIC to instruction memory and writing the low byte of the instruction followed by the high byte of the instruction to dual port memory. The program counter then auto-increments allowing the next instruction to be loaded. At any time, the program counter may be modified, followed by instruction loads, to allow areas of instruction memory to be skipped. The remaining instructions are loaded in the same manner. When all the instructions have been loaded, the system is started and configured as requested by the user.

Interrupts can occur prior to the execution of the first instruction loaded into instruction memory if a BCP program has been previously running in the MPA-II system with interrupts enabled. This is because the BCP uses a "dummy" instruction to fetch the first instruction in instruction memory and this "dummy" instruction does not disable interrupts. The following is a scenario that describes this: the MPA-II is running with a BCP program that has receiver interrupts enabled. The BCP is then stopped by clearing [STRT] in {RIC} and instruction memory soft-loaded with a new BCP program. Although the BCP's CPU is stopped, the receiver is operating independently and, therefore, the receiver still monitors the line for activity. If the receiver becomes active, it generates an interrupt to the BCP's CPU. When the BCP's CPU is started with the intention of running the BCP program just loaded, it will instead service the receiver interrupt immediately after the "dummy" instruction cycle. This will result in problems if the first and second BCP programs do not use the same interrupt table location because the new interrupt table location will not have been loaded into {IBR} yet. Therefore, the BCP will vector to an instruction address determined by the current contents of {IBR}, set by the first BCP program. Since the second BCP program has already been loaded into instruction memory, the interrupt table that is vectored to is meaningless and will create unexpected results. There are various methods which can be used to disable interrupts until the first instruction of the new code can be executed; for example, resetting the BCP. Since the Loader cannot reset the BCP, we chose to single step the BCP immediately after "soft-loading" the BCP's instruction memory and prior to starting the system running. This allows an interrupt, such as the receiver interrupt generated in the previous example, to be serviced during the single step. Servicing the interrupt automatically disables the Global Interrupt Enable by clearing [GIE]. After single stepping the BCP, its program counter must be reset. The BCP may now be safely started.

For convenience the Loader notation is repeated and the options which apply to the soft-loading are discussed here:

LD [Config_options . . .] [Options . . .]<Filename>

Filename: The file specified by the Filename contains the BCP microcode to be soft-loaded into the MPA-II system. The file format must be either BCX or FMT as described earlier in this section. The −B and −F options can be used to specify the file format as BCX or FMT, respectively. The file format can also be specified implicitly with a file extension of .BCX for BCX format files or .FMT for FMT format files. The Loader defaults to BCX format, and, if no file extension is entered, the Loader will append the appropriate file extension (i.e., either .BCX or .FMT). A file with no extension can be loader by ending the file name with a ".".

Options:

−**B**— Specifies that the format of the file to be loaded is binary or "BCX" format. This option provides the user with the flexibility to load a file with an extension other than .BCX as a BCX format file.

−**F**— Specifies that the format of the file to be loaded is ASCII PROM or "FMT" format. This option provides the user with the flexibility to load a file with an extension other than .FMT as a FMT format file.

−**N**[=][**l_addr**]— Soft-loads the file into instruction memory beginning at the hex address, l_addr, but does not start the MPA-II after the load. This feature can be useful for debugging code using tools such as Capstone's monitor debugger, BSID. The load address, l_addr, defaults to the hex address 0000.

−**R**[=][**addr**][,**r_addr**]— Soft-loads the file into instruction memory beginning at the hex address l_addr, then sets the program counter to r_addr and starts the BCP.

The instruction address where the BCP begins running, r_addr, defaults to the value of l_addr if r_addr is not specified. l_addr defaults to the hex address 0000.

−U[=][seg_id]— Enables dual port RAM in the PC memory map to the PC memory segment seg_id, where seg_id is the upper byte of the PC memory segment. This allows the MPA-II system to avoid PC memory conflicts. The Loader defaults to seg_id=CE. The value for the seg_id must be on an even 8K boundary. Therefore, seg_id=CD is invalid.

Examples using the file, MPA2.BCX, provided in the MPA-II Design/Evaluation Kit are shown below. This file is a BCX formatted file. The following examples all load the file MPA2.BCX in the same format and demonstrate the −B and −F options:

```
LD MPA2          Loader defaults to BCX
                 format and applies the .BCX
                 file extension.
LD MPA2.BCX      Loader determines that
                 format is BCX from the file
                 extension.
LD MPA2.BCX −B   Loader determines that the
                 file format is BCX from
                 the -B option.
```

The following example demonstrates options which affect how the file is soft-loaded:

```
LD MPA2 -R=0000, 0126 -U=CC
```

In this example, the Loader soft-loads code through dual port memory mapped at the PC memory address CC000, from the BCX format file MPA2.BCX, starting at instruction memory 0000h. The Loader then sets the program counter to 0126h and starts the BCP.

Configuring The MPA-II

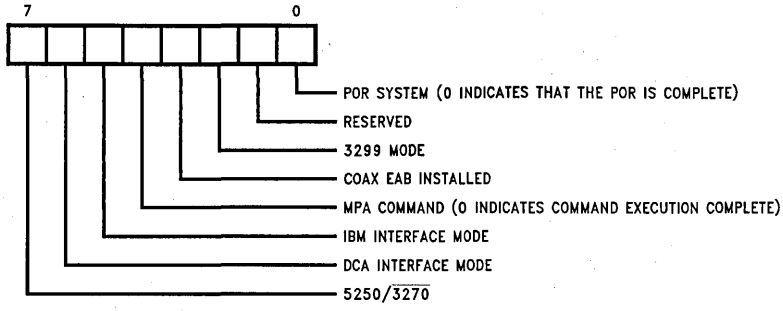
The Loader configures the MPA-II terminal emulation interface mode as requested by the user through the Configuration Options. Configuring the MPA-II interface mode enables the MPA-II to emulate the standard PC terminal emulation interfaces including DCA's IRMA and Smart Alec interface modes; and IBM's 3270 CUT and DFT interface modes. In addition, the MPA-II extends the DCA and IBM 3270 modes to support single session 3299. (Multi-session 3299 support is possible for the BCP, but not for the DCA or IBM interfaces.) The terminal emulation interface which the MPA-II emulates is implemented by the MPA-II as described in Chapter 6. The Loader Configuration Options available to the user will be discussed later in this section.

The Loader configures the MPA-II interface mode by writing the configuration to the MPA-II's Configuration Register. *Figure 7-2* shows the bit definitions for the MPA-II Configuration Register. The Loader writes to the configuration register immediately after starting the BCP's CPU. The MPA-II configuration register is located at the PC I/O location 2DCh. Writing to this register will set the BIRQ interrupt, and thus, could lock out the PC if this feature has been activated by previous BCP microcode. If the BCP's CPU is stopped when the configuration register is written, then the next access of the BCP's memory (both dual port and I/O) made by the PC could be held off indefinitely since the BIRQ interrupt can not be cleared by the BCP's microcode. Therefore, when the Loader Option −N, as described in the previous section, is selected, the Loader will not set the configuration requested. (The Loader notifies the user that the configuration has not been set.) See Chapter 5 for further information regarding BIRQ and the PC lock out feature.

The Loader uses the following handshaking protocol with the BCP microcode to verify that the configuration has been recognized by the MPA-II system. The Loader sets [POR] in the MPA-II Configuration register when it writes a configuration to the MPA-II Configuration Register. The Loader then polls the MPS-II Configuration Register looking for [POR] to be cleared by the BCP microcode. This indicates that the BCP microcode has processed the requested configuration. The value in the MPA-II Configuration Register now contains the actual MPA-II interface configuration implemented by the BCP microcode. If [POR] is not cleared within a predefined time period, then the Loader reports a failure. If [POR] is cleared within the predefined time limit, the Loader then compares the configuration implemented with the configuration requested by the user. If they are not the same, the Loader reports the differences. This feature allows the BCP microcode to determine valid configurations.

The Loader Configuration Options are discussed here. Where applicable, these options can be combined to create a customized configuration for the interface mode. Once again, for convenience the Loader notation is repeated:

```
LD[Config_options...][Options...]<Filename>
```



TL/F/10488–46

FIGURE 7-2. MPA-II Configuration Register

Config_options:

-C—The Loader clears the 5250/$\overline{3270}$ bit of the MPA-II Configuration Register. This selects a 3270 Coax-Twisted Pair terminal emulation interface mode for the MPA-II interface.

-D—The Loader sets the DCA Interface Mode bit of the MPA-II Configuration Register. This selects a DCA terminal emulation interface mode for the MPA-II interface. The 5250/$\overline{3270}$ bit of the MPA-II Configuration Register is used to determine which DCA Interface mode, IRMA or Alec, is actually set.

-E—The Loader sets the EAB bit of the MPA-II Configuration Register. This selects the Coax Extended Attribute Buffer.

-I—The Loader sets the IBM Interface Mode bit of the MPA-II Configuration Register. This selects the IBM 3270 terminal emulation interface mode for the MPA-II interface.

-T—The Loader set the 5250/$\overline{3270}$ bit of the MPA-II Configuration Register. This selects a 5250 Twinax terminal emulation interface mode for the MPA-II interface.

-X[=]<**addr**>—The Loader sets the 3299, mux, bit of the MPA-II Configuration Register. This selects 3299 coax mode for the MPA-II interface. A decimal muX address is required, and is passed to the MPA-II through the MPA-II parm/response register, 2DBh, which is written prior to the configuration being set, but after the BCP's CPU is started.

-Z—The Loader does not set the MPA-II Configuration Register. This option provides the flexibility to use the Loader to load microcode other than the MPA-II microcode.

-M[=]<**mode**>—This option allows for automatic configuration of the standard terminal emulation modes, i.e.,: DCA's IRMA, DCA's Smart Alec and IBM's interface modes. Valid MODE options are IRMA, IBM, and ALEC. These modes set the MPA-II Configuration Register as follows: When the mode is ALEC, the Loader sets the 5250/$\overline{3270}$ bit and the DCA Interface Mode bit in the MPA-II Configuration Register. For IBM mode, the Loader clears the 5250/$\overline{3270}$ bit and sets the IBM Interface Mode bit. For IRMA mode, the Loader clears the 5250/$\overline{3270}$ bit, sets the DCA Interface bit and the Coax EAB bit. This option also allows a hex value to be entered directly into the MPA-II Configuration Register with the <MODE> =CONFIG[=]<config>, where config is the hex byte value to be loaded into the MPA-II Configuration Register. The Loader defaults to configure the MPA-II interface mode for IRMA.

As an example of how to use the configuration options, lets assume that the IRMA interface mode is required along with coax 3299 support using the 3299 station address 3. The following command lines all perform this task using the Configuration Options discussed above:

```
LD MPA2.BCX -M=IRMA -X=3
LD MPA2.BCX -C -D -E -X=3
```

For further flexibility, the Loader also provides an autoload option, -a, to configure the MPA-II on the fly. The autoload function is actually a "smart hotswitch", allowing the user to change the MPA-II's configuration without necessarily reloading BCP microcode. The autoload is "smart" in that the Loader verifies that the MPA-II is "alive" before it changes configurations. If the MPA-II is not alive (i.e., running with the correct version of microcode), the Loader will automatically load the BCP microcode and configure the MPA-II as requested.

The autoload function is useful when the Loader is used in a batch file such as the AUTOEXEC.BAT file. If the PC is rebooted then the Loader will not destroy an ongoing terminal emulation session. In addition, the error levels returned by the Loader may be used to skip loading of the PC terminal emulator if the MPA-II board is not present. The following is an example of how to use the autoload function to implement the IRMA interface mode in a batch file:

```
LD MPA2.BCX -M=IRMA -A
IF ERRORLEVEL 8 GOTO SKIPIRMA
E78 /R
:SKIPIRMA
```

MPA-II Diagnostics

The Loader can run diagnostics to test the functionality of the MPA-II hardware. These diagnostics are implemented with the Loader and the BCP microcode; MPADIAG.BCX, provided in MPA-II Design/Evaluation Kit. Note, the Loader expects the file MPADIAG.BCX to be located in the same directory as the file LD.EXE.

Figure 7-3, The MPA-II Diagnostics Flow Chart, provides a good overview of the extent of testing performed by the MPA-II diagnostics. For the actual implementation of these tests, refer to the source code, which is well documented. The first four diagnostic tests do not require BCP microcode. These diagnostics include testing RIC, the BCP's Program Counter, dual port memory, and instruction memory. In all of these diagnostics, the Loader writes patterns to the device under test, and expects to read the pattern back from the device under test.

If all these initial tests pass, then the BCP microcode, MPA-DIAG.BCX is soft-loaded into instruction memory and the BCP is started. The Loader maintains ultimate control over the diagnostics. This is accomplished through a handshaking protocol in which dual port memory is used to pass codes to and from the Loader program and the BCP microcode program, MPADIAG.BCX. The Loader passes a start code through dual port memory. The BCP microcode polls dual port memory until it receives the start code. Once the BCP microcode recognizes the start code, it executes the next test in sequence. Each diagnostic test that the BCP microcode executes writes codes into dual port memory to indicate both the completion of the test and if the test passed or failed. When appropriate, the BCP microcode also indicates the failure mechanism. The BCP microcode then polls dual port memory for the start code of the next test. After the Loader writes a start code to dual port memory, it polls dual port memory for the code from the BCP microcode indicating completion of the test. If the completion code is not received within a predefined time limit, the Loader indicates a failure. If the completion code is received, the Loader then checks dual port memory to determine if the test passed or failed.

Either of the two Loader Options, -s or -l, cause the Loader to implement the MPA-II diagnostics. For convenience the Loader notation is repeated and the options which apply to the MPA-II diagnostics are discussed here:

```
LD[Config_options...][Options...]<Filename>
```

Options:

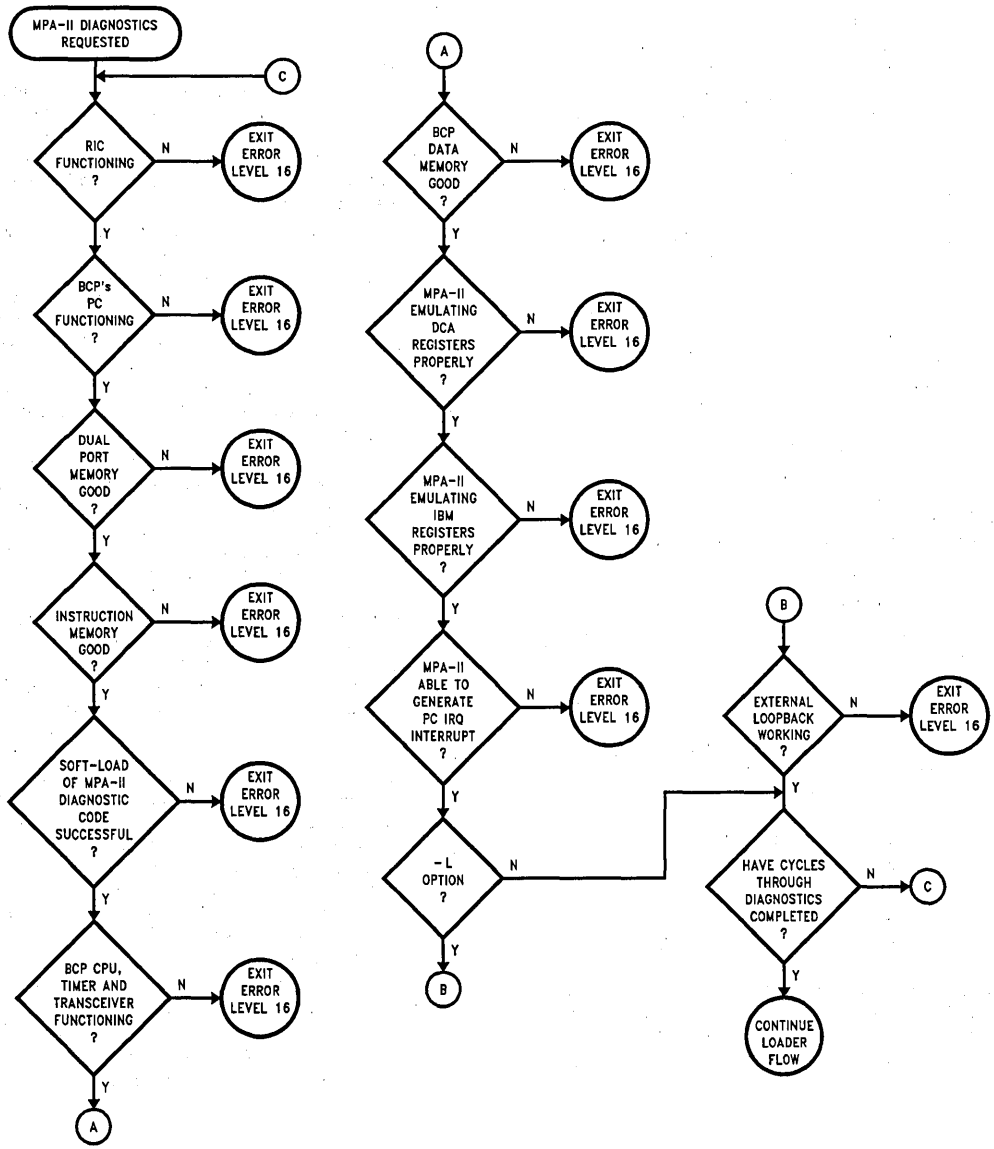
-S[=][count[,Irq#]]— Selftest option of the Loader. Cycles through the MPA-II Diagnostics count (default count = 1) times. The irq# refers to the PC IRQ interrupt level to be tested. irq# = 2 (default) tests the PC IRQ2 interrupt (i.e., jumper JP6 connected). irq#3 tests the PC IRQ3 interrupt (i.e., jumper JP4 connected). irq# = 4 tests the PC IRQ4 interrupt (i.e., jumper JP5 connected).

-L— In addition to the selftest, the BCP's transceiver is tested by implementing an external Loopback feature. In loopback, the BCP's receiver and transmitter are allowed to be active at the same time. This allows the BCP to test the external transmitter and receiver logic on the MPA-II board. This test should not be performed when the MPA-II is connected to a controller as it may cause the controller to detect line errors.

The following examples demonstrate using the Loader options to implement the MPA-II diagnostics:

LD -S = 3, 4 Cycles through the MPA-II diagnostics three times (the external loopback test is not performed), the PC IRQ interrupt level 4 is tested.

LD -L -S Cycles through the MPA-II diagnostics one time, the loopback test is performed, and PC IRQ interrupt level 2 is tested.



TL/F/10488-47

FIGURE 7-3. MPA-II Diagnostics Flow Chart

APPENDIX A

HARDWARE REFERENCE

```
"_____________________________________
"       name:           MPAII_AC - U3   V3.02
"       description:    auxillary control register
"
"       Provides line interface logic, including the coax and
"       twinax TX_ACT signals as well as the ex-or of /DATA_OUT and
"       DATA_DLY for the twinax logic.
"          AD6 -> COAX   (if AD6 lo, COAX enabled)
"       IRQ output to pc is registered output of AD7 in this pal.  If
"       the BCP puts a 1 to AD7 during a write to this register, the
"       PC interrupt will be asserted.  The interrupt is cleared by
"       a BCP write with AD7 = 0.
"
"
"       history:V0.1    msa     12/13/87        create
"               V0.2    msa     12/16/87        Abel version
"               V0.3    msa     12/31/87        added IRQ
"               V0.4    msa     02/27/88        u9 -> u4
"               V1.0    msa     04/07/88        u4 -> aux_ctl
"               V3.00   wvm     08/10/88        eliminated manual reset
"               V3.01   wvm     09/07/89        make signal names match
"                                               schematic and add test vectors
"               V3.02   wvm     09/11/89        add BIRQ_EN function
"
"_____________________________________
"               COPYRIGHT NATIONAL SEMICONDUCTOR, INC. 1987,1988,1989,1990
"_____________________________________

module mpaii_aux_ctl
title 'data register'

"declarations
                        mpaii_ac        device 'P16R4';
        TX_ACT          pin     9;
        DATA_DLY        pin     8;
        DATA_OUT        pin     7;
        AD5             pin     6;
        BCP_RST~        pin     5;
        PC_RST          pin     4;
        AD6             pin     3;
        AD7             pin     2;

        OE~             pin     11;
        DREG_CLK~       pin     1;

        COAX_ACT        pin     19;
        TWX_ACT         pin     18;
        act_swtch       pin     17;
        not_used1       pin     16;
        BIRQ_EN         pin     15;
        IRQ             pin     14;
        INTENSE         pin     13;
        IRST~           pin     12;

        H,L,C,Z,X       =       1,0,.C.,.Z.,.X.;
        outputs         =       [COAX_ACT,TWX_ACT,INTENSE,IRST~];
        r_outputs       =       [act_swtch,not_used1,BIRQ_EN,IRQ];
```

TL/F/10488-48

```
equations
        COAX_ACT        =       !act_swtch & TX_ACT & BCP_RST~;
        TWX_ACT         =       act_swtch & TX_ACT & BCP_RST~;
        !INTENSE        =       DATA_OUT & !DATA_DLY # !DATA_OUT & DATA_DLY;
        IRST~           =       !PC_RST;

"       ****    registered outputs      ****
        BIRQ_EN         :=      AD5;
        act_swtch       :=      AD6;
        IRQ             :=      AD7;

 enable outputs         =       ^b1111;
 enable r_outputs       =       !OE~;


end mpaii_aux_ctl
```

TL/F/10488-49

```
"_____________________________________
"       name:           MPAII_CT - U8    V3.03
"       description:    BCP Data memory Decode,
"                       PC transceiver control timing
"
"       history:V0.1    msa     12/13/87        create
"               V0.2    msa     12/16/87        Abel version
"               V0.3    msa     12/17/87        used all outputs, 1 in free
"               V0.4    msa     12/23/87        more vectors, remote
"               V0.5    msa     02/19/88        moved areg map next to RAM
"                                               areg now cleared by BCP reads,
"                                               Host reads will not affect
"               V0.6    msa     02/27/88        u5a -> u9a
"               V1.0    msa     04/07/88        u9a -> ctl_tim
"               V1.1    msa     07/07/88        fixed DATA_G~ bus contention
"                                               (REMRD~ -> REMWR~)
"               V3.00   wvm     08/10/89        made revisions for MPAII:
"                                               1) eliminated unused chip
"                                                  selects
"                                               2) removed bcp_rst, it was an
"                                                  unused signal input
"                                               3) moved the pc_rdy signal to
"                                                  MPAII_RI
"                                               4) added PRE_BIRQ decode
"                                               5) added A13 and A14 decodes
"                                                  for remote accesses
"               V3.01   wvm     09/7/89         make signal names match
"                                               schematic and add test vectors
"               V3.02   tas     10/17/89        change name of DATA_CBA to
"                                               DATA_CAB and corrected equation
"                                               to use XACK instead of BCP_RD.
"               V3.03   tas     10/25/89        replace IOD0 with IBM_REG~,
"                                               simplified PRE_BIRQ,and modified
"                                               out_A13 and out_A14 to include
"                                               IBM_REG~
"
"_____________________________________
"               COPYRIGHT NATIONAL SEMICONDUCTOR, INC. 1987,1988,1989,1990
"_____________________________________

module mpaii_ctl_tim
title 'PC iface - data control'

"declarations
                        mpaii_ct        device 'P20L10';

        XACK            pin     1;
        REMRD~          pin     2;
        REMWR~          pin     3;
        IBM_REG~        pin     4;
        RAE~            pin     5;
        IO_ACCESS~      pin     7;
        BCP_RD~         pin     10;
        BCP_WR~         pin     11;
        LCL~            pin     13;
        A15,A14,A13     pin     9,6,8;
```

TL/F/10488-50

```
        DMEM_CS~        pin     14;
        AREG_OC~        pin     23;
        AREG_CLK~       pin     15;
        out_A13         pin     16;
        DATA_DIR~       pin     17;
        DATA_CAB~       pin     18;
        DATA_G~         pin     19;
        out_A14         pin     20;
        DREG_CLK~       pin     21;
        PRE_BIRQ        pin     22;

        H,L,Z,X         =       1,0,.Z.,.X.;
        a_dec           =       [A15..A13];
        outputs         =       [DMEM_CS~,AREG_OC~,AREG_CLK~,DREG_CLK~,
                                 DATA_DIR~,DATA_CAB~,DATA_G~,PRE_BIRQ];
        bcp_oc          =       [out_A14,out_A13];

equations
        !DMEM_CS~   = LCL~ # (!A15 & !LCL~);
        !AREG_CLK~  = !IO_ACCESS~ & LCL~ & !BCP_WR~;        " pc access to set
        !AREG_CLK~  = !LCL~ & !BCP_WR~ & (a_dec == ^b100);    " bcp access to clear
        !AREG_OC~   = !BCP_RD~ & !LCL~ & (a_dec == ^b100);

        out_A13     = !IO_ACCESS~ # !IBM_REG~;
        out_A14     = !IO_ACCESS~ # !IBM_REG~;

        PRE_BIRQ    = LCL~;

        !DREG_CLK~  = !LCL~ & !BCP_WR~ & (a_dec == ^b101);

        !DATA_G~    = (!REMWR~ & LCL~) # (!RAE~ & !REMRD~);
        !DATA_CAB~  = !REMRD~ & !XACK & LCL~;
        DATA_DIR~   = !REMRD~ & !RAE~;              "Abus -> Bbus if rem_read cycle

 enable outputs     = ^b111111111;
 enable bcp_oc      =  LCL~;

end mpaii_ctl_tim
```

TL/F/10488-51

```
"_____________________________________
"       name:            MPAII_PD - U9    V3.03
"       description:     Upper PC address buffer and I/O decode
"
" This PAL decodes the PC address lines A12-A4,PC_A19_16 ,PC_A16_14 and
" IOACCESS~, REMRD~ to provide the BCP A15, A12-8 outputs and the
" two partial decodes IBM,DCA indicating which system (IBM, or DCA)
" is being accessed.  Note that this scheme will not allow the MPAII board
" to co-exist w/ any  board using these PC I/O addresses.  A15, A12-8
" are enabled by LCL~ going high, indicating a remote access cycle.
"
" If IOACCESS~ and LCL~ are asserted, A12-8 are driven high to translate
" the PC I/O access to the top page of the BCP's data RAM (7FFX - unless
" it is a read of IRMA space, which is translated to 7FEX,) required to
" emulate the dual-ported registers used on this board.
"
" If the PC accesses the BCP's data memory, LCL~ will be asserted but not
" IOACCESS~, in which case no translation will occur, and A12-8 will only
" be buffered onto the BCP's address lines.
"
"     IBM_REG~ DCA_REG~   -type of decode-
"        1        1       not an MPAII IO decode
"        1        0       DCA IO access (addresses 0022x)
"        0        1       IBM IO access (addresses 002dx)
"        0        0       not an MPAII IO decode
"
"
"       history:V0.1     msa      12/13/87         create
"               V0.2     msa      12/16/87Abel     version
"               V0.3     tjq      12/17/87         simulate
"               V0.4     msa      02/27/88         u3 -> u23
"               V0.5     msa      03/03/88         corrected address pin nums
"               V0.6     msa      03/12/88         edits for TN's irma code
"               V1.0     msa      04/07/88         u23 -> pad_dec
"               V3.00    wvm      08/10/89         made revisions for MPAII:
"                                                  1) moved REMOTE decode to an
"                                                     inverter
"                                                  2) moved A13 and A14 decodes
"                                                     for remote accesses to
"                                                     MPAII_CT
"                                                  3) include decode of PC_A14
"                                                     through PC_A19 in IOD0
"                                                     and IOD1
"               V3.01    wvm      09/07/89         make signal names match
"                                                  schematic and add test vectors
"               V3.02    wvm      09/11/89         rearrange pins
"               V3.03    tas      10/25/89         renamed IOD0 and IOD1 to DCA_REG~
"                                                  and IBM_REG~,respectively. Swapped
"                                                  IOD0/1 pins. Added IBM_REG~ to
"                                                  A12-A8.
"
"_____________________________________
"               COPYRIGHT NATIONAL SEMICONDUCTOR, INC. 1987,1988,1989,1990
"_____________________________________

module mpaii_pad_dec
title 'PC iface - i/o decode'
```

TL/F/10488-52

```
"declarations
                        mpaii_pddevice 'P20L10';

        PC_A12,PC_A11,PC_A10      pin      1,3,4;
        PC_A9,PC_A8,PC_A7         pin      5,6,8;
        PC_A6,PC_A5,PC_A4         pin      9,10,11;
        PC_A14_16                 pin      15;
        PC_A17_19                 pin      16;

        REMRD~                    pin      2;
        IO_ACCESS~                pin      7;
        LCL~                      pin      13;

        A15                       pin      14;
        A12,A11                   pin      17,18;
        A10,A9,A8                 pin      19,20,23;

        IBM_REG~,DCA_REG~         pin      21,22;


        H,L,X,Z  =   1,0,.X.,.Z.;
        low      =   ^b11;
        pc_a     =   [PC_A12..PC_A4];
        pc_abuf  =   [PC_A12..PC_A8];
        bcp_oc   =   [A12..A8];
        bcp_a    =   [A12..A8];
        io_dec   =   [IBM_REG~,DCA_REG~];

equations
        !IBM_REG~        =        (pc_a == ^h02d) & PC_A14_16 & PC_A17_19;
        !DCA_REG~        =        (pc_a == ^h022) & PC_A14_16 & PC_A17_19;

        A15              =        L;

        A12              =        PC_A12 # !IO_ACCESS~ # !IBM_REG~;
        A11              =        PC_A11 # !IO_ACCESS~ # !IBM_REG~;
        A10              =        PC_A10 # !IO_ACCESS~ # !IBM_REG~;
        A9               =        PC_A9  # !IO_ACCESS~ # !IBM_REG~;
        A8               =        PC_A8  # (!IO_ACCESS~ & REMRD~)  # !IBM_REG~;


        enable io_dec    =        low;
        enable bcp_oc    =        LCL~;
        enable A15       =        LCL~;

end mpaii_pad_dec
```

TL/F/10488-53

```
"_____________________________________
"       name:           MPAII_RD - U7    V3.04
"       description:    PC I/O register decode
"
" Decodes the low 4 bits of the PC address lines to determine enables for
" data memory, RIC, and SREG.  All four PC read and write
" strobes as well as the IBM_REG~/DCA_REG~, PC_A13, PC_AEN, REM_enable and /MMATCH
" signals are inputs. /RAE, CM /MEM_CS and /REM_RD, /REM_WR, /IOACCESS
" and /SREG_EN are outputs.
"
"       history:V0.1    msa     12/13/87        create
"               V0.2    msa     12/16/87        Abel version
"               V0.3    msa     12/17/87        added pc_clk edit
"               V0.4    msa     12/17/87        to 2018
"               V0.5    msa     12/31/87        added bcp reset input
"               V0.6    msa     02/27/88        u4 -> u8
"               V0.7    msa     03/12/88        edited for tn's irma
"               V1.0    msa     04/07/88        u8 -> reg_dec
"               V3.00   wvm     08/10/89        revisions made to MPAII:
"                                               1) eliminate PC_HI_OC and
"                                                  PC_LO_OC
"                                               2) remove IO_MAYBE and replace
"                                                  it with PC_AEN
"                                               3) input PC_A13 for address
"                                                  decodes
"                                               4) input REM_enable to control
"                                                  accesses during rest-time
"                                               5) Make RAE~ a full decode of
"                                                  every access
"               V3.01   wvm     09/7/89         make signal names match
"                                               schematic and add test vectors
"               V3.02   wvm     09/11/89        rearrange pin assignments
"               V3.03   tas     10/19/89        added PC_AEN to SREG_EN to
"                                               eliminate accidental clock of SREG.
"               V3.04   tas     10/25/89        Rename IOD0, IOD1 to DCA_REG~,
"                                               IBM_REG~, repectively.
"                                               Swap IOD0, IOD1 pins. Changed
"                                               IO_ACCESS to avoid SREG.
"
"_____________________________________
"               COPYRIGHT NATIONAL SEMICONDUCTOR, INC. 1987,1988,1989,1990
"_____________________________________

module mpaii_reg_dec
title 'PC iface - i/o decode'

"declarations
                        mpaii_rd    device 'P20L8';

        BCP_RST~        pin     1;
        DCA_REG~        pin     2;
        PC_AEN          pin     3;
        IBM_REG~        pin     4;
        PC_A3,PC_A2     pin     5,6;
        PC_A1,PC_A0     pin     7,8;
        PC_MEMR~        pin     9;
        PC_MEMW~        pin     10;
        PC_IOR~         pin     11;
```

TL/F/10488-54

```
        PC_IOW~          pin      13;
        BCP_WR~          pin      14;
        PC_A13           pin      16;
        REM_enable       pin      17;
        MMATCH~          pin      23;


        RAE~             pin      22;
        REMRD~           pin      21;
        REMWR~           pin      20;
        CMD              pin      19;
        IO_ACCESS~       pin      18;
        SREG_EN~         pin      15;

        CMD              ISTYPE   'feed_pin';

        H,L,X,Z =        1,0,.X.,.Z.;
        pc_a    =        [PC_A3..PC_A0];
        pc_hi   =        (pc_a == ^b1110);
        pc_lo   =        (pc_a == ^b1101);
        io_dec  =        [IBM_REG~,DCA_REG~];
        no_acc  =        (io_dec == ^b11);
        dca_acc =        (io_dec == ^b10);
        ibm_acc =        (io_dec == ^b01);
        strobes =        [PC_MEMR~,PC_IOR~,PC_MEMW~,PC_IOW~];
        outputs =        [CMD,RAE~,REMWR~,REMRD~,
                         IO_ACCESS~,SREG_EN~];

equations
        !RAE~   =        ((!PC_IOR~ # !PC_IOW~)&!no_acc&!PC_AEN&!PC_A13)
                          # (!MMATCH~ & !PC_AEN & (!PC_MEMW~ # !PC_MEMR~));

        CMD     =        (pc_a == ^b1111)&(ibm_acc)&(!PC_IOR~ # !PC_IOW~);

        !REMRD~ =        !PC_IOR~ & REM_enable
                          # !PC_MEMR~ & REM_enable;

        !REMWR~ =        !PC_IOW~ & REM_enable
                          # !PC_MEMW~ & REM_enable;

       !SREG_EN~ =       (!PC_IOW~)&(pc_a == ^b0111)&(ibm_acc)&(!PC_A13)&!BCP_WR~
                          &!PC_AEN # !BCP_RST~;

       !IO_ACCESS~= !DCA_REG~ & !CMD & !PC_A13 & !PC_AEN & (!PC_IOW~ # !PC_IOR~)
                   # !IBM_REG~ & !CMD & !PC_A13 & !PC_AEN & !PC_IOR~
                   # !IBM_REG~ & !CMD & !PC_A13 & !PC_AEN & !PC_IOW~ &
                   (!PC_A0 # !PC_A1 # !PC_A2);
" IO_ACCESS~ is active if:
"       1) it's an IRMA register read or write
"       2) it's an IBM register read
"       3) it's an IBM register write except to xxx7 or xxxF.
"               xxx7 = Segment Register (U16)
"               xxxF = the BCP's RIC register
"
 enable outputs =        ^b111111;

end mpaii_reg_dec
```

TL/F/10488-55

```
"_____________________________________
"       name:           MPAII_RI - U4   V3.02
"       description:    birq register and rest-time circuit
"
"       This PAL sends the BIRQ interrupt to the BCP whenever a
"       IBM, IRMA or SMART_ALEC I/O access is made.  The BIRQ
"       interrupt is cleared when the BIRQ register is read by
"       the BCP.
"       This PAL also contains all of the rest-time state machine
"       needed to pevent missed or inproper accesses.  The signal
"       REM_enable is fed back to MPAII_RD and prevents remote accesses
"       during rest-time.  The PC_RDY signal to the PC bus is also
"       controlled by XACK from the BCP and the rest-time state
"       machine.
"
"       history:V3.00   wvm     08/10/89        create
"               V3.01   wvm     09/07/89        make signal names match
"                                               schematic
"               V3.02   wvm     09/11/89        change BIRQ decode
"               V3.02   tas     01/03/90        corrected some test vectors
"
"_____________________________________
"               COPYRIGHT NATIONAL SEMICONDUCTOR, INC. 1989-1990
"_____________________________________


module remote_interface_pal
title 'REST-TIME Compliance State Machine';

        MPAII_RI device  'P16RA8';

"inputs
        PL~             pin 1;
        XACK            pin 2;
        CLK_OUT         pin 3;
        BCP_rst~        pin 4;
        RAE~            pin 5;
        unused_1        pin 6;
        BIRQ_EN         pin 7;
        AREG_CLK~       pin 8;
        PRE_BIRQ        pin 9;
        OE~             pin 11;


"outputs
        PC_RDY~         pin 19;
        q0              pin 18;
        q1              pin 17;
        q2              pin 16;
        q3              pin 15;
        wait_start      pin 14;
        REM_enable      pin 13;
        BIRQ            pin 12;


"definitions
        x,z,L,H         = .X., .Z.,0,1;
```

TL/F/10488-56

```
equations

        enable PC_RDY~    = (!RAE~);

        PC_RDY~.RE        = 1;
        PC_RDY~.PR        = 1;

        !PC_RDY~          := (!XACK
                          # q0 & !RAE~
                          # !q2 & !RAE~
                          # q3 & !RAE~);


        REM_enable.RE     = 1;
        REM_enable.PR     = 1;

        !REM_enable       := wait_start
                          # RAE~ & !q3 & q2 & q1 & !q0;


        q3.PR             = !BCP_rst~;
        q3.C              = CLK_OUT;

        !q3               := (!q0 & !q2 & !q3
                          # !q0 & !q1 & !RAE~
                          # !q0 & !q3 & !RAE~ & !wait_start
                          # !q0 & !q1 & q2);


        q2.RE             = !BCP_rst~;
        q2.C              = CLK_OUT;

        !q2               := (!q0 & !q1 & !q2 & !q3
                          # !q1 & q3 & !RAE~
                          # q1 & !q2 & q3
                          # q0 & !q1 & q3);


        q1.PR             = !BCP_rst~;
        q1.C              = CLK_OUT;

        !q1               := (!q0 & !q1 & q3
                          # !q0 & !q2 & q3
                          # q0 & q2 & q3
                          # !q0 & !q1 & q2 & RAE~);


        q0.PR             = !BCP_rst~;
        q0.C              = CLK_OUT;

        !q0               := (!q0 & !q1
                          # !q0 & !q2
```

TL/F/10488-57

```
                    # BIRQ & q1 & !q2 & q3
                    # !q0 & !q3);


wait_start.PR     = q3 & !q2 & !q1 & !q0;
wait_start.C      = RAE~ & BCP_rst~;

!wait_start       := !q3 & q2 & !q1 & !q0;


BIRQ.RE = !BCP_rst~;
BIRQ.C            = AREG_CLK~;

!BIRQ             := PRE_BIRQ & BIRQ_EN;

end remote_interface_pal
```

TL/F/10488-58

APPENDIX B

Timing Analysis

This section will first discuss the timing analysis used in selecting appropriate data memory and instruction memory for use in the MPA-II system. Following this is a description of the timing involved in interfacing the MPA-II system with the PC-XT/AT.

As discussed in Chapter 5—Hardware Architecture, the BCP utilizes a Harvard Architecture, where the data memory and instruction memory are organized into two independent memory banks, each with their own address and data buses. The data memory is dual ported enabling both the BCP and the remote processor to have access. The instruction memory, on the other hand, is exclusively owned by the BCP. Any remote processor accesses to this memory occur through the BCP, and only when the BCP is idle.

The MPA-II system runs with the BCP operating at full speed, 18.8696 MHz ({DCR[CCS]} = 0), with zero instruction (n_{IW}) and one data (n_{DW}) wait state resulting in a T-state of 53 ns. For a system running the BCP at half speed, 9.45 MHz ({DCR[CCS]} = 1), with zero instruction and zero data wait states, the T-state is 106 ns. The T-state is calculated as shown:

$$\text{T-state} = 1/(\text{CPU Clock Frequency})$$

Interfacing Memory to the DP8344B

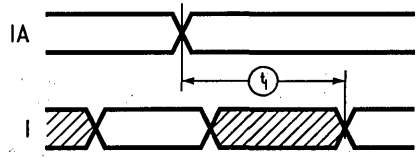
As with most other aspects of a design, choosing memory is a cost vs. performance trade off. Maximum performance is achieved running no wait-states with fast, expensive memory. Slower, less expensive memory can be used, but wait-states must be added, slowing down the BCP. Therefore one needs to choose the slowest memory possible while still meeting design specifications. While this appendix assumes RAM is used for instruction and data memory, the information is relevant to memory devices in general.

Instruction Memory Timing

The BCP needs separate data and instruction RAM, each with their own requirements. Instruction read time is the major constraint when choosing instruction RAM. Instruction read time t_I, as shown in *Figure B-1*, is measured from when the instruction address becomes valid to when the next instruction is latched into the BCP. Instruction read time for various clock frequencies and wait states are given in Table B-1. Clock frequency and wait state combinations other than those given in the table can be calculated using parameter 1 in Table 5-5, Instruction Memory Read Timing, of the DP8344B data sheet:

$$t_I = (1.5 + n_{IW})\,T - 19\text{ ns}$$

where t_I is the instruction read time (ns), n_{IW} is the number of instruction memory wait states, and T is the 7-state time (ns). The RAM chosen needs to have a faster access time than the read time for the desired combination of clock frequency and wait states. Since the MPA-II system runs at full speed (18.8696 MHz) with $n_{IW} = 0$, the RAM chosen for instruction memory must have an access time which is faster than 60.5 ns (See Table B-1). Note that 55 ns Static Rams will work for both full speed and half speed operation of the MPA-II.



TL/F/10488–59

FIGURE B-1. Instruction Memory Read Timing

TABLE B-1. Instruction Read Times, t_I (ns)

CPU Clock Freq. (MHz)	Wait States n_{IW}			
	0	1	2	3
9.43	140	246	352	458
18.86	60.5	115.5	166.5	219.5
20.00	56	106	156	206

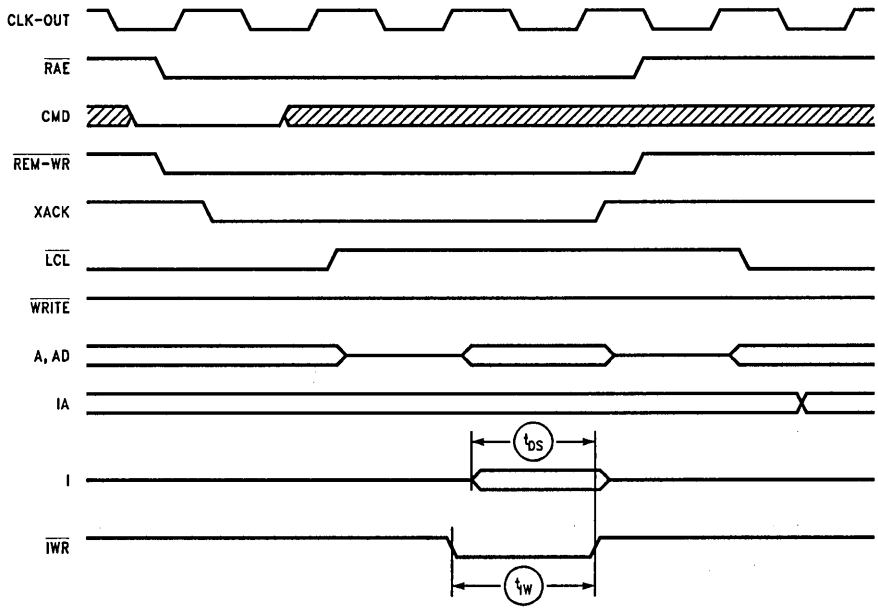
However, instruction read time is not the only timing consideration when choosing instruction RAM. If the BCP is used in an application which requires full speed softloading of instruction RAM, there are two other timing relationships which require evaluation. These are data setup time and write pulse width (see *Figure B-2*). The relevant BCP timing parameters are I valid before $\overline{IWR}$ rising, t_{DS}, and $\overline{IWR}$ low time, t_{IW}. The value of these timing parameters depends on the Remote Interface mode of operation, which is Fast Buffered Write for the MPA-II system. Using Table 5-22, Fast Buffered Write of IMEM, of the DP8344B datasheet, the data setup time (parameter 19) is:

$$t_{DS} = (n_{IW} + 1)T - 18\text{ ns}$$

and the write pulse width t_{IW} (parameter 20) is:

$$t_{IW} = (n_{IW} + 1)T - 10\text{ ns}$$

Table B-2a and B-2b give various data setup times and write pulse widths. Once again, the RAM chosen must have a faster RAM data setup time and quicker RAM write strobe width than the corresponding desired data setup time and write pulse width. Thus, for the MPA-II system, the selected Instruction RAM data setup time must be less than 35 ns (Table B-2a), and the RAM Write Strobe Width must be less than 43 ns (Table B-2b). In a typical application of the BCP, softloading occurs after reset with the BCP operating with CLK/2 and full wait states. Under these conditions the instruction read time value is the critical parameter for choosing the instruction RAM. In the MPA-II system, softloading can also occur under the full speed conditions. First, softloading occurs upon a first load of instruction memory into the MPA-II on power up. The MPA-II system can then be reloaded without powering down. In this situation, the MPA-II system is set to full speed. Therefore, the RAM selected must meet all the parameters listed thus far.



TL/F/10488-60

FIGURE B-2. Data Setup Time and Write Pulse Width for Fast Buffered Write of IMEM

TABLE B-2a. Data Setup Times, t_{DS} (ns) for Fast Buffered Write of Instruction Memory

CPU Clock Freq. (MHz)	Wait States n_{IW}			
	0	1	2	3
9.43	88	194	300	406
18.86	35	88	141	194
20.00	32	82	132	182

TABLE B-2b. Write Pulse Width, t_{IW} (ns) for Fast Buffered Write of Instruction Memory

CPU Clock Freq. (MHz)	Wait States n_{IW}			
	0	1	2	3
9.43	96	202	308	414
18.86	43	96	149	202
20.00	40	90	140	190

The MPA-II uses two 55 ns 8K x 8 CMOS Static RAMs for instruction memory. The output enable is tied low and the chip select enables are both enabled. Therefore, the RAMs are always selected. The write enable is the instruction write signal ($\overline{IWR}$) from the BCP. Table B-3 compares the selected instruction memory RAM parameters with required parameters for the DP8344B.

Data Memory Timing

The MPA-II system uses a 100 ns 32K x 8 CMOS Static RAM to implement the system data memory.

The selection of data memory RAM requires the evaluation of several important timing parameters. The RAM access time, strobe width, and data setup times are three of the most critical timing parameters and must all be matched to equivalent BCP timing parameters. The RAM access time should be compared to the data read time of the BCP.

Data read time, t_D, *(Figure B-3)* is measured from when the data address is valid to when data from the RAM is latched into the BCP. Table B-4 gives data read times. The equation for calculating data read time is similar to the one given for instruction read time, and is taken from Table 5-3 (Parameter 14) of the DP8344B data sheet:

$$t_D = (2.5 + \mathrm{MAX}\,(n_{DW}, n_{IW} - 1))T - 40$$

where t_D is the data read time (ns), n_{DW} is the number of data memory wait states, n_{IW} is the number of instruction memory wait states, and T is the T-state time (ns). Since the lower address byte (AD) is externally latched, the latch propagation delay needs to be subtracted from the available read time when determining the required RAM access time.

TABLE B-3. Instruction Memory Read and Write Parameters

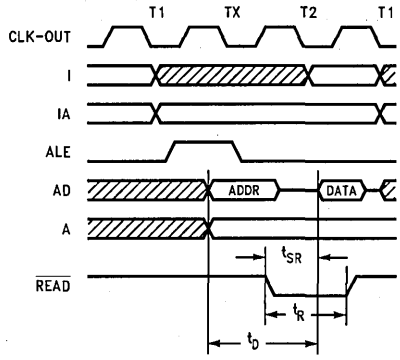
Parameter	Fujitsu RAM (Minimum) (55 ns)	DP8344B BCP (Minimum)					
		Read			Write		
		Full Speed	Half Speed	POR	Full Speed	Half Speed	POR
Access Time (t_I)	55	60.5	140	458			
Write Pulse Width (t_W)	40				43	96	414
Data Setup (t_{DS})	25				35	88	406

Measurements are in ns.

Full Speed is 53 ns T-state with $n_{IW} = 0$ and $n_{DW} = 1$.

Half Speed is 106 ns T-state with $n_{IW} = 0$ and $n_{DW} = 0$.

POR is 106 ns T-state with $n_{IW} = 3$ and $n_{DW} = 7$.



TL/F/10488-61

FIGURE B-3. Data Memory Read Timing

TABLE B-4. Data Read Time, t_D (ns)

CPU Clock Freq. (MHz)	Wait States Max (n_{DW}, n_{IW} − 1)		
	0	1	2
9.43	225	331	437
18.86	92.5	145.5	198.5
20.00	85	135	185

An octal latch (74ALS573) is used in the MPA-II system to demultiplex and latch the address. There is a delay associated with latching of the address and it is dependant on the latch considered. The latch enable is the ALE signal from the BCP. While ALE is high, the outputs follow the inputs. When ALE falls the address is latched on the outputs. The latch has a propagation delay of 20 ns which corresponds to the time it takes for the data on the inputs to reach the outputs.

Therefore, for the MPA-II system the RAM access time is:

$$t_{acc} = t_D - 20 \text{ ns}$$

Using Table B-4, the required RAM access time can be calculated to be:

$$t_{acc} = 145.5 - 20 = 125.5 \text{ ns}$$

for full speed operation with one wait state.

Another important timing parameter is the RAM strobe width. The BCP $\overline{READ}$ and $\overline{WRITE}$ outputs will typically be used to strobe data out of and into the RAM. The signal relationships for a data memory access are shown in *Figure B-3* for a read and in *Figure B-4* for a write. Table B-5 contains $\overline{READ}$ and $\overline{WRITE}$ pulse width values for various clock frequencies and wait state combinations. The equation for calculating $\overline{READ}$ and $\overline{WRITE}$ pulse widths are taken from parameter 8 of Table 5-4 and parameter 12 of Table 5-3 in the DP8344B data sheet:

$$t_R = t_W = (1 + \text{MAX}(n_{DW}, n_{IW} - 1))T - 10$$

where t_W (= t_R) is the pulse width (ns), n_{DW} is the number of data memory wait states and n_{IW} is the number of instruction memory wait states. The RAM chosen should require shorter strobe widths than the pulse width listed in Table B-5 for the desired combination of clock frequency and wait states. Thus, for the MPA-II system, the RAM strobe width must be shorter than 96 ns.

The last important consideration when choosing the data memory RAM is setup times into the BCP on a read and into the RAM on a write. In a typical application, $\overline{READ}$ is connected to the output enable pin on the RAM. When reading from the RAM, the data becomes valid when $\overline{READ}$ falls and activates the RAM outputs. The data must become valid fast enough to meet the setup time required by the BCP. This setup time t_{SR}, as shown in *Figure B-3*, is listed in Table B-6 for various combinations of clock frequencies and wait states. Using Table 5-3 (parameter 7) of the DP8344B datasheet, t_{SR} can be calculated as follows:

$$t_{SR} = (1 + \text{MAX}(n_{DW}, n_{IW} - 1))T - 22$$

where t_{SR} is the maximum time allowed for the data to become valid (ns), n_{DW} is the number of data memory wait states and n_{IW} is the number of instruction memory wait states. The data memory RAM used needs to have a faster output enable time than the time listed in Table B-6 for the desired combination of clock frequency and wait states.

When writing to data memory, the data must be valid in time to meet the setup time requirement of the RAM. In a typical application, this time is measured from the data becoming valid out of the BCP to $\overline{WRITE}$ going high. *Figure B-4* shows this timing relationship, t_{DW}, and Table B-7 contains times for various combinations of clock frequencies and wait states. The equation for calculating this time is from Table 5-4 (parameter 4) of the DP8344B datasheet.

$$t_{DW} = (1 + \text{MAX}(n_{DW}, n_{IW} - 1))T - 20$$

where t_{DW} is the minimum data valid time before $\overline{WRITE}$ rising (ns), n_{DW} is the number of data memory wait states and n_{IW} is the number of instruction memory wait states. This time should be at least as long as the data setup time of the RAM.

TABLE B-5. $\overline{READ}$ and $\overline{WRITE}$ Pulse Width, $t_R = t_W$ (ns)

CPU Clock Freq. (MHz)	Wait States Max (n_{DW}, n_{IW} − 1)		
	0	1	2
9.43	96	202	308
18.86	43	96	149
20.00	40	90	140

TABLE B-6. Data Read Setup Time, t_{SR} (ns)

CPU Clock Freq. (MHz)	Wait States Max (n_{DW}, n_{IW} − 1)		
	0	1	2
9.43	84	190	296
18.86	31	84	137
20.00	28	78	128

TABLE B-7. Data Write Valid Time, t_{DW} (ns)

CPU Clock Freq. (MHz)	Wait States Max (n_{DW}, n_{IW} − 1)		
	0	1	2
9.43	86	192	278
18.86	33	86	139
20.00	30	80	130

Instruction RAM has the greatest affect on execution speed. Each added instruction memory wait state slows the BCP by about 40% as compared to running with no instruction memory wait states. Each added data memory wait state slows a data access by 33% as compared to running with no data memory wait states. RAM costs are coming down, but higher speed RAM still carries a price premium. So there is the trade-off.

Table B-8 compares the BCP data memory requirements with the selected data RAM.

PC System

The MPA-II expansion board is an 8-bit board, which runs in a PC-XT, PC-AT or compatible system. The timings of the two systems have many differences, but the 8 MHz PC-AT bus specifications are more stringent than those of the 4.77 MHz PC-XT bus. So, this evaluation will cover the 8 MHz PC-AT.

The critical timing in this system will be the amount of time the MPA-II will have before it must deassert IO-CHRDY low in order to extend the access cycle by adding wait states. By deasserting IOCHRDY the MPA-II can extend a read or write cycle until the correct data is available or written, respectively.

As stated before, the MPA-II is an 8-bit board so both the I/O and memory cycles will have 8-bit access cycles. In the PC-AT, 8-bit I/O and memory cycles have the exact same timing. There is always one command delay (0.5 T-states) from the time ALE falls until the command strobe ($\overline{IOR}$, $\overline{IOW}$, $\overline{MEMR}$ or $\overline{MEMW}$) goes active low. Four programmed wait states are inserted, forcing the command strobe to stay active low for a minimum of 4.5 T-states. *Figure B-5* shows the relationship between ALE, the command strobes and the bus cycles T-states.

For the following calculations the original IBM PC-AT schematic has been used. This schematic can be found in IBM Technical Reference Personal Computer AT.

In the PC-AT, both ALE and all of the command strobes are controlled by an 82288 bus controller. The command strobes will go active a short delay time after the 82288 inserts the command delay. (The delay time for an 8 MHz 82288 is T (delay 82288) = 25 ns.) After leaving the 82288, $\overline{MEMR}$ and $\overline{MEMW}$ pass through a 74ALS244 before reaching the expansion bus.

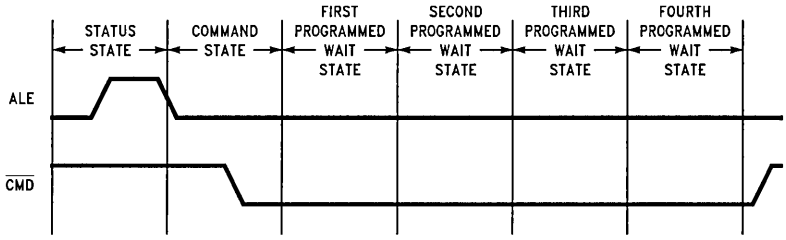
TABLE B-8. Data Memory Read and Write Parameters

Parameter	Hitachi HM62256 RAM (Minimum)	DP8344B BCP (Minimum)			
		Read		Write	
		Full Speed	Half Speed	Full Speed	Half Speed
Access Time (t_{acc})	100	125.5	205		
Write Pulse Width (t_W)	60			96	96
Data Setup (t_{DW})	40			86	86
Output Enable (t_{SR})		84	84		

Measurements are in ns.

Full Speed is 53 ns T-state with $n_{IW} = 0$ and $n_{DW} = 1$.

Half Speed is 106 ns T-state with $n_{IW} = 0$ and $n_{DW} = 0$.



TL/F/10488-63

FIGURE B-5. Relationship of ALE, $\overline{CMD}$, and Bus Timing

So, T(delay 82288) + T(delay 74ALS244) is equal to the maximum amount of time from the end of the command delay until the command strobe reach the MPA-II.

T(strobes valid) = T(delay 82288) + T(delay 74ALS244)
= 25 ns + 10 ns
= 35 ns

In order to add wait states any expansion board must deassert IOCHRDY low in time for it to propagate through a 74ALS32, then through a 74F74 (from preset to output), and then setup to the 82284 by the end of the third programmed wait state (which is also the beginning of the fourth wait state). If the IOCHRDY signal also meets the 82284's hold requirement, then a fifth wait state will be added. Then again, at the end of the fourth wait state if IOCHRDY is still deasserted low a sixth wait state will be added. This will continue until IOCHRDY is asserted high. On the other hand, if IOCHRDY is deasserted too late (i.e. after the end of the third programmed wait state), then the cycle will end without adding any additional wait states.

The following is a calculation of the minimum amount of time before the end of the third wait state that IOCHRDY must be deasserted to add wait states:

T(add wait) = T(delay 74ALS32 H–L) + T(74F74 P–Q) + T(setup 82284)
= 12 ns + 25 ns + 0 ns
37 ns

The maximum amount of time an expansion board has before it must deassert IOCHRDY (to add wait states) from the command strobe being valid is:

T(Max IOCHRDY) = 3.5T − T(strobes) − T(add wait)

where, T = 125 ns in a 8 MHz expansion bus. Therefore,

T(MAX IOCHRDY = 3.5 (125 ns) − 35 ns − 37 ns
= 365.5 ns

This means that the MPA-II has 365.5 ns to deassert IOCHRDY (if wait states are needed) from the time it receives a valid remote access command strobe.

On the MPA-II, the command strobes are buffered by a 20L8B PAL to the BCP's $\overline{\text{REM-RD}}$ and $\overline{\text{REM-WR}}$ inputs. The BCP will respond to a valid remote access by deasserting XACK a delay time after receiving a valid remote access $\overline{\text{REM-RD}}$ or $\overline{\text{REM-WR}}$ strobe. XACK controls IOCHRDY via a 16RA8 PAL.

The maximum delay from receiving a valid remote access command strobe to deasserting IOCHRDY follows:

T(MPA-II IOCHRDY) = T(delay 20L8B) + T(XACK) 1 + T(delay 16RA8)
= 15 ns + 26 ns + 35 ns
= 76 ns

The MPA-II will deassert IOCHRDY a maximum of 76 ns after it receives a valid remote access command strobe. One should notice 76 ns is much less than the maximum allowable time of 365.5 ns.

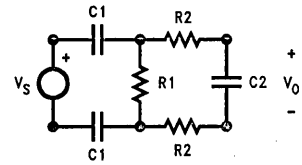
As a final note, the reader should be aware that most faster PC-AT's still run their expansion buses at 8 MHz to remain compatible. This means that the timing on these expansion buses should remain the same as those on any other PC-AT no matter how fast the CPU runs. Thus, the MPA-II will run in all PC-AT's with 8 MHz expansion buses that follow the original 8 MHz PC-AT's expansion bus design. In fact, as can be seen above, the MPA-II will run with bus speeds faster than 8 MHz.

APPENDIX C

Filter Equations

Derivation of Filter Equations for the Combined Coax/Twisted Pair Interface

The basic operation of the filter can be understood by studying the figure below. The actual circuit includes the effects of the terminating resistors, DC isolation capacitors, and the transformer; furthermore, a thorough investigation of bandwidth and gain characteristics should employ the use of a circuit simulator such as SPICE.



TL/F/10468–64

Simple loop analysis yields the following transfer function for the filter:

$$\frac{V_O}{V_S} = \frac{\frac{1}{2R_2C_2}(s)}{s^2 + s\left[\frac{R_1C_1 + C_2(4R_2 + 2R_1)}{2R_1R_2C_1C_2}\right] + \frac{1}{R_1R_2C_1C_2}}$$

If it is assumed $R_1 >> R_2$ and $C_1 >> C_2$, we can then simplify the equation and solve for the poles to obtain the following form:

$$|f| = \frac{\frac{1}{2R_2C_2} \pm \sqrt{\frac{1}{4R_2^2C_2^2} - 4\left(\frac{1}{R_1R_2C_1C_2}\right)}}{4\pi}$$

After splitting the above equation to solve each pole and using a binomial expansion to simplify each pole's equation, we get:

$$f_l \approx \frac{1}{\pi R_1C_1} \approx 20\text{ kHz}$$

(vs. 30 kHz from simulation and testing)

$$f_h \approx \frac{1}{4\pi R_2C_2} \approx 40\text{ MHz}$$

(vs. 30 MHz from simulation and testing)

APPENDIX D

References

DP8344B Biphase Communcations Processor Data Sheet, National Semiconductor, 2900 Semiconductor Drive, P.O. Box 58090, M/S 16-197, Santa Clara, CA 95052-8090, Version 4.2.

DP8344 Biphase Communcations Processor Assembler User Manual (DP8344BSM-M5) National Semiconductor, 2900 Semiconductor Drive, P.O. Box 58090, Santa Clara, CA 95052-8090, February 1988.

DP8344 Biphase Communications Processor Application Notes, National Semiconductor, 2900 Semiconductor Drive, P.O. Box 58090, Santa Clara, CA. 95052-8090.

IBM PC 3270 Emulation Program Entry Level Version 1.10 (84X0280), International Business Machines Corporation, Department 52Q, Neighborhood Read, Kingston, NY 12401, 1981.

IRMA™ User's Manual (40-97910-001), Digital Communications Associates, Inc., 1000 Alderman Drive, Alpharetta, GA, 30201.

Smart Alec™ User's Guide (40-98100-007), Digital Communcations Associates, Inc., 1000 Alderman Drive, Alpharetta, GA 30201.

Guide to Operations Personal Computer AT (1502241), International Business Machines Corporation, P.O. Box 1328-C, Boca Raton, FL 33432, 1984.

Guide to Operations Personal Computer XT (6936810), International Business Machines Corporation, P.O. Box 1328-C, Boca Raton, FL 33432, 1983.

IBM 3270 Connection Technical Reference (GA23-0339-02), Information Development, Department 802, P.O. Box 12195, Research Triangle Park, NC 27709, 1988.

IBM 3174/3274 Control Unit to Device Product Attachment Information, International Business Machines Corporation, Armonk, NY 10504, October 1986.

IBM 3274 Control Unit to Distributed Function Device Product Attachment Information, International Business Machines Corporation, Armonk, NY 10504, June 1985.

5250 Information Display System to System/36, System/38, and Applications System/400, System Units Product Attachment Information, International Business Machines Corporation, Armonk, NY 10504, October 1988.

Technical Reference Personal Computer AT (502243), International Business Machines Corporation, P.O. Box 1328-C, Boca Raton, FL 33432, 1984.

Technical Reference Personal Computer XT (6936808), International Business Machines Corporation, P.O. Box 1328-C, Boca Raton, FL 33432, 1983.

abel™(880004), Data I/O Corporation, 10525 Willows Road NE, P.O. Box 97046, Redmond, WA 93073-9746, 1988.

BRIEF™ User's Guide, Underware, Inc., 84 Gainborough St., Suite 103W, Boston, MA 02115, June 1987.

BSID, Capstone Technology, 47354 Fremont Blvd., Fremont, CA 94538.

DP8344 BCP Demonstration/Development Kit, Capstone Technology, 47354 Fremont Blvd., Fremont, CA 94538.

Microsystem Components Handbook—Volume I (230843-002), Intel Corporation, Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

8088 Microprocessor Data Sheet
8288 Bus Controller Data Sheet
80286 Microprocessor Data Sheet
82288 Bus Controller Data Sheet
8284 Clock Generator and Driver Data Sheet

iAPX 86/88, 186/188 User's Manual Hardware Reference 1985 (210912-001), Intel Corporation, Literature Distribution, Mail Stop SC6-714, 3065 Bowers Avenue, Santa Clara, CA 95051.

iAPX286 Hardware Reference Manual 1983 (210760-001), Intel Corporation, Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

S/LS/TTI Logic Data Book (1985—400050), National Semiconductor, 2900 Semiconductor Drive, Santa Clara, CA 95051.

Contacts

For further information on the MPA-II or the DP8344 BCP contact:

BCP Product Marketing
National Semiconductor
2900 Semiconductor Drive
Mail Stop: D3800
Santa Clara, CA 95052-8090
Phone: (408) 721-5000

For Technical Information on the MPA-II or the DP8344 BCP contact:

DATACOM Applications Support
National Semiconductor
1111 W. Bardin Road
Mail Stop: A2190
Arlington, TX 76017
Phone: (817) 468-6676
Fax: (817) 468-1468

For requesting IBM Product Attachment Information manuals (PAI's) contact:

Industry Relations Dept.
IBM
2000 Purchase Street
Purchase, New York 10577
Phone: (914) 697-7227

For ordering IBM manuals other than PAI's contact your local IBM Sales Office.

For ordering products from Azure Technologies contact:

Azure Technologies, Inc.
38 Pond Street
Franklin Massachusetts 02038
Phone: (508) 520-3800
Fax: (508) 528-4518

For ordering products from Capstone Technology contact:

Richard L. Drolet
Capstone Technology
47354 Fremont Blvd.
Fremont, CA 94538
Phone: (510) 438-3500

For ordering products from Hewlett Packard contact your local Hewlett Packard Sales Office:

For ordering products from DCA contact:

Digital Communications Associates, Inc.
1000 Alderman Drive
Alpharetta, Georgia 30201
Phone: (404) 740-0300

For ordering products from Simware, such as SimPC Master, contact:

Simware, Inc.
20 Colonnade Road
Ottawa, Ontario
Canada K2E 7M6
Phone: (613) 727-1779
Fax: (613) 727-9409

For ordering products from Relay Communications, such as Relay Gold, contact:

Relay Communications, Inc.
41 Kenosia Avenue
Danbury, CT 06810
Phone: (800) 222-8672

For ordering products from Fischer International Systems, such as Xeus, contact:

Fischer International Systems Corp.
P.O. Box 9107
4073 Merchantile Avenue
Naples, Florida 33942
Phone: (813) 643-1500

A Combined Coax-Twisted Pair 3270 Line Interface for the DP8344 Biphase Communications Processor

National Semiconductor
Application Note 624
Tim Davis and David Weinman

This paper will discuss the design of an improved 3270 transceiver interface for the National Semiconductor DP8344 combining increased error-free performance and the ability to communicate over both coax and twisted pair transmission lines. At this date, the largest installed base of terminals is the 3270 protocol terminal which primarily utilizes coax cabling. Because of phone wire's easy accessibility and lower cost, twisted pair cabling has become popular among end users for new terminal installations. In the past, baluns have been used to augment existing coax interfaces, but their poor performance and cost considerations leave designers seeking new solutions. In addition, the integration of coax and twisted pair on the same board has become a market requirement, but this is a considerable design challenge. A brief summary of the interface concepts, a discussion of the proposed design, and a description of the results are included in this application note.

CONCEPTS

Coax cable is normally driven on the center conductor with the shield grounded. Conversely, unshielded twisted pair cable is driven on both lines. Because of the way that each is driven, coax operation is often called unbalanced and twisted pair operation balanced.

Transmission line characteristics of coax and twisted pair cables can be envisioned as essentially those of a low-pass filter with a length-dependent bandwidth.[1] In 3270 systems, different data combinations generate dissimilar transmission frequencies because of the Manchester format.[2] These two factors combine to produce data pulse widths that vary according to the data transmitted and the length and type of cable used. This pulse-width variation is often described as "data jitter."[3]

In addition to line filtering, noise can cause jitter. Coax cable employs a shield to isolate the signal from external noise. Electromagnetically balanced lines minimize differential noise in unshielded twisted pair cable. In other words, the twisted pair wires are theoretically equidistant from any noise source, and all noise superimposed on the signal should be the common-mode type. Although these methods diminish most noise, they are not totally effective, and environmental interference from other nearby wiring and circuitry may still cause problems.

Besides the effects of jitter, reflections can produce undesirable signal characteristics that introduce errors. These reflections may be caused by cable discontinuities, connectors, or improper driver and receiver matching. Signal edge rates may aggravate reflection problems since faster edges tend to produce reflections that may dramatically distort the signal.[3] Most reflection difficulties occur over short cable (less than 150 ft.) because at these distances reflections suffer little attenuation and can significantly distort the signal. Since the timing of the reflections is a function of cable length, it may be possible to operate at some short distance and not at some greater length.

An effective receiver design must address each of the above concerns. To counteract the effects of line filtering and noise, there must be a large amount of jitter tolerance. Some filtering is needed to reduce the effects of environmental noise caused by terminals, computers, and other proximate circuitry. At the same time, such filtering must not introduce transients that the receiver comparator translates into data jitter.

Like the receiver design, a successful driver design should compensate for the filtering effects of the cable. As cable length is increased, higher data frequencies become attenuated more than lower frequency signals, yielding greater disparity in the amplitudes of these signals.[4] This effect generates greater jitter at the receiver. The 3270 signal format allows for a high voltage (predistorted) magnitude followed by a low voltage (nondistorted) magnitude within each data half-bit time.[2] Increasing the predistorted-to-nondistorted signal level ratio counteracts the filtering phenomenon because the lower frequency signals contain less predistortion than do higher frequency signals. Thus, the amplitude of the higher frequency components are greater than the lower frequency components at the transmitter. Implementation of this compensation technique is limited because nondistorted signal levels are more susceptible to reflection-induced errors at short cable lengths. Consequently, proper impedance matching and slower edge rates must be utilized to eliminate as much reflection as possible at these lengths.

Besides improved performance, both unbalanced and balanced operation must be adequately supported. Electromagnetic isolation for coaxial cabling can be provided by a properly grounded shield. Electrically and geometrically symmetric lines must be maintained for twisted pair operation. For both cable types, proper termination should be employed, although terminations slightly greater than the characteristic impedance of the line may actually provide a larger received signal with insignificant reflection.[3] In the board layout, the comparator traces should be as short as possible. Lines should be placed close together along their entire path to avoid the introduction of differential noise. These traces should not pass near high frequency lines and should be isolated by a ground plane.

BCP LINE INTERFACE DESIGN

An extensive characterization of the BCP comparator was done to facilitate this interface design. The proposed design enhances some of the BCP transceiver's characteristics and incorporates the aforementioned suggestions.

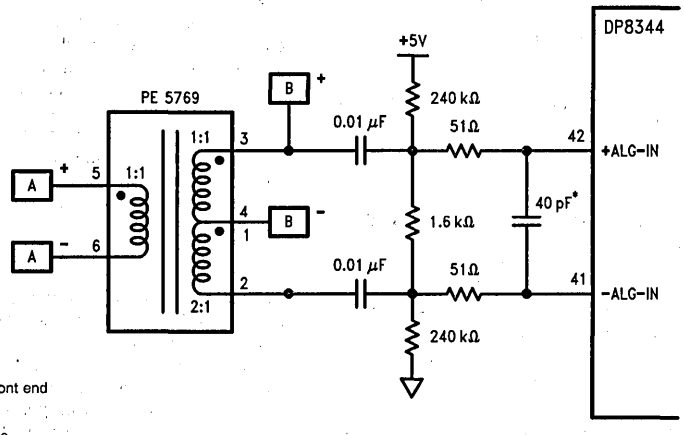
The interface design takes into account the common comparator attributes of power supply rejection, variable switching offset, finite voltage sensitivity, and fast edge rate sensitivity. V_{CC} noise can affect the comparator output when the inputs are biased to the same voltage. This particular type of biasing may render portions of the comparator susceptible to supply noise. Variable switching offset and finite voltage sensitivity cause the receiver decoding circuitry to see a

substantial amount of data jitter when signal amplitudes approach the sensitivity limits of the comparator. At these signal magnitudes, considerable variation in the output of the comparator is observed. Finally, edge sensitivity may allow a fast edge to introduce errors as charge is coupled through the inputs during a rapid predistorted-to-nondistorted level transition, especially as the nondistorted level is reduced in magnitude.

The receiver interface design *(Figure 1)* addresses each of the BCP comparator's characteristics. A small offset (about 17 mV) separates the inputs to eliminate V_{CC}-coupled noise. This offet is relatively large compared to possible fabrication variations, resulting in a more consistent, device-independent operation. The offset has the added benefit of making the comparator more immune to ambient noise that may be present on the circuit board. A 2:1:1 transformer (arranged as a 3:1) restores any voltage sensitivity lost by introducing the offset. A bandpass filter is employed to reduce the edge rate of the signal at the comparator and to eliminate environmental noise. The bandwidth (30 kHz to 30 MHz) was chosen to provide sufficient noise attenuation while producing minimum data jitter. Refer to Appendix 2 for a derivation of the filter equations.

Like many present 3270 circuits, the driver design *(Figure 2)* utilizes a National Semiconductor DS3487 and a resistor network to generate the proper signal levels. The predistorted-to-nondistorted ratio was chosen to be about 3 to 1. This ratio was observed to offer good noise immunity at short cable lengths (less than 150 feet) and error-free transmission to an IBM 3174 controller at long cable lengths (greater than 5000 feet).

To allow for two interfaces in the same circuit design, the coax/twisted pair front end *(Figure 3)* includes an ADC Telecommunications brand TPC connector to switch between coax and twisted pair cable. This connector allows different male connectors for coax and twisted pair cable to switch in different interfaces for the particular cable type. The coax interface has only the shield capacitively coupled to ground. The 510Ω resistor and the filter loading produce a termination of about 95Ω. The twisted pair interface balances both lines and possesses an input impedance of about 100Ω. This termination is somewhat higher than the characteristic impedance (about 96Ω) of twisted pair. Terminations of this type produce reflections that do not tend to generate mid-bit errors, as well as having the benefit of creating a larger voltage at the receiver over longer cable lengths.



Legend

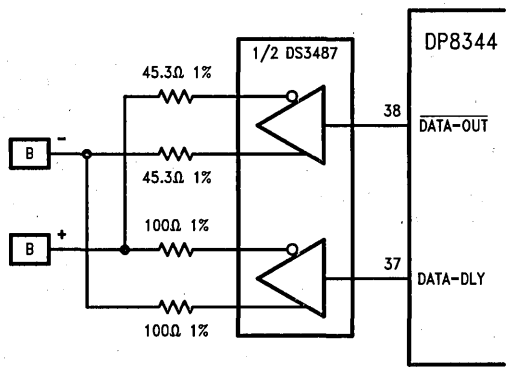
Ⓐ To coax/twisted pair front end

Ⓑ To line driver circuitry

*Includes board capacitance

TL/F/10448-1

FIGURE 1. BCP Receiver Filter Design

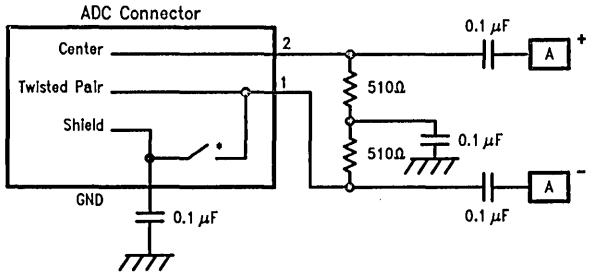


Legend

Ⓑ To 2:1:1 Transformer

TL/F/10448-2

FIGURE 2. BCP Driver Design



Legend

Ⓐ To 2:1:1 Transformer

TL/F/10448-3

*Connector closes switch for coax and opens switch for twisted pair.

FIGURE 3. BCP Coax/Twisted Pair Front End

RESULTS AND COMPARISONS

The evaluation involved producing multiple data transfers between an IBM 3174-81R and the device under test during a live 3270 session. The preferred method of testing would be to transfer extremely large files to the host. Since terminals and muxes cannot transfer files and all devices being tested needed to be evaluated under similar conditions, a screen-oriented approach was taken for testing. The screen-oriented approach involved using common methods for forcing the controller to send an entire screen of characters to the device. Procedural specifics are included in Appendix 1.

Performance of the BCP interface typically extended over 7000 feet of RG62A/U coax and 1700 feet of AT&T DIW 4 pair/24 AWG unshielded twisted pair. This operation met or exceeded many of the current 3270 solutions. The performance of other 3270 products was obtained from production stock of competitors' equipment and should be taken as typical operation. Although these long distances are possible, it is recommended that companies specify their products to IBM's PAI^2 specifications of 5000 feet of coax cable. The extra long distance capability of the new interface will assure the designer a comfortable guardband of performance. Similarly, a 50% margin on the unshielded twisted pair capability will give approximately a 900 foot specification.

It should be noted that the BCP receiver detects errors before the controller does. This is because of comparator skew, a mechanism that occurs when the amplitude of the signal approaches the sensitivity of the comparator. At these small levels, propagation symmetry for high-to-low and low-to-high transitions is lost. The failure mechanisms of competitors include insufficient receiver jitter tolerance, filter transients, and comparator skew. Operational distance may be extended by the utilization of transformers with higher turn ratios as long as considerations are taken for impedance matching, driver loading, and component quality tolerances (higher turn ratios may demand circuits with very low tolerance percentages).

There are also economical advantages in using the BCP comparator. The number of active and passive components required to build the line interface is small compared to competing solutions. The proposed design is extremely cost competitive with current media solutions.

CONCLUSION

An effective and economical 3270 interface solution has been demonstrated using only passive components and a line driver. Guidelines have also been suggested to facilitate the design and layout of such an interface. Criteria concerning board layout and noise suppression must be considered to be at least as important as the components themselves; for example, adjustments should be made for variations in board capacitances and inductances. With only slight modification of the components given for this design, it is thus likely that optimum performance can be obtained for a specific layout. Implementation of these design principles should prove advantageous for the development of an efficient and competitive 3270 line interface.

REFERENCES

1. H.P. Neff, Jr., *Basic Electromagnetic Fields,* New York: Harper and Row, 1981, Chapter 13.

2. *IBM 3174/3274 Control Unit to Device Product Attachment Information,* Communication Products Information Development, International Business Machines Corporation, Research Triangle Park, NC, October 1986.

3. K.M. True, *The Interface Handbook: Line Drivers and Receivers,* Semiconductor Components Group, Fairchild Camera and Instrument Corporation, Mountain View, CA, 1975. Chapters 3 and 4.

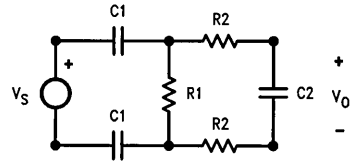
4. N.S. Nahman, "A Discussion on the Transient Analysis of Coaxial Cables Considering High Frequency Losses," *IRE Trans. Circuit Theory,* vol. CT-9, pp. 144–152, June 1962.

APPENDIX 1: TEST PROCEDURE FOR LONG AND SHORT DISTANCE TESTING

1. Enter Test mode on the 3174 controller.
2. Clear the error counters.
3. Hit the Clear key rapidly 30 times. This will repaint the screen with the test menu very rapidly. This is a quick and easy method to cause an entire screen of characters to be sent to either an emulation card or a terminal over the coax.
4. Exit Test mode.
5. LOGON to a session on the host.
6. Issue the FILELIST command.
7. Hit the Clear key 20 times. After the controller clears the screen, it will repaint the FILELIST menu each time. This will again cause an entire screen of characters to be sent over the coax to the device under test.
8. XEDIT a 40k file text file.
9. Page through the entire file forwards once, then backwards once. Again, this will cause a varied stream of transmissions to be sent to the device under test.
10. LOGOFF the session.
11. Enter Test mode again.
12. Check for errors on the error test screen.

APPENDIX 2: DERIVATION OF FILTER EQUATIONS

The basic operation of the filter can be understood by studying the figure below. The actual circuit includes the effects of the terminating resistors, DC isolation capacitors, and the transformer; furthermore, a thorough investigation of bandwidth and gain characteristics should employ the use of a circuit simulator such as SPICE.



TL/F/10448–4

Simple loop analysis yields the following transfer function for the filter:

$$\frac{V_O}{V_S} = \frac{\dfrac{1}{2\,R2\,C2}(S)}{S^2 + S\left[\dfrac{R1\,C1 + C2\,(4\,R2 + 2\,R1)}{2\,R1\,R2\,C1\,C2}\right] + \dfrac{1}{R1\,R2\,C1\,C2}}$$

If it is assumed R1 >> R2 and C1 >> C2, we can then simplify the equation and solve for the poles to obtain the following form:

$$|f| = \frac{\dfrac{1}{2\,R2\,C2} \pm \sqrt{\dfrac{1}{4R2^2\,C2^2} - 4\left(\dfrac{1}{R1\,R2\,C1\,C2}\right)}}{4\pi}$$

After splitting the above equation to solve each pole and using a binomial expansion to simplify each pole's equation, we get:

$$f_l \approx \frac{1}{\pi R1\,C1} \approx 20\text{ kHz}$$

(vs. 30 kHz from simulation and testing)

$$f_h \approx \frac{1}{4\pi\,R2\,C2} \approx 40\text{ MHz}$$

(vs. 30 MHz from simulation and testing)

Interfacing Memory to the DP8344B

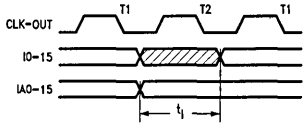
National Semiconductor
Application Note 623
Bill Fisher,
Mark Koether

As with most other aspects of a design, choosing memory is a cost vs. performance trade off. Maximum performance is achieved running no wait-states with fast, expensive memory. Slower, less expensive memory can be used, but wait-states must be added, slowing down the BCP. Therefore one needs to choose the slowest memory possible while still meeting design specifications. While this article assumes RAM is used for instruction and data memory, the information is relevant to memory devices in general.

The BCP needs separate data and instruction RAM, each with their own requirements. Instruction read time is the major constraint when choosing instruction RAM. Instruction read time, t_I, as shown in *Figure 1*, is measured from when the instruction address becomes valid to when the next instruction is latched into the BCP. Instruction read time for various clock frequencies and wait states are given in Table I. Clock frequency and wait state combinations other than those given in the table can be calculated by the following equation:

$$t_I = 10^3 (1 + \frac{T_L}{T} + n_{IW})/f_{CPU} - 19$$

where t_I is the instruction read time (ns), n_{IW} is the number of instruction memory wait states, T_L is the CPU clock low pulse width (ns), T is the CPU clock period (ns), and f_{CPU} is the clock frequency (MHz) at which the CPU is running. The RAM chosen needs to have a faster access time than the read time for the desired combination of clock frequency and wait states. However, instruction read time is not the only timing consideration when choosing instruction RAM. If the BCP is used in an application which requires full speed softloading of instruction RAM, there are two other timing relationships which require evaluation. These are data setup time and write pulse width. The relevant BCP timing parameters are I valid before $\overline{IWR}$ rising, $t_{PD\text{-}I\text{-}IWR}$, and $\overline{IWR}$ low time, $t_{W\text{-}IWR}$. The value of these timing parameters depends on the Remote Interface mode of operation. More detailed information can be found in the Device Specifications and the Remote Interface and Arbitration System sections of the BCP data manual. Note that in a typical application of the BCP, softloading occurs after reset with the BCP operating with CLK/2 and full wait states. Under these conditions the instruction read time value is the critical parameter for choosing the instruction RAM.



TL/F/10447-1

FIGURE 1. Instruction Read Time

TABLE I. Instruction Read Times, t_I (ns)

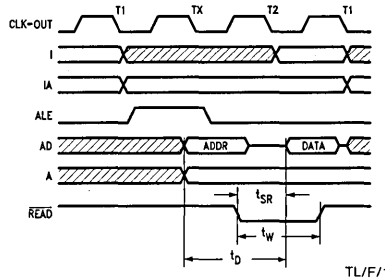
CPU Clock Freq. (MHz) ($T_L/T = 0.5$)	Wait States		
	0	1	2
9.43	140	246	352
18.86	60	113	166
20.00	56	106	156

The selection of data memory RAM requires the evaluation of several important timing parameters. The RAM access time, strobe width, and data setup times are three of the most critical timing parameters and must all be matched to equivalent BCP timing parameters. The RAM access time should be compared to the data read time of the BCP. The following discussion assumes 3 T-state data memory read timing ([4TR] = 0). However, the basic approach is applicable to the less critical 4 T-state data memory read timing. Detailed information on this mode can be found in the CPU Description and the Device Specifications sections of the BCP data manual.

Data read time, t_D, *(Figure 2)* is measured from when the data address is valid to when data from the RAM is latched into the BCP. Table II gives data read times. The equation for calculating data read time is similar to the one given for instruction read time:

$$t_p = 10^3 (2 + \frac{T_L}{T} + \text{MAX}(n_{DW}, n_{IW} - 1))/f_{CPU} - 40$$

where t_D is the data read time (ns), n_{DW} is the number of data memory wait states, n_{IW} is the number of instruction memory wait states, T_L is the CPU clock low pulse width (ns), T is the CPU clock period (ns), and f_{CPU} is the clock frequency (MHz) at which the CPU is running. Since the lower address byte (AD) is externally latched, the latch propagation delay needs to be subtracted from the available read time when determining the required RAM access time.



TL/F/10447-2

FIGURE 2. Data Memory Read Timing

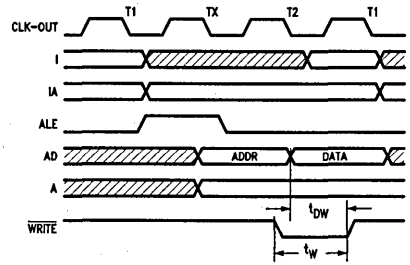
TABLE II. Data Read Time, t_D (ns)

CPU Clock Freq. (MHz) ($T_L/T = 0.5$)	Wait States MAX(n_{DW}, n_{IW} − 1)		
	0	1	2
9.43	225	331	437
18.86	92	145	198
20.00	85	135	185

Another important timing parameter is the RAM strobe width. The BCP $\overline{READ}$ and $\overline{WRITE}$ outputs will typically be used to strobe data out of and into the RAM. The signal relationships for a data memory access are shown in *Figure 2* for a read and in *Figure 3* for a write. Table III contains $\overline{READ}$ and $\overline{WRITE}$ pulse width values for various clock frequencies and wait state combinations. The equation for calculating $\overline{READ}$ and $\overline{WRITE}$ pulse width is:

$$t_W = 10^3(1 + \text{MAX}(n_{DW}, n_{IW} - 1)/f_{CPU} - 10$$

where t_W is the pulse width (ns), n_{DW} is the number of data memory wait states, n_{IW} is the number of instruction memory wait states, and f_{CPU} is the clock frequency (MHz) at which the CPU is running. The RAM chosen should require shorter strobe widths than the pulse width listed in Table III for the desired combination of clock frequency and wait states.



TL/F/10447-3

FIGURE 3. Data Memory Write Timing

TABLE III. $\overline{READ}$ and $\overline{WRITE}$ Pulse Width, t_W (ns)

CPU Clock Freq. (MHz)	Wait States MAX(n_{DW}, n_{IW} − 1)		
	0	1	2
9.43	96	202	308
18.86	43	96	149
20.00	40	90	140

The last important consideration when choosing the data memory RAM is setup times into the BCP on a read and into the RAM on a write. In a typical application, $\overline{READ}$ is connected to the output enable pin on the RAM. When reading from the RAM, the data becomes valid when $\overline{READ}$ falls and activates the RAM outputs. The data must become valid fast enough to meet the setup time required by the BCP. This setup time t_{SR}, as shown in *Figure 2*, is listed in Table IV for various combinations of clock frequencies and wait states. It can be calculated from the following equation:

$$t_{SR} = 10^3(1 + MAX(n_{DW}, n_{IW} - 1)/f_{CPU} - 22$$

where t_{SR} is the maximum time allowed for the data to become valid (ns), n_{DW} is the number of data memory wait states, n_{IW} is the number of instruction memory wait states, and f_{CPU} is the clock frequency (MHz) at which the CPU is running. The data memory RAM used needs to have a faster output enable time than the time listed in Table IV for the desired combination of clock frequency and wait states.

TABLE IV. Data Read Setup Time, t_{SR} (ns)

CPU Clock Freq. (MHz)	Wait States MAX(n_{DW}, n_{IW} − 1)		
	0	1	2
9.43	84	190	296
18.86	31	84	137
20.00	28	78	128

When writing to data memory, the data must be valid in time to meet the setup time requirement of the RAM. In a typical application, this time is measured from the data becoming valid out of the BCP to $\overline{WRITE}$ going high. *Figure 3* shows this timing relationship, t_{DW}, and Table V contains times for various combinations of clock frequencies and wait states. The equation for calculating this time is:

$$t_{DW} = 10^3(1 + MAX(n_{DW}, n_{IW} - 1))/f_{CPU} - 20$$

where t_{DW} is the minimum data valid time before $\overline{WRITE}$ rising (ns), n_{DW} is the number of data memory wait states, n_{IW} is the number of instruction memory wait states, and f_{CPU} is the clock frequency (MHz) at which the CPU is running. This time should be at least as long as the data setup time of the RAM.

TABLE V. Data Write Valid Time, t_{DW} (ns)

CPU Clock Freq. (MHz)	Wait States MAX(n_{DW}, n_{IW} − 1)		
	0	1	2
9.43	94	200	306
18.86	41	94	147
20.00	30	80	130

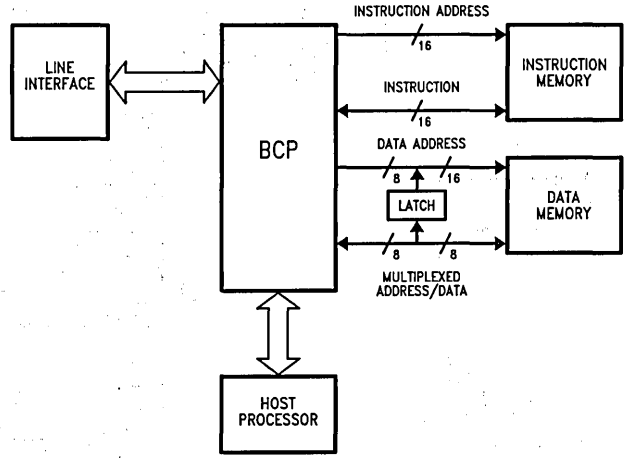
Instruction RAM has the greatest effect on execution speed. Each added instruction memory wait state slows the BCP by about 40% as compared to running with no instruction memory wait states. Each added data memory wait state slows a data access by 33% as compared to running with no data memory wait states. RAM costs are coming down, but higher speed RAM still carries a price premium. So there is the trade-off.

DP8344 BCP Stand-Alone Soft-Load System

National Semiconductor
Application Note 504
Jim Margeson

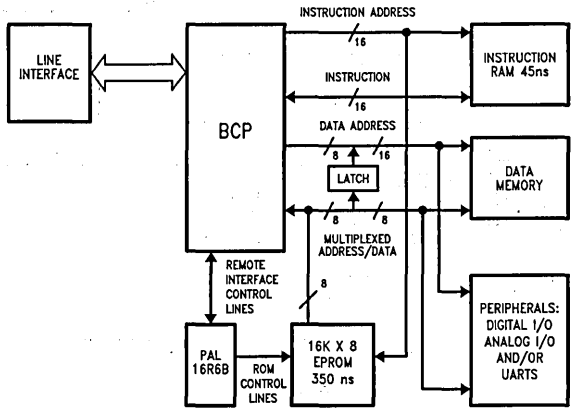
INTRODUCTION

The DP8344 Biphase Communications Processor (BCP) is a 20 MHz Harvard architecture microprocessor with an on-chip transmitter and receiver. The BCP can be used to implement several biphase communication protocols: IBM 3270, IBM 3299, IBM 5250, and National's general purpose 8-bit protocol. This application note shows how DP8344 software can be loaded from EPROM into instruction RAM. It is particularly valuable in stand-alone systems where the BCP is not interfaced to a host processor. Possible applications include: protocol converters, multiplexers, high-speed remote data acquisition systems and remote process control systems.



TL/F/9403-1

FIGURE 1. BCP System with Host Processor



TL/F/9403-2

FIGURE 2. BCP Stand-Alone System with EPROM Soft Load Circuit

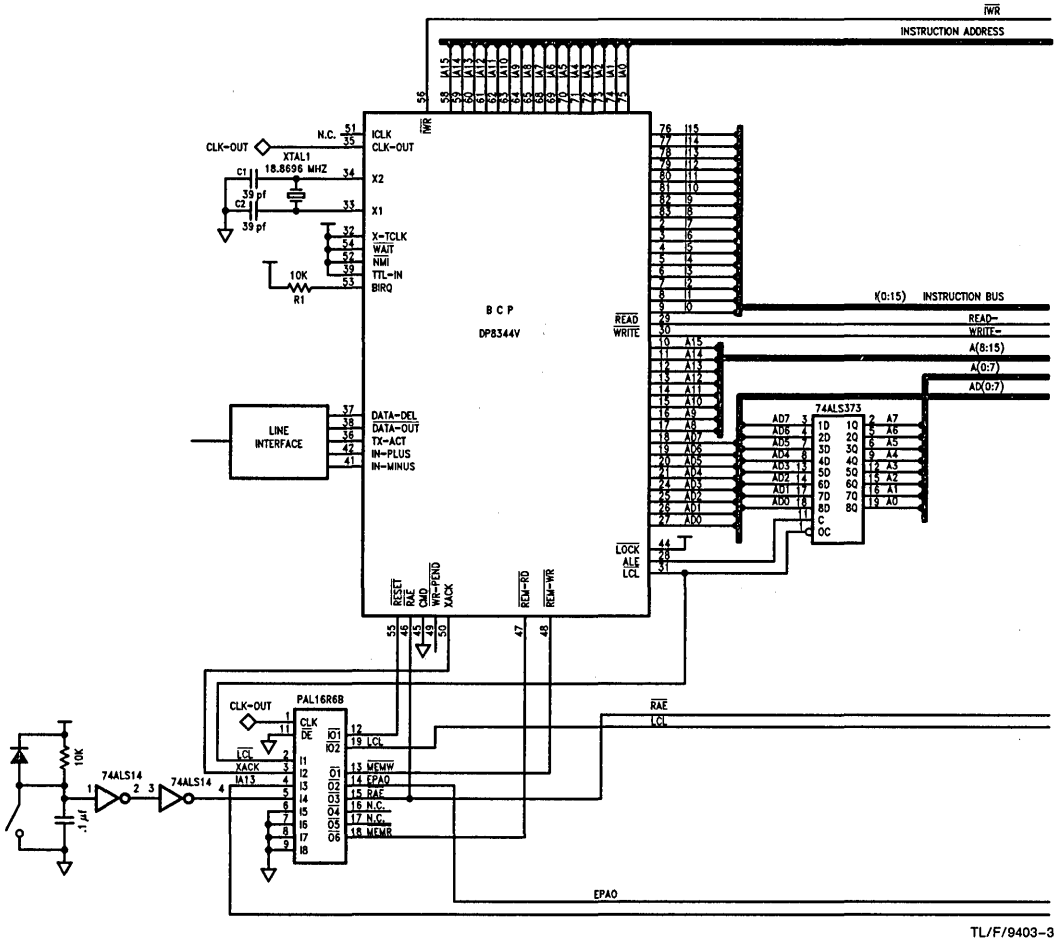
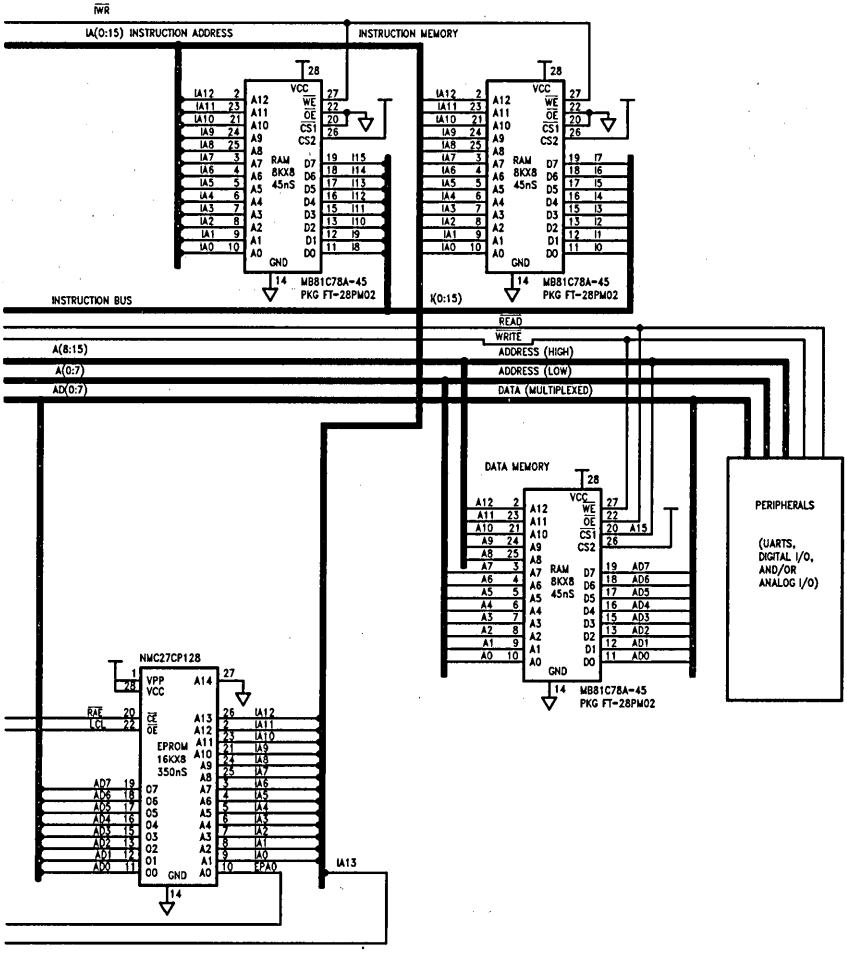


FIGURE 3. Schematic



TL/F/9403-4

FIGURE 3. Schematic (Continued)

WHY EPROM SOFT-LOAD?

In a stand-alone application, the BCP instruction code must be kept in non-volatile memory. Instruction memory with 45 ns access time is required to run the BCP at full speed.

EPROM at this speed can be quite expensive, much more than 45 ns RAM or 350 ns EPROM. RAM with 45 ns access time can be used for instruction memory if a scheme is employed to load the BCP code into the RAM from slow (350 ns), inexpensive EPROM, upon power-up.

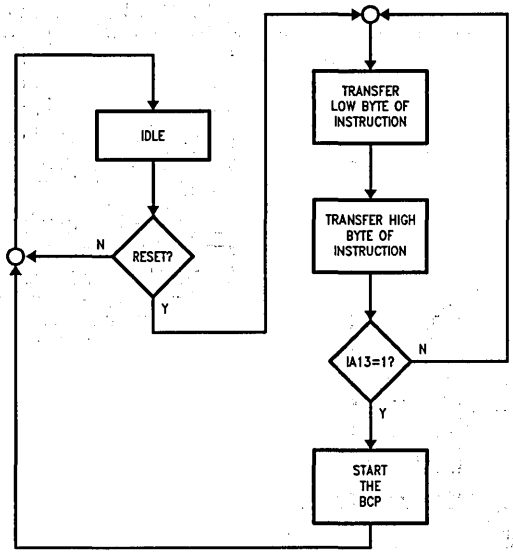
In non-stand-alone applications, a host processor would communicate with the BCP through the BCP's built-in remote interface *(Figure 1)*. In such a system, BCP code would be loaded from the host into the BCP's instruction RAM using the remote interface. In a stand-alone system, however, the BCP is not interfaced to a host; the program is loaded from EPROM through the remote interface. As shown in *Figure 2* a PAL® sequencer controls the loading of the program, generating handshaking signals similar to those of a typical host processor. When the load is complete, the sequencer tells the BCP to begin execution of the program.

HOW THE SOFT-LOAD CIRCUIT WORKS

The BCP, as configured in this system, comes up halted after reset *(Figure 3)*. The program counter is set to zero, and the remote interface is configured to receive 16-bit instructions in 8-bit pieces and write them into instruction memory. The BCP has the feature that it can be configured to come up stopped or to begin program execution after a reset has occurred. If the following conditions are true when reset is de-asserted then the processor will begin running: RAE~ (Remote Access Enable, active low) = High, REMWR~ (Remote Write, active low) = low, REMRD~ (Remote Read, active low) = low. Otherwise, it will come up halted.

The PAL sequencer begins the software load by writing the low byte of the first instruction to the remote interface. A simplified flowchart of the sequence operation is shown in *Figure 4*.

This byte comes from address 0000H of the EPROM. The corresponding locations of EPROM and RAM are shown in *Figure 5*. The least significant address line of the EPROM is controlled by the sequencer; the other address lines are driven by the instruction address bus of the BCP. The instruction address bus reflects the contents of the BCP's program counter (PC), which contains the destination of the instruction currently being loaded. After the low byte of the first instruction is written to the remote interface, the sequencer brings the least significant address line of the EPROM high. Now location 0001H of the EPROM is addressed, and the high byte of the first instruction is written to the remote interface. At this point the BCP writes both bytes into address 0000H of instruction RAM, and increments its program counter.



TL/F/9403-5

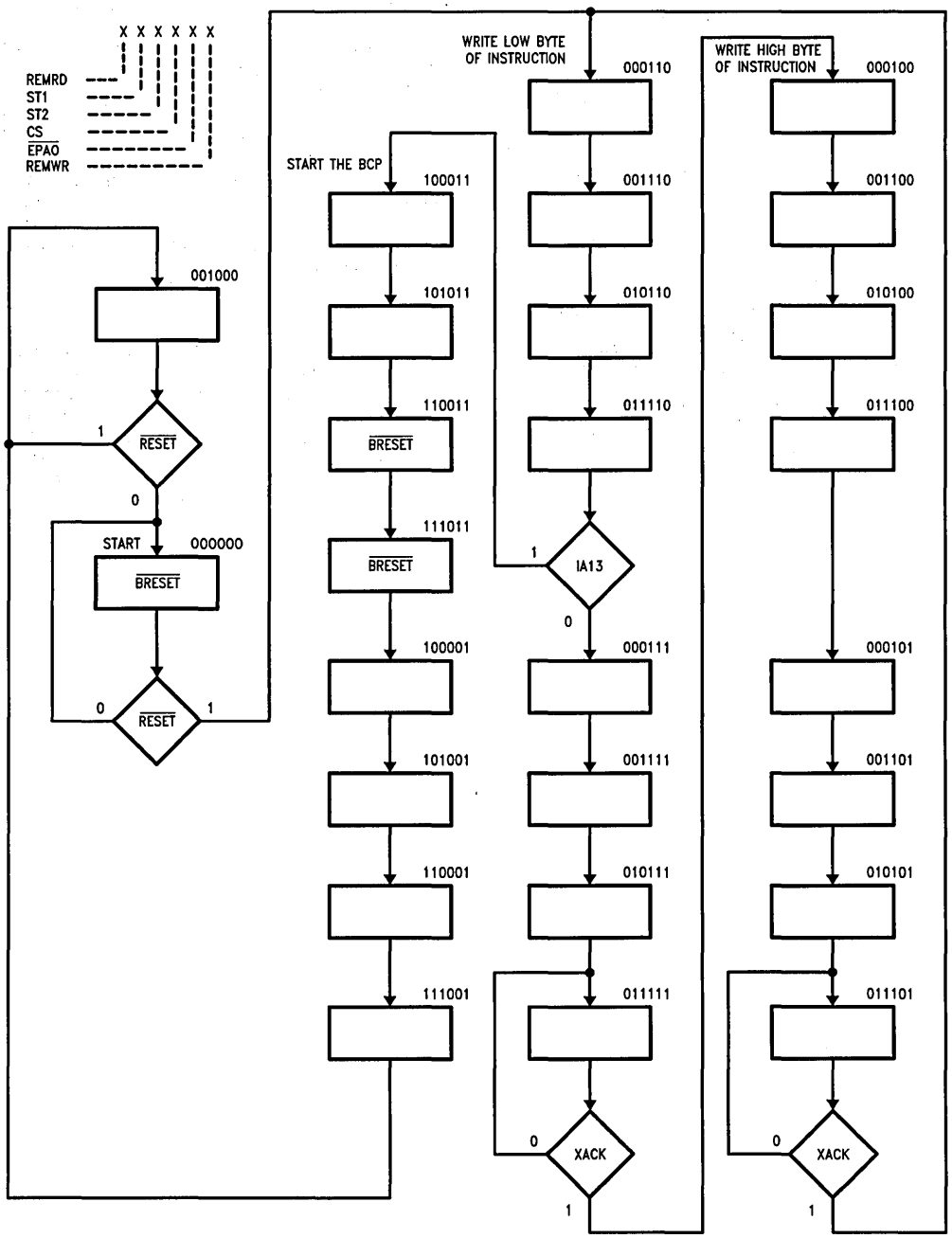
FIGURE 4. Sequencer Operation

EPROM Address	Instruction Memory Address	
0	0	(Low Byte)
1	0	(High Byte)
2	1	(Low Byte)
3	1	(High Byte)
4	2	(Low Byte)
5	2	(High Byte)
•	•	•
•	•	•
•	•	•
•	•	•
16382	8190	(Low Byte)
16383	8191	(High Byte)

FIGURE 5. EPROM to RAM Address Mapping

The first 16-bit instruction has been transferred; the second is done in a similar manner. The sequencer brings the least significant address line of the EPROM low again. The PC now contains 0001H, which is output on the instruction address bus. Location 0002H of the EPROM is addressed, and the low byte of the second instruction is written to the remote interface. The sequencer then brings the least significant address line of the EPROM high (to address location 0003H) and the high byte of the second instruction is transferred. The BCP writes the second 16-bit instruction to location 0001H of instruction RAM. This process is repeated until the last instruction is transferred.

The sequencer senses that the load is complete when instruction address line 13 comes high. This occurs when the program counter is incremented to a value of 4000H, indicating that 8K instruction words have been transferred. At this point the BCP must be started. To achieve this, the sequencer resets the BCP again, while holding RAE~ high, REMRD~ low, and REMWR~ low. A reset during these conditions brings the processor up running, and also clears the program counter. The BCP begins execution at instruction address 0000H and the sequencer and EPROM go into an inactive state, transparent to the software being executed. A detailed version of the sequencer flowchart is shown in *Figure 6.* A hardware compiler/minimizer was used to obtain the equations shown in *Figure 7.* These equations were used to program a National PAL16R6B. Typical timing waveforms of the soft-load are shown in *Figure 8.*



TL/F/9403-8

FIGURE 6. Sequencer Flowchart

There are several advantages to using the remote interface to load the BCP software. If a scheme like the one in *Figure 9* was used to load the program directly from EPROM to instruction RAM, much more hardware would be required and the access time of the RAM would need to be shorter. Two EPROMs would have to be used instead of one because the transfer would be 16 bits wide instead of 8 bits. In this case the BCP's program counter could not be used to increment through the memory locations, thus an external 13-bit counter would be needed. TRI-STATE® buffers would isolate the RAM and EPROM from the instruction data and instruction address busses during soft-load. These buffers would add propagation delays to memory accesses demanding that faster RAM be used. Soft-loading through the remote interface requires fewer I.C.'s and does not degrade the performance of the processor.

```
DMPAL16R6B;   SOFTLOAD
CK LCL XACK IA13 RESET NC6 NC7 NC8 IWR GND
/OE /BRESET /REMWR /EPAO /CS /ST2 /ST1 /REMRD /LCLINV VCC
/REMRD := RESET*     /REMRD*           CS*/EPAO*/REMWR
        + RESET*     /REMRD*      ST2* CS*      /REMWR
        + RESET*     /REMRD* ST1*      CS*      /REMWR
        + RESET*IA13* REMRD*/ST1*/ST2*/CS*/EPAO* REMWR
/ST1   := RESET*      REMRD*/ST1* ST2*/CS
        + RESET*      REMRD* ST1*/ST2*/CS
        + RESET*     /REMRD* ST1*/ST2* CS*      /REMWR
        + RESET*     /REMRD*/ST1* ST2* CS*      /REMWR
        + RESET*/XACK*REMRD*     /ST2*/CS*      /REMWR
/ST2   := RESET*      REMRD*      ST2*/CS
        + RESET*/XACK*REMRD*/ST1*     /CS*      /REMWR
        + RESET*     /REMRD*      ST2* CS*      /REMWR
        + RESET*     /REMRD*/ST1*      CS* EPAO*/REMWR
        + RESET*      REMRD* ST1*/ST2* CS* EPAO* REMWR
/CS    := RESET*      REMRD*          /CS*      /REMWR
        + RESET*      REMRD* ST1*     /CS
        + RESET*      REMRD*          /CS* EPAO
        + RESET*      REMRD*      ST2*/CS
        + RESET*      REMRD* ST1* ST2*     EPAO* REMWR
        + RESET*/IA13*REMRD*          /CS
*/EPAO  := RESET*      REMRD*      ST2*/CS*/EPAO
        + RESET*/XACK*REMRD*          /CS*/EPAO
        + RESET*      REMRD*          /CS*/EPAO* REMWR
        + RESET*      REMRD* ST1*     /CS*/EPAO
        + RESET*     /REMRD* ST1*      CS*/EPAO*/REMWR
        + RESET*     /REMRD*      ST2* CS*/EPAO*/REMWR
        + RESET*XACK* REMRD*/ST1*/ST2*/CS*EPAO*/REMWR
        + RESET*      REMRD* ST1* ST2* CS*EPAO* REMWR
/REMWR := RESET*     /REMRD*       ST2*/CS*     /REMWR
        + RESET*      REMRD* ST1*      /CS*     /REMWR
        + RESET*     /REMRD*           CS*/EPAO*/REMWR
        + RESET*     /REMRD*      ST2* CS*      /REMWR
        + RESET*      REMRD*/ST1*/ST2*/CS*       REMWR
        + RESET*     /REMRD* ST1*      CS*      /REMWR
        + RESET*/XACK*REMRD*          /CS*      /REMWR
/BRESET = /RESET +   /REMRD*/ST1*      CS*/EPAO*/REMWR
/LCLINV = LCL
```

FIGURE 7

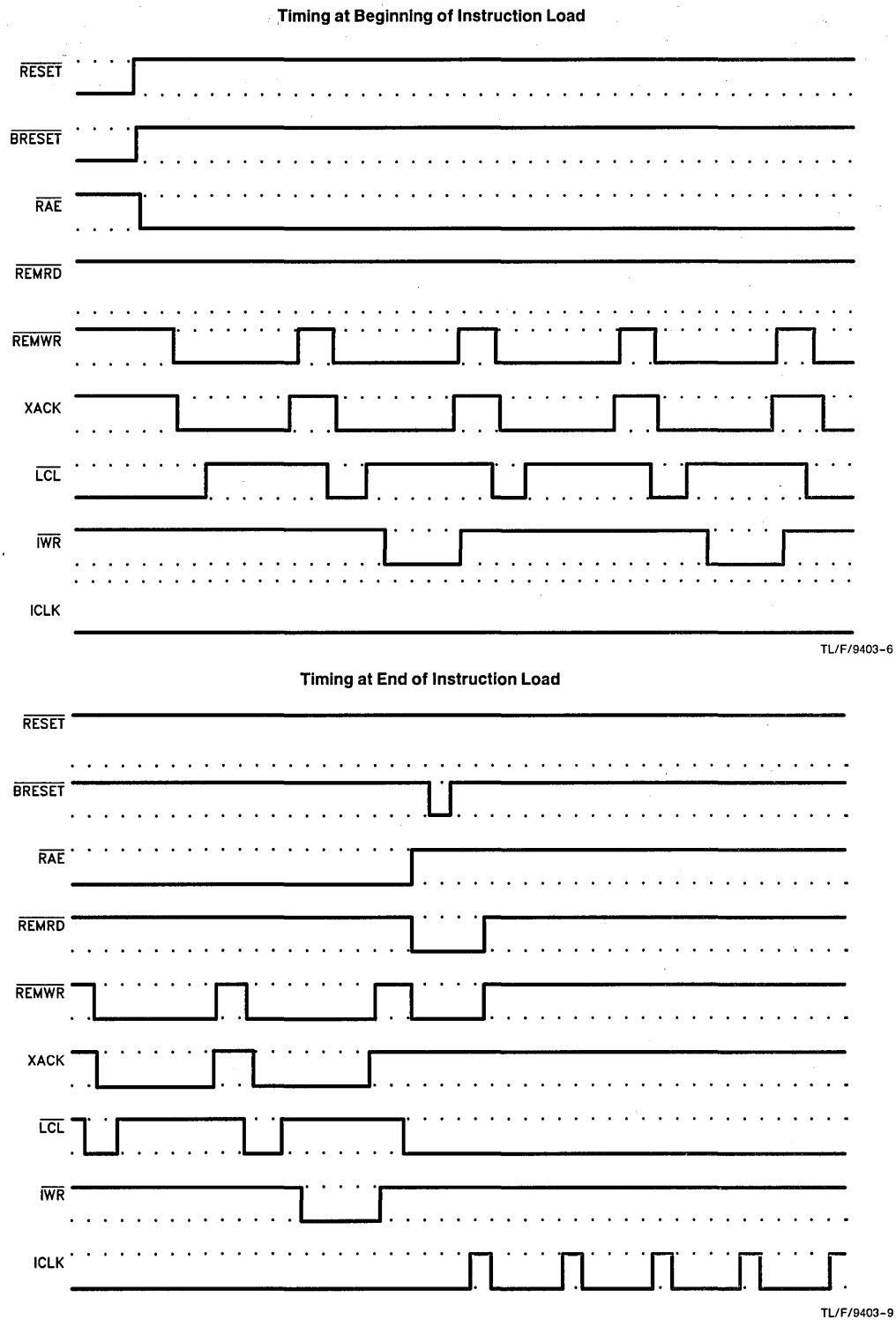
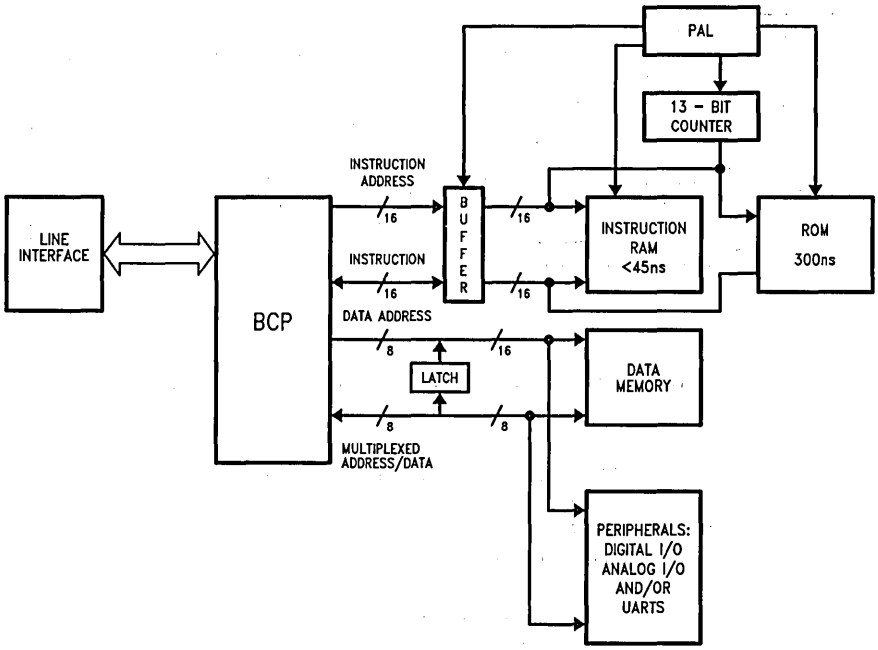


FIGURE 8. Example of Timing Waveforms



TL/F/9403-7

FIGURE 9. Another Method of Soft-Loading (A Non-Ideal Solution)

MODIFYING THE SOFT-LOAD SYSTEM FOR LARGER MEMORY

The soft-load system as documented loads 8K x 16 bits of instruction memory. Large programs may require more memory; smaller, lower cost systems may use less. The soft-load system can easily be altered to load larger or smaller instruction memory by changing one connection.

Connecting a different instruction address line to pin 4 of the PAL changes how much instruction memory is loaded: These connections are shown in *Figure 10.*

Instruction Memory Size:	Connect Pin 4 of PAL to:
32k x 16	IA15
16k x 16	IA14
8k x 16	IA13
4k x 16	IA12
2k x 16	IA11

FIGURE 10. Connections for Altering Instruction Memory Size

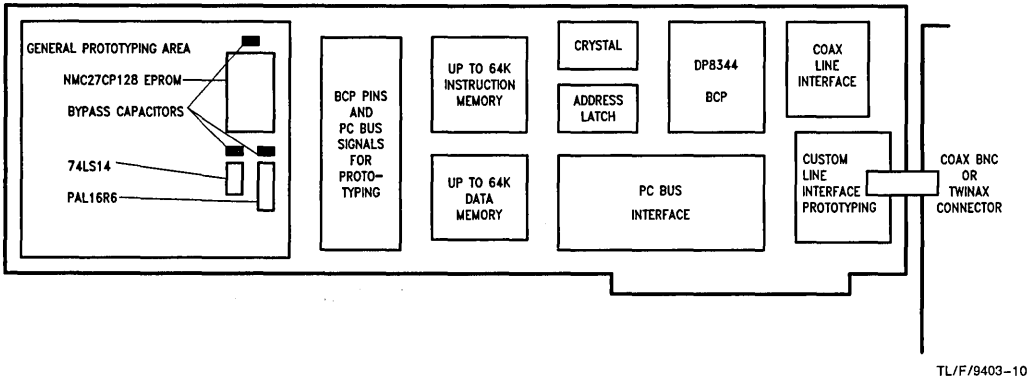
USING THE CAPSTONE CT-104 DEVELOPMENT BOARD TO EVALUATE THE SOFT-LOAD APPLICATION

A DP8344 biphase Communications Process development board is available from Capstone Technology Inc., of Fremont, California. The board is designed to reside in an IBM® PC. A breadboard area is provided on the board so that custom circuitry can be added. It can be converted into a stand-alone soft-load system by wire-wrapping three additional I.C.'s into the breadboard area. A diagram of the CT-104 board with the additional components is shown in *Figure 11.* Note that most of the prototyping area remains available, enabling the addition of other circuitry specific to the application being developed. A parts list is shown in *Figure 12.* The PAL16R6 is programmed with the equations shown in *Figure 7.* U22 and U23 must be removed from the CT-104 board and be replaced with specially wired 20-pin headers. The wiring on these headers, shown in *Figure 13,* provides access to the RESET~ signal and disables the unused interface circuitry on the board. Pin 11 of the header that replaces U23 must be wired to pin 13 of the 74LS14. A wiring list is shown in *Figure 14.* Power supply connections must be added because the board can no longer reside in the PC. Development of a stand-alone soft-load application can be done easily and quickly by using the CT-104 board because minimal circuit construction is required.

SUMMARY

The soft-load circuit uses the BCP's remote interface to load BCP code from slow EPROM to fast RAM, with a minimum of extra hardware. This method is useful in systems where there is no host processor directly interfaced to the BCP and the full processing speed of the BCP is needed.

The circuit can easily be modified to load different sizes of memory. The Capstone Technology, Inc. CT-104 development board can easily be converted to a stand-alone soft-load system for evaluation of the application.

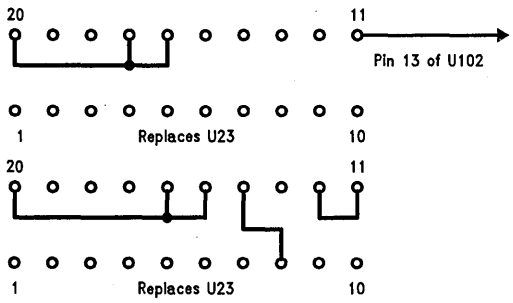


TL/F/9403–10

FIGURE 11. CT-104 Development Board with Soft-Load Circuitry

NMC27CP128 350 ns access time or faster
PAL16R6B
DM74LS14N
28-pin wire-wrap socket
20-pin wire-wrap socket
14-pin wire-wrap socket
3 Bypass capacitors, 0.1 μF
2 50-pin wire-wrap strips, 2 pins wide
2 20-pin headers

FIGURE 12. Parts List for Conversion of CT-104 Board



TL/F/9403–11

FIGURE 13. Header Wiring for Conversion of CT-104 Board

Pin	Unit	to	Pin	Unit
1	U100		—	VCC
2	U100		12	W1
3	U100		7	W1
4	U100		6	W1
5	U100		5	W1
6	U100		4	W1
7	U100		3	W1
8	U100		2	W1
9	U100		1	W1
10	U100		14	U101
11	U100		33	W1
12	U100		34	W1
13	U100		35	W1
14	U100		—	GND
15	U100		36	W1
16	U100		37	W1
17	U100		38	W1
18	U100		39	W1
19	U100		40	W1
20	U100		46	W2
21	U100		10	W1
22	U100		19	U101
23	U100		11	W1
24	U100		9	W1
25	U100		8	W1
26	U100		13	W1
27	U100		—	VCC

Pin	Unit	to	Pin	Unit
28	U100		—	VCC
1	U101		17	W2
2	U101		11	W2
3	U101		7	W2
4	U101		14	W1
5	U101		10	U102
6	U101		—	GND
7	U101		—	GND
8	U101		—	GND
9	U101		50	W1
10	U101		—	GND
11	U101		49	W2
12	U101		8	W2
13	U101		48	W2
15	U101		46	W2
18	U101		47	W2
20	U101		—	VCC
1	U102		—	GND
3	U102		—	GND
5	U102		—	GND
7	U102		—	GND
9	U102		—	GND
11	U102		12	U102
13	U102		11	U23 HEADER
14	U102		—	VCC
45	W2		—	GND

FIGURE 14. Wiring List for Conversion of CT-104 Board

"Interrupts"—A Powerful Tool of the Biphase Communications Processor

National Semiconductor
Application Note 499
Mark Koether

When you have only 5.5 μs to respond you have to act fast. This is the amount of time specified in the IBM 3270 Product Attachment Information document as the maximum time allowed to respond to a message in a 3270 environment. This 5.5 μs is why the DP8344 interrupts are specifically tailored for the task of managing a communications line and feature very short latency times. This article contains information that will help the user to take better advantage of the extensive interrupt capability found in the DP8344.

The DP8344 has two external and four internal interrupt sources. The external interrupt sources are the Non-Maskable Interrupt pin, ($\overline{NMI}$), and the Bi-directional Interrupt Re-Quest pin ($\overline{BIRQ}$). A Non-Maskable Interrupt is detected by the CPU when $\overline{NMI}$ receives a falling edge. The falling edge is captured internally and the interrupt is processed when it is detected by the CPU as described later. $\overline{BIRQ}$ can function as both an interrupt into the DP8344 and as an output which can be used to interrupt other devices. When $\overline{BIRQ}$ is configured as an input an interrupt will occur if the pin is held low. Note that $\overline{BIRQ}$ is not edge sensitive and if the pin is taken back high before the interrupt is processed by the CPU then no interrupt will occur.

The internal interrupts consist of the Transmitter FIFO Empty (TFE) interrupt, the Line Turn Around (LTA) interrupt, the Time Out (TO) interrupt, and a user selectable receiver interrupt source.

The receiver interrupt source is selected from either the Receiver FIFO Full (RFF) interrupt, the Data Available (DA) interrupt, or the Receiver Active (RA) interrupt. The RFF interrupt occurs when the receive FIFO is full or if the receiver detects an error condition. This interrupt enables the user to handle packets of data as opposed to handling every data word individually. It also allows the program to spend additional time performing other tasks. However, since the RFF interrupt is only asserted when the receive FIFO is full, the LTA interrupt should be used in conjunction with RFF to allow the program to check the FIFO for additional words at the end of a message. The DA interrupt indicates valid data is present in the receive FIFO and also occurs if the receiver detects an error condition. It should be used when it is desirable to handle each data word individually. The DA interrupt also allows the program to utilize the time between receiving each data word for performing other tasks. The RA interrupt is asserted when the receiver detects a valid start sequence. It provides the user with an early indication of data coming into the receiver. This allows the program time to perform any necessary overhead activity before handling the receiver data. The RA interrupt is asserted approximately 90 transceiver clock cycles prior to data becoming available in the receive FIFO when using 3270 mode. Consequently, if the transceiver and CPU are operating at the same clock frequency, approximately 90 clock cycles (T-states) are available for interrupt latency and taking care of overhead prior to handling the received data.

A TFE interrupt occurs when the last word in the transmit FIFO is loaded into the encoder. This interrupt allows a program to continue working on another task while the transmitter is sending data. It is especially useful when sending a long message. When the transmit FIFO becomes empty the program is alerted by the TFE interrupt and may continue the message by loading additional words into the FIFO. This approach frees up a significant amount of processing time. For example, after the transmit FIFO is loaded it takes the transmitter approximately 264 transceiver clock cycles to send the starting sequence and two data words in 3270 mode. With the CPU operating at the transceiver clock frequency, the program has approximately 264 T-states available before the TFE interrupt will occur.

Once the TFE interrupt occurs the CPU has approximately 80 transceiver clock cycles to load the transmit FIFO in order to continue a multiframe message in 3270 mode. If the CPU is operating at the transceiver clock frequency, the program has approximately 80 T-states to accomplish the load operation. Since the load to the Receive/Transmit Register, {RTR}, only takes 2 T-states, 78 T-states are available for interrupt latency and processing overhead after the interrupt occurs.

The LTA interrupt provides an easy means for determining the end of a message. This allows a program to quickly begin transmitting after the end of a reception. The LTA interrupt indicates that the receiver detected a valid end sequence in 3270 mode of operation. In 5250 operating mode, the LTA interrupt occurs when the last fill bit has been received and no further input transitions are detected by the receiver. However, a LTA interrupt does not occur in 5250 or 8-bit non-promiscuous modes of operation unless an address match was decoded by the receiver.

The TO interrupt occurs when the CPU timer counts down to zero. The timer provides a flexible means for timing events. It is a sixteen bit counter which can be loaded by accessing CPU registers {TRH} and {TRL} and is controlled by the [TCS], [TLD] and [TST] bits in the Auxiliary Control Register, {ACR}.

After an interrupt occurs the event that generated it must be handled in order to clear the interrupt. The exception to this is $\overline{NMI}$. Since it is falling edge triggered, it is cleared internally when the CPU processes the interrupt. The actions necessary to clear the interrupts are listed in Table I.

In the case where $\overline{BIRQ}$ is asserted, the response will be dependent on the system design. Ordinarily, this response would involve some hardware handshaking such as reading or writing a specific data memory location. When internal interrupts become asserted there are specific actions which must be taken by a program to clear these interrupts. The RFF interrupt is cleared when the receive FIFO is no longer full and any errors detected by the receiver are cleared. Data is read from the receive FIFO by reading {RTR}. Reading the Error Code Register, {ECR}, clears any errors detected by the receiver. The DA interrupt is cleared when the receive FIFO is empty and any errors detected by the receiver are cleared. The RA interrupt is cleared by reading {RTR} or {ECR}. All three receiver interrupts are cleared when the transceiver is reset. In many cases, resetting the transceiver is the preferable response to an error detected

TABLE I. Clearing Interrupts

Interrupt	How to Clear Interrupt
$\overline{NMI}$	Internally Cleared When Recognized by the CPU.
RFF	Read {RTR} When Receive FIFO is Full. Read {ECR} When an Error Occurs. Read {ECR} and {RTR} When an Error Occurs and Receive FIFO is Full. Reset the Transceiver. Reset the DP8344.
DA	Read {RTR} When Receive FIFO is Not Empty. Read {ECR} When an Error Occurs. Read {ECR} and {RTR} When an Error Occurs and Receive FIFO is Not Empty. Reset the Transceiver. Reset the DP8344.
RA	Read {RTR} or {ECR}. Reset the Transceiver. Reset the DP8344.
TFE	Write to {RTR}.
LTA	Write to {RTR}. Reset the Transceiver. Reset the DP8344. Write a One to {NCF} Bit 4.
$\overline{BIRQ}$	System Dependent.
TO	Write a One to {CCR} Bit 7. Stop the Timer. Reset the DP8344.

by the receiver. The TFE interrupt is cleared by writing to {RTR}. Unlike the receiver interrupts, the TFE interrupt is asserted when the transceiver is reset. The LTA interrupt is also cleared by writing to {RTR} or resetting the transceiver. In addition, it may be cleared by writing a one to bit 4 of the Network Command Flags register, {NCF}. The last internal interrupt is TO. It is cleared by writing a one to bit 7 in the Condition-Code Register, {CCR} or by stopping the timer. Note that the timer reloads itself and continues to count after the interrupt has been generated regardless of whether a one is written to bit 7 in {CCR}.

With the exception of $\overline{NMI}$, all of the interrupts are disabled when the DP8344 is reset. In order to make use of the interrupts they must be enabled in software. Software enabling and disabling of the interrupts is performed by changing the state of the Global Interrupt Enable, [GIE], bit in {ACR} and the state of the individual interrupt mask bits in the Interrupt Control Register, {ICR}.

[GIE] is a read/write register bit and so may be changed by using any instruction that can write to {ACR}. In addition, the RET, RETF, and EXX instructions have option fields which can be used to alter the state of [GIE]. RET and RETF are the return instructions in the DP8344 and EXX is used to exchange register banks. The EXX instruction can set or clear [GIE] as well as leaving it unchanged. The RET and RETF instructions can restore [GIE] to the value that was saved on the address stack at the time the interrupt was recognized. They also provide the options of clearing or setting [GIE] or leaving it unchanged. [GIE] is cleared when an interrupt is recognized by the CPU in order to prevent other interrupts from occurring during an interrupt service routine. The [GIE] options described above facilitate enabling and disabling interrupts when returning from an interrupt service routine. The restore option is especially useful with the $\overline{NMI}$. Since a Non-Maskable Interrupt can occur whether [GIE] is set or cleared, the restore [GIE] option can be used in the return instruction to put [GIE] back to its state prior to the interrupt occurring.

As the name implies, [GIE] affects all the maskable interrupts. However, in order to use any of these interrupts they must be unmasked by changing the state of their associated mask bit in {ICR}. When set high, bits [IM0], [IM1], [IM2], [IM3], and [IM4] in {ICR} mask the receiver interrupt, TFE interrupt, LTA interrupt, $\overline{BIRQ}$ interrupt, and TO interrupt respectively. To enable an interrupt, its mask bit must be set low. The interrupts and associated mask bits are shown in Table II. These bits are set high when the DP8344 is reset. Bits [RIS1] and [RIS0] in {ICR} are used to select the source of the receiver interrupt as shown in Table III. Note that only one of these interrupts can be active as the source of the receiver interrupt.

TABLE II. {ICR} Interrupt Mask Bits and Interrupt Priority

Interrupt	Mask Bit	Priority
$\overline{NMI}$	—	Highest
RFF, DA, RA	IM0	
TFE	IM1	
LTA	IM2	
$\overline{BIRQ}$	IM3	
TO	IM4	Lowest

TABLE III. {ICR} Receiver Interrupt Select Bits

RIS1	RIS0	Receiver Interrupt Source
0	0	RFF
0	1	DA
1	0	Reserved
1	1	RA

As stated earlier, [GIE] is cleared when an interrupt is recognized by the CPU. This prevents other interrupts from occurring in the interrupt service routine. In cases where it is desirable to allow nesting of interrupts, [GIE] should be set high within the interrupt routine. An example of nesting interrupts is using the RA interrupt in the main program and switching to the RFF or DA interrupt in the RA interrupt routine. Note that the internal address stack is twelve words deep and there is no recovery from a stack overflow. Therefore, care should be taken when nesting interrupts.

When more than one interrupt is unmasked and asserted, the CPU processes the interrupt with the highest priority first. $\overline{NMI}$ has the highest priority followed by the receiver interrupt, TFE, LTA, $\overline{BIRQ}$, and TO. Therefore, if DA and $\overline{BIRQ}$ were both active, DA would be processed first followed by $\overline{BIRQ}$. However, if a higher priority interrupt occurred while the DA interrupt was being handled then it would be processed before $\overline{BIRQ}$. Each time the interrupts are sampled, the highest priority interrupt is processed first, regardless of how long a lower priority interrupt has been active. Interrupt priority is summarized in Table II.

A call to the interrupt address is generated when an interrupt is detected by the CPU. The address for each interrupt is constructed by concatenating the Interrupt Base Register, {IBR}, contents with the individual interrupt code as shown in Table IV. There is room between the interrupt addresses for a maximum of four instruction words. Normally, at each interrupt address there would be a jump instruction to an interrupt service routine. The return instruction at the end of the interrupt service routine would then return to the address at which the interrupt occurred. By changing {IBR} it is possible to locate the interrupt jump table in memory wherever it is convenient or for one program to use more than one interrupt jump table.

TABLE IV. Interrupt Vector Generation

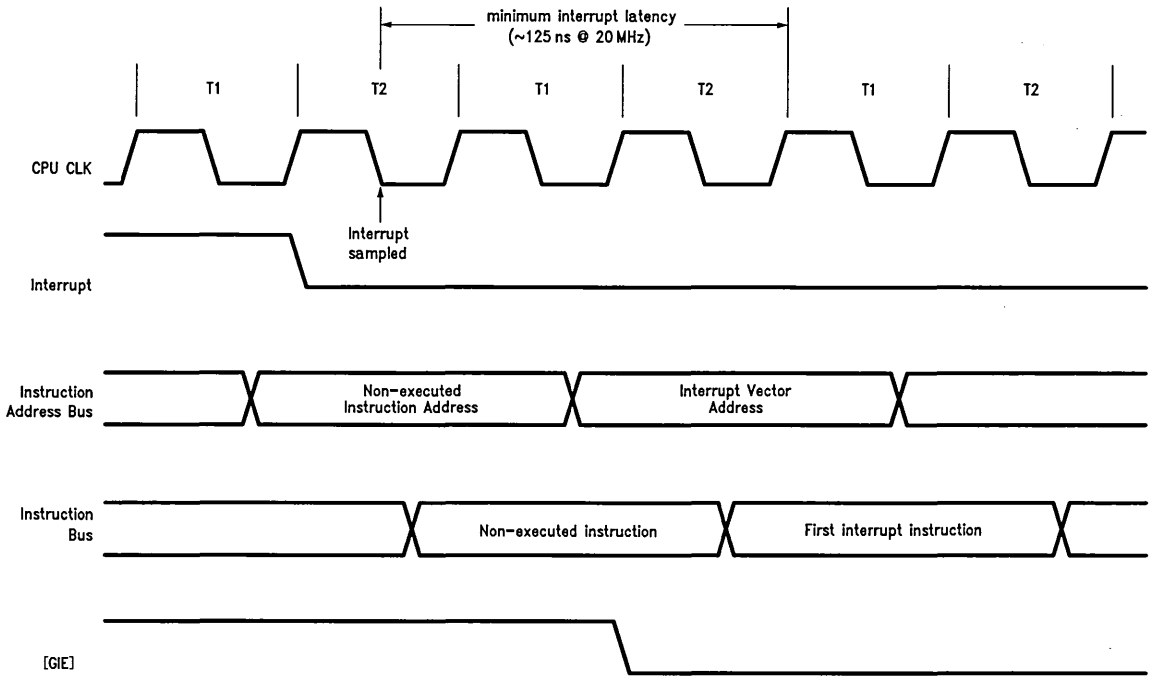
Interrupt	Code
NMI	111
RFF, DA, RA	001
TFE	010
LTA	011
BIRQ	100
TO	101

Interrupt Vector

{IBR} Contents	0 0 0	Code	0 0

15 8 4 2 0

As mentioned previously, the interrupts are sampled in the CPU prior to the start of each instruction. To be precise, they are sampled by each falling edge of the CPU clock with the last falling edge prior to the start of the next instruction determining whether an interrupt will be processed. The timing of a typical interrupt event is shown in *Figure 1*. The interrupt occurs during the current instruction and is sampled by the falling edge of the CPU clock. The next instruction is not operated on and its address is stored in the internal address stack. In addition, the current state of [GIE] and the states of the ALU flags and bank positions are stored in the internal address stack. A 2 T-state call is now executed in place of the non-executed instruction. This call will cause a branch to the interrupt address that is generated in the first half of T-state T1. [GIE] is then cleared during the first half of T-state T2. From this description it is evident that the shortest interrupt latency is 2.5 T-states. This assumes that an interrupt occurs during the first half of T2 and is sampled by the next falling edge of the CPU clock. However, a number of factors can increase the interrupt latency. If the interrupt misses the setup time to the falling edge of the last CPU clock the response time will increase by a minimum of 2 T-states. This increase is caused by the execution of one additional instruction. Of course, if the additional instruction takes more than 2 T-states to execute the interrupt latency will be greater.



TL/F/9361-1

FIGURE 1. Minimum Interrupt Timing

Running the DP8344 with wait states will also increase interrupt latency. Instruction memory wait states increase latency by increasing the length of each instruction, including the call to the interrupt service routine. Data memory wait states will increase interrupt latency if an interrupt must wait for an instruction which accesses data memory to execute before it can be processed. A less obvious factor that can increase interrupt latency is data memory accesses by the remote system. If the DP8344 is attempting a data memory access and the remote system already has control of the data memory bus, the CPU will be waited. If an interrupt occurs at this time it will not be processed until the DP8344 is able to complete the instruction which is accessing data memory. This implies that a system with a lot of data memory arbitration occurring between the DP8344 and the remote system may have a longer average interrupt latency. The worst case interrupt latency will occur when the external $\overline{\text{LOCK}}$ or $\overline{\text{WAIT}}$ pins are asserted. Clearly, if the CPU is stopped by the assertion of the $\overline{\text{WAIT}}$ pin any interrupts ocurring will not be processed until the CPU is released from the wait state. Asserting the $\overline{\text{LOCK}}$ pin would have the same affect if the DP8344 attempts to make a data memory access. Note that interrupts are not disabled or cleared when the CPU is stopped by the remote system deasserting [STRT] in the Remote Interface Configuration, {RIC}, register. When the CPU is restarted any asserted interrupts will be processed. From the above discussion it is evident that calculating the interrupt latency is not trivial and will be dependent on the program and the system.

The interrupts on the DP8344 are powerful tools for controlling events in a time critical environment. They are one of the many reasons why the DP8344 Bi-phase Communications Processor provides a superior solution to managing communications interfaces.

JRMK Speeds Command Decoding

National Semiconductor
Application Note 625
David Weinman

The Biphase Communications Processor (BCP) has several features that make it ideal to use in a high speed communications environment. The relative Jump with Rotate and MasK on register command, JRMK, is designed to allow quick and efficient decoding of register fields. Fast decoding of command, data, and address fields allows the BCP to spend most of an interrupt handler's code and time on the protocol's actual instruction execution, instead of on decoding it. This helps meet the stringent 5.5 μs turn around times demanded in 3270 communications.

JRMK rotates and masks a copy of its source register to form a signed program counter offset which is often used to point to a jump table. The JRMK instruction first makes a copy of the source register. All actions will be performed on this copy, not on the original. The register then is rotated to the right zero to seven places. Next, JRMK masks (zeros out) the LSB in addition to as many bits as the mask field indicates, starting at the MSB. Finally, JRMK adds this result to the Program Counter (PC), providing a relative range of +128, −126 instruction words. In practice, relative jumps (JMP) and long jumps (LJMP) are usually placed in the table, but there are no restrictions on which instructions may fit in. Each entry has a minimum space of two instruction words allowing LJMP's to fit. *Figure 1* demonstrates the BCP's internal execution of a JRMK instruction.

Example Code

```
JRMK      RTR,3,3          ;decode feature address
```

Instruction Execution

(a) Copy {RTR} into JRMK's displacement register

(b) Rotate displacement register 3 bits right

(c) AND result with "00011110"

(d) Sign extend resulting displacement and add it to the program counter, (PC). If the bits F4–F1 equal "0001" then +2 is added to the PC.

JRMK Displacement Register Contents

(a)	F4	F3	F2	F1	x	x	x	x
(b)	x	x	x	F4	F3	F2	F1	x
(c)	0	0	0	F4	F3	F2	F1	0

FIGURE 1. JRMK Instruction Example

The JRMK instruction contains four (4) fields that control its operation—a source register field, a rotate field, a mask field, and the opcode itself. The source register may be any register in the BCP that is always available or is currently bank switched in. The source register is not modified by the operation of the JRMK instruction. Even in the case of the {RTR} register, the receiver FIFO is not changed and the same byte remains at the top of the FIFO after executing JRMK. The rotation field directs the BCP to rotate the source register to the right by 0–7 bits. The mask field indicates how many bits to mask from the source register starting at the MSB after the rotation is complete. Up to 7 bits may be masked off in addition to the LSB. If the mask field equals zero (0), only the LSB will be masked. If the mask field equals one (1), the MSB will be masked as well as the LSB. Similarly, if the mask field equals two (2), bits 7,6 and the LSB will be masked. *Figure 2* shows the construction of the JRMK instruction opcode.

Opcode

1	0	0	0	0	m	m	m	b	b	b	Rs

m—Mask Field
b—Bit Places to Rotate
Rs—Source Register

FIGURE 2. JRMK Opcode Construction

JRMK can be set up to provide more than two instruction words per table entry, if the source register data format is known. If the rotation causes a zero bit to always appear in bit 1 of the rotated register, then each table entry will have four instruction words.

The JRMK instruction executes in 4 T-states if there are no instruction wait states. If the BCP's CPU clock is running at a speed of 20 MHz, a T-state is 50 ns in duration. In this case, each JRMK instruction will complete in 200 ns.

AN EXAMPLE

A good example of how to use the JRMK instruction is found in the Multi-Protocol Adaptor (MPA). The MPA is a design/evaluation kit available from National Semiconductor. It provides complete link level source code, hardware, and development notes for creating a 3270 or 5250 PC terminal emulator card.

This example comes from actual MPA code in the Data Available interrupt handler for 3270 terminal emulation. All overhead such as bank switching, register saving, and index register setting have been previously executed, and the 3270 command is at the top of the receiver FIFO. The actual implementation of executing each 3270 instruction, as well as the decode tables for devices other than the base, is not shown. Additionally, the code for handling data is not presented. These are all included with the MPA source code.

When a 3270 message is available in the receiver FIFO, a determination is made whether that message is a command or data at the *rxcx_fast* label as shown in *Figure 3*. If the receiver contains data, the BCP vectors to a location held in the index register equated to DATA_VECTOR. If the message is a command, the BCP will jump to the label *cx_comm* to check for common commands. The Network Control Flag (NCF) register contains bits for hardware decoded commands, POLL, POLL/ACK, and TT/AR. POLL and POLL/ACK will jump to their respective command handlers. Since a TT/AR shold not be received by a terminal, its decode will jump to the *cx_perr* error handler. A no-operation, NOOP, is inserted after the first jump because the JRMK instruction is set in this case to jump to every other address. The NOOP takes up an instruction location to ensure that the table conforms to this specification. A NOOP is a macro that stands for MOVE ACC,ACC. If the command is not one of these three, then the address of the command must be checked.

At the label *addr_dec*, the BCP will vector to different command handlers based on the feature address of the received command. All unimplemented features jump to the *cx_dec_err* error handler. The JRMK instruction is used to look at bits 4–7 of {RTR} which point to the 3270 feature that the command is for. Based on these bits, the different feature command decoders will be jumped to as shown in *Figure 4*.

```
;
; setup code here
;
        .
        .
        .
rxcx_fast:
        ljmp        TSR,1,S,cx_comm         ; command or data?
                                            ; jump if command
        ljmp        [DATA_VECTOR]           ; data, jump to appropriate
                                            ; handler


;
; check for quick command decodes
;
cx_comm:
        jrmk        NCF,7,4                 ; jump on immediate decode prior to
                                            ; advancing FIFO
cx_immed:
        jmp         addr_dec                ; not an immediate decode command
        NOOP
        ljmp        cx_poll                 ; poll command decoded
        ljmp        cx_pack                 ; pack
        ljmp        cx_perr                 ; should not get here (TT/AR)
```

FIGURE 3. JRMK Fast Command Determination

```
;
; find out which feature that the command is addressed to
;
addr_dec:
        jrmk        RTR,3,3         ; jump based on 4 bit address field

; address parse table

cx_addr:
        jmp         base_dec        ; 0 decode base/keyboard command
        NOOP
        jmp         base_dec        ; 1 decode base/keyboard
        NOOP
        ljmp        cx_dec_err      ; 2 light pen
        ljmp        cx_dec_err      ; 3 reserved
        ljmp        cx_dec_err      ; 4 magnetic stripe reader
        ljmp        cx_dec_err      ; 5 PC adapter
        ljmp        cx_dec_err      ; 6 3180 advanced
        ljmp        eab_dec         ; 7 EAB
        ljmp        cx_dec_err      ; 8 reserved
        ljmp        cx_dec_err      ; 9 reserved
        ljmp        cx_dec_err      ; A reserved
        ljmp        cx_dec_err      ; B convergence
        ljmp        cx_dec_err      ; C reserved
        ljmp        cx_dec_err      ; D reserved
        ljmp        cx_dec_err      ; E reserved
        ljmp        cx_dec_err      ; F reserved
```

FIGURE 4. JRMK Feature Determination

At the base feature decoder *base_dec*, the actual command is decoded and jumps are taken to the different addresses to handle each one. *Figure 5* details this operation.

```
;
; base command parse table
;
base_dec:
        jrmk        RTR,7,2         ; decode base command
cx_base:
        ljmp        cx_ignore       ; 00 should not get here
        ljmp        cx_poll         ; 01 poll command
        ljmp        cx_reset        ; 02 reset device
        ljmp        cx_readata      ; 03 read data
        ljmp        cx_lach         ; 04 load address counter high
        ljmp        cx_rach         ; 05 read address counter high
        ljmp        cx_clear        ; 06 clear
        ljmp        cx_rdex         ; 07 read extended terminal ID
        ljmp        cx_start        ; 08 start operation
        ljmp        cx_rdid         ; 09 read terminal ID
        ljmp        cx_lcont        ; 0A load control register
        ljmp        cx_rdmul        ; 0B read multiple
        ljmp        cx_write        ; 0C write data
        ljmp        cx_rdstat       ; 0D read status
        ljmp        cx_insert       ; 0E insert byte
        ljmp        cx_ignore       ; 0F reserved
        ljmp        cx_sforward     ; 10 search forward
        ljmp        cx_pack         ; 11 poll with acknowledge set
        ljmp        cx_sback        ; 12 search backward
        ljmp        cx_ignore       ; 13 reserved
        ljmp        cx_lacl         ; 14 load address counter low
        ljmp        cx_racl         ; 15 read address counter low
        ljmp        cx_mask         ; 16 load mask
        ljmp        cx_ignore       ; 17 reserved
        ljmp        cx_ignore       ; 18 reserved
        ljmp        cx_ignore       ; 19 reserved
        ljmp        cx_lscont       ; 1A load secondary control
        ljmp        cx_ignore       ; 1B reserved
        ljmp        cx_diagreset    ; 1C diagnostic reset
        ljmp        cx_ignore       ; 1D reserved
        ljmp        cx_ignore       ; 1E reserved
        ljmp        cx_ignore       ; 1F reserved
```

FIGURE 5. JRMK Decoding of 3270 Instructions

If our command was a Load Control Register command (00001010), the JRMK instruction at label *cx_comm* would send us to a jump to *addr_dec* to decode which feature the command is directed to. At that label, JRMK would send us to the jump to *base_dec* since our address is "0000". Since the command is "01010", the JRMK relative jump will move to the instruction ***ljmp cx_jcont*** which jumps to the appropriate code to handle that instruction.

From *rxcx_fast* to the proper command to the base feature, there are 24 T-states of time used. At 20 MHz with no wait states, this translates to 1.2 μs. With a maximum interrupt latency of 225 ns, this leaves at least 4.075 μs to handle all other aspects of each command to the base. Commands to other features will probably take 1 T-state longer for the long jump to the command decode table (also using JRMK) for that feature, whereas the base feature used a relative jump.

The JRMK instruction is one example of how the BCP is optimized for high speed communications.

DP8344 Remote Processor Interfacing

National Semiconductor
Application Note 627
William V. Miller

This application note is provided to help the reader understand the information given in Table 26: Remote Rest Time of the DP8344BV 6.0 datasheet.*

For the BCP to operate properly, remote accesses to the BCP must be separated by a minimum amount of time. This minimum amount of time has been termed 'rest time'.

To give the reader a better understanding of rest time, the following items will be discussed in this application note:

1. The causes of remote rest time.
2. The way to interpret Table 26 and the worst case rest time.
3. The desirable features of a rest time circuit.
4. A design example of a rest time circuit for the CT-104 board.

Before proceeding any further, it must be stated that the design of DP8344BV did not introduce remote rest time. Remote rest time exists on all versions of the BCP.

*All specifications used in this application note are from the DP8344BV 6.0 datasheet. Please refer to the latest datasheet available for the most current specifications.

CAUSES OF REMOTE REST TIME

There are two causes for remote rest time. The first cause is implied in the state diagrams for remote accesses and can be explained as follows:

At the beginning of every T-state the validity of a remote access is sampled for that T-state. To guarantee that the BCP recognizes the end of a remote cycle, the time between remote accesses must be a minimum of one T-state plus setup and hold times. This worst case rest time for the DP8344BV is:

$$\begin{aligned} \text{rest time} &= 1T + t\,(\text{setup time}) + t\,(\text{hold time}) \\ &= 1T + 22\text{ ns} + 10\text{ ns} \\ &= 1T + 32\text{ ns} \end{aligned}$$

In the case of Latched Read and Fast Buffered Write, the validity of a remote access is not sampled on the first rising edge of the CPU-CLK following XACK rising. However, on all subsequent rising edges of the CPU-CLK, the validity of the remote access is sampled. As a result, if the remote processor can terminate its remote access quickly after XACK rises (within a T-state), up to a T-state may be added to the above equation for Latched Read and Fast Buffered Write modes. On the other hand, if the remote processor does not terminate its remote access within a T-state of XACK rising, the above equation remains valid for Latched Read and Fast Buffered Write modes.

If this specification is not adhered to, the BCP may sample the very end of one valid remote access and one T-state later sample the very beginning of a second valid remote access. Thus, the BCP will treat the second access as a continuation of the first remote access and will not perform the second read/write. The second access will be ignored. (Reference *Figure 1* for timing diagrams which demonstrate how two remote accesses can be mistaken as one.)

The second source of remote rest time is due to the manner in which the BCP samples the CMD signal. (Please note that when CMD is high all remote accesses are to the Remote Interface Control register {RIC}. When CMD is low all remote accesses are to where RIC's Memory Select Bits point.) CMD is sampled once at the beginning of each remote access. Due to the manner in which CMD is sampled, CMD will not be sampled again if a second remote access begins within 1.5(T-states) plus a hold time, after the BCP recognizes the end of the first remote access. If this happens, the BCP will use the value of CMD from the previous remote access during the second remote access. If the value of CMD is the same for both accesses, the second access will proceed as intended. However, if the value of CMD is different for the two remote accesses, the second remote access would read/write the wrong location.

The reader should note that the timing of the second source of rest time begins at the same time that the BCP first samples the end of the previous remote access. Thus, when the first source of rest time ends, the second source of rest time begins. (Reference *Figure 2* for timing diagrams for rest time in all modes except latched write.)

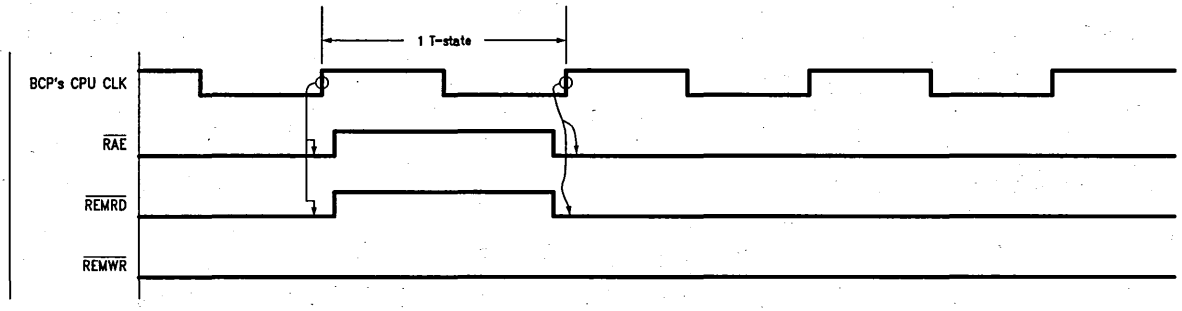
LATCHED WRITE MODE

Latched write mode is a special case of rest time and needs to be discussed separately from the other modes. The first cause of rest time affects every mode including latched write. In regards to the second source of rest time, latched write mode was designed to allow a second remote access to start while a write is still pending (i.e., $\overline{\text{WR-PEND}}$ = 0). Thus, when $\overline{\text{WR-PEND}}$ rises (signaling the end of the previous write) the value of CMD is sampled for the seccond remote access. This will result in sampling the correct value of CMD for the second access. This allows latched write to avoid the second cause of rest time mentioned above.

However, if a remote access begins within half a T-state after $\overline{\text{WR-PEND}}$ rises, CMD will not be sampled again. For this case, if the value of CMD changed just after $\overline{\text{WR-PEND}}$ rose and at the same time the remote access began, the BCP would read/write the wrong location. (Reference *Figure 3* for timing diagrams of rest time for latched write mode.)

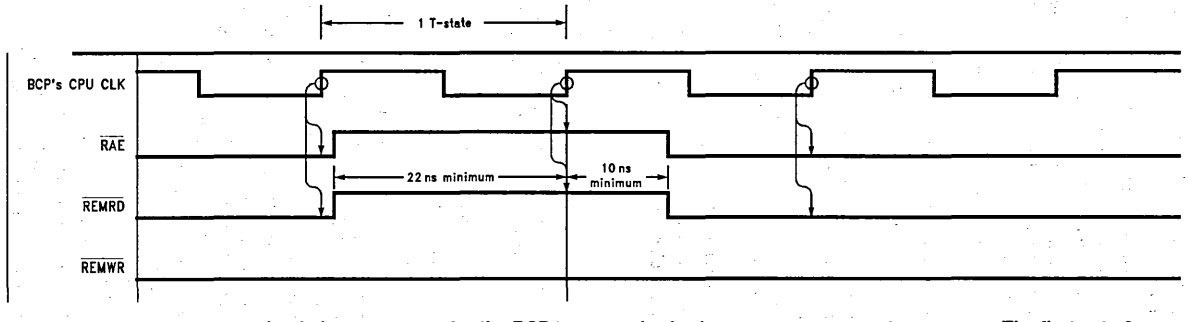
HOW TO INTERPRET TABLE 26 AND WORST CASE REST TIMES

At this time it is desirable to review how to interpret Table 26 and to review what the actual worst case rest time is. To interpret the specifications in Table 26, the reader must understand the differences between running the BCP at full speed (i.e., [CCS] = 0) and half speed (i.e., [CCS] = 1). At full speed both the CPU-CLK and CLK-OUT operate at the same frequency as OCLK. When the BCP runs at half speed, CLK-OUT remains at the same frequency as OCLK, but the CPU-CLK operates at half the frequency of OCLK. In the data sheet, one T-state is defined as one CPU-CLK cy-



TL/F/10451–1

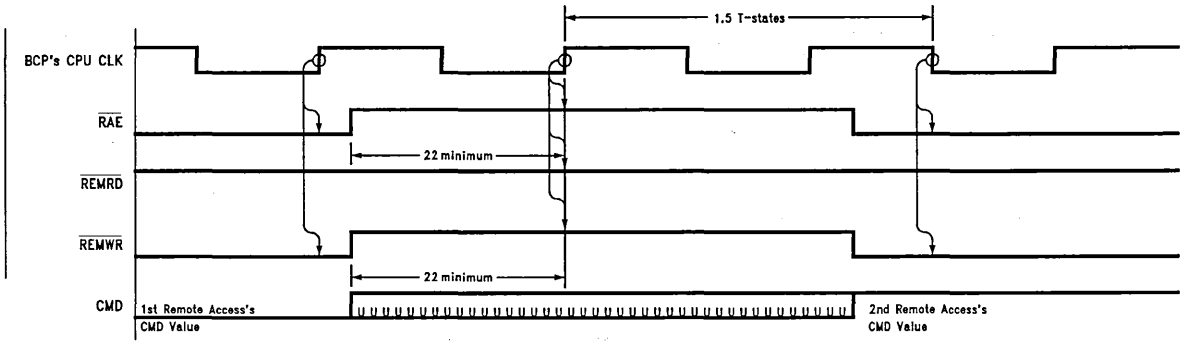
(a) This timing diagram shows two remote accesses within one T-state. The first set of arrows shows the BCP sampling a valid remote read. The next time the BCP samples the validity of the remote access is shown by the second set of arrows (1 T-state later). In this case, it will sample the second remote access and mistake it as a continuation of the first remote access.



TL/F/10451–2

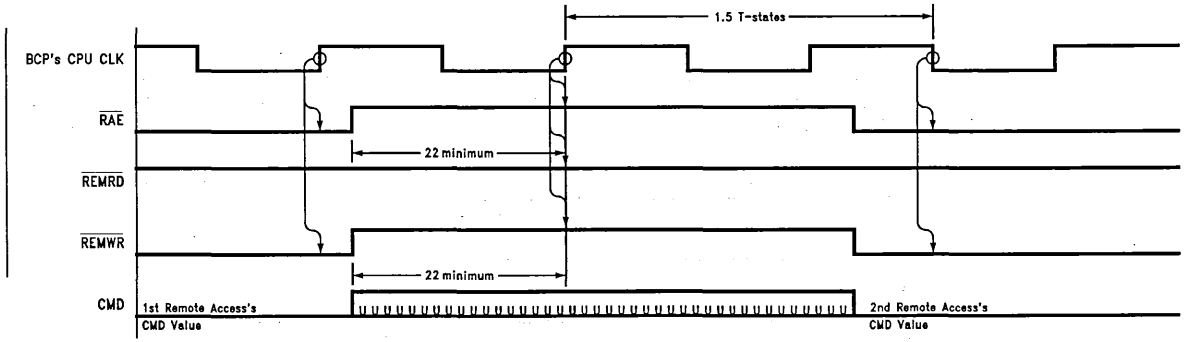
(b) This timing diagram shows the timing necessary for the BCP to recognize both accesses as separate accesses. The first set of arrows shows the BCP sampling a valid remote read. One T-state later at the second set of arrows, the BCP will sample the end of the first remote access. Another T-state later at the third set of arrows, the BCP will sample the beginning of the second remote access.

FIGURE 1. Mistaking Two Remote Accesses as Only One



TL/F/10451-3

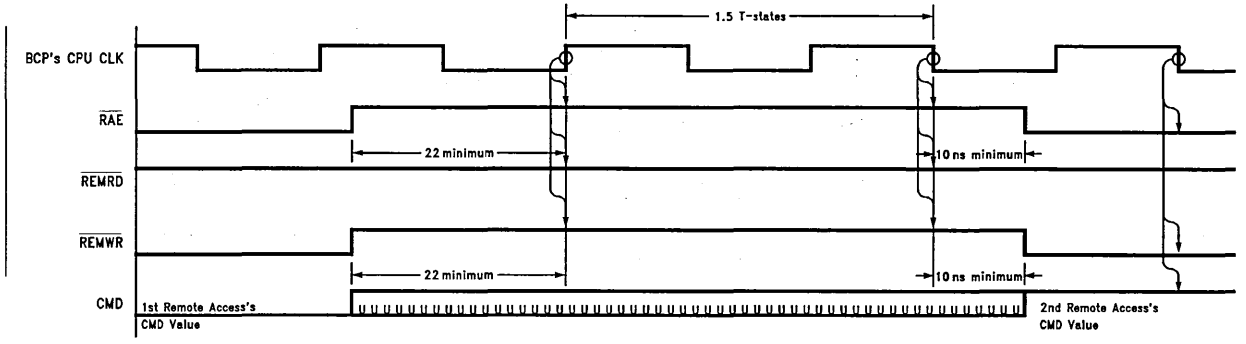
(a) This timing diagram shows the second remote access violating rest time. The first set of arrows shows the BCP sampling a valid remote write. The second set of arrows (1 T-state later), shows the BCP sampling the end of the first remote access. If a second remote access starts before the position of the third set of arrows (another 1.5 T-states later), the value of CMD will not be sampled. The value of CMD has changed from the first remote access, so the BCP will write to the wrong location during the second access.



TL/F/10451-4

(b) This timing diagram shows the second remote access violating rest time. The first set of arrows shows the BCP sampling a valid remote write. The second set of arrows (1 T-state later), shows the BCP sampling the end of the first remote access. If a second remote access starts before the position of the third set of arrows (another 1.5 T-states later), the value of CMD will not be sampled. The value of CMD does not change from the first remote access, so the BCP will write to the intended location during the second remote access.

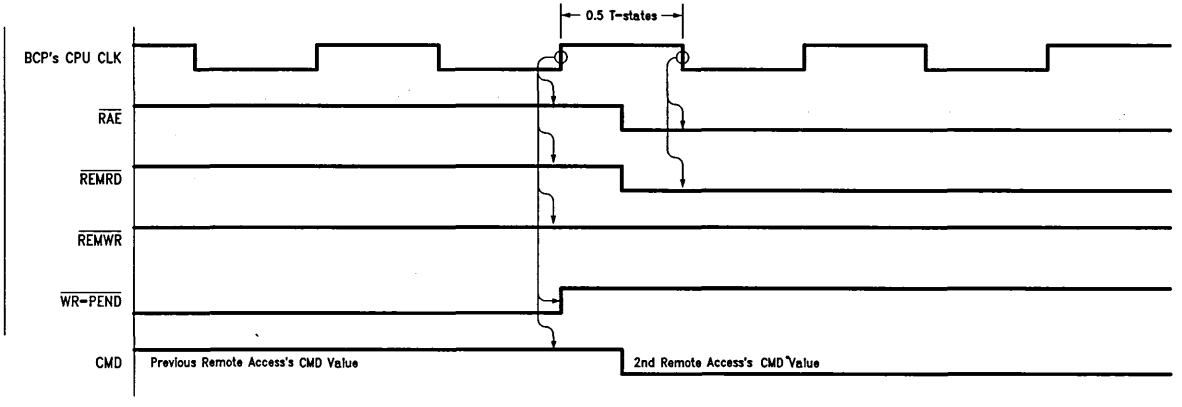
FIGURE 2. Remote Rest Time for All Modes except Latched Write



TL/F/10451-5

(c) This timing diagram shows the timing needed to avoid rest time for all modes except latched write. The first set of arrows shows the BCP sampling the end of the first remote access. The second set of arrows (1.5 T-states later), shows the BCP recognizing no remote access has started and the value of CMD will be sampled for the next remote access. The third set of arrows shows the BCP sampling the correct value of CMD for the second remote access.

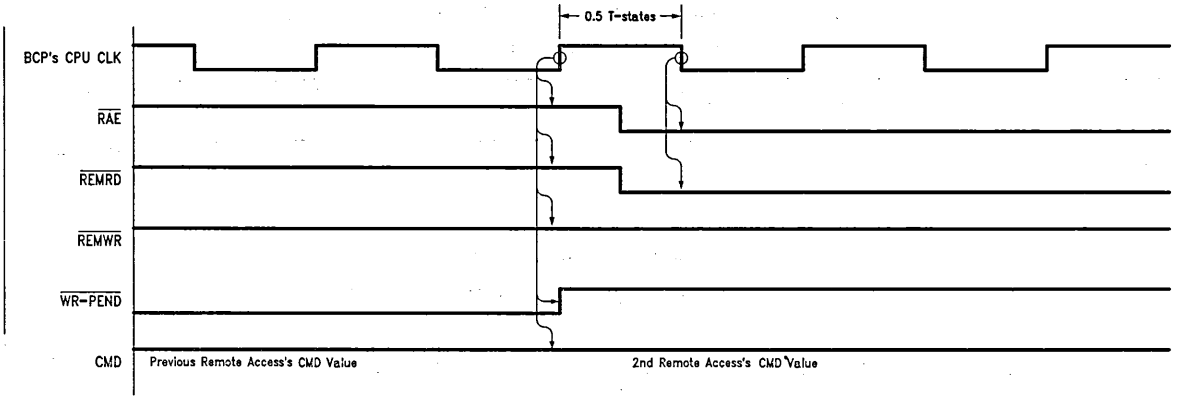
FIGURE 2. Remote Rest Time for All Modes except Latched Write (Continued)



TL/F/10451-6

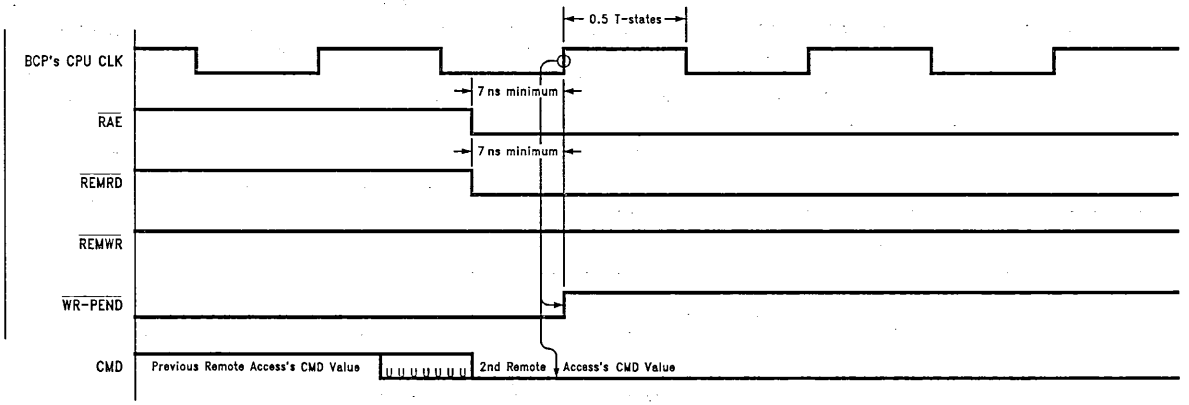
(a) This timing diagram shows a remote access violating remote rest time. The first set of arrows shows the BCP sampling the value of CMD when $\overline{\text{WR-PEND}}$ rises. If a remote access begins after $\overline{\text{WR-PEND}}$ rises and before the position of the second set of arrows (0.5 T-states later), the value of CMD will not be sampled again. The value of CMD has changed since $\overline{\text{WR-PEND}}$ rose, so the BCP will read the wrong location.

FIGURE 3. Rest Time for Latched Write Mode



TL/F/10451-7

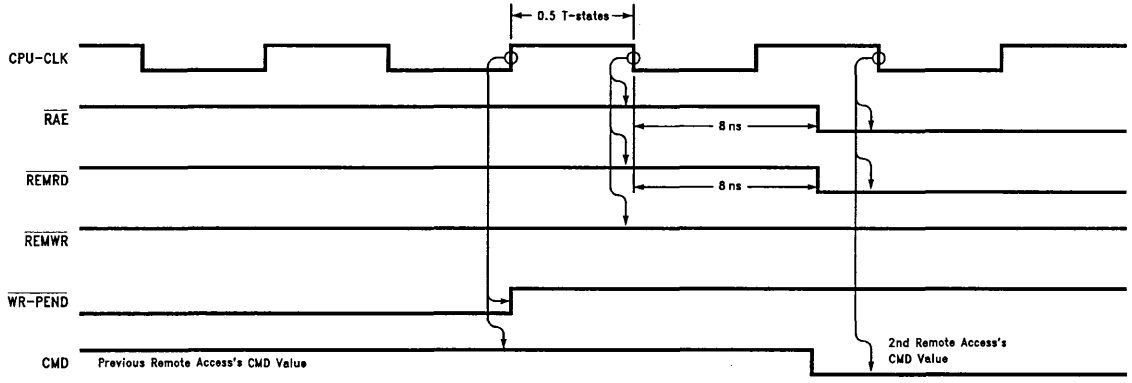
(b) This timing diagram shows a remote access violating remote rest time. The first set of arrows shows the BCP sampling the value of CMD when WR-PEND rises. If a remote access begins after WR-PEND rises and before the position of the second set of arrows (0.5 T-states later), the value of CMD will not be sampled again. The value of CMD has not changed since WR-PEND rose, so the BCP will read the intended location.



TL/F/10451-8

(c) This timing diagram shows a remote access setting up in time for WR-PEND rising to latch in the proper value of CMD. The only set of arrows shows the BCP sampling the second remote access's CMD value when WR-PEND rises. The value of CMD will not be sampled again. The BCP will carry out the second remote access as it was intended.

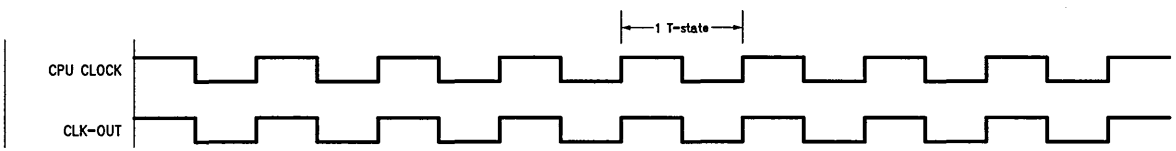
FIGURE 3. Rest Time for Latched Write Mode (Continued)



TL/F/10451-13

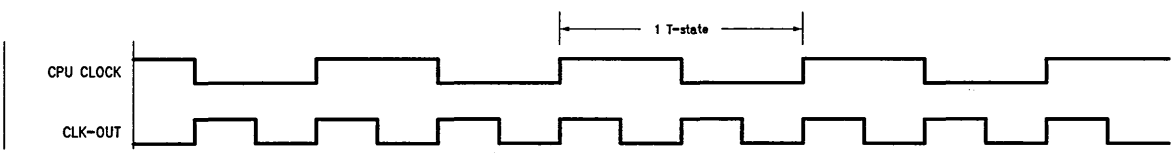
(d) This timing diagram shows a remote access starting after a half T-state plus a hold time since $\overline{\text{WR-PEND}}$ rose. The first set of arrows shows the BCP sampling the value of CMD when $\overline{\text{WR-PEND}}$ rises. The second set of arrows shows the BCP recognizing that no remote access has started and the value of CMD will be sampled for the next remote access. The third set of arrows shows the BCP sampling the correct value of CMD for the second remote access. The BCP will carry out the second remote access as it was intended.

FIGURE 3. Rest Time for Latched Write Mode (Continued)



TL/F/10451-9

(a) BCP Running at Full Speed



TL/F/10451-10

(b) BCP Running at Half Speed

FIGURE 4. Relationship between the BCP's CPU-Clock and CLK-OUT

cle. As a result, at full speed one T-state equals one CLK-OUT cycle, but at half speed one T-state equals two CLK-OUT cycles. (Reference *Figure 4* to see the relationship between the BCP's CPU-CLK and CLK-OUT at full speed and half speed.) The specifications in Table 26 are all measured with the BCP running at full speed. All of the rest time specifications are dependent on the CPU-CLK and not on CLK-OUT. At full speed, the CPU-CLK and CLK-OUT are the same, and this fact allows specifications to CLK-OUT in place of the CPU-CLK. On the other hand, at half speed the specifications to CLK-OUT are no longer valid because one cannot tell if a rising edge of CLK-OUT is a rising or falling edge of the CPU-CLK.

Earlier the worst case rest time for the BCP mistaking two fast back to back accesses as only one was given as:

rest time = 1T + t (setup time) + t (hold time)
(mistaking two accesses as one)

The real time worst case for the BCP mistaking two accesses as one, happens when the BCP runs at half speed. So for the BCP running at half speed and OCLK = 18.8696 MHz, the worst case rest time for mistaking two accesses as one is:

rest time = 2(CLK-OUT cycles) + t_{SU} + t_h
(mistaking two accesses as one)

rest time = 2(53 ns) + 22 ns + 10 ns
(mistaking two accesses as one)

rest time = 135 ns
(mistaking two accesses as one)

Up to a full T-state (or two CLK-OUT cycles) may be added to the above equation if one is using Latched Read or Fast Buffered Write modes. As explained in the CAUSES of Remote Rest Time section, this extra T-state is only added if the remote processor can terminate the remote access quickly after XACK rises (within a T-state). Otherwise, the above equation remains valid as written. The reader should note that this extra T-state is not mentioned or included in the following calculations because it takes place coincidentally with that cause of rest time.

As mentioned previously, the absolute worst case rest time for all modes, except latched write mode, may be calculated by adding the above case of rest time to the second source of rest time caused by fast back to back accesses with different values for CMD. This rest time can be calculated as follows:

rest time = first source + second source
(CMD changes)

rest time = [1T + t (setup time) + t (hold time)]
(CMD changes) + [1.5T + t (hold time)]

Note: The first hold time is during the second source's 1.5 T-states, so in the following formula it disappears.

rest time = 2.5T + t (setup time) + t (hold time)
(CMD changes)

For the BCP running at half speed and OCLK = 18.8696 MHz, the absolute worst case rest time is:

rest time = 5(CLK-OUT cycles) + t_{SU} + t_h
(CMD changes)

rest time = 5(53 ns) + 22 ns + 10 ns
(CMD changes)

rest time = 297 ns
(CMD changes)

For latched write mode the remote rest time starts when $\overline{\text{WR-PEND}}$ rises. The rest time for this case can be calculated as follows:

rest time = 0.5T + t (hold time)
(CMD changes)

The real time worst case for rest time in latched write mode is with the BCP running at half speed. The following is a calculation of this rest time with the BCP running at half speed and OCLK = 18.8196 MHz.

rest time = 1(CLK-OUT cycle) + t (hold time)
(CMD changes)

rest time = 53 ns + 7 ns = 60 ns
(CMD changes)

Please refer to the latest datasheet for more information and the most current specifications.

DESIRABLE FEATURES OF A REST TIME CIRCUIT

In regards to designing with the rest time specifications, the first suggestion is to determine if rest time is an issue in one's design(s). If one's present or future design(s) is for systems which can never violate the rest time specification, the whole issue of rest time is a moot point.

On the other hand, designs such as terminal emulation boards, which may be placed in faster and faster PC buses, must address rest time. In slower PCs one's product may never violate rest time, but in faster PCs rest time may become an issue.

All remote accesses are susceptible to having two fast back to back accesses recognized as only one. The worst case rest time for this was determined earlier as:

rest time = 135 ns
(mistaking two accesses as one)
(where OCLK = 18.8696 MHz and the BCP runs at half speed, [CCS] = 1)

All designs with the BCP must guarantee this minimum amount of time between every access.

The second issue of remote rest time involves fast back to back accesses that have different values for CMD. The worst case for this was also calculated earlier as:

rest time = 297 ns
(CMD changes)
(where OCLK = 18.8696 MHz and [CCS] = 1)

Two ways to handle this rest time issue are:

1. Prevent all remote accesses to the BCP for at least 297 ns after the end of every remote access.
2. Hold off remote accesses that change the value of CMD for a minimum of 297 ns after the last remote access. However, allow remote accesses that do not change the value of CMD to occur a minimum of 135 ns after the last access. (When the value of CMD does not change from one access to the next, this will allow accesses up to 162 ns sooner than option 1).

When designing with rest time one must decide if the increase in speed of option 2) is worth the extra logic. However, as is demonstrated by the design example for the CT-104 (Next section), the increase in logic between option 1) and option 2) may be minimal.

Again, latched write mode is addressed separately. Unlike the other modes, latched write's rest time starts when $\overline{\text{WR-PEND}}$ rises. Two possible design options are:

1. Hold off all remote accesses for at least 60 ns (If OCLK = 18.8696 MHz) after $\overline{\text{WR-PEND}}$ rises. However, doing this will result in slowing every remote access to the BCP. Furthermore, it should be noted that $\overline{\text{WR-PEND}}$ will not rise until a minimum of three T-states after the previous access has ended. If no accesses are allowed until after $\overline{\text{WR-PEND}}$ rises, then the second access will never be mistaken as a continuation of the previous access.
2. Similar to the previous options, allow accesses after 136 ns if CMD has not changed between accesses. Then hold off access for at least 60 ns after $\overline{\text{WR-PEND}}$ rises when CMD changes between accesses.

The last design issue that must be addressed is how to wait the host processor while preventing remote accesses to the BCP. Normally the wait signal of a remote processor is driven by the XACK signal out of the BCP. (Please note that the XACK signal can be active low only when a remote access to the BCP is in progress.) During rest time, the rest time circuit prevents remote accesses to the BCP, so the XACK signal will not wait the remote processor. PC buses specify the maximum amount of time before the bus must be waited (if it is going to be waited). It is possible that not allowing remote accesses to the BCP (during rest time) may delay the XACK signal long enough to violate this bus specification. To prevent this, designs which wait a PC bus, must use logic to wait the bus whenever a remote access begins during rest time. Furthermore, the logic that starts waiting the bus before remote access is allowed to the BCP, must continue to wait the bus until XACK takes over waiting the bus.

DESIGN EXAMPLE FOR THE CT-104

The four major goals in designing a rest time circuit for the CT-104 were:

1. Keep the component count to a minimum.
2. Keep the impact to the original CT-104 design to a minimum.
3. Allow the CT-104 to operate in every mode.
4. Take advantage of the faster accesses allowed when CMD does not change from one access to the next.

The rest time circuit is implemented on one PAL16R4B and one 74ALS74. Only a single signal (REM_enable) is fed back into the original CT-104 design. In addition, the XACK signal from the BCP is now fed into the rest time PAL16R4B and the IO_CHRDY signal to the PC bus is controlled by this PAL®. This rest time circuit implements all modes and takes advantage of the increase in speed possible when CMD does not change from one access to the next.

First, how the REM_enable signal controls remote accesses will be discussed. Then, the functions implemented by the two positive-edge-triggered D flip-flops in the 74ALS74 will be discussed. Finally, a description of the operation of the rest time state machine, in the PAL16R4B, will be given. *Figure 5* is the schematic for the CT-104's rest time circuit.

The REM_enable *(Figure 5)* signal is produced in the rest time PAL16R4B and is low during rest time. After rest time is over the REM_enable signal goes high until the end of the next access, when it once again goes low during rest time.

The signal REM_enable is fed back into U22 (a PAL16L8) on the CT-104. (Note that this PAL had one unused pin so the design of this PAL was only slightly altered.)

On the original CT-104, the $\overline{\text{REMRD}}$ and $\overline{\text{REMWR}}$ outputs of U22 were buffered signals of $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ respectively. With the new rest time circuit both $\overline{\text{REMRD}}$ and $\overline{\text{REMWR}}$ are held high when REM_enable = 0. This prevents all remote accesses during rest time. When rest time is over REM_enable = 1 and once again, $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ control $\overline{\text{REMRD}}$ and $\overline{\text{REMWR}}$ respectively.

One of the D flip-flops in the 74ALS74 stores the value of the previous access's CMD (L_CMD). This value (L_CMD) was latched at the beginning of the previous valid remote access. With this value stored in a flip-flop, the rest time state machine can determine if the present value of CMD has changed since the last remote access.

The other D flip-flop acts as a part of the rest time circuit's state machine. When $\overline{\text{RAE}}$ rises (signaling the end of that access) a one (1) is latched into this flip-flop. This signal (WAIT_START) forces the state machine to move through the next three states in sequence. If this latch is not used, the rest time state machine may also miss the ending of an access if back to back accesses occur within one CLK-OUT cycle plus the setup time for a PAL16R4B's register input. If OCLK = 18.8696 MHz this time will be:

time = 1(CLK-OUT cycle) + t (setup time for PAL16R4B)

time = 53 ns + 20 ns

= 73 ns

This in effect, trades a rest time of 136 ns for one of 73 ns. However, while the output of this latch (WAIT_START, *Figure 5*) equals one, REM_enable will be low and the state machine will be forced to start the rest time states. In the third rest time state the WAIT_START latch is cleared by the $\overline{\text{CLR\_START}}$ *(Figure 5)* signal going low. $\overline{\text{CLR\_START}}$ is produced in the rest time PAL16R4B and $\overline{\text{CLR\_START}}$ equals zero (0) only when in the third rest time state. In this way the WAIT_START signal guarantees the minimal rest time of 136 ns by keeping REM_enable equal to zero through at least three CLK-OUT cycles (i.e., 3[53 ns] = 159 ns if OCLK = 18.8696 MHz).

To describe the operation of the state machine, a state by state description follows. When reading through the states one should remember that the state machine can only change states on the rising edge of CLK-OUT. A flow chart of this state machine is provided as *Figure 6*. *Figure 7* is a PAL program (written in the ABEL program language) for the PAL16R4, rest time PAL. *Figure 8* shows the reduced equations that result for the PAL program given in *Figure 7*.

STATE: IDLE

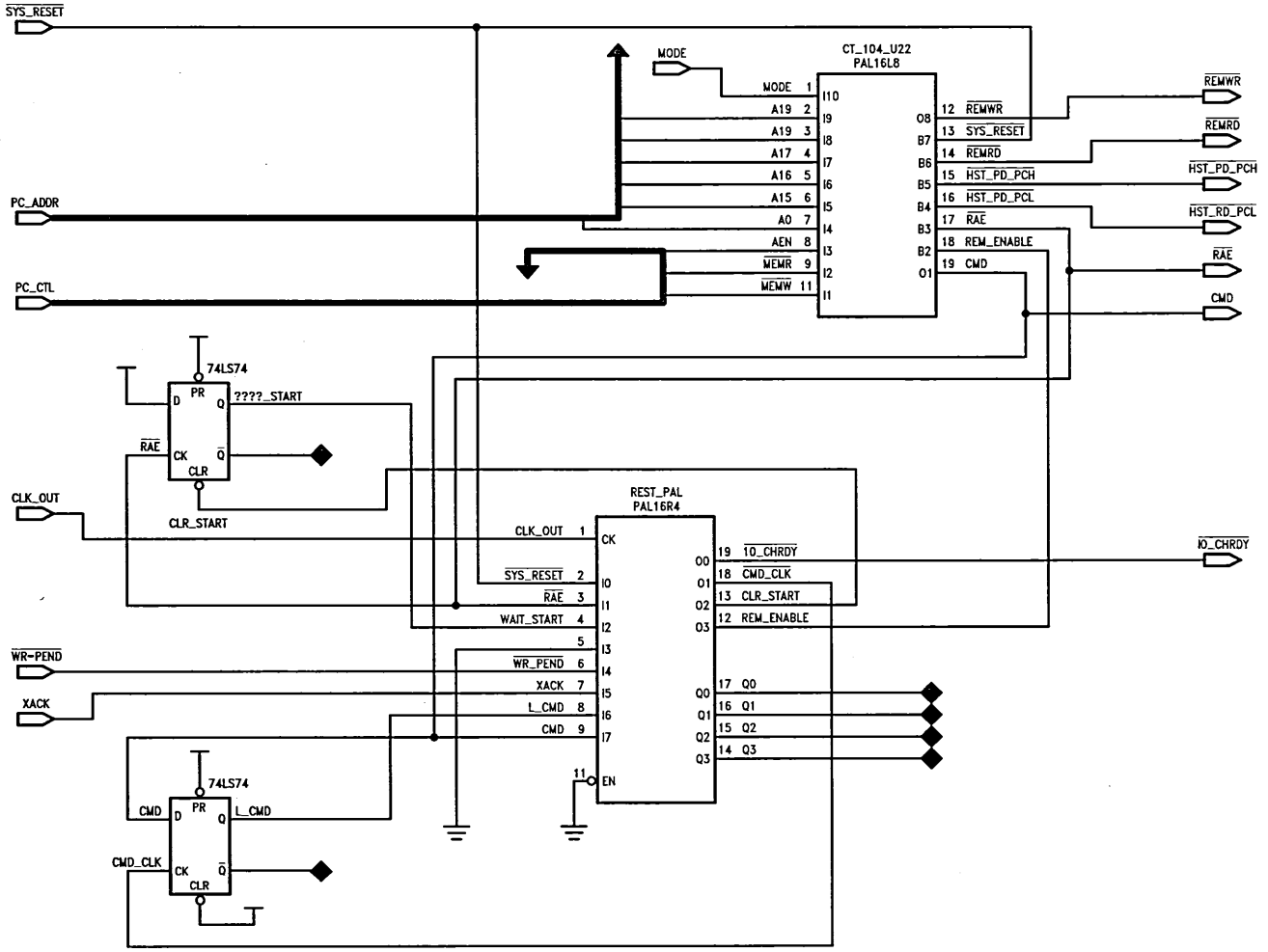
This state is entered when a system reset occurs. In this state REM_enable = 1, CMD_clk = 0, and XACK controls the state of IO_CHRDY.

The state machine will stay in this state until a valid remote access starts (i.e., $\overline{\text{RAE}}$ = 0). Then the state machine moves to CYCLE_START.

Note: On the CT-104, the signal $\overline{\text{RAE}}$ is a full decode of a valid access. This means that it decodes a valid address and a valid $\overline{\text{MEMR}}$ or $\overline{\text{MEMW}}$. If $\overline{\text{RAE}}$ is only an address decode, it alone would not indicate that a valid access had started.

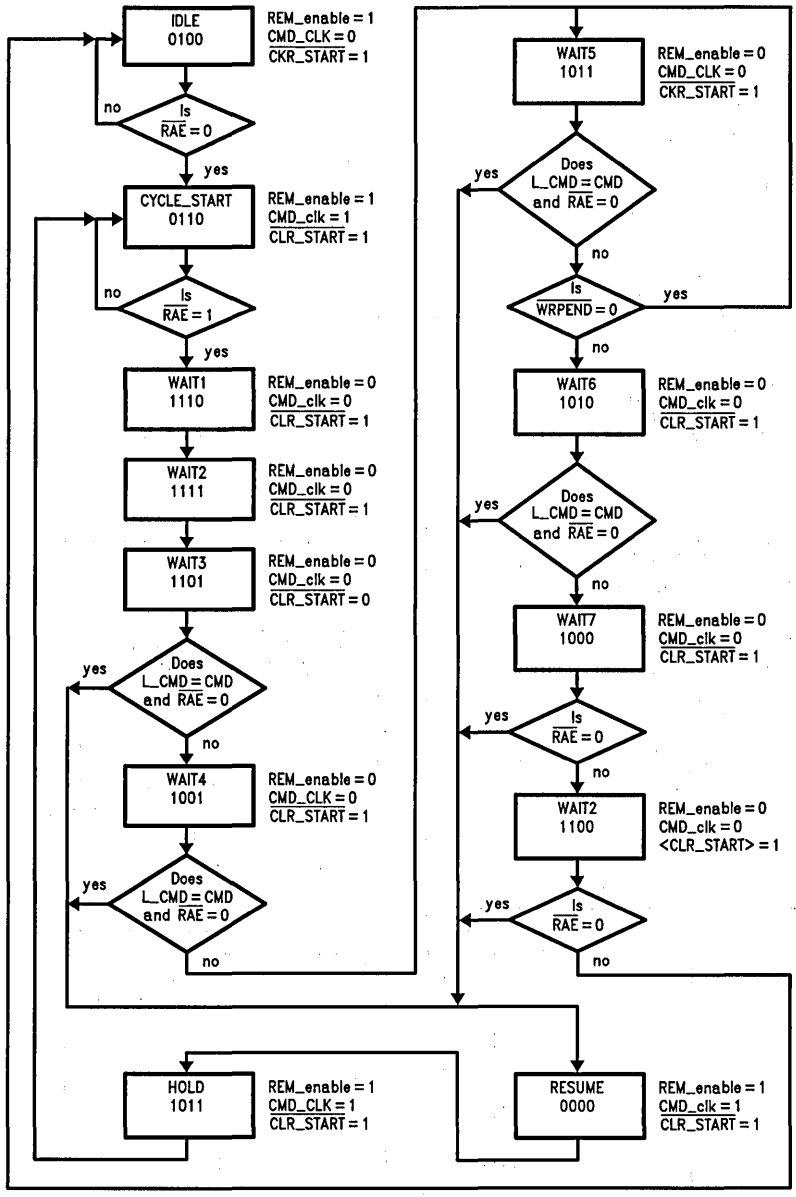
STATE: CYCLE_START

In this state REM_enable = 1, CMD_clk = 1 as long as $\overline{\text{RAE}}$ = 0, $\overline{\text{CLR\_START}}$ = 1, and XACK controls the state of IO_CHRDY. Note, when CMD_clk rises it latches in the present value of CMD. The state machine will stay in this state until the remote access ends, indicated by either $\overline{\text{RAE}}$ = 1 or WAIT_START = 1. Then the state machine moves to WAIT1.



TL/F/10451-11

FIGURE 5



TL/F/10451–12

FIGURE 6. State Diagram of Rest Time Circuit

STATE: WAIT1

In this state REM_enable = 0, CMD_clk = 0, $\overline{\text{CLR\_START}}$ = 1, and if a remote access starts, IO_CHRDY is driven low whenever $\overline{\text{RAE}}$ = 0. While in this state WAIT_START remains equal to one because it has not been cleared yet. Thus, after one CLK_OUT cycle the state machine moves to WAIT2.

STATE: WAIT2

In this state REM_enable = 0, CMD_clk = 0, $\overline{\text{CLR\_START}}$ = 1, and IO_CHRDY is driven low whenever $\overline{\text{RAE}}$ = 0. Again WAIT_START = 1 and after another CLK-OUT cycle the state machine moves to WAIT3.

STATE: WAIT3

In this state REM_enable = 0, CMD_clk = 0, $\overline{\text{CLR\_START}}$ = 0 which clears WAIT_START, and IO_CHRDY is driven low whenever $\overline{\text{RAE}}$ = 0. Since WAIT_START is cleared, on the next rising edge of CLK-OUT the state machine will make a decision:

IF L_CMD equals CMD (indicating no change in the value of CMD between cycles) and a valid remote access has started (i.e., $\overline{RAE}$ = 0), then the state machine will move to the RESUME state. (The RESUME state is covered after the WAIT8 state.) However, if those conditions are not met then the state machine moves to WAIT4.

STATE: WAIT4

In this state REM_enable = 0, CMD_clk = 0, $\overline{CLR\_START}$ = 1, and IO_CHRDY is driven low whenever $\overline{RAE}$ = 0. If L_CMD equals CMD and $\overline{RAE}$ = 0, then on the next rising edge of CLK-OUT the state machine will move to the RESUME state. Otherwise the state machine moves to state WAIT5.

STATE: WAIT5

In this state REM_enable = 0, CMD_clk = 0, $\overline{CLR\_START}$ = 1, and IO_CHRDY is driven low whenever $\overline{RAE}$ = 0. IF L_CMD equals CMD and $\overline{RAE}$ = 0 then the next state will be RESUME.

As long as the above condition is not met and $\overline{WR\text{-}PEND}$ = 0, the state machine will remain in this state. $\overline{WR\text{-}PEND}$ = 0 indicates that the previous access was a write with the BCP in latched write mode. Holding the state machine at WAIT5 prevents remote accesses, that changes the value of CMD, for the required latched write rest time.

If both of the above conditions are false then the next state will be WAIT6.

STATE: WAIT6

In this state REM_enable = 0, CMD_clk = 0, $\overline{CLR\_START}$ = 1, and IO_CHRDY is driven low whenever $\overline{RAE}$ = 0. If L_CMD equals CMD and $\overline{RAE}$ = 0, then on the next rising edge of CLK-OUT the state machine will move to the RESUME state. Otherwise the state machine moves to state WAIT7.

STATE: WAIT7

In this state REM_enable = 0, CMD_clk = 0, $\overline{CLR\_START}$ = 1, and IO_CHRDY is driven low whenever $\overline{RAE}$ = 0. Any remote access that has changed the value of CMD will be prevented until the end of this state. That would be a minimum of seven CLK-OUT cycles between accesses or 371 ns if OCLK = 18.8696 MHz.

Also, all remote accesses which follow a latched write and change the value of CMD have been prevented at least two CLK-OUT cycles or 106 ns, if OCLK = 18.8696 MHz. Thus after one CLK-OUT cycle, if $\overline{RAE}$ = 0 the next state will be RESUME. Otherwise, it will be WAIT8.

STATE: WAIT8

In this state REM_enable = 1, (allows accesses), CMD_clk = 0, $\overline{CLEAR\_START}$ = 1, and IO_CHRDY is driven low

```
module rest_pal        flag '-r3'
title 'REST-TIME Compliance State Machine';

        REST_PAL    device        'p16r4';


"inputs:
        clock,enab     pin 1,11;
        !sys_reset     pin 2;
        !rae           pin 3;
        wait_start     pin 4;
        !wr_pend       pin 6;
        xack           pin 7;
        L_cmd          pin 8;
        cmd            pin 9;

"outputs:
        rem_enable     pin 12;
        clr_start      pin 13;
        q3,q2,q1,q0    pin 14,15,16,17;
        cmd_clk        pin 18;
        IO_chrdy       pin 19;

        sreg       = [q3,q2,q1,q0];
        outputs    = rem_enable;

"definitions:
        ck,x,z,L,H     = .C., .X., .Z.,0,1;
        access         = rae;
        st             = [q3,q2,q1,q0];

"State Values...

        idle       = ^b0100;    " 4h
        start      = ^b0110;    " 6h
        wait1      = ^b1110;    " Eh
        wait2      = ^b1111;    " Fh
        wait3      = ^b1101;    " Dh
        wait4      = ^b1001;    " 9h
        wait5      = ^b1011;    " Bh
        wait6      = ^b1010;    " Ah
        wait7      = ^b1000;    " 8h
        wait8      = ^b1100;    " Ch
        resume     = ^b0000;    " 0h
        hold       = ^b0010;    " 2h
        notused1   = ^b0111;    " 7h
        notused2   = ^b0011;    " 3h
        notused3   = ^b0101;    " 5h
        notused4   = ^b0001;    " 1h
```

TL/F/10451–14

FIGURE 7. PAL Program File (Written in the ABEL Program Language)

```
equations

    enable outputs = 1;

    enable IO_chrdy = access;

    !IO_chrdy    = ( q3 * access ) # (!q2 * access ) # ( q0 * access )
                 # (!xack) # ( wait_start * access );

    !clr_start = ((q3) * (q2) * (!q1) * (q0))
               # (sys_reset);

    cmd_clk    = (access * !q3 * !q2 * !q0 * !wait_start)
               #(access * !q3 * q1  * !q0 * !wait_start)
               # (access * cmd_clk * !wait_start);

    !rem_enable  = (!q2 * q3) # q0 # (q1 * q3) # wait_start;

state_diagram sreg;

      State idle:              " Remain in idle while sys_reset is active.
        IF (sys_reset) THEN idle;
         ELSE IF (access) THEN start;
        ELSE idle;

    State start:          " Begin normal access.
        IF (sys_reset) THEN idle;
        ELSE IF (!access # wait_start) THEN wait1;
        ELSE start;

    State wait1:          " First wait cycle.
        IF (sys_reset) THEN idle;
        ELSE IF (access & L_cmd & cmd & !wait_start) THEN resume;
        ELSE IF (access & !L_cmd & !cmd & !wait_start) THEN resume;
        ELSE wait2;

    State wait2:
        IF (sys_reset) THEN idle;
        ELSE IF (access & L_cmd & cmd & !wait_start) THEN resume;
        ELSE IF (access & !L_cmd & !cmd & !wait_start) THEN resume;
        ELSE wait3;

    State wait3:
        IF (sys_reset) THEN idle;
        ELSE IF (access & L_cmd & cmd & !wait_start) THEN resume;
        ELSE IF (access & !L_cmd & !cmd & !wait_start) THEN resume;
        ELSE wait4;
```

TL/F/10451-15

FIGURE 7. PAL Program File (Written In the ABEL Program Language) (Continued)

whenever $\overline{RAE}$ = 0. This state was included in the state machine to reduce the state machine's logic. Otherwise it would have been logical to return to the IDLE state from WAIT7 if $\overline{RAE}$ = 1 (no access in progress). If $\overline{RAE}$ = 0, then the next state will be RESUME. Otherwise the state machine returns to IDLE.

STATE: RESUME

In this state REM_enable = 1, CMD_clk = 1 (rising edge of CMD_clk latches in the present value of CMD), $\overline{CLR\_START}$ = 1, and IO_CHRDY is driven low while $\overline{RAE}$ = 0. When the state machine moves to this state, it means that a remote access took place quickly after the previous access. The state machine has allowed the remote access to proceed. However, the state machine must have waited the PC-bus for some period of time before entering this state. As a result, the PC-bus should be waited until the XACK signal can take over control of driving IO_CHRDY. For the design of the CT-104, it was determined that once REM_enable = 1, the XACK signal would take over control within two CLK-OUT cycles. So the state machine will wait the PC-bus through this state and the next. On the next rising edge of CLK-OUT the state machine will move to the HOLD state.

STATE: HOLD

In this state REM_enable = 1, CMD_clk = 1, $\overline{CLR\_START}$ = 1, and IO_CHRDY is driven low while $\overline{RAE}$ = 0. Again, this state is provided to wait the PC-bus for a second CLK-OUT cycle while still allowing remote access. The next state is CYCLE_START. In CYCLE_START, XACK will take over control of IO_CHRDY.

```
State wait4:
    IF (sys_reset) THEN idle;
    ELSE IF (access & L_cmd & cmd & !wait_start) THEN resume;
    ELSE IF (access & !L_cmd & !cmd & !wait_start) THEN resume;
     ELSE wait5;

State wait5:
    IF (sys_reset) THEN idle;
    ELSE IF (access & L_cmd & cmd & !wait_start) THEN resume;
    ELSE IF (access & !L_cmd & !cmd & !wait_start) THEN resume;
    ELSE IF (wr_pend) THEN wait5;
    ELSE wait6;

State wait6:
    IF (sys_reset) THEN idle;
    ELSE IF (access & L_cmd & cmd & !wait_start) THEN resume;
    ELSE IF (access & !L_cmd & !cmd & !wait_start) THEN resume;
    ELSE wait7;

State wait7:
    IF (sys_reset) THEN idle;
    ELSE IF (access) THEN resume;
    ELSE wait8;

State wait8:
    IF (sys_reset) THEN idle;
    ELSE IF (access) THEN resume;
    ELSE idle;

State resume:
    IF (sys_reset) THEN idle;
    ELSE hold;

State hold:
    IF (sys_reset) THEN idle;
    ELSE start;

State notused1:
    IF (sys_reset) THEN idle;
    ELSE wait2;

State notused2:
    IF (sys_reset) THEN idle;
    ELSE wait2;

State notused3:
    IF (sys_reset) THEN idle;
    ELSE wait2;

State notused4:
    IF (sys_reset) THEN idle;
    ELSE wait2;

end
```

TL/F/10451–16

FIGURE 7. PAL Program File (Written in the ABEL Program Language) (Continued)

REST-TIME Compliance State Machine
Equations for Module rest_pal

Device REST_PAL

Reduced Equations:

```
enable rem_enable = (1);

enable IO_chrdy = (!~rae);

!IO_chrdy = (!~rae & wait_start
        # !xack
        # q0 & !~rae
        # !q2 & !~rae
        # q3 & !~rae);


!clr_start = (!~sys_reset # q0 & !q1 & q2 & q3);

!cmd_clk = (wait_start
        # !cmd_clk & q0
        # !cmd_clk & !q1 & q2
        # !cmd_clk & q3
        # ~rae);


!rem_enable = (wait_start # q1 & q3 # q0 # !q2 & q3);

!q3 := (!q0 & !q2 & !q3
     # !q0 & !q1 & !~rae
     # !L_cmd & !cmd & q3 & !~rae & !wait_start
     # L_cmd & cmd & q3 & !~rae & !wait_start
     # !q0 & !q3 & !~rae & !wait_start
     # !~sys_reset
     # !q0 & !q1 & q2);


!q2 := (!q0 & !q1 & !q2 & !q3 & ~sys_reset
     # !q1 & q3 & !~rae & ~sys_reset
     # q1 & !q2 & q3 & ~sys_reset
     # q0 & !q1 & q3 & ~sys_reset
     # !L_cmd & !cmd & q3 & !~rae & ~sys_reset & !wait_start
     # L_cmd & cmd & q3 & !~rae & ~sys_reset & !wait_start);

!q1 := (!q0 & !q1 & q3
     # !q0 & !q2 & q3
     # q0 & q2 & q3
     # !L_cmd & !cmd & q3 & !~rae & !wait_start
     # L_cmd & cmd & q3 & !~rae & !wait_start
     # !q0 & !q1 & q2 & ~rae
     # !~sys_reset);
```

TL/F/10451-17

REST-TIME Compliance State Machine
Equations for Module rest_pal

Device REST_PAL

```
!q0 := (!q0 & !q1
     # !q0 & !q2
     # q1 & !q2 & q3 & ~wr_pend
     # !L_cmd & !cmd & q3 & !~rae & !wait_start
     # L_cmd & cmd & q3 & !~rae & !wait_start
     # !~sys_reset
     # !q0 & !q3);
```

TL/F/10451-18

FIGURE 8. Reduced Equations for Rest Time State Machine PAL

DP8344 Timer Application

National Semiconductor
Application Note 626
William V. Miller

INTRODUCTION

The DP8344 is a communications processor which handles IBM 3270, 3299 and 5250 protocols along with NSC general 8-bit protocol. In order to reduce the impact on the DP8344's CPU the timer was designed to stand-alone and count independently of the CPU.

The timer's circuitry includes a unique holding register. This holding register can be loaded with a sixteen-bit countdown-value, which will remain unchanged until a new value is loaded or the DP8344 is reset.

When the timer counts to zero it takes two actions: 1) it sets both the timer interrupt and the Time Out flag [TO], and 2) the timer reloads the sixteen-bit countdown-value stored in the holding register and continues the countdown cycle. This demonstrates a significant advantage of the DP8344's timer; the timer continues keeping accurate time while notifying the CPU that the timer has completed a cycle. The timer does not wait for the CPU to service it, instead the timer notifies the CPU of the completion of a cycle and allows the CPU to take the desired action when it has the time.

With the use of the holding register, a multiple number of timer cycles of the exact same duration can be performed consecutively. The other major advantage of the holding register is that it allows the interleaving of any number of countdown-values. Loading the holding register with a new countdown-value does not affect the countdown-value presently in the timer's countdown circuitry. In this way a countdown-value (call it A) can be counting down and the holding register can be loaded with a new countdown-value (call it B). When the value A reaches zero, both the timer interrupt and Time Out flag [TO] are set, and the value B is loaded into the countdown circuitry and starts its countdown. Then the value A can be loaded back into the holding register when the CPU has the time. This demonstrates how countdown-values with different durations can be interleaved and once again how the timer does not have to wait to be serviced by the CPU, making both the timer and CPU more efficient.

The CPU can load the upper and lower bytes of the holding register by writing the desired value to the CPU registers {TRH} and {TRL} respectively.

Control of the timer's countdown circuitry is maintained via three bits in the Auxiliary Control Register {ACR}.

Timer STart [TST] (bit 7 of {ACR}) is the start/stop control bit for the timer. Writing a one to [TST] starts the timer counting down from the present value in the countdown circuitry. When [TST] is zero the timer stops and the timer interrupt is cleared.

The second control bit is Timer LoaD [TLD] (bit 6 of {ARC}). This bit allows the CPU to immediately load the timer's countdown circuitry with the value in the timer's holding register. This capability is required after the DP8344 is reset; the value in the timer's countdown circuitry will be the reset value and not the desired value. CPU controlled loading can also be used to load higher priority countdown-values before a lower priority countdown is completed. The 5250 Protocol application implements the timer in this manner.

Writing a one to [TLD] will load the timers countdown circuitry with the value in the timer's holding register and initializes the timer clock in preparation to start counting down. Upon completing the load operation [TLD] is cleared by internal hardware.

When the timer is loaded by writing a one to [TLD], the timer is re-initialized to prevent the timer's circuitry from decrementing the newly loaded countdown-value prematurely. By initializing the countdown circuitry after a CPU load, the newly loaded countdown-value's duration will be accurately measured. The reader should note that there is no way to precisely measure the total elapse time of two or more countdown-values if the CPU loads them (using [TLD]) into the countdown circuitry. However, the error due to CPU loading will be a maximum of one period of the timer for each CPU load and can often be ignored if the countdown values are large.

EXAMPLE:	countdown-value	= 1000
	maximum count error	= 1
	maximum error	= 0.1%

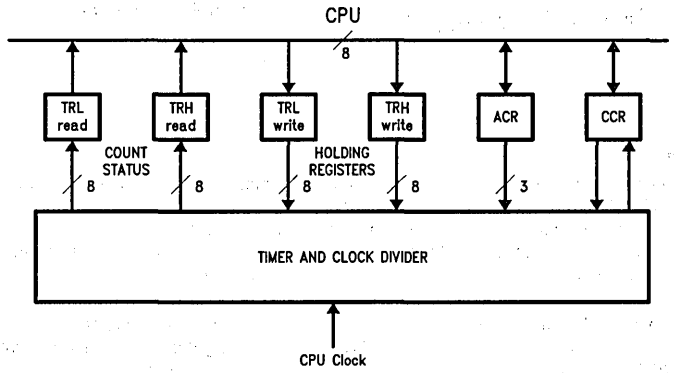
The last control bit is TiMer Clock select (bit 5 of {ACR}). This bit determines the rate at which the countdown-value will be decremented. When [TMC] is low, the timer decrements the countdown-value at one-sixteenth the CPU's clock frequency. When [TMC] is high the rate is one-half the CPU's clock frequency. The reader should note that the timer's decrement rate is based on the CPU's clock frequency, which is controlled by CPU Clock Select [CCS] (bit 7 of {DCR}). When [CCS] is low the CPU's clock frequency equals the oscillator's clock frequency, and when [CCS] is high the CPU's clock frequency equals one-half the oscillator's clock frequency.

The last portion of the timer's circuitry is a sixteen-bit output register. This output register is loaded with the present value of the countdown-value in the countdown circuitry, at the end of every execution cycle. This register is loaded even if the timer is stopped.

The CPU can read the upper and lower bytes of this output register by reading the CPU registers {TRH} and {TRL} respectively.

The reader should note that when the CPU reads and writes to the registers {TRH} and {TRL} the timer's circuitry accesses different registers. All writes will load the timer's holding register and all reads will read the timer's output register.

The count status of the timer can be monitored by reading {TRL} and/or {TRH}. When the registers are read, the value in the timer's output register is presented to the CPU and not the value in the input holding register. To read back what was written to {TRL} and {TRH}, the timer must be loaded first, followed by a one instruction delay before reading {TRL} and {TRH} to allow the output register to be updated after the load operation. *Figure 1* is a block diagram of the Timer-CPU interface.



TL/F/10450–1

FIGURE 1. Block Diagram of Timer-CPU Interface

TIMER OPERATION

This section of the application note reviews the general operation of the timer. Constraints and suggestions for software are included as well as a short review of the timing equations.

After the desired sixteen bit time-out value is written into the timer's holding register via {TRL} and {TRH}, the start, load and clock selection can be achieved in one write to {ACR}. A glitch, which will cause a loss of timer accuracy, may occur if the timer's clock frequency is changed while the timer is running. To prevent this, a restriction exists on changing the timer's clock frequency in that [TMC] should not be changed while the timer is running (i.e., [TST] is high). After the write to {ACR}, the timer starts counting down at the selected frequency starting with the loaded value from the timer's holding register. Upon reaching a count of zero the timer reloads the current word in its holding register and recycles through the count.

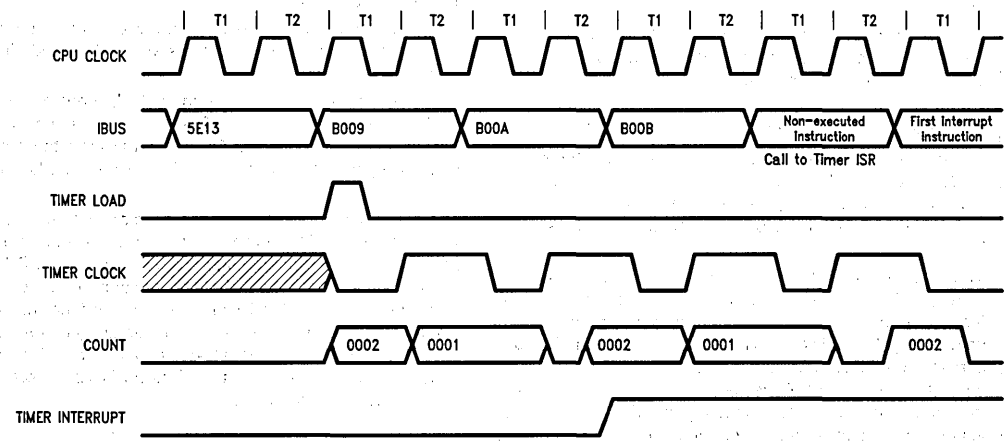
The timing waveforms shown in *Figure 2* show a write to {ACR} that loads, starts and selects the divide by two of the CPU clock rate. The timer interrupt has also been selected. Prior to the write to {ACR}, the holding register in the timer was loaded with 0002 (Hex) by writing 02 (Hex) and 00 (Hex) to {TRL} and {TRH} respectively. The timer interrupt has also been selected.

The timer can be selected as an interrupt source by unmasking it in the Interrupt Control Register {ICR}. This is achieved by writing a zero to bit 4 of {ICR} and asserting the Global Interrupt Enable [GIE] (bit 0 of {ARC}). The timer interrupt is the lowest priority interrupt and is latched and maintained until it is cleared in software. If the timer times out prior to T2 of an instruction, the call to the interrupt service routine will occur in the next instruction. When the time out occurs in T2, the call will occur in the instruction after the next instruction.

The timer may also be used in a polled configuration. This is achieved by masking the timer interrupt bit (i.e., writing a one to bit 4 of {ICR}) and writing software which will poll the [TO] flag (bit 7 of {CCR}). Both [TO] and the timer interrupt are set high when the timer counts to zero.

Then the timer reloads the current word in its holding register and recycles through the count. This means that the timer continues to keep track of time while leaving the task of handling the timer interrupt and/or the [TO] poll to be performed by the CPU. To operate correctly in the polled configuration, software must be written that will guarantee that [TO] is polled and cleared at a rate that prevents [TO] being set twice before it is polled again.

The interface between the CPU and the timer allows only one byte of information to be transferred at a time. This



TL/F/10450–2

FIGURE 2. Timing Waveforms of Timer Operation

prevents the CPU from accessing both {TRL} and {TRH} in the same instruction. Since the timer's output register is updated after every instruction cycle, two consecutive reads of {TRL} and {TRH} will not correspond to the same count status in the timer. This error will be slight except when one of the output register values rolls over or when the count in the timer reaches zero and the timer reloads between instructions.

The suggested software for this situation is to read {TRH}, then read {TRL} and then read {TRH} again. If the values for both reads of {TRH} are the same, then the output register values did not roll over and the timer did not reload. This eliminates the error due to rolling over or reloading, but increases the amount of software.

The reader must be aware that stopping the timer (i.e., writing a zero to [TST]) will clear both the [TO] flag and the timer interrupt. For the case where the timer counts down to zero just prior to or during a stop timer instruction, the [TO] flag and timer interrupt will be cleared before the software can take the desired action. Thus, the information that the timer counted to zero will also be lost. The software to handle this situation should check the [TO] flag one instruction before the stop instruction and then check the value in the timer's output register one instruction after the stop instruction. Checking the [TO] flag before the stop instruction will insure that any previous count to zero will be verified. On the other hand, if the [TO] flag was low and the value in the output register is the same as the value stored in the holding register, then the timer counted to zero and reloaded just prior to or during the stop instruction.

For any value except 0000 (Hex) loaded into the timer's holding register, the following equations can be used to determine the time out delay for that value:

$$\text{Timeout} = (\text{value in the holding register}) \times \text{Tcpu} \times \begin{cases} 2\ ; & [\text{TMC}] = 1 \\ 16; & [\text{TMC}] = 0 \end{cases}$$

where:

Tcpu = The period of the CPU clock
CPU clock = oscillator clock rate ; [CCS] = 0
CPU clock = ½ oscillator clock rate ; [CCS] = 1

Timeout = The amount of time after the end of the instruction that asserts [TST]

When the value of 0000 (Hex) is loaded into the timer, the maximum time out is obtained and is calculated as follows:

$$\text{Timeout} = 65536 \times \text{Tcpu} \times \begin{cases} 2\ ; & [\text{TMC}] = 1 \\ 16; & [\text{TMC}] = 0 \end{cases}$$

With the CPU running with a 18.8 MHz crystal, the maximum single loop time out attainable would be 55.6 ms ([TMC] = 0). The minimum time out with the same constraints is 106 ns ([TMC] = 1). For accumulating time out intervals, the total time out is simply the number of loops accumulated multiplied by the calculated Timeout. The equations above do not account for any overhead for processing the timer interrupt and for precision timing this may need to be included.

INTRODUCTION TO APPLICATIONS

In a communications environment a timer may be needed to determine the appropriate response time, the polling rate of a device or the length of a signal.

The first two applications discussed are for the communications environment.

In the first application the response time for the BCP operating in the 5250 protocol mode is controlled by the timer.

In the second application, the serial input from a keyboard is connected to the DP8344's BIRQ pin and the timer determines at what rate the input is sampled to read in the valid keystroke serial data.

To further demonstrate the timer's versatility the last two applications discuss how to implement basic timer uses not restricted to the communications environment; namely blinking the terminal's cursor and a real time clock.

All four applications implement the timer as an interrupt source, none poll the [TO] flag. Using the interrupt reduces the amount of software needed and it also results in the fastest responses to a time out. However, the reader should note that the [TO] flag may be read even during other interrupt routines while the timer's interrupt is masked off. This may be important if the other interrupt routines are long and could delay the service of the timer interrupt longer than the

time out length. Thus the [TO] flag can be used to guarantee the timer is serviced even during anothers interrupts service routine.

5250 PROTOCOL

Introduction

The DP8344 Biphase Communication Processor (BCP) is capable of responding to received data within 5.5 μs. This is a stringent requirement for the IBM 3270 protocol. However, the IBM 5250 protocol requires a response time of 45 μs ±15 μs. Obviously the powerful BCP will respond too rapidly if it is not programmed to wait at least 30 μs before responding.

Also while operating in the IBM 5250 protocol mode, the BCP often expects to be polled at some minimum rate. In this discussion the BCP expects to be polled within every two seconds. If the BCP is not polled within this time it is assumed that a problem exists and the BCP is programmed to reset.

In this application the timer and some DP8344 software are used to guarantee the proper response time, and to determine how long it has been since the last poll.

General Description

For the majority of time, the timer will be used to keep track of the real-time which has transpired since the BCP was last polled. However, once a receiver interrupt is set, the timer loads and counts down a 45 μs delay value. This count down is used to delay the BCP's response so that it will lie between 30 μs and 60 μs. After the 45 μs delay value is handled, the timer returns to keeping track of the two seconds of real-time.

Resetting the BCP, after two seconds have passed since it was last polled, is not a stringent requirement. Thus the 45 μs delays are not included in the two seconds. In effect a receiver interrupt 1) stops the timer, 2) records the present value of the two second count down value, 3) loads and counts down the 45 μs delay value, and 4) reloads and continues the two second count down from the value recorded after the receiver interrupt stopped the timer.

Detailed Description

After a reset the timer must be programmed to operate in the desired configuration before the BCP can start its operation in the 5250 protocol mode. For this application the timer is pre-configured to divide the CPU clock by sixteen (CPU clock = ½ oscillator clock, OCLK = 18.8696 MHz). All interrupts are unmasked and enabled. The time out value 60BE (Hex) is then loaded into the timer's holding register via {TRL} and {TRH}.

After the timer is programmed properly the timer is loaded and started with one write to {ACR}. The reader will note that the count down value of 60BE (Hex) corresponds to 21 ms not two seconds. As shown earlier in this application note (OPERATION section), the maximum single loop time out attainable for this mode is 55.6 ms. Since it is impossible to load the timer with a countdown-value of two seconds, software is written to record the number of times the 21 ms count down value reaches zero. Still more software is responsible for resetting the BCP if one hundred time outs occur before the BCP is polled again. The use of 21 ms time outs instead of 20 ms time outs will guarantee that a minimum of two seconds has passed, even if there are small timing errors by either the controller or the BCP's oscillator clock.

The timer will continue to count down the 21 ms time outs until a receiver interrupt is set. The BCP's software then calls a receiver interrupt service routine. (Refer to *Figure 3* for the BCP code for this service routine.) The timer is stopped. The present value of the 21 ms count down is stored in a temporary memory location. The time out value 0011 (Hex) which corresponds to 28 μs is loaded into the timer's holding register via {TRL} and {TRH}. (Adding the delay due to setting up and responding to the timer together with the 28 μs time out, results in a total elapse time delay which guarantees the response between 30 μs and 60 μs.) The timer is loaded and restarted. Then the partially completed 21 ms countdown-value stored in a temporary memory location is loaded into the timer's holding register via {TRL} and {TRH}. Once the 28 μs time out counts to zero, the partially completed 21 ms time out is resumed as the timer is loaded with the value in the holding register and continues the 21 ms count down.

Every time the timer counts down to zero, it sets the timer interrupt and the DP8344 is programmed to call a timer interrupt service routine. In order to operate correctly the service routine must first determine if a 21 ms or a 28 μs time out has occured. If a 28 μs time out has taken place, the timer is stopped. The value in the timer's holding register will not be the 21 ms count down value; it will be the value which was in the timer when the receiver interrupt stopped the timer. So the 21 ms countdown-value 60BE (Hex) is loaded into the timer's holding register via {TRL} and {TRH}. Then the timer is started. If a 21 ms timer interrupt is pending it will be serviced, otherwise the software will return with all interrupts unmasked and enabled.

In the case of a 21 ms timer interrupt, the number of 21 ms time outs is recorded for all seven sessions in data memory. For every 21 ms timer interrupt a one is added to the value stored in data memory for each session. An exception is made when FF (Hex) is the value stored in data memory. Adding a one would result in the value 00 (Hex) replacing FF (Hex) in memory. This would falsely indicate that less than 21 ms has passed since the BCP was last polled. As a result if FF (Hex) is the number in memory nothing is added to it. As before the software returns with all interrupts unmasked and enabled.

The software which 1) clears the number of 21 ms time outs recorded when the BCP is polled and 2) resets the BCP after two seconds have passed without the BCP being polled, is not discussed in this application note because it does not effect the normal operation of the timer.

This application describes the use of the timer in the 5250 protocol mode as it is implemented in the Multi-Protocol Adapter (MPA).

```
;-----------------------------------------------------------------
;       name: tw_timer_int
;
;        description: The timer interrupt service routine is responsible
;                     for:
;                      1) Maintains a real time clock counter for each session:
;                         - Increments a real time clock counter which controls
;                          System Available flag, auto reset and reset
;                        complete;
;                         - Prevents counter roll over by keeping a max count
;                       of FFh;
;                       2) Provides 45us time out signal for poll response
;                         - If interrupt is due to 45us poll response, unmasks
;                        Tx int to
;                         allow for response.
;                 note : The timer interrupt service routine lock out host
;                       access and
;                       other interrupts except TFE interrupt.
;      scope: global
;
;        entry: timer interrupt hits, ie. timer reaches a count of zero.
;               the timer is pre-configured to use 1/16 cpu clock with a
;               count value of 305Fh which corresponds to 21ms.
;        inputs: 1) tw_sysa_por_cnt(0-6)
;                   real time clock counters, reset to 0 by receiver when
;                 Poll received, and
;                 by session task when going to do a POR.
;               2) tw_sysa_resp_flag (in RSTATE)
;               - TW_TIMER_RESP
;                    timer response flag, set by receiver for 45us poll ]
;                 response.
;                - TW_TO_PEND
;                    timer interrupt pending flag, set by receiver if it sees
;                   a pending
;                   timer interrupt.
;
;      exit:
;        outputs: 1) tw_sysa_por_cnt(0-6)
;                   for all sessions, counters are incremented by 1.
;                  Counters will remain in 'FF' without roll over.
;               2) tw_sysa_resp_flag (in RSTATE)
;                 - TW_TIMER_RESP
;                    reset if interrupt is due to 45us poll response.
;               - TW_TO_PEND
;                    reset if there is a pending timer interrupt.
;
```

TL/F/10450–3

FIGURE 3

```
;
;       COPYRIGHT NATIONAL SEMICONDUCTOR, INC. 1987,1988
;-----------------------------------------------------------
;
;               pseudo code
;
;tw_rel_time_clk_timer()
;
;{
;       lock out remote access
;        reset time-out flag by setting [TO] to 1;
;        if (time-out of 45us)
;       {
;               stop timer;
;               reload timer input register;
;               start timer;
;                   allow poll response by unmasking transmitter interrupt;
;                reset poll response flag;
;               if NOT (timer interrupt pending)
;               {
;                       call check birq;
;               enable interrupt;
;                       return with flags and reg banks restored;
;               }
;       enable interrupt;
;       }
;       push regs being used in following section;
;        for all 7 sessions do
;       {
;                if tw_sysa_por_cnt = 5sec next session
;              else
;              {
;                       increment tw_sysa_por_cnt;
;                     next session;
;              }
;       }
;        restore registers;
;         set interrupt mask to enable all interrupts;
;       call check birq;
;        return with flags and reg banks restored;
;}
;
;-----------------------------------------------------------------
```

TL/F/10450-4

FIGURE 3 (Continued)

```
;
        .GLOBAL dca_fast_birq, tw_tx_tm_entry

TW_TIMER.BCP:        .SECT  X


  tw_timer_int::

          exx     MA,AB                    ; switch reg bank
          PUSHP   IZ                       ; save IZ
          or      CCR_TO,CCR               ; clear time out of timer
          LJMPBP  RSTATE,TW_TIMER_RESP,NS,tm_relti_clk
                                           ;jump to real time clock counter

; timer poll/activate read response timeout and offline response
; timing

          and  ~ACR_TST,ACR                ; stop timer
          move        TM_21MS_HI,ACC       ; prepare timer input
                                              upper byte
          move        ACC,TRH              ; move to TRH
          move        TM_21MS_LO,ACC       ; prepare timer input
                                              lower byte
          move        ACC,TRL              ; move to TRL
          or     ACR_TST,ACR               ; start timer
          LJMPBP      RSTATE,RX_RESPONSE_WAIT,S,tm_skip_tfe
                                           ; if offline response
                                             timeout, skip tfe call
          and     ~ICR_TX,ICR              ; unmask Tx interrupt
                                             since interrupt expects
                                             to be unmasked
          lcall  tw_tx_tm_entry            ; go handle response
                                             via TFE interrupt
tm_skip_tfe:
          LJMPBP      RSTATE,TW_TO_PEND,S,tm_relti_clk_1
                                           ; jump to real time clock
                                             if interrupt pending
          and    ~(TW_TIMER_RESP|RX_RESPONSE_WAIT),RSTATE;
                                           ; reset poll response flag
          jmp  tm_check_birq               ; go check birq, do birq
                                             if needed [V0.5]


; real timer clock counter

tm_relti_clk_1:
   and ~(TW_TIMER_RESP|RX_RESPONSE_WAIT|
        TW_TO_PEND),RSTATE
                                  ; reset poll response, response
                                    wait, and int pending FLAGS
tm_relti_clk:
          move    DCPHI,IZHI                ; setup IZHI
          move     LOW(tw_sysa_por_cnt0-1),ACC      ; setup IZLO
          move        ACC,IZLO             ;
          move        1,ACC                ; set a '1' for
                                             later use
          move    [+IZ],GP7                ; get counter
          cmp     GP7,TM_5SEC              ; equal to 5.4sec?
          jz      tm_next_1                ; yes, goto next session
                                             without counter +1
          adda    GP7,[IZ]                 ; increment counter
```

TL/F/10450-5

FIGURE 3 (Continued)

```
tm_next_1:
        move    [+IZ],GP7           ; get counter
        cmp     GP7,TM_5SEC          ; equal to 5.4sec?
        jz      tm_next_2           ; yes, goto next session
                                      without counter +1
        adda    GP7,[IZ]            ; increment counter
tm_next_2:
        move    [+IZ],GP7           ; get counter
        cmp     GP7,TM_5SEC         ; equal to 5.4sec?
        jz      tm_next_3           ; yes, goto next session
                                      without counter +1
        adda    GP7,[IZ]            ; increment counter
tm_next_3:
        move    [+IZ],GP7           ; get counter
        cmp     GP7,TM_5SEC         ; equal to 5.4sec?
        jz      tm_next_4           ; yes, goto next session
                                      without counter +1
        adda    GP7,[IZ]            ; increment counter
tm_next_4:
        move    [+IZ],GP7           ; get counter
        cmp     GP7,TM_5SEC         ; equal to 5.4sec?
        jz      tm_next_5           ; yes, goto next session
                                      without counter +1
        adda    GP7,[IZ]            ; increment counter
tm_next_5:
        move    [+IZ],GP7           ; get counter
        cmp     GP7,TM_5SEC         ; equal to 5.4sec?
        jz      tm_next_6           ; yes, goto next session
                                      without counter +1
        adda    GP7,[IZ]            ; increment counter
tm_next_6:
        move    [+IZ],GP7           ; get counter
        cmp     GP7,TM_5SEC         ; equal to 5.4sec?
        jz      tm_next_end         ; yes, goto next session
                                      without counter +1
        adda    GP7,[IZ]            ; increment counter

tm_next_end:
tm_check_birq:                      ; following codes added
                                      in [V0.5]
        ljmp    CCR,BIRQ,S,tm_no_birq ; check pending
                                      birq
        lcall   dca_fast_birq       ; yes, go do it
tm_no_birq:
        POPP    IZ                  ; restore Z


        UNLOCK                      ; unlock remote
        ret     RI,RFB              ; return with GIE, ALU flags
                                      and reg bank restored

        .end
```

TL/F/10450–6

FIGURE 3 (Continued)

DP8344 AS A SERIAL INPUT FROM A KEYBOARD

Introduction

To keep the cost of terminals low, the 3270 protocol was designed to place all of the intelligence of the system in the cluster controller while all of the memory remained in the terminal. In this protocol the terminal is responsible for recording all keystrokes until the cluster controller can poll the terminal and process the keystroke data.

With that in mind this application uses the timer along with the BIRQ interrupt pin as a serial port to read in keystrokes from a serial keyboard. The timer in this application is used to read the serial keystroke dataword at the proper baud rate. (Refer to *Figure 4* for the actual BCP code used for this application.)

Description

The specifications for the serial dataword produced by the serial keyboard and read by the DP8344 are as follows: it is 1) asynchronous, 2) ten bits long (1 startbit, 8 databit with the most significant bit first, and 1 stopbit), and 3) transmitted at 1200 baud. When no serial data is being transmitted

the serial data line will be held high. The start bit will be a zero to indicate the beginning of the serial bit string.

As mentioned in the introduction, the serial data line from the keyboard is connected to the BIRQ interrupt pin (Pin 53 on the DP8344). The BIRQ interrupt pin acts as the serial port through which the serial keystroke dataword is read into the DP8344.

First, BCP software programs the timer to determine the baud rate at which the DP8344 will read the serial dataword presented at the BIRQ pin. The timer is pre-configured to divide the CPU clock by two (i.e., [TMC] = 1) with the CPU clock set equal to the oscillator clock at 18.8696 MHz (i.e., [CCS] = 0). The time out value of 03D7 (Hex) is loaded into the timer's holding register via {TRL} and {TRH}. The time out value of 03D7 (Hex) corresponds to 0.104188748 ms or approximately one eighth of 0.833333 . . . ms, which is the period of one bit at 1200 baud. After the holding register is loaded, the timer is loaded but not started. Both the timer and BIRQ interrupts are unmasked and enabled. Now the DP8344 is ready to read an asynchronous, ten bit long serial keystroke dataword at 1200 baud via its BIRQ interrupt pin.

After the timer is configured the DP8344's software can perform other operations until a zero on the serial data line activates the BIRQ interrupt (Note: the BIRQ interrupt is active low). The software then jumps to a BIRQ interrupt service routine. The service routine will mask off the BIRQ interrupt and start the timer and then return to perform other operations. After four consecutive timer interrupts the middle of the startbit should be present at the BIRQ pin. To ensure that a glitch or noise did not produce a zero momentarily and that the zero is actually a startbit, the BIRQ interrupt is unmasked. If the value at the BIRQ pin is a one instead of a startbit zero, the timer is stopped and reloaded with the countdown-value 03D7 (Hex). The BIRQ interrupt will remain unmasked waiting for the next zero. However, if the value at the BIRQ pin is a zero (indicating a valid startbit), the software jumps to the BIRQ interrupt service routine. The BIRQ interrupt is masked off and the timer continues to run and the software returns to perform other operations.

For the case of a true startbit the DP8344 needs to read in the value of the serial keystroke dataword. The value of each data bit must be read one at a time. After each value is read, it is added to a temporary value stored in a register. Once the value of the keystroke dataword has been calculated it is transferred to data memory and the temporary register is cleared so that the next keystroke value may be calculated there. The following is a more detailed description of this process, starting in the middle of the start bit.

After eight more timer interrupts the middle of the first and most significant data bit is present at the BIRQ pin. So the software unmasks the BIRQ interrupt. If a zero is present at the BIRQ pin, the software calls the BIRQ interrupt service routine. The BIRQ interrupt is masked off and nothing is added to the value in the temporary register. After masking off the BIRQ interrupt the software returns to perform other operations. On the other hand, if the value at the BIRQ pin is a one, the BIRQ interrupt is masked off and the value 80 (Hex) (Most Significant Bit) is added to the value (initially 00 (Hex)) in the temporary register.

Likewise, after eight more timer interrupts the middle of the second serial databit is present at the BIRQ pin. So the BIRQ interrupt is unmasked and goes through the same procedure as above to decide if 40 (Hex) should be added to the value in the temporary register. Similarly this method will continue for the next six data bits. After the least significant bit has been evaluated, the value in the temporary register is moved to data memory. The reader should realize that this value can also be stored in another register if desired. Then the temporary register is cleared so that the next keystroke value can be recorded there.

The software continues to mask off the BIRQ interrupt until the end of the stopbit. At the end of the stopbit the timer is stopped and the time out value 03D7 (Hex) is loaded into the timer. The BIRQ interrupt is unmasked to wait for the next start bit zero.

This application has demonstrated how the timer and the BIRQ input/output pin can be used as a serial input. However there should be a note of warning that a production program should sample the serial input signal more than once every bit-time to guarantee valid data at a given baud rate. Furthermore, the software for the DP8344 must guarantee that the timer and/or BIRQ interrupts are not masked off by higher priority interrupts for too great a time; this could delay the sampling of the serial input signal for more than a bit-time, resulting in invalid data being read.

```
;-----------------------------------------------------------------
;;
;       This program will receive serial data using the BIRQ pin as
;     a serial input pin.
;       The timer will be used to determine when the middle of a bit
;     is present at the BIRQ pin. Then the value of the bit is sampled
;     with the use of the BIRQ interrupt along with software to decide
;     if the bit is a one or a zero. Then the software takes the
;     appropriate action for each case.
;       In this program all keystoke values are stored in consecutive
;     memory locations.
;
;   ******   National Semiconductor  Copyright 1988   *****
;-----------------------------------------------------------------
        .input          "stdequ.hdr"

CODE:                   .sect  x
initialization:
                exx     AA,AB,DI
                move    5Fh,DCR         ;set CPU-CLK equal to OCLK
                move    02h,IBR         ;set up interrupts
                exx     MA,MB,DI
                move    0E7h,ICR        ;unmask timer and BIR interrupts
                move    10,IWLO         ;clear IW
                move    00,IWHI
                move    0,IX            ;clear IX
                move    0,GP0           ;clear temporary registers
                move    0,GP1
                move    0,GP2
                move    0,GP3
                move    0,GP4
                move    0,GP5
                move    GP5,IZLO        ;load IZ with
                move    1,GP6           ;base address in data memory
                move    GP5,IZHI        ;for bit constant values
                move    80h,GP7         ;storing constants for
                move    GP7,[IZ+2]      ; the most significant bit
                move    40h,GP7
                move    GP7,[IZ+3]      ; bit 6
                move    20h,GP7
                move    GP7,[IZ+4]      ; bit 5
                move    10h,GP7
                move    GP7,[IZ+5]      ; bit 4
                move    08h,GP7
                move    GP7,[IZ+6]      ; bit 3
                move    04h,GP7
                move    GP7,[IZ+7]      ; bit 2
                move    02h,GP7
                move    GP7,[IZ+8]      ; bit 1
                move    01h,GP7
                move    GP7,[IZ+9]      ; the least significant bit
                move    0D7h,GP7
                move    GP7,TRL         ;load timer's holding register
                move    03h,GP7         ;with count down value
```

TL/F/10450–7

FIGURE 4

```
            move  GP7,TRH
            move  61h,ACR          ;load timer
                  ;END of initialization
back:
            cmp   GP0,0
            jz    back             ;waiting for BIRQ interrupt
            cmp   GP0,0
            jz    back             ;protection
            cmp   GP1,0            ;Is this a true start bit?
            jnz   next_1
            move  0,GP2            ;NO, then clear GP2
            jmp   next_2
next_1:
            move  0EFh,ICR         ;mask off the BIRQ interrupt
            move  GP1,GP4
            move  [IZ+A],GP4       ;load value of present bit
            adda  GP5,GP5          ;add value of present bit
            cmp   GP1,9            ;Is this bit 0?
            jnz   next_2
            move  GP5,[IW+]        ;YES, store byte of imformation
            move  0,GP5            ;clear temporary registers
            move  0,GP6
next_2:
            move  0,GP0
            ljmp  back

      ;Timer Interrupt Service Routine
      ;*******************************
tm:
            or    80h,CCR          ;clear [TO] flag
            cmp   GP6,0            ;Is GP6 = 0?
            jnz   next_10
            add   1,GP3            ;loop until the stop
            cmp   GP3,14h          ;bit has passed
            jnz   next_11
            move  60h,ACR          ;stop and load timer
            move  0,GP1            ;clear GP1
            move  0,GP2            ;clear GP2
            move  0,GP3            ;clear GP3
            move  1,GP6            ;set GP6 = 1
            move  0E7h,ICR         ;unmask the BIRQ interrupt
next_11:
            ret   RI
next_10:
            cmp   GP1,0            ;Is this the start bit?
            jnz   next_12
            add   1,GP3            ;YES, add one to GP3
            cmp   GP3,4            ;Is it the middle of the
            jnz   next_13          ;start bit?
            move  1,GP0            ;YES, set GP0 = 1
            move  0,GP3            ;clear GP3
            move  0E7h,ICR         ;unmask BIRQ interrupt
next_13:
            ret   RI
```

TL/F/10450–8

FIGURE 4 (Continued)

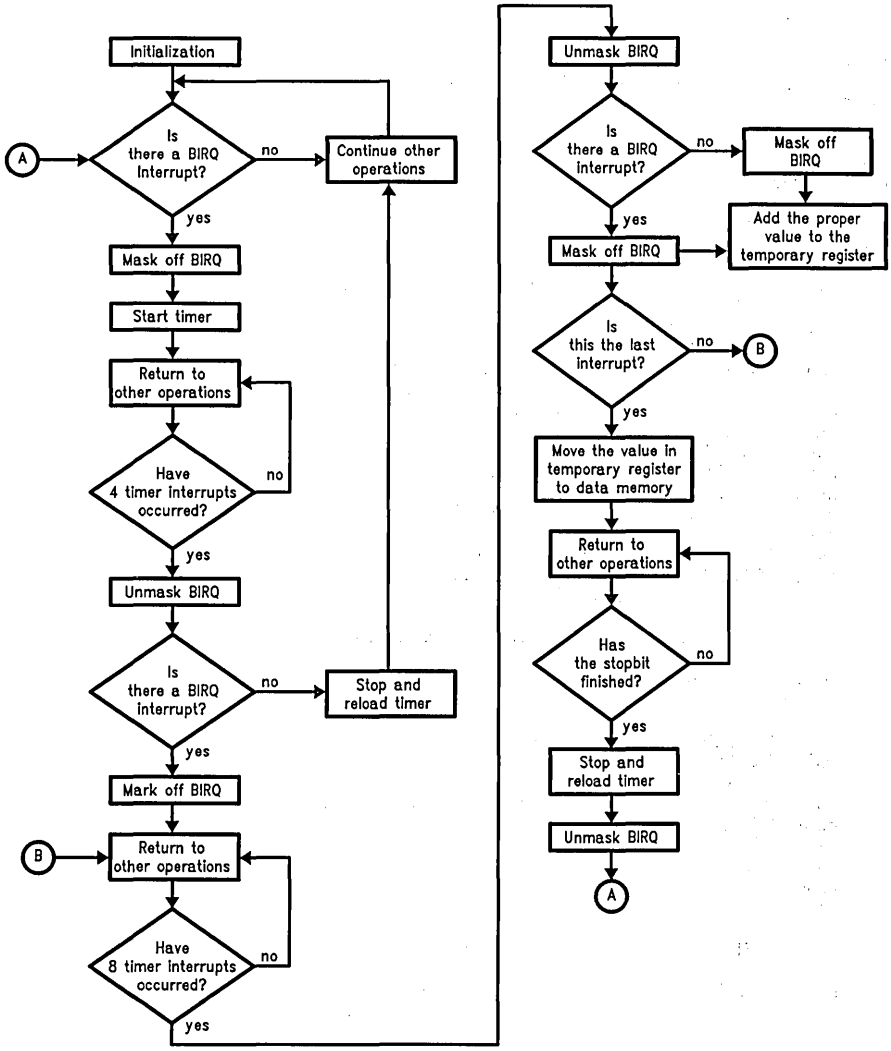
```
next_12:
            add     1,GP3
            cmp     GP3,08h             ;Is it the middle of a
            jnz     next_14             ;data bit?
            move    1,GP0               ;YES, set GP0 = 1
            add     1,GP1               ;update which bit it is
            move    0,GP3               ;clear GP3
            move    0E7h,ICR            ;unmask the BIRQ interrupt
next_14:
            ret     RI

        ;BIRQ Interrupt Service Routine
        ;******************************
bq:
            move    0EFh,ICR            ;mask off the BIRQ interrupt
            move    0,GP0               ;clear GP0
            cmp     GP1,0               ;Is this the start bit?
            jnz     next_20
            cmp     GP2,0               ;YES, is this the first
            jnz     next_21             ;indication?
            move    0A0h,ACR            ;YES, start the timer
            move    1,GP2               ;set GP2 =1
            ret     RI
next_21:
            move    1,GP1               ;set GP1 = 1
next_20:
            cmp     GP1,9
            jnz     next_22
            move    GP5,[IW+]
            move    0,GP5
            move    0,GP6
next_22:
            ret     RI

CODE:           .sect   ax
                .org    210h
                ljmp    bq
                .org    214h
                ljmp    tm
            .END
```

TL/F/10450-9

FIGURE 4 (Continued)



TL/F/10450–10

FIGURE 5. A Flow Chart of the Basic Application of the DP8344 as a Serial Input Keyboard

```
;-----------------------------------------------------------------
;
;     This is the very simple cursor program for the BCP.
;     The timer along with the software toggles the state of the
;   cursor every 200msec. The state of the cursor is stored in data
;   memory.
;
;-----------------------------------------------------------------
    .input        "stdequ.hdr"

CODE:           .sect  x
initialization:
            exx            AA,AB,DI
            move  5Fh,DCR            ;set CPU-CLK equal to OCLK
            move  01,IBR             ;set up interrupt
            exx            MA,MB,DI
            move  0EFh,ICR           ;unmasktimer interrupt
            move  0,GP0              ;clear temperary register
            move  0,GP1              ;clear temperary register
            move  0,GP2              ;clear temperary register
            move  GP2,IZLO           ;clear IZ
            move  GP2,IZHI
            move  GP2,[IZ+0]         ;clear data memory location
            move  5Ch,GP2
            move  GP2,TRH            ;load high bit of the timer
            move  23h,GP2
            move  GP2,TRL            ;load low bit of the timer
            move  0C1h,ACR           ;load and start timer
                              ;END of initialization
loop:
            jmp           loop                ;wait for timer interrupt

        ;Timer Interrupt Service Routine
        ;*******************************
dest:
            or    80h,CCR            ;clear [TO] flag
            add   1,GP0
            cmp   GP0,0Ah            ;Has 200msec passed?
            rnz   R                          ;NO, return with interrupts on
            move  0,GP0              ;YES,
            cmp   GP1,0              ;Toggle
            jnz   next               ;    the
            move  1,GP1              ;        value
            jmp   send               ;              of
next
            move  0,GP1              ;                the
send:
            move  GP1,[IZ+0]         ;                    cursor
            ret   RI                 ;return with interrupts enabled

CODE:           .sect        ax
                .org         114h
                ljmp         dest
            .END
```

TL/F/10450-11

FIGURE 6

BLINK THE CURSOR

Introduction

Blinking the cursor is performed on virtually all computers. With its powerful CPU and programmable timer, the DP8344 can easily implement this basic function without any additional components.

In this application the timer along with a small amount of software will turn the cursor on and off at a periodic rate. The following is a description of the way the timer is programmed and the DP8344's software used to implement this function. (Refer to *Figure 6* for the actual BCP code for this application.)

Description

The following is one of many ways to perform the blinking cursor operation.

First, timer must be programmed to operate in the desired configuration. For this application the timer is pre-configured to divide the CPU clock by sixteen with the CPU clock set equal to the oscillator clock at 18.8696 MHz (i.e., [CCS] = 0). The timer interrupt is unmasked and enabled. The time out value of 5C23 (Hex) is loaded into the timer's holding register via {TRL} and {TRH}.

After the timer is programmed properly, the timer is loaded and started by writing to {ACR}. Once the timer is loaded

and started, it will continually cycle through the time out value of 5C23 (Hex), which corresponds to 20 ms. When the timer counts down to zero, it will set the timer interrupt and the BCP's software is programmed to call a timer interrupt service routine. The timer interrupt service routine updates and stores the state of the cursor in data memory.

State of Cursor: 0 → Cursor is OFF
1 → Cursor is ON

After ten timer interrupts, the service routine will toggle the state of the cursor. Thus, the cursor will blink 2.5 times a second.

Blinking the cursor every 200 ms is not a stringent requirement. As a result the 5250 protocol's 45 μs delays may be interleaved with the 20 ms time outs. This once again demonstrates that the flexibility of the timer can enhance the performance of two functions at the same time.

REAL-TIME CLOCK

Introduction

An added feature on most personal computers is a real-time clock. The clock is used to provide the time, the day, the month and the year. With the BCP's programmable timer and powerful CPU, this clock function can be performed by the DP8344 without any additional components.

In this application the timer along with a small amount of DP8344 software keeps track of the time. Software also allows the user to set the initial time, then the DP8344's timer and software takes over and accurately keeps track of the time. The following is a description of the way the timer is programmed and the DP8344's software used to implement a real-time clock. (Refer to *Figure 7* for the actual BCP code for this application.)

Description

The following describes one basic way to implement a real-time clock using the BCP.

First, the timer must be programmed to operate in the desired configuration. For this application the timer is pre-configured to divide the CPU clock by sixteen with the CPU clock set equal to the oscillator clock at 18.8696 MHz (i.e., [CCS] = 0). The timer interrupt is unmasked and enabled. The countdown-value of 5C23 (Hex) is loaded into the timer's holding register via {TRL} and {TRH}.

After the timer is programmed properly the timer is loaded and started by writing to {ACR}. Once the timer is loaded and started, it should remain on and continually cycle through the time out value of 5C23 (Hex), which corresponds to 20 ms exactly. When the timer counts down to zero, it sets the timer interrupt and the CPU is programmed to call a timer interrupt service routine.

The timer interrupt service routine is very basic. The number of seconds, minutes, hours, days and years are all recorded in separate data memory locations. The service routine will add one to the seconds value after fifty timer interrupts. After sixty seconds, the seconds value is reset to zero and a one is added to the minutes value. After sixty minutes, the minutes value is reset to zero and a one is added to the hours value. Likewise the number of hours, days and years are recorded in a similar manner.

As mentioned in the introduction, in order to set the present date and time after powering up requires software which allows the user to define the present time and date. This software would be remote processor software, not DP8344 software. This remote processor's software should allow the user to enter the present time and date, then this software must transform the entered time and data into data which can be transferred to the real-time clock's data memory locations. This starts the clock at the entered time, and the timer and DP8344 software will be responsible for updating the clock accurately.

The reader may desire a clock which records time in increments as small as a hundredth of a second. In this case the timer should be programmed to count down 10 ms time outs, and another data memory location must be used to record the number of these hundredths of a second.

For the application of a real-time clock, the timer cannot interleave two timing values as in the 5250 Protocol application. The timer must be pre-configured and allowed to run without interruption. Otherwise, timing errors will occur and the clock will not record time accurately.

However, the reader may notice that the BLINK THE CURSOR application uses the same time out value (i.e., 5C23 (Hex)) as the real-time clock. This demonstrates how the BCP can be programmed to use one countdown-value to implement two desirable functions without effecting the performance of either operation.

A final warning to the reader. The oscillator clock must be extremely accurate for this application. For the program provided, and error of 0.0002 MHz in the oscillator clock (OCLK = 18.8696 MHz) will result in an error of 0.916 seconds a day or 5 minutes 34 seconds per year. The best way to prevent timing problems is to accurately measure the oscillator clock frequency first, then calculate and implement all time out values based on that measurement.

```
;-----------------------------------------------------------------
;
;       This is the third version of the real-time clock.
;       This version like the second uses the timer interrupt to make
;   service calls, instead of polling the TO flag (bit 7 of CCR)
;   to see when the timer has counted to zero. The timer is pre-
;   configured to use CPU-CLK/16 and the CPU-CLK is set equal to
;   OCLK (oscillation clock, in this case 18.8696 MHz). The countdown
;   value is 5C23 Hex, which corresponds to 20 ms.
;       The IW register is incremented every 20 ms interrupt until
;   it contains 32(Hex) or 50(Dec), which corresponds to every second
;   exactly. Then the IX register is incremented.
;       Unlike version 2 this version uses data memory to store and
;   record the time that has elapsed. The following table gives
;   the memory locations of the stored time values.
;
;               value           |  memory location (HEX)
;               ----------------|-----------------------
;               seconds         |       00 40
;               minutes         |       00 30
;               hours           |       00 20
;               days            |       00 10
;               years           |       00 00  (not implimented in program)
;-----------------------------------------------------------------
        .input          "stdequ.hdr"

CODE:           .sect   x
initialization:
                exx     AA,AB,DI
                move    5Fh,DCR         ;set CPU-CLK equal to OCLK
                move    01,IBR          ;set up interrupt
                exx     MA,MB,DI
                move    0EFh,ICR        ;unmask timer interrrupt
                move    0,IW            ;clear IW
                move    0,IX            ;clear IX
                move    0,GP5
                move    GP5,IYLO        ;clear IY
                move    GP5,IYHI
                move    GP5,IZLO        ;clear IZ
                move    GP5,IZHI
                move    GP5,[IZ+0]      ;clear year
                move    GP5,[IZ+10h]    ;clear days
                move    GP5,[IZ+20h]    ;clear hours
                move    GP5,[IZ+30h]    ;clear minutes
                move    GP5,[IZ+40h]    ;clear seconds
                move    1,GP6
                move    5Ch,GP5
                move    GP5,TRH         ;load high byte of the time out value
                move    21h,GP5
                move    GP5,TRL         ;load low byte of the time out value
                move    40h,ACR         ;load timer from the holding reg.
                move    81h,ACR         ;start timer
                        ;END of initialization
loop:
```

TL/F/10450–12

FIGURE 7

```
            jmp     loop

        ;Timer interrupt service routine
        ;*******************************
dest:
            or      80h,CCR             ;clear [TO] flag
            add     1,IWLO              ;increment IW
            cmp     IWLO,32h            ;does IW = 50 decimal
            rnz     RI                  ;NO, then return with interrupt on
            move    0,IWLO              ;clear IW
            move    [IZ+40h],IXLO
            add     1,IXLO              ;YES, then increment IX
            cmp     IXLO,3Ch            ;does seconds = 60
            jnz     next_1
            move    0,IXLO              ;YES, then clear seconds
next_1:     move    IXLO,[IZ+40h]       ;move seconds to data memory
            rnz     RI
            move    [IZ+30h],IXLO
            add     1,IXLO              ;increment minutes
            cmp     IXLO,3Ch            ;does minutes = 60
            jnz     next_2
            move    0,IXLO              ;YES, then clear minutes
next_2:     move    IXLO,[IZ+30h]       ;move minutes to data memory
            move    0,IXLO
            rnz     RI
            move    [IZ+20h],IXLO
            add     1,IXLO              ;increment hours
            cmp     IXLO,18h            ;does hours = 24
            jnz     next_3
            move    0,IXLO              ;YES, then clear hours
next_3:     move    IXLO,[IZ+20h]       ;move hours to data memory
            move    0,IXLO
            rnz     RI
            move    [IZ+10h],IXLO
            add     1,IXLO              ;increment days
            move    IXLO,[IZ+10h]
            move    0,IXLO
            ret     RI

CODE:       .sect   ax
            .org    114h
            ljmp    dest
        .END
```

TL/F/10450-13

FIGURE 7 (Continued)

Interfacing the DP8344 to Twinax

National Semiconductor
Application Note 516
Thomas J. Quigley

OVERVIEW

The DP8344, or **Biphase Communications Processor** from National Semiconductor's Advanced Peripherals group brings a new level of system integration and simplicity to the IBM® connectivity world. Combining a 20 MHz RISC architecture CPU with a flexible multi-protocol transceiver and remote interface, the BCP is well suited for IBM 3270, 3299 and 5250 protocol interfaces. This Application Note will show how to interface the BCP to twinax, as well as provide some basics about the IBM 5250 environment.

5250 ENVIRONMENT

The IBM 5250 environment encompasses a family of devices that attach to the IBM System/34, 36 and 38 mid-size computer systems. System unit model numbers include the 5360, 5362, 5364, 5381, and 5382, and remote controller models 5294 and 5251 model 12. The system units have integral work station controllers and some may support up to 256 native mode twinax devices locally. Native mode twinax devices are ones that connect to one of these host computers or their remote control units via a multi-drop, high speed serial link utilizing the 5250 data stream. This serial link is primarily implemented with twinaxial cable but may be also found using telephone grade twisted pair. Native mode 5250 devices include mono-chrome, color and graphics terminals, as well as a wide range of printers and personal computer emulation devices.

TWINAX AS A TRANSMISSION MEDIA

The 5250 environment utilizes twinax in a multi-drop configuration, where eight devices can be "daisy-chained" over a total distance of 5000 ft. and eleven splices, (each physical device is considered a splice) see *Figure 1*. Twinax can be routed in plenums or conduits, and can be hung from poles between buildings (lightning arrestors are recommended for this). Twinax connectors are bulky and expensive, but are very sturdy. Different sorts may be purchased from IBM or a variety of third party vendors, including Amphenol. Twinax should not be spliced; to connect cables together both cables should be equipped with male connectors and a quick-disconnect adapter should be used to join them (Amphenol #82-5588).

Twinaxial cable is a shielded twisted pair that is nearly ⅓ of an inch thick. This hefty cable can be either vinyl or teflon jacketed and has two internal conductors encased in a stiff polethylene core. The cable is available from BELDEN (type #9307) and other vendors, and is significantly more expensive than coax.

The cable shield must be continuous throughout the transmission system, and be grounded at the system unit and each station. Since twinax connectors have exposed metal connected to their shield grounds, care must be taken not to expose them to noise sources. The polarity of the two inner conductors must also be maintained throughout the transmission system.

The transmission system is implemented in a balanced current mode; every receiver/transmitter pair is directly coupled to the twinax at all times. Data is impressed on the transmission line by unbalancing the line voltage with the driver current. The system requires passive termination at both ends of the transmission line. The termination resistance value is given by:

$R_t = Z_{O/2}$; where

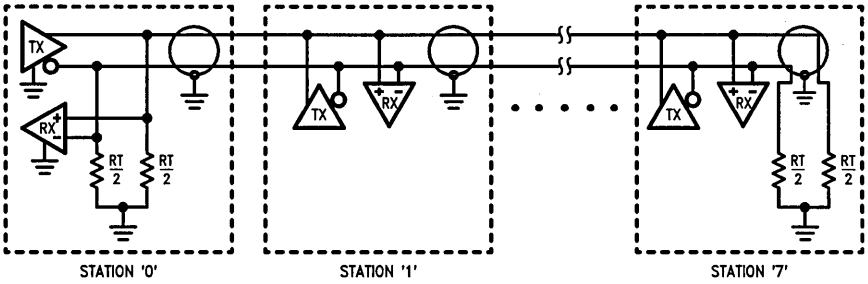
R_t: Termination Resistance

Z_o: Characteristic Impedance

In practice, termination is accomplished by connecting both conductors to the shield via 54.9Ω, 1% resistors; hence the characteristic impedance of the twinax cable of 107Ω ±5% at 1.0 MHz. Intermediate stations must not terminate the line; each is configured for "pass-through" instead of "terminate" mode. Stations do not have to be powered on to pass twinax signals on to other stations; all of the receiver/transmitter pairs are DC coupled. Consequently, devices must never output any signals on the twinax line during power-up or down that could be construed as data, or interfere with valid data transmission between other devices.

WAVEFORMS

The bit rate utilized in the 5250 protocol is 1 MHz ±2% for most terminals, printers and controllers. The IBM 3196 dis-



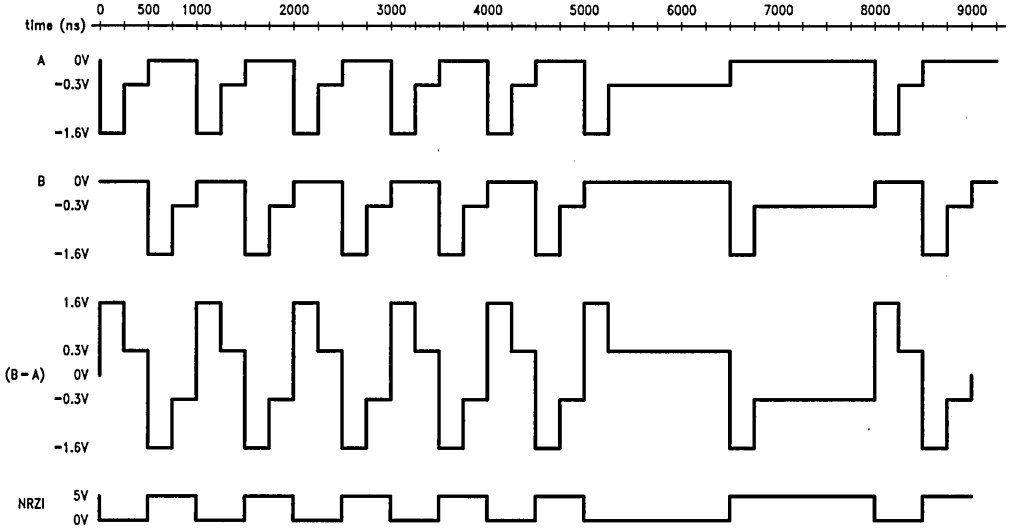
TL/F/9635–1

FIGURE 1. Multi-Drop Transmission Lines

The eight stations shown include the host device as a station. The first and last stations are terminated while intermediate stations are not.

play station has a bit rate of 1.0368 MHz ±0.01%. The data are encoded in biphase, NRZI (non-return to zero inverted) manner; a "1" bit is represented by a positive to negative transition, a "0" is a negative to positive transition in the center of a bit cell. This is opposite from the somewhat more familiar 3270 coax method. The biphase NRZI data is encoded in a pseudo-differential manner; i.e. the signal on the "A" conductor is subtracted from the signal on "B" to form the waveform shown in *Figure 2*. Signals A and B are not differentially driven; one phase lags the other in time by 180°. *Figures 3* and *4* show actual signals taken at the driver and receiver after 5000 ft. of twinax, respectively.

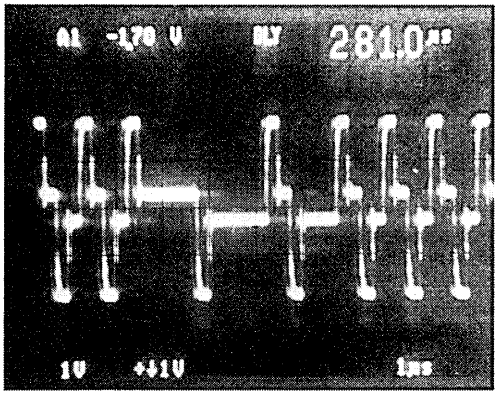
The signal on either the A or B phase is a negative going pulse with an amplitude of −0.32V ±20% and duration of 500 ±20 ns. During the first 250 ±20 ns, a predistortion or pre-emphasis pulse is added to the waveform yielding an amplitude of −1.6V ±20%. When a signal on the A phase is considered together with its B phase counterpart, the resultant waveform represents a bit cell or bit time, comprised of two half-bit times. A bit cell is 1 μs ±20 ns in duration and must have a mid bit transition. The mid bit transition is the synchronizing element of the waveform and is key to maintaining transmission integrity throughout the system.



TL/F/9635-2

FIGURE 2. Twinax Waveforms

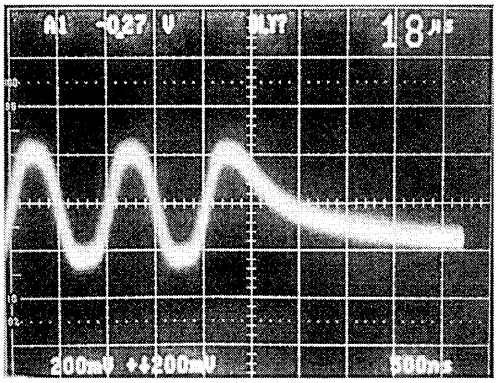
The signal on phase A is shown at the initiation of the line quiesce/line violation sequence.
Phase B is shown for that sequence, delayed in time by 500 ns.
The NRZI data recovered from the transmission.



TL/F/9635-3

FIGURE 3. Signal at the Driver

The signal shown was taken with channel 1 of an oscilloscope connected to phase B, channel 2 connected to A, and then channel 2 inverted and added to channel 1.



TL/F/9635-4

FIGURE 4. Signal at the Receiver

The signal shown was viewed in the same manner as *Figure 3*. The severe attenuation is due to the filtering effects of 5000 ft. of twinax cable.

As previously mentioned, the maximum length of a twinax line is 5000 ft. and the maximum number of splices in the line is eleven. Devices count as splices, so that with eight devices on line, there can be four other splices. The signal 5000 ft. and eleven splices from the controller has a minimum amplitude of 100 mV and a slower edge rate. The bit cell transitions have a period of 1 μs $\pm$30 ns.

5250 BIT STREAM

The 5250 Bit stream used between the host control units and stations on the twinax line consists of three separate parts; a bit synchronization pattern, a frame synchronization pattern, and one or more command or data frames. The bit sync pattern is typically five one bit cells. This pattern serves to charge the distributed capacitance of the transmission line in preparation for data transmission and to synchronize receivers on the line to the bit stream. Following the bit sync or line quiesce pattern is the frame sync or line violation. This is a violation of the biphase, NRZI data mid bit transition rule. A positive going half bit, 1.5 times normal duration, followed by a negative going signal, again 1.5 times normal width, allows the receiving circuitry to establish frame sync.

Frames are 16 bits in length and begin with a sync or start bit that is always a 1. The next 8 bits comprise the command or data frame, followed by the station address field of three bits, a parity bit establishing even parity over the start, data and address fields, and ending with a minimum of three fill bits (fill bits are always zero). A message consists of a bit sync, frame sync, and some number of frames up to 256 in total. A variable amount of inter-frame fill bits may be used to control the pacing of the data flow. The SET MODE command from the host controller sets the number of bytes of zero fill sent by attached devices between data frames. The zero fill count is usually set to zero. The number of zero fill bits injected between frames by the BCP is set by the Fill Bit select register {FBR}. This register contains the one's complement of the number of BITS sent, not bytes.

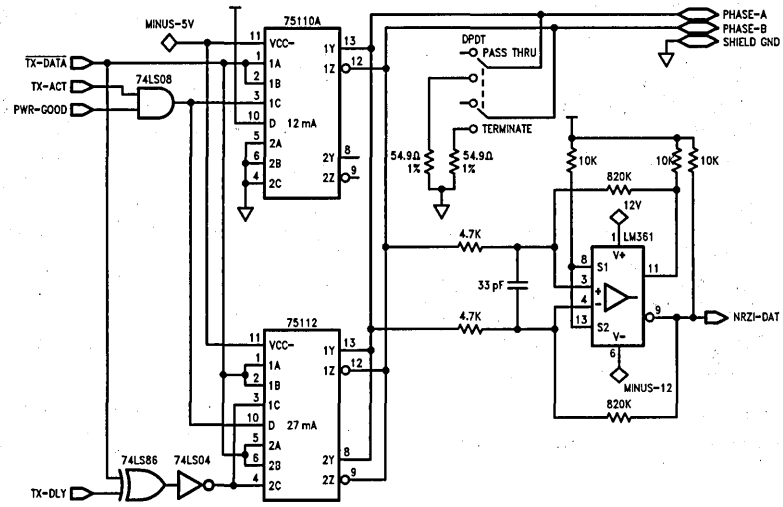
Message routing is accomplished through use of the three-bit address field and some basic protocol rules. As mentioned above, there is a maximum of eight devices on a given twinax line. One device is designated the controller or host and the remaining seven are slave devices. All communication on the twinax line is host initiated and half duplex. Each of the seven devices is assigned a unique station address from zero to six. Address seven is used for an End Of Message delimiter, or EOM. The first or only frame of a message from controller to device must contain the address of the device. Succeeding frames do not have to contain the same address for the original device to remain selected, although they usually do.

The last frame in a sequence must contain the EOM delimiter. For responses from the device to the controller, the responding device places its own address in the address field in frames 1 to (n − 1),where n $\leq$ 256, and places the EOM delimiter in the address field of frame n. However, if the response to the controller is only one frame, the EOM delimiter is used. The controller assumes that the responding devices was the one addressed in the initiating command.

Responses to the host must begin in 60 $\pm$20 μs, although some specifications state a 45 $\pm$15 μs response time. In practice, controllers do not change their time out values per device type so that anywhere from 30 μs to 80 μs response times are appropriate.

DRIVER CIRCUITS FOR THE DP8344

The transmitter interface on the DP8344 is sufficiently general to allow use in 3270, 5250, and 8-bit transmission systems. Because of this generality, some external hardware is needed to adapt the outputs to form the signals necessary to drive the twinax line. The chip provides three signals: $\overline{\text{DATA-OUT}}$, DATA-DLY, and TX-ACT. $\overline{\text{DATA-OUT}}$ is biphase serial data (inverted). DATA-DLY is the biphase serial data output (non-inverted) delayed one-quarter bit-time. TX-ACT, or transmitter active, signals that serial data is being transmitted when asserted. $\overline{\text{DATA-OUT}}$ and DATA-DLY can be used to form the A and B phase signals with their three levels by the circuit shown in *Figure 5*. TX-ACT is used as an external transmitter enable. The BCP can invert the sense of the $\overline{\text{DATA-OUT}}$ and DATA-DLY signals by asserting TIN {TMR[3]}. This feature allows both 3270 and 5250 type biphase data to be generated, and/or utilization of inverting or non-inverting transmitter stages.



TL/F/9635-5

FIGURE 5. Schematic

The current mode drive method used by native twinax devices has both distinct advantages and disadvantages. Current mode drivers require less power to drive properly terminated, low-impedance lines than voltage mode drivers. Large output current surges associated with voltage mode drivers during pulse transition are also avoided. Unwanted current surges can contribute to both crosstalk and radiated emission problems. When data rate is increased, the surge time (representing the energy required to charge the distributed capacitance of the transmission line) represents a larger percentage of the driver's duty cycle and results in increased total power dissipation and performance degradation.

A disadvantage of current mode drive is that DC coupling is required. This implies that system grounds are tied together from station to station. Ground potential differences result in ground currents that can be significant. AC coupling removes the DC component and allows stations to float with respect to the host ground potential. AC coupling can also be more expensive to implement.

Drivers for the 5250 environment may not place any signals on the transmission system when not activated. The power-on and off conditions of drivers must be prevented from causing noise on the system since other devices may be in operation. *Figure 5* shows a "DC power good" signal enabling the driver circuit. This signal will lock out conduction in the drivers if the supply voltage is out of tolerance.

Twinax signals can be viewed as consisting of two distinct phases, phase A and phase B, each with three levels, off, high and low. The off level corresponds with 0 mA current being driven, the high level is nominally 62.5 mA, +20%−30%, and the low level is nominally 12.5 mA, +20%−30%. When these currents are applied to a properly terminated transmission line the resultant voltages impressed at the driver are: off level is 0V, low level is 0.32V ±20%, high level is 1.6V ±20%. The interface must provide for switching of the A and B phases and the three levels. A bi-modal constant current source for each phase can be built that has a TTL level interface for the BCP.

An integrated solution can be constructed with a few current mode driver parts available from National and Texas Instruments. The 75110A and 75112 can be combined to provide both the A and B phases and the bi-modal current drive required as in *Figure 5*. The external logic used adapts the coax oriented BCP outputs to the twinax interface circuit, and prevents spurious transmissions during power-up or down. The serial NRZ data is inverted prior to being output by the BCP by setting TIN, {TMR[3]}.

RECEIVER CIRCUITS

The pseudo-differential mode of the twinax signals make receiver design requirements somewhat different than the coax 3270 world. Hence, the analog receiver on the BCP is not well suited to receiving twinax data. The BCP provides both analog inputs to an on-board comparator circuit as well as a TTL level serial data input, TTL-IN. The sense of this serial data can be inverted by the BCP by asserting RIN, {TMR[4]}.

The external receiver circuit must be designed with care to ensure reliable decoding of the bit-stream in the worst environments. Signals as small as 100 mV must be detected. In order to receive the worst case signals, the input level switching threshold or hysteresis for the receiver should be nominally 29 mV ±20%. This value allows the steady state, worst case signal level of 100 mV 66% of its amplitude before transitioning.

To achieve this, a differential comparator with complementary outputs can be applied, such as the National LM361. The complementary outputs are useful in setting the hysteresis or switching threshold to the appropriate levels. The LM361 also provides excellent common mode noise rejection and a low input offset voltage. Low input leakage current allows the design of an extremely sensitive receiver, without loading the transmission line excessively.

In addition to good analog design techniques, a low pass filter with a roll-off of approximately 1 MHz should be applied to both the A and B phases. This filter essentially conducts high frequency noise to the opposite phase, effectively making the noise common mode and easily rejectable.

Layout considerations for the LM361 include proper bypassing of the ±12V supplies at the chip itself, with as short as possible traces from the pins to 0.1 μF ceramic capacitors. Using surface mount chip capacitors reduces lead inductance and is therefore preferable in this case. Keeping the input traces as short and even in length is also important. The intent is to minimize inductance effects as well as standardize those effects on both inputs. The LM361 should have as much ground plane under and around it as possible. Trace widths for the input signals especially should be as wide as possible; 0.1 inch is usually sufficient. Finally, keep all associated discrete components nearby with short routing and good ground/supply connections.

Design equations for the LM361 in a 5250 application are shown here for example. The hysteresis voltage, V_h, can be expressed the following way:

$$V_h = V_{rio} + ((R_{in}/(R_{in} + R_f) \times V_{ol}) - (R_{in}/(R_{in} + R_f) \times V_{ol}))$$

where

V_h — Hysteresis Voltages, Volts

R_{in} — Series Input Resistance, Ohms

R_f — Feedback Resistance, Ohms

C_{in} — Input Capacitance, Farads

V_{rio} — Receiver Input Offset Voltage, Volts

V_{oh} — Output Voltage High, Volts

V_{ol} — Output Voltage Low, Volts

The input filter values can be found through this relationship:

$$V_{cin} = V_{in1} - V_{in2/1} + jwC_{in}(R_{in1} + R_{in2})$$

where $R_{in1} = R_{in2} = R_{in}$:

$$Fro = w/2\pi$$

$$Fro = 1/(2\pi \times R_{in} \times C_{in})$$

$$C_{in} = 1/(2\pi \times R_{in} \times Fro)$$

where

V_{in1}, V_{in2} — Phase A and B signal voltages, Volts

V_{cin} — Voltage across C_{in}, or the output of the filter, Volts

R_{in1}, R_{in2} — Input resistor values, $R_{in1} = R_{in2}$, Ohms

Fro — Roll-Off Frequency, Hz

W — Frequency, Radians

The roll-off frequency, Fro, should be set nominally to 1 MHz to allow for transitions at the transmission bit rate. The transition rate when both phases are taken together is 2 MHz, but then R_{in1} and R_{in2} must be considered, so:

$$Fro2 = 1/(2\pi \times (R_{in1} + R_{in2}) \times C_{in})$$

or,

$$Fro2 = 1/(2\pi \times 2 \times R_{in} \times C_{in})$$

where Fro2 = 2 × Fro, yielding the same results.

The following table shows the range of values expected:

TABLE I

Value	Maximum	Minimum	Nominal	Units	Tolerance
R_{IN}	4.935E+03	4.465E+03	4.700E+03	Ω	0.05
R_F	8.295E+05	7.505E+05	7.900E+05	Ω	0.05
C_{IN}	4.4556E−11	2.6875E−11	3.3863E−11	F	
V_{OH}	5.250E+00	4.750E+00	5.000E+00	V	
V_{OL}	4.000E−01	2.000E−01	3.000E−01	V	
V_{IN+}	1.920E+00	1.000E−01		V	
V_{IN-}	1.920E+00	1.000E−01		V	
V_{RIO}	5.000E−03	0.000E+00	1.000E−03	V	
R	6.533E−03	5.354E−03	5.914E−03	Ω	
Fro	1.200E+06	8.000E+05	1.000E+06	Hz	0.2
V_H	3.368E−02	2.691E−02	2.880E−02	V	
Xc	7.4025E+03	2.9767E+03	4.7000E+03	Ω	

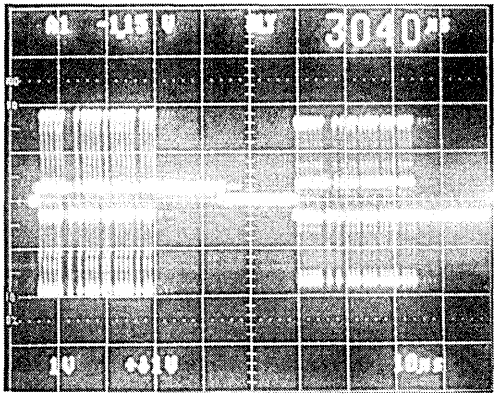
The BCP has a number of advanced features that give designers much flexibility to adapt products to a wide range of IBM environments. Besides the basic multi-protocol capability of the BCP, the designer may select the inbound and outbound serial data polarity, the number of received and transmitted line quiesces, and in 5250 modes, a programmable extension of the TX-ACT signal after transmission.

The polarity selection on the serial data stream is useful in building single products that handle both 3270 and 5250 protocols. The 3270 biphase data is inverted with respect to 5250.

Selecting the number of line quiesces on the inbound serial data changes the number of line quiesce bits that the receiver requires before a line violation to form a valid start sequence. This flexibility allows the BCP to operate in extremely noisy environments, allowing more time for the transmission line to charge at the beginning of a transmission. The selection of the transmitted line quiesce pattern is not generally used in the 5250 arena, but has applications in 3270. Changing the number of line quiesces at the start of a line quiesce pattern may be used by some equipment to implement additional repeater functions, or for certain inflexible receivers to sync up.

The most important advanced feature of the BCP for 5250 applications is the programmable TX-ACT extension. This feature allows the designer to vary the length of time that the TX-ACT signal from the BCP is active after the end of a transmission. This can be used to drive one phase of the

twinax line in the low state for up to 15.5 μs. Holding the line low is useful in certain environments where ringing and reflections are a problem, such as twisted pair applications. Driving the line after transmitting assures that receivers see no transitions on the twinax line for the specified duration. The transmitter circuit shown in *Figure 5* can be used to hold either the A or B phase by using the serial inversion capability of the BCP in addition to swapping the A and B phases. Choosing which phase to hold active is up to the designer; 5250 devices use both. Some products hold the A phase, which means that another transition is added after the last half bit time including the high and low states, with the low state helf for the duration, see *Figure 6*. Alternatively, some products hold the B phase. Holding the B phase does not require an extra transition and hence is inherently quieter.



TL/F/9635-6

FIGURE 6. Line Hold Options

The signal was viewed in the same manner as *Figures 3* and *4*. The lefthand portion of the signal is a transmitting device utilizing line hold on phase A. The right hand side shows the IBM style (phase B) line hold.

To set the TX-ACT hold feature, the upper five bits of the Auxilliary Transceiver Register, {ATR [3–7]}, are loaded with one of thirty-two possible values. The values loaded select a TX-ACT hold time between 0 μs and 15.5 μs in 500 ns increments.

SOFTWARE INTERFACE

The BCP was designed to simplify designing IBM communications interfaces by providing the specific hardware necessary in a highly integrated fashion. The power and flexibility of the BCP, though, is most evident in the software that is written for it. Software design for the BCP deserves careful attention.

When designing a software architecture for 5250 terminal emulation, for example, one concern the designer faces is how to assure timely responses to the controller's commands. The BCP offers two general schemes for handling the real time response requirements of the 5250 data stream: interrupt driven transceiver interface mode, and polled transceiver interface mode. Both modes have strengths that make them desirable. The excellent interrupt response and latency times of the BCP make interrupts very useful in most 5250 applications.

Although factors such as data and instruction memory wait states and remote processors waiting BCP data memory accesses can degrade interrupt response times, the minimum latency is 2.5 T-states. The BCP samples all interrupt sources by the falling edge of the CPU clock; the last falling edge prior to the start of the next instruction determines whether an interrupt will be processed. When an interrupt is recognized, the next instruction in the present stream is not executed, but its address is pushed on the address stack. A two T-state call to the vector generated by the interrupt type and the contents of {IBR} is executed and [GIE] (Global Interrupt Enable) is cleared. If the clock edge is missed by the interrupt request or if the current instruction is longer than 2 bytes, the interrupt latency is extended.

Running in an interrupt driven environment can be complex when multiple sessions are maintained by the same piece of code. The software has the added overhead of determining the appropriate thread or session and handling the interrupt accordingly. For a multi-session 5250 product, the transceiver interrupt service routines must determine which session is currently selected through protocol inferences and internal semaphores to keep the threads separate and intact.

In a polled environment, the biggest difficulty in designing software is maintaining appropriate polling intervals. Polling too often wastes CPU bandwidth, not polling frequently enough loses data and jeopardizes communication integrity. Standard practice in servicing polled devices is to count CPU clock cycles in the program flow to keep track of when to poll. A program change can result in lengthy recalculations of polling intervals and requalifications of program functionality. Using the programmable timer on board the BCP to set the polling interval alleviates the need to count instructions when code is changed or added. In both polled or interrupt environments, the latency effects of remote processors waiting memory accesses must be limited to a known length of time and figured into both polling intervals and worst case interrupt latency calculations. Using the programmable timer on the BCP makes both writing and maintaining polled software easier.

SOFTWARE ARCHITECTURE FOR 5250 EMULATION

The 5250 data rate is much lower than that of the 3270 data stream, hence it is possible for the BCP to emulate all seven 5250 sessions with a CPU frequency of 8 MHz. Choosing a 16 MHz crystal allows the transceiver to share the CPU clock at OCLK/2, eliminating an extra oscillator circuit. The 8 MHz rate yields a 125 ns T-state, or 250 ns for most instructions. Interrupt latency is typically one instruction (assuming no wait states or remote accesses) which is suitable for 5250 operation. If more speed is desired, the CPU could be switched to 16 MHz operation.

A MULTI-MODE TRANSCEIVER

The BCP provides two 5250 protocol modes, promiscuous and non-promiscuous. These two modes afford the designer a real option only when the end product will attach to one 5250 address at a time. The non-promiscuous mode is configured with an address in the {ATR} register and only re-

ceives messages whose first frame address matches that address, or an error occurs in the first frame of the message. Filtering out unwanted transmissions to other addresses leaves more CPU time for other non-protocol related tasks, but limits the device to one address at a time. The promiscuous mode allows messages to any and all addresses to be received. Resetting the transceiver during a message destined to another device forces the transceiver to begin looking for a start sequence again, effectively discarding the entire unwanted message. Because of its flexibility, the promiscuous mode is used in this illustration.

REAL TIME CONSIDERATIONS

Choosing a scheme for servicing the transceiver is basic to the design of any emulation device. The BCP provides both polled and interrupt driven modes to handle the real time demands of the chosen protocol. In this example, the interrupt driven approach is used. This implies the extra overhead of setting up interrupt vectors and initializing the interrupt masks appropriately. This approach eliminates the need to figure polling intervals within the context of other CPU tasks.

5250 CONFIGURATION

Configuring a complex device like the BCP can be difficult until a level of familiarity with the device is reached. To help the 5250 product designer through an initial configuration, a register by register description follows, along with the reasons for each configuration choice. Certainly, most applications will use different configurations than the one shown here. The purpose is to illustrate one possible setup for a 5250 emulation device.

There are two major divisions in the BCP's configuration registers: CPU specific and transceiver specific ones.

CPU SPECIFIC CONFIGURATION REGISTERS:

{**DCR**}**—Device Control Register**—This register controls the clocks and wait states for instruction and data memory. Using a value of H#A0 sets the CPU clock to the OCLK/2 rate, the transceiver to OCLK/2, and no wait states for either memory bank. As described above, the choice of a 16 MHz crystal and configuring this way allows 8 MHz operation now, with a simple software change for straight 16 MHz operation in the future.

{**ACR**}**—Auxiliary Control Register**—Loading this register with H#20 sets the timer clock source to CPU-CLK/2, sets [BIC], the Bidirectional Interrupt Control to configure BIRQ as an input, allows remote accesses with [LOR] cleared, and disables all maskable interrupts through [GIE] low. When interrupts are unmasked in {ICR}, [GIE] must be set high to allow interrupts to operate. [GIE] can be set and cleared by writing to it, or through a number of instructions including RET and EXX.

{**IBR**}**—Interrupt Base Register**—This register must be set to the appropriate base of the interrupt vector table located in data RAM. The DP8344 development card and monitor software expect [IBR] to be at H#1F, making the table begin at H#1F00. The monitor software can be used without the interrupt table at H#1F00, but doing so is simplest for this illustration.

{**ICR**}**—Interrupt Control Register**—This register contains both CPU and transceiver specific controls. From the CPU point of view, the interrupt masks are located here. In this illustration, the system requires receiver, transmitter, BIRQ, and timer interrupts, so that in operation those interrupt bits should be unmasked. For initialization purposes, though, interrupts should be masked until their vectors are installed and the interrupt task is ready to be started. Therefore, loading [ICR] with H#7F is prudent. This also sets the receiver interrupt source, but that will be discussed in the next section.

TRANSCEIVER CONFIGURATION REGISTERS:

{**TMR**}**—Transceiver Mode Register**—This register controls the protocol selection, transceiver reset, loopback, and bit stream inversion. Loading this register with H#0D sets up the receiver in 5250 promiscuous mode, inverts serial data out, does not invert incoming serial data, does not allow the transmitter and receiver to be active at the same time, disables loopback, and does not reset the transceiver. Choosing to set [RIN] low assumes that serial data will be presented to the chip in NRZI form. Not allowing the receiver and transmitter to operate concurrently is not an issue in 5250 emulation, since there is no defined repeater function in the protocol as in 3270 (3299). Bits [B5, 6], [RPEN] and [LOOP] are primarily useful in self testing, where [LOOP] routes the transmitted data stream into the receiver and simultaneous operation is desirable. Please note that for loopback operation, [RIN] must equal [TIN]. [TRES] is used regularly in operation, but should be left off when not specifically needed.

{**TCR**}**—Transceiver Command Register**—This register has both configuration and operation orientated bits, including the transmitted address and parity bits. For this configuration, the register should be set to H#00 and the specific address needed summed into the three LSBs, as appropriate. The [SEC] or Select Error Codes bit is used to enable the {ECR} register through the {RTR} transceiver FIFO port, and should be asserted only when an error has been detected and needs to be read. [SLR], or Select Line Receiver is set low to enable the TTL-IN pin as the serial data in source. The BCP's on chip comparator is best suited to transformer coupled environments, and National's LM361 high speed differential comparator works very well for the twinax line interface. [ATA], or Advance Transmitter Active is normally used in the 3270 modes to change the form of the first line quiesce bit for transmission. Some twinax products use a long first line quiesce bit, although it is not necessary. The lower four bits in {TCR} are used to form the frame transmitted when data is written into {RTR}, the transceiver FIFO port. Writing into {RTR} starts the transmitter and/or loads the transmit FIFO. The least significant three bits in {TCR} form the address field in that transmitted frame, and B3, [OWP] controls the type of parity that is calculated and sent with that frame. [OWP] set to zero calculates even parity over the eight data bits, address and sync bit as defined in the IBM 5250 PAI.

{**ATR**}**—Auxiliary Transceiver Register**—Since this application is configured for promiscuous mode, the {ATR} register serves only to set the line hold function time. In non-promiscuous mode, the three least significant bits of this register are the selected address. Setting this register to H#50 allows a 5 μs hold time and clears the address field to 0, since promiscuous mode is used.

{FBR}—**Fill Bit Register**—This register controls the number of biphase zeros inserted between concatenated frames when transmitting. This register should be set upon reception of the SET MODE instruction from the host. {FBR} contains the one's complement of the number of inter-frame fill bits so that H#FF sends no extra fill bits.

{ICR}—**Interrupt Control Register**—As discussed in the CPU configuration section, this register sets [RIS] or Receiver Interrupt Select as well as the interrupt mask. Setting the register to H#7F selects [DA + ERR], Data Available or transceiver ERRor, as the interrupt source. This interrupt is asserted when either a valid frame has been clocked into the receive FIFO or an error has occurred. Other interrupt options are available including: [RA], Receiver Active; and [RFF + ERR], Receive FIFO Full or transceiver ERRor. For 5250 protocols the [DA + ERR] is most efficient. The [RFF + ERR] interrupt will not assert until the FIFO is full ... regardless of whether the incoming message is single or multi-frame. [RA] provides plenty of notice that a frame is incoming, but due to the speed of the BCP, this advanced warning is not generally needed. [DA + ERR] provides a notification just after the parity bit has been decoded from the incoming frame which is almost 3 μs prior to the end of the frame. With the CPU running at 8 MHz, that allows typically nine instructions ([(4 * 3) − 3)]) for interrupt latency, trap and bank switch after interrupting.

MULTI-SESSION POWER

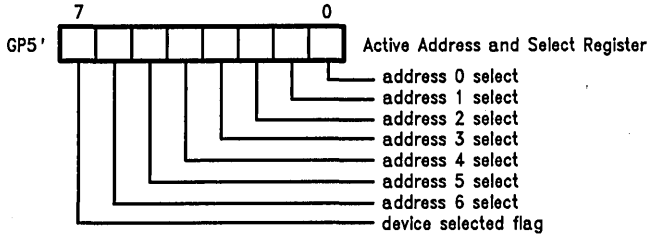
Handling multiple sessions in software is not trivial, and making the receiver service routines interrupt driven complicates the task further. The BCP is so fast, that at 8 MHz handling a multi-frame message by interrupting on the first frame and polling for succeeding frames is very inefficient. To maximize bandwidth for non-protocol related tasks, the CPU should handle each frame separately on interrupt and exit. To do this, a number of global state variables must be maintained. Since the alternate B register bank is primarily used for transceiver functions anyway, dedicating the other registers in that bank permanently as state variables is acceptable in most cases; doing so speeds and simplifies access to them. Defining the following registers as:

enables the software to keep track of the various states the protocol must handle.

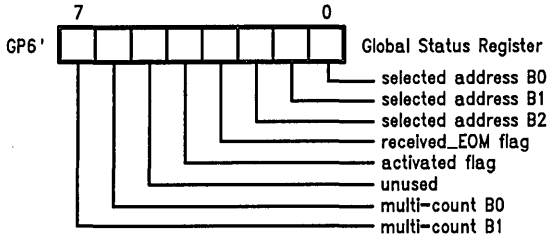
The active address bits in GP5′ allow individual addresses to be active, or any combination of addresses. When interrupted by a message to a non-selected address, [TRES] is toggled to reset the receiver until the beginning of the next message is detected. [B7] is used to determine if any particular address is "selected" and in the process of receiving data. The selected flag is set and cleared according to specific protocol rules set up in the IBM PAI.

Register GP6′ contains the selected address storage [B0–2], where the address of the device expecting at least one other frame is stored when exiting the interrupt service routine, so that upon interruption caused by the reception of that frame, the address is still available. The received_EOM flag, [B3] is set when a message is decoded that contains B#111 or EOM delimiter. It is stored in this global status register to allow the protocol to determine the end of a transmission. In most multi-byte transmissions, the number of data frames expected is dictated by the protocol. However, ACTIVATE WRITE commands to printers can have any number of data frames associated with them up to 256. In this situation, the activated flag, [B4] is set to signal a variable length stream. Certain host devices also concatenate commands within messages, obscuring the determination of end of message. This scheme allows the software to keep track during such scenarios. The multi-count bits, [B6–7] are used in addition to the EOM delimiter to determine the end of a transmission. The number of additional frames expected in a given multi-byte command is written into these bits (note that a maximum of three bytes can be planned for in this way). When the count is terminated and no EOM delimiter is present, the algorithm then assumes a multi-command message is in progress. [B5] is unused.

Register GP7′ is used to store the received data or error code for passage to other routines. The data can be passed on the stack, but dedicating a register to this function simplifies transactions in this case. Keeping track of received data is of utmost importance to communications devices.



TL/F/9635–7



TL/F/9635–8

GP7′—Bits [0–7] Received Data or Error Register

RECEIVER INTERRUPT

The receiver interrupt algorithm handles any or all seven addresses possible on the twinax line. The same code is used for each address by utilizing a page oriented memory scheme. Session specific variables are stored in memory pages of 256 bytes each. All session control pages, or SCPs are on 256-byte even boundaries. By setting the high order byte of a BCP index register to point to a particular page or SCP, the low order byte then references an offset within that page. Setting up data memory in such a way that the first SCP begins at an address of B#xxxx x000 0000 0000 further enhances the usefulness of this construct. In this scheme, the high byte of the SCP base pointer can be used to set the particular SCP merely by summing the received or selected address into the lower three bits of the base register.

NORMAL OPERATION

In normal operation, the configuration described thus far is used in the following manner: After initializing the registers, data structures are initialized, and interrupt routines should be activated. This application utilizes the receiver, transmitter, timer, and bi-directional interrupts. Since {IBR} is set to H#1F, the interrupt table is located at H#1F00. A LJMP to the receiver interrupt routine should be installed at location H#1F104, the transmitter interrupt vector at H#1F08, the BIRQ interrupt vector at H#1F10, and the Timer interrupt vector at H#1F14. Un-masking the receiver interrupt and BIRQ at start up allows the device to come on-line.

When interrupt by the receiver, the receiver interrupt service routine first checks the [ERR] flag in {TSR [B5]}. If no errors have been flagged, the received_EOM flag is either set or cleared. This is accomplished by comparing {TSR [B0–2]} with the B#111 EOM delimiter. A test of the selected flag, {GP5′ [B7]} determines if any of the active addresses are selected. Assuming that the system is just coming on line, none of the devices would be selected. If the frame is addressed to an active device, the SCP for that device is set, and the command is parsed. Parsing the command sets the appropriate state flags, so that upon exiting, the interrupt routine will be prepared for the next frame. Once parsed, the command can be further decoded and handled. If the command is queue-able, the command is pushed on the internal command queue, and the receiver interrupt routine exits. If the command requires an immediate response, then the response is formulated, the timer interrupt is setup, and the routine is exited.

The timer interrupt is used in responding to the host by waiting an appropriate time to invoke the transmit routine. The typical response delay is 45 ±15 μs after the last valid fill bit received in the command frame. Some printers and terminals are allowed a full 60 ±20 μs to respond. In either case, simply looping is very inefficient. The immediate response routine simply sets the timer for the appropriate delay and unmasks the timer.

In the transmit routine, the data to be sent is referenced by a pointer and an associated count. The routine loads the appropriate address in the three LSBs of {TCR}, and writes the data to be sent into {RTR}. This starts the transmitter. If the data count is greater than the transmit FIFO depth (three bytes), the Transmit FIFO Empty interrupt [TFE] is setup. This vectors to code that refills the FIFO and re-enables that interrupt again, if needed. This operation must be carried out before the transmitter is finished the last frame in the FIFO or the message will end prematurely.

The last frame transmitted must contain the EOM delimiter. It can be loaded into {TCR} and data into {RTR} while the transmitter is running without affecting the current frame. In other words, the transmit FIFO is 12 bits wide, including address and parity with data; the address field is clocked along with the data field. In this way, multi-byte response may be made in efficient manner.

ERROR HANDLING

In 5250 environments, the time immediately after the end of message is most susceptible to transmission errors. The BCP's receiver does not detect an error after the end of a message unless transitions on the line continue for a complete frame time or resemble a valid sync bit of a multiframe transmission. If the twinax line is still active at the end of what could be an error frame, the receiver posts the LMBT error. For example, if noise on the twinax line continues for up to 11 μs after the three required fill bits, the receiver will reset without flagging an error. If noise resembles a start bit, the receiver now expects a new frame and will post an error if a loss of synchronization occurs. If the noisy environment is such that transitions on the receiver's input continue for 11 μs, or the receiver really has lost sync on a real frame, the error is posted.

Basically, the receiver samples [LA] in addition to the loss of synchronization indication to determine when to reset or to post an error. After a loss of synchronization in the fill bit portion of a frame, if the [LA] flag's time-out of 2 μs is reached prior to the end of what could be the next frame, the receiver will reset. If the transitions prevent [LA] from timing out for an entire 11 μs frame time, a LMBT error is posted. This method for resetting the receiver is superior in that not only are the spurious loss of mid bit errors eliminated, the receiver performs better in noisy environments than other designs.

SUMMARY

The IBM 5250 twinax environment is less understood and in some ways more complex than the 3270 environment to many developers. This application note has attempted to explain some basics about twinax as a transmission medium, the hardware necessary to interface the DP8344 to that medium, and some of the features of the BCP that make that task easier. Schematics are included in this document to illustrate possible designs. Details of the twinax waveforms were discussed and figures included to illustrate some of the more relevant features. Also, some different software approaches to handling the transceiver interface were discussed.

REFERENCES

5250 Information Display to System/36 and System/38 System Units Product Attachment Information, IBM, November 1986.

Transmission Line Characteristics, Bill Fowler, National Semiconductor Application Note AN-108.

Basic Electromagnetic Theory, D.T. Paris, F.K. Hurd McGraw-Hill Inc., 1969.

APPENDIX A: EXAMPLE CODE

The following code was assembled with the HILEVEL assembler. Table II shows the correlation between HILEVEL mnemonics and the mnemonics used in National data sheets for the DP8344V.

TABLE II

HILEVEL	National Semiconductor
MOVE Rs,Rd	MOVE Rs,Rd
LD Ptr,Rd{,Mde}	MOVE [mIr],Rd
ST Rs,Ptr{,Mde}	MOVE Rs,[mIr]
LDAX Ptr,Rd	MOVE [Ir + A],Rd
STAX Rs,Ptr	MOVE Rs,[Ir + A]
LDNZ n,Rd	MOVE [IZ + n],rd
STNZ Rs,n	MOVE rs,[IZ + n]
LDI n,Rd	MOVE n,rd
STI n,Ptr	MOVE n,[Ir]
ADD Rs,Rd	ADDA Rs,Rd
ADDRI Rs,Ptr{,Mde}	ADDA Rs,[mIr]
ADDI n,Rsd	ADD n,rsd
ADC Rs,Rd	ADCA Rs,Rd
ADCRI Rs,Ptr{,Mde}	ADCA Rs,[mIr]
SUBT Rs,Rd	SUBA Rs,Rd
SUBRI Rs,Ptr{,Mde}	SUBA Rs,[mIr]
SUBI n,Rsd	SUB n,rsd
SBC Rs,Rd	SBCA Rs,Rd
SBCRI Rs,Ptr{,Mde}	SBCA Rs,[mIr]
AND Rs,Rd	ANDA Rs,Rd
ANDRI Rs,Ptr{,Mde}	ANDA Rs,[mIr]
ANDI n,Rsd	AND n,rsd
OR Rs,Rd	ORA Rs,Rd
ORRI Rs,Ptr{,Mde}	ORA Rs,[mIr]
ORI n,Rsd	OR n,rsd
XOR Rs,Rd	XORA Rs,Rd
XORRI Rs,Ptr{,Mde}	XORA Rs,[mIr]
XORI n,Rsd	XOR n,rsd
CMP Rs,n	CMP rs,n
CPL Rsd	CPL Rsd
BIT Rs,n	BIT rs,n
SRL Rsd,n	SHR Rsd,b
SLA Rsd,n	SHL Rsd,b
ROT Rsd,n	ROT Rsd,b

TL/F/9635-9

```
JMP  n                JMP  n
LJMP n                LJMP nn
JMPR Rs               JMP  Rs
JMPI Ptr              LJMP [Ir]
JRMK Rs,n,m           JRMK Rs,b,m
JMPB Rs,s,p,n         LJMP Rs,p,s,nn
JMPF s,f,n            JMP  f,s,n
                      Jcc  n   - opt. syntax for JMP f-

CALL n                CALL n
LCALL n               LCALL      nn
CALLB Rs,s,p,n        LCALL      Rs,p,s,nn

RET  {g{,rf}}         RET  {g {,rf}}
RETF s,f{,g{,rf}}     RETF f,s,{,{g} {,rf}}
                      Rcc  {g {,rf}}   - opt. syntax -

EXX  a,b{,g}          EXX  ba,bb,{,g}

TRAP n{,g}            TRAP n {,gU}
```

Table 2.

```
Addr      Line

            1          .REL
            2   TAB    8
            3   WIDTH  132
            4   LIST   S,F
            5   TITLE  RXINT
            6   ;-----------------------------------------------------------------
            7   ;      RXINT - 9/21/87
            8   ;
            9   ;      pseudo code
           10   ;
           11   ;bool         selected;                        /* station is selected
           12   ;byte         seladdr;                         /* address of selected station
           13   ;byte         multicount;                      /* number of frames in this multi
           14   ;bool         activated;                       /* command has been activated
           15   ;
           16   ;rxint()
           17   ;byte         data;                            /* data storage
           18   ;bool         rx_eom;                          /* received EOM
           19   ;bool         lta;                             /* line turn around flag
           20   ;{
           21   ;      if (error) {
           22   ;              if (logerror()== true) return;   /* receiver errors
           23   ;              }
           24   ;      else {
           25   ;              if (TSR == EOM) rx_eom = true;   /* set received EOM flag
           26   ;              else rx_eom = false;
           27   ;
           28   ;              if (!selected) {
```

TL/F/9635-10

```
29   ;                    if (active) {
30   ;                            if (!rx_eom) {
31   ;                                    seladdr = (TSR * EOM);
32   ;                                    IZ = (SCPBASE + seladdr); /* set SCP to appropriate session */
33   ;                                    data = rtr;
34   ;                            else {
35   ;                                    proto_error(); /* should not get here
36   ;                                    reset_xcvr();
37   ;                                    return();
38   ;                                    }
39   ;                            }
40   ;                    else {
41   ;                            reset_xcvr();          /* not of interest
42   ;                            return();
43   ;                            }
44   ;                    if (multiframe) {              /* activate write, etc...
45   ;                            multicount = parse(data); /* set number of frames */
46   ;                            selected = true;       /* only way to select   */
47   ;                            queue(data);
48   ;                            }
49   ;                    else {                         /* not multi
50   ;                            if ((var = single_decode(data)) == queable)
51   ;                                    queue(data);
52   ;                            else if (var == immed) immediate(data);
53   ;                    }
54   ;            else {                                 /* selected         */
55   ;                    IZ = (SCPBASE + seladdr);

Addr     Line RXINT

56   ;                    data = rtr
57   ;            if (activated) {                       /* in the middle of transmission
58   ;                act_data(data);
59   ;                if (rx_eom) {                      /* end of message
60   ;                    selected = false;
61   ;                    activated = false;
62   ;                    }
63   ;                return();
64   ;                }
65   ;                    if (multicount > 0) {
66   ;                            queue(data);
67   ;                            if (multicount-= 0) {
68   ;                                    if (rx_eom) selected = false;
69   ;                                    }
70   ;                            }
71   ;                    else {
72   ;                            if (multiframe) {
73   ;                                    multicount = parse(data);
74   ;                                    queue(data);
75   ;                                    }
76   ;                            else {
77   ;                                    if ((var = single_decode(data)) == queable)
78   ;                                            queue(data);
```

TL/F/9635-11

```
79   ;                                   else if (var == immed) immediate(data);
80   ;                                   if (rx_eom) selected = false;
81   ;                                   }
82   ;                          }
83   ;                    }
84   ;              }
85   ;       }
86   ;       return();
87   ;}
88   ;logerror()
89   ;{
90   ; bool result;
91   ;     switch (error_type) {
92   ;     case RDIS:
93   ;          result = err_rdis();                    /* receiver diabled while active
94   ;          break;
95   ;     case LMBT:
96   ;          result = err_lmbt();                    /* loss of midbit error
97   ;          break;
98   ;     case PARR:
99   ;          result = err_parr();                    /* parity error
100  ;          break;
101  ;     case OVF:
102  ;          result = err_ovf();                     /* receiver FIFO overrun
103  ;          break;
104  ;     default:
105  ;          result = err_unknown();                 /* strange error handler
106  ;          break;
107  ;     }
108  ;     return(result);
109  ;   }
110  ;
```

Addr Line RXINT

```
111  ;err_lmbt()
112  ;{
113  ;     if (!DA && !selected && !delay(LA)) return(false); /* delay of 6 usec
114  ;     else {
115  ;          log();                                  /* bump error counters
116  ;          return(true);                           /* admit defeat
117  ;          }
118  ; }
119  ;-----------------------------------------------------------------
120  ;       name:           RXINT
121  ;       description:    receiver interrupt handler
122  ;
123  ;               received datum is sent to other routines thru gp7'
124  ;               SCP is set appropriately in IZ
125  ;               GP5P - active addresses:bits 0-6
126  ;                      selected flag:   bit 7
127  ;               GP6P - multicount:      bit 7-6
128  ;                      unused:          bit 5
```

TL/F/9635-12

```
          129   ;                    activated:      bit 4
          130   ;                    rx_eom flag:    bit 3
          131   ;                    seladdr:        bits 2-0
          132   ;              GP7P - received data
          133   ;
          134   ;       entry:         DA interrupt, GP5', GP6'
          135   ;       exit:          ACC',GP7' ARE DESTROYED
          136   ;       history:       tjq  9/16/87   create
          137   ;-----------------------------------------------------------------
          138           PUBLIC  RCVRINT
          139
          140           EXTRN   PARSE,QUEUE,IMMEDECODE,RESXCVR
          141           EXTRN   MIDERRL,MIDERRH,OVFERRL,OVFERRH,PARERRL,PARERRH
          142           EXTRN   RXERRL,RXERRH,RSPCTL,RSPCTH,BASESCP,IESERRL,IESERRH
          143
          144
          145   SELERR: EQU     B#01000000      ; select the error register
          146   RXEOM:  EQU     B#00001000      ; rxeom flag
          147   EOM:    EQU     B#00000111      ; EOM delimeter
          148   MULTI:  EQU     B#11000000      ; multicount
          149   SELECT: EQU     B#10000000      ; selected flag
          150   LTA:    EQU     B#101           ;     "
          151   CFLAG:  EQU     B#00000010      ; CARRY FLAG
          152
00000     153   RCVRINT:
          154           EXX     MA,AB,DI        ; SET APPROPRIATE BANK
00000 AEE8  154
00001 D500  155         JMPF    NS,RERR,NOERROR
00002 CC00  156         CALL    RXERROR         ; ERROR IN FRAME
00003 D900  157         JMPF    S,C,EXIT        ; ABORT
00004 D900  158   NOERROR:
00004 B078  159         LDI     EOM,ACC         ; LOAD MASK
            160         AND     TSR,GP7         ; FORM ADDRESS
00005 F165  160
            161         CMP     GP7,EOM         ; TEST
00006 307B  161
00007 D000  162         JMPF    NS,Z,C1RXINT    ; IF NOT EQUAL, JUMP

 Addr       Line RXINT

0000B 508A  163         ORI     RXEOM,GP6       ; ELSE SET EOM FLAG
00009 CB00  164         JMP     C2RXINT         ;
0000A CB00  165   C1RXINT:
0000A 4F7A  166         ANDI    RXEOM*,GP6      ; CLEAR IT
            167   ;
            168   ; DECIDE IF WE'RE ALREADY SELECTED
            169   ;
0000B       170   C2RXINT:
            171         JMPB    GP5,S,B7,DEVSELECT ; IF ALREADY SELECTED
0000B BDE9  171
0000C 0000  171
            172   ;
            173   ; NOT SELECTED...DECIDE IF ADDRESS IS ACTIVE, IE; VALID FOR US
```

TL/F/9635–13

```
                 174    ;
0000D            175    DEVTABLE:                          ; ELSE, SEE IF ACTIVE
                 176            JRMK    TSR,ROT6,MSK3      ; JUMP BASED ON THE ADDRESS FIELD*4
0000D B3C5       176
                 177            JMPB    GP5,NS,B0,RSTRX    ; ADDR 0 - IF NOT ACTIVE, RESET RX
0000E BC09       177
0000F 0000       177
                 178            LJMP    LOADSCP            ; ACTIVE DEVICE, SET scp
00010 CE00       178
00011 0000       178
                 179            JMPB    GP5,NS,B1,RSTRX    ; ADDR 1 - IF NOT ACTIVE, RESET RX
00012 BC29       179
00013 0000       179
                 180            LJMP    LOADSCP            ; ACTIVE DEVICE, SET scp
00014 CE00       180
00015 0000       180
                 181            JMPB    GP5,NS,B2,RSTRX    ; ADDR 2 - IF NOT ACTIVE, RESET RX
00016 BC49       181
00017 0000       181
                 182            LJMP    LOADSCP            ; ACTIVE DEVICE,
00018 CE00       182
00019 0000       182
                 183            JMPB    GP5,NS,B3,RSTRX    ; ADDR 3 - IF NOT ACTIVE,
0001A BC69       183
0001B 0000       183
                 184            LJMP    LOADSCP            ; ACTIVE DEVICE,
0001C CE00       184
0001D 0000       184
                 185            JMPB    GP5,NS,B4,RSTRX    ; ADDR 4 - IF NOT ACTIVE,
0001E BC89       185
0001F 0000       185
                 186            LJMP    LOADSCP            ; ACTIVE DEVICE,
00020 CE00       186
00021 0000       186
                 187            JMPB    GP5,NS,B5,RSTRX    ; ADDR 5 - IF NOT ACTIVE,
00022 BCA9       187
00023 0000       187
                 188            LJMP    LOADSCP            ; ACTIVE DEVICE,
00024 CE00       188
00025 0000       188
                 189            JMPB    GP5,NS,B6,RSTRX    ; ADDR 6 - IF NOT ACTIVE,
00026 BCC9       189

 Addr        Line RXINT

00027 0000       189
                 190            LJMP    LOADSCP            ; ACTIVE DEVICE,
00028 CE00       190
00029 0000       190
                 191            LCALL   RESXCVR            ; ADDR 7 - RECEIVED EOM ...WE'RE NOT INTERESTED
0002A CEB0       191
0002B 0000       191
0002C CB00       192            JMP     EXIT               ; QUIT
```

TL/F/9635-14

```
                ;
                ; LOAD THE SCP POINTER, IZ
                ;
0002D           LOADSCP:
                        XOR     ACC,ACC         ; CLEAR
0002D F908
                        MOVE    ACC,ZLO         ; LOW BYTE
0002E FE48
0002F B008              LDI     BASESCP,ACC     ; SET UP UPPER BYTE OF SCP POINTER
                        MOVE    ACC,ZHI         ;
00030 FE68
00031 B07B              LDI     EOM,ACC         ; EOM MASK
                        AND     TSR,ACC         ; LEAVE IN ACC
00032 F105
                        ADD     ZHI,ZHI         ; ADD INTO Z POINTER
00033 E273
                ;
                ; DECODE THE COMMAND FRAME
                ;
00034           DECODE:
                        MOVE    RTR,GP7         ; GET RX DATA
00034 FD64
                        JMPB    GP7,S,B0,MULTIFRM; IF MULTIFRAME
00035 BD0B
00036 0000
                        LCALL   IMMEDECODE      ; ELSE, IMMEDIATE ACTION REQUIRED
00037 CEB0
0003B 0000
00039 CB00              JMP     EXIT
0003A CB00      MULTIFRM:
                        LCALL   PARSE           ; SET MULTI COUNT
0003A CEB0
0003B 0000
0003C 5B09              ORI     H#B0,GP5        ; SELECTED = TRUE
0003D 4F8A              ANDI    EOM*,GP6        ; CLEAR SELECTED ADDRESS
0003E B07B              LDI     EOM,ACC         ; MASK ADDRESS
                        AND     TSR,ACC         ; LEAVE IN ACC
0003F F105
                        OR      GP6,GP6         ; SET NEW ADDRESS
00040 F54A
                        LCALL   QUEUE           ; PLACE ON QUEUE
00041 CEB0
00042 0000
00043 CB00              JMP     EXIT            ;
                ;
                ; THIS CODE IS BRANCHED TO IF THE DEVICE IS SELECTED
                ;       FIRST, SET SCP BASED ON SELECTED ADDRESS
```

```
Addr        Line RXINT

                ;
00044           DEVSELECT:
                        XOR     ACC,ACC         ; CLEAR ACC
```

TL/F/9635-15

```
00044 F908  226
            227         MOVE    ACC,ZLO         ; CLEAR LOW BYTE OF POINTER
00045 FE48  227
00046 B008  228         LDI     BASESCP,ACC     ; BASE OF SESSION CONTROL PAGE
            229         MOVE    ACC,ZHI         ; UPPER BYTE
00047 FE68  229
00048 B078  230         LDI     EOM,ACC         ; MASK ADDRESS
            231         AND     GP6,ACC         ; LEAVE IN ACC
00049 F10A  231
            232         ADD     ZHI,ZHI         ; FORM SCP POINTER
0004A E273  232
            233   ;
            234   ; NOW DECIDE ABOUT MULTIFRAME POSSIBILITIES
            235   ;
            236         MOVE    RTR,GP7         ; GET DATA
0004B FD64  236
0004C BC08  237         LDI     MULTI,ACC       ; MULTI MASK
            238         AND     GP6,ACC         ; COUNT IN UPPER NIBBLE
0004D F10A  238
            239         SRL     ACC,ROT6        ; POSITION IN LOWER NIBBLE
0004E C8C8  239
0004F D800  240         JMPF    S,Z,NEWCOMM     ; NOT in A MULTIBYTE
            241         LCALL   QUEUE           ; MULTI, SO PUSH ON QUEUE
00050 CE80  241
00051 0000  241
00052 2018  242         SUBI    H#01,ACC        ; DECREMENT MULTICOUNT
00053 D800  243         JMPF    S,Z,TERMULTI    ; IF ZERO, MULTI HAS TERMINATED
            244   ;
            245   ; MULTI STILL IN PROGRESS
            246   ;
00054 43FA  247         ANDI    MULTI*,GP6      ; CLEAR OUT OLD COUNT
            248         SLA     ACC,ROT6        ; REPOSITION COUNT
00055 C948  248
            249         OR      GP6,GP6         ; SUM INTO STATUS
00056 F54A  249
00057 CB00  250         JMP     EXIT
            251   ;
            252   ; MULTICOUNT HAS REACHED ZERO, SO TERMINATE
            253   ;
00058       254   TERMULTI:
00058 43FA  255         ANDI    MULTI*,GP6      ; CLEAR OLD COUNT TO ZERO
            256         JMPB    GP6,NS,B3,C1TERM; IF NOT EOM,
00059 BC6A  256
0005A 0000  256
0005B 47F9  257         ANDI    SELECT*,GP5     ; ELSE, SELECT = FALSE
0005C CB00  258         JMP     RSTRX           ; RESET THE TRANSCEIVER
0005D CB00  259   C1TERM:
0005D CB00  260         JMP     EXIT
            261   ;
            262   ; NEW COMMAND; MULTI OR SINGLE
            263   ;
0005E       264   NEWCOMM:
```

TL/F/9635-16

```
 Addr       Line RXINT

            265          JMPB    GP7,NS,B0,SINGLE; IF NEW COMMAND IS NOT MULTI,
0005E BC0B  265
0005F 0000  265
            266          LCALL   PARSE           ; IS MULTI, SET COUNT
00060 CEB0  266
00061 0000  266
            267          LCALL   QUEUE           ; PUSH ON QUEUE
00062 CEB0  267
00063 0000  267
00064 CB00  268          JMP     EXIT            ; QUIT, TIL NEXT FRAME
            269   ;
            270   ; NEW COMMAND IS SINGLE AND/OR NEEDS IMMEDIATE RESPONSE
            271   ;
00065       272   SINGLE:
            273          LCALL   IMMEDECODE      ; SINGLE...GO DO IT
00065 CEB0  273
00066 0000  273
            274          JMPB    GP6,NS,B3,EXIT  ; IF NOT EOM...
00067 BC6A  274
00068 0000  274
00069 47F9  275          ANDI    SELECT*,GP5     ; CLEAR SELECTED BIT
0006A 47F9  276   RSTRX:
            277          LCALL   RESXCVR         ; RESET, CLEAR DATA OUT
0006A CEB0  277
0006B 0000  277
0006C 0000  278   EXIT:
0006C AFB0  279          RET     RI,RF           ; RETURN GRACEFULLY
            280
            281   ;-----------------------------------------------------------------
            282   ;      name:          RXERROR
            283   ;      description:   receiver ERROR handler
            284   ;
            285   ;      entry:         DA + ERR interrupt, GP5', GP6'
            286   ;      exit:          ACC',GP7' ARE DESTROYED
            287   ;      history:       tjq  9/16/87   create
            288   ;-----------------------------------------------------------------
            289   ;
            290   ; RECEIVER ERROR HANDLER
            291   ;
0006D       292   RXERROR:
0006D 5406  293          ORI     SELERR,TCR      ; SET ECR BIT
            294          MOVE    RTR,GP7         ; GET ERROR TYPE
0006E FD64  294
0006F 4BF6  295          ANDI    SELERR*,TCR     ; RESET TCR
            296          JMPB    GP7,S,B1,LMBTERR; LOSS OF MIDBIT
00070 BD2B  296
00071 0000  296
            297          JMPB    GP7,S,B3,PARERR ; PARITY
00072 BD6B  297
00073 0000  297
            298          JMPB    GP7,S,B4,OVFERR ; OVERFLOW
```

TL/F/9635-17

```
00074 BDBB  298
00075 0000  298
00076 0000  299   ILLEGAL:
00076 B008  300          LDI     ILLEGAL,ACC     ; WHAT ERROR IS THIS?

 Addr       Line RXINT

00077 CB00  301          JMP     BUMPERR         ; SHOULD NOT GET HERE!!
00078 CB00  302   LMBTERR:
00078 DE00  303          JMPF    5,DA,CLEARC     ; if DA, THEN NO ERROR
            304          JMPB    GP5,S,B7,LOGIT  ; IF SELECTED, POST
00079 BDE9  304
0007A 0000  304
0007B CC00  305          CALL    SDLY            ; DELAY FOR 6 USEC
            306          JMPB    NCF,NS,B5,CLEARC; IF NOT ACTIVE - DISCARD, ELSE POST
0007C BCA1  306
0007D 0000  306
0007E 0000  307   LOGIT:
0007E B008  308          LDI     MIDERRL,ACC     ; LOSS OF MIDBIT
0007F CB00  309          JMP     BUMPERR         ; INCREMENT COUNTER
00080 CB00  310   PARERR:
00080 B008  311          LDI     PARERRL,ACC     ; PARITY
00081 CB00  312          JMP     BUMPERR
00082 CB00  313   OVFERR:
00082 B008  314          LDI     OVFERRL,ACC     ; OVERFLOW...VERY BAD!
00083 B008  315   BUMPERR:
            316          ADD     ZLO,YLO         ; FORM NEW POINTER
00083 E212  316
00084 B018  317          LDI     H#01,ACC        ; INCREMENT
            318          LD      PTRY,GP6        ; FETCH OLD COUNT
00085 C0CA  318
            319          ADDRI   GP6,PTRY,POSTD  ; WRITE OUT NEW
00086 A04A  319
00087 D100  320          JMPF    NS,C,RXEXIT     ; GET OUT
            321          LD      PTRY,GP6        ; FETCH UPPER BYTE
00088 C0CA  321
            322          ADDRI   GP6,PTRY        ;
00089 A0CA  322
0008A 5020  323          ORI     CFLAG,CCR       ; SET CARRY
0008B 5020  324   RXEXIT:
0008B AFB0  325          RET                     ; DO NOT restore flags
0008C AFB0  326   CLEARC:
0008C 4FD0  327          ANDI    CFLAG*,CCR      ; CLEAR CARRY
0008D CB00  328          JMP     RXEXIT
            329   ;-----------------------------------------------------------------
            330   ;       name:           SDLY
            331   ;       description:    delay routine, MULTIPLES OF 4.8usec,
            332   ;                       1.4 usec OVERHEAD, MAX OF 410usec
            333   ;       entry:          delay count on stack
            334   ;       exit:           acc destroyed
            335   ;       WARNING:        DONT CALL THIS WITH COUNT = 0!
            336   ;       history:        tjq  9/16/87   create
            337   ;-----------------------------------------------------------------
```

TL/F/9635-18

```
                 338
0008E            339    SDLY:
                 340            EXX     MA,MB,NAI     ; BANK, ALLOW INTERRUPTS
0008E AE80       340
                 341            MOVE    DS,ACC        ; GET COUNT
0008F FD1F       341
                 342            MOVE    GP7,DS        ; PUSH GP7 REGISTERS USED
00090 FFEB       342
                 343            MOVE    GP6,DS
 Addr           Line RXINT

00091 FFEA       343
                 344            MOVE    ACC,GP7       ; USE GP7 FOR COUNT ALSO
00092 FD68       344
00093 FD68       345    SDLYLP1:
00093 B03A       346            LDI     H#03,GP6      ; LOAD FOR 4.8usec COUNTS
00094 B03A       347    SDLYLP2:
00094 201A       348            SUBI    H#01,GP6      ; DECREMENT COUNT
00095 D000       349            JMPF    NS,Z,SDLYLP2  ; CONTINUE UNTIL EXHAUSTED
00096 201B       350            SUBI    H#01,GP7      ; DECREMENT OUTER COUNT
00097 D000       351            JMPF    NS,Z,SDLYLP1  ; CONTINUE IF NOT ZERO
                 352            MOVE    DS,GP6        ; POP REG
00098 FD5F       352
                 353            MOVE    DS,GP7        ;
00099 FD7F       353
0009A AFB0       354            RET     RI,RF         ; RETURN, RESTORE FLAGS
                 355
                 356
                 357            END

Assembly Phase complete.
    0 error(s) detected.
```

TL/F/9635-19

The DP8344 BCP® Inverse Assembler

National Semiconductor
Application Note 688
Laura Johnson

OVERVIEW

The DP8344 BCP Inverse Assembler is a software package for use in a Hewlett Packard Logic Analyzer. It was developed by National Semiconductor's Arlington Design Center to allow disassembly of the DP8344 op-code mnemonics.

When developing systems using a RISC processor such as the DP8344, the need often arises to know the sequence of events that occurred in real time in the system. The actual execution flow that occurred in the system can be determined by monitoring the states on the Instruction memory Address bus and the Instruction memory bus of the DP8344 with a Hewlett Packard Logic Analyzer. The DP8344 BCP Inverse Assembler enhances this development tool by displaying the BCP instruction op-code mnemonics on the logic analyzer's screen. This Application Note lists the equipment needed as well as the necessary information to set up, use, and obtain the DP8344 BCP Inverse Assembler. Additionally, the source code flow chart for the DP8344 BCP Inverse Assembler is provided in Appendix A of this Application Note.

EQUIPMENT REQUIRED

The following equipment is required to use the DP8344 BCP Inverse Assembler:

1. DP8344 BCP Inverse Assembler; Available from National Semiconductor.
2. HP1650A or HP1651A Logic Analyzer, or HP16500A Logic Analysis System with an HP16510A State/Timing Card installed.
3. DP8344 Biphase Communications Processor in a System.

It is assumed that the reader is familiar with the operation of the HP Logic Analyzer. For further information refer to the Operation Reference Manual provided with the HP1650A or 1651A Logic Analyzers, or with the HP16510A Logic Analyzer Module. Information pertaining to the operation of the logic analyzer in a state mode will be useful.

SYSTEM SETUP

A block diagram of the setup of the system for using the DP8344 BCP Inverse Assembler is shown in *Figure 1*. The target system refers to a system containing a BCP which is running. The DP8344 BCP Inverse Assembler is software which has been loaded into the HP Logic Analyzer. The target system is interfaced to the DP8344 BCP Inverse Assembler through the HP Logic Analyzer's channels.

An example of a target system is a Multi-Protocol Adapter (MPA™) installed in a personal computer. The MPA Design/Evaluation Kit includes both the hardware and software that allows the MPA to emulate a 3270 or 5250 display terminal and to support industry standard PC emulation software. The MPA Design/evaluation Kit is available from National Semiconductor (Part No. D88344MPA-EB). All the examples in this document were generated using an MPA board and it's associated software for the target system.

Additional equipment which one may find useful includes an extender card and an 84-pin PLCC Adapter. The extender card brings a PC board out of the PC chasis, allowing easier access to the BCP. An 84-pin PLCC Adapter allows one to directly connect the channels of the logic analyzer to the pins on the BCP. Emulation Technology, Inc., makes an 84-pin PLCC Adapter which it calls a BUG KATCHER. (It is Part No. BC-4-084-PCC5-00000).

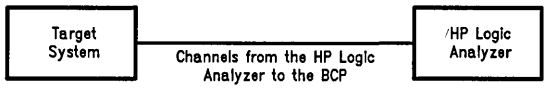
The sample target system described above includes the following equipment:

1. IBM® Personal Computer or compatible
2. MPA Development Kit
3. Extender Card (optional)
4. 84-Pin PLCC Adapter

The DP8344 BCP Inverse Assembler requires information from both the Instruction memory Address bus and the Instruction memory data bus of the BCP in the target system. Thus, these pins must be connected to the logic analyzer. The 84-pin PLCC Adapter allows one to directly connect the logic analyzer channels to the BCP. *Figure 2* provides a detailed view of the pin connections from the DP8344 to the logic analyzer. The pins can be connected to any of the pods as long as the channel and label definitions are defined accordingly in the FORMAT Menu as described later in this Application Note.

STARTING THE DP8344 BCP INVERSE ASSEMBLER

Once the system hardware has been set up, the DP8344 BCP Inverse Assembler software needs to be installed in the HP Logic Analyzer. The 3½ inch diskette provided in the DP8344 BCP Inverse Assembler Package contains the software for the HP Logic Analyzer. Load the DP8344 BCP Inverse Assembler Software into the HP Logic Analyzer by selecting either LOAD ALL from file BCP, or LOAD State/Timing E, from File BCP.E as in *Figure 3*. This automatically loads the DP8344 BCP Inverse Assembler as well as the stored State/Timing configuration into the HP Logic Analyzer.



TL/F/10814-1

FIGURE 1. Block Diagram of the System Set Up

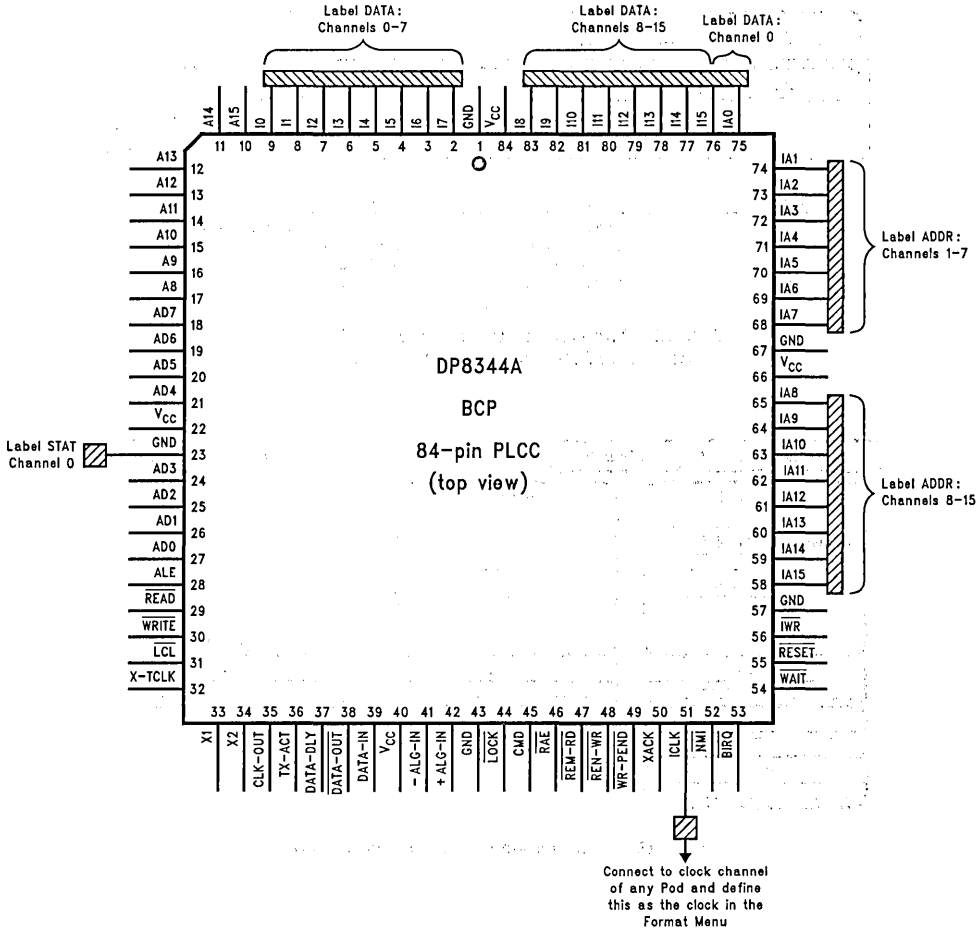
CONFIGURING THE HP LOGIC ANALYZER

The DP8344 Inverse Assembler software contains a State/Timing configuration which one may use without any changes. The designer can change this default configuration, or define an entirely new configuration to meet their own systems needs. However, certain parameters must exist in the configuration for the DP8344 Inverse Assembler to work. These parameters will be described using the default configuration as an example.

Internal communication variables are set as the logic analyzer collects data from the target system. Therefore, the logic analyzer's configuration must follow the setup described here. *Figures 4–6* show the configuration provided on the DP8344 BCP Inverse Assembler diskette. One may create their own configuration by adding more labels and connecting more channels to the target system than shown in the examples in this document. This will allow one to monitor the system activity according to their needs. However, the logic analyzers system configuration must include the following:

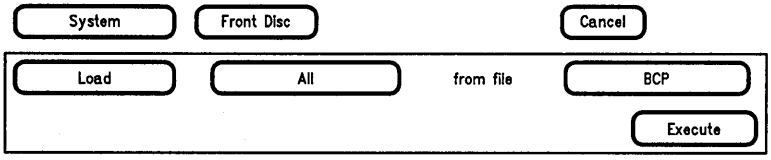
In the Configuration Menu, as in *Figure 4,* one must:

1. Define the Analyzer Type to be a State Analyzer.
2. Assign at least two pods to the State Analyzer.



TL/F/10814–2

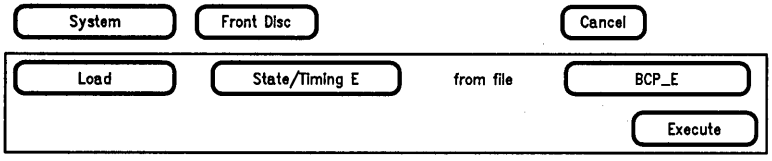
FIGURE 2. Pins Connected to Logic Analyzer Pods



TL/F/10814-3

Filename	File Type	File Description
BCP	inverse_assem	DP8344 BCP INVERSE ASSEMBLER
BCP_D	16530A_config	DP8344 BCP INVERSE ASSEMBLER
BCP_E	16510A_config	DP8344 BCP INVERSE ASSEMBLER
BCP_	16500A_config	DP8344 BCP INVERSE ASSEMBLER

(a)

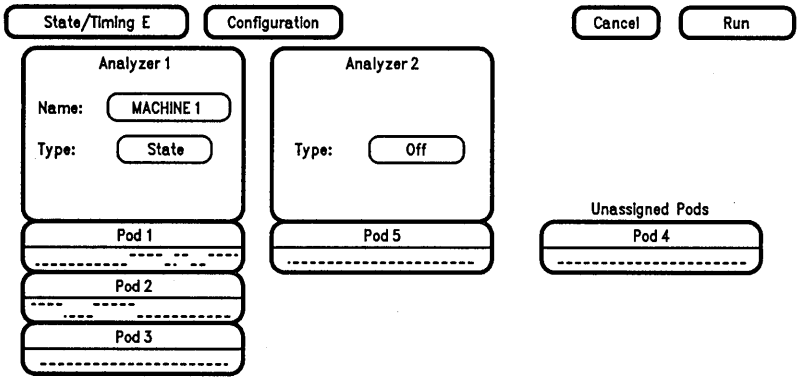


TL/F/10814-4

Filename	File Type	File Description
BCP	inverse_assem	DP8344 BCP INVERSE ASSEMBLER
BCP_D	16530A_config	DP8344 BCP INVERSE ASSEMBLER
BCP_E	16510A_config	DP8344 BCP INVERSE ASSEMBLER
BCP_	16500A_config	DP8344 BCP INVERSE ASSEMBLER

(b)

FIGURE 3. Two Methods to Load DP8344 BCP Inverse Assembler Software from the Front Disk Menu

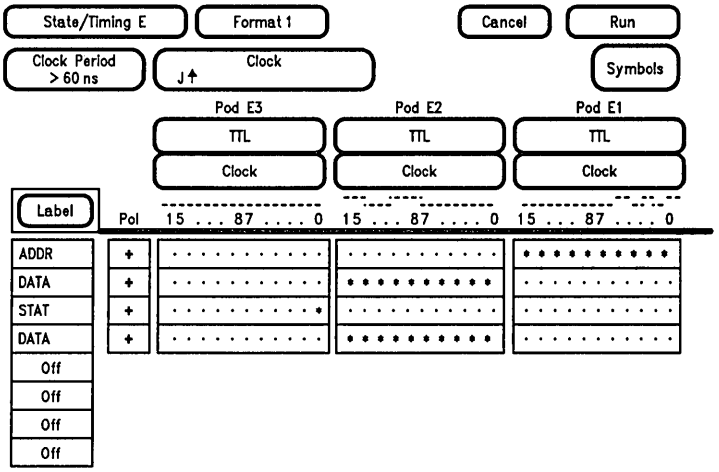


TL/F/10814-5

FIGURE 4. Configuration Menu on Logic Analyzer

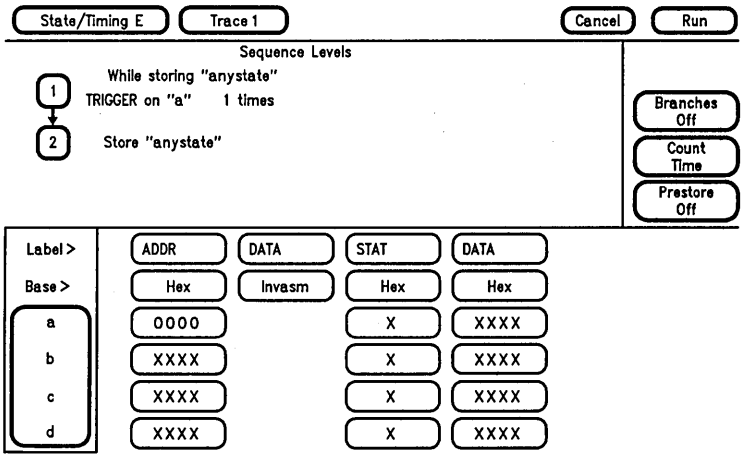
In the Format Menu, see *Figure 5*, define the labels and assign the channels in the following manner:

1. Create labels ADDR, DATA, and STAT.
2. Assign the channels connected to the labels as follows:
   i. Label ADDR refers the channels connected to the Instruction memory Address Bus on the DP8344. From *Figure 2*, these are pins 75 through 68, and pins 65 through 58. To use the default configuration the pins from the Instruction memory Address bus must be connected to channel 0 through 15 of Pod E1.
   ii. The DATA label refers to the channels connected to the Instruction memory data bus on the DP8344. From *Figure 2*, these are pins 9–2, and pins 83–76. To use the default configuration the pins from the Instruction memory Data bus must be connected to channels 0 through 15 of Pod E2.
   iii. For the label STAT it is not necessary to actually connect any of the defined channels to the BCP. However, it is recommended that one does connect all defined channels to a pin such as ground. This is because the BCP does not use a STATUS bus. The STAT label **must** be defined in the Format Menu. In the example shown in *Figure 5*, the channel assigned to the STAT label corresponds to a ground pin on the BCP connected to channel 0 of Pod E3.
3. Define the Clock to be the channel which corresponds to the connection from the pod clock connection to pin 51, ICLK, on the DP8344. In the example shown in *Figure 5*, the J clock means that ICLK is connected to the clock channel of pod E1. Set the clock to trigger on the rising edge of ICLK.



TL/F/10814-6

FIGURE 5. Format Menu on Logic Analyzer



TL/F/10814-7

FIGURE 6. Trace Menu on Logic Analyzer

The trigger may be defined in the Trace Menu according to the information desired. For example, in *Figure 6*, the trace is set to trigger when the BCP executes the program, i.e., the Instruction memory Address bus is 0 hex.

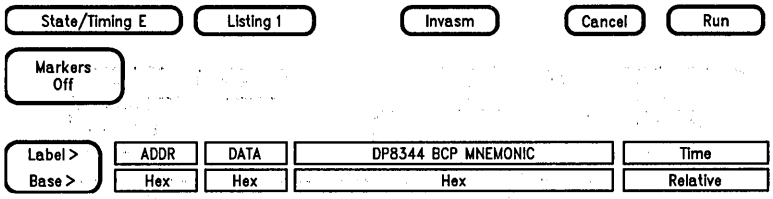
Once the system configuration has been developed, it must be linked with the inverse assembler software. First, load the DP8344 BCP Inverse Assembler Software by either method shown in *Figure 3*. Second, create a configuration by either:

i. modifying the configuration file which was loaded into the HP Logic Analyzer with the DP8344 BCP Inverse Assembler, or

ii. by loading another State/Timing Configuration which has been stored on diskette.

Third, verify that the three labels: ADDR, DATA and STAT exist in the Format Menu. Fourth, in the State Listing Display, shown in *Figure 7*, select the base field below the label DATA. This will generate seven pop-outs. Select the "Invasm" pop-out to allow the mnemonics to be displayed. Finally, store the new configuration to the DP8344 BCP Inverse Assembler using one of the two methods shown in *Figure 8*. Whenever this configuration file is loaded, the inverse assembler will automatically load. Note that storing the configuration to the Inverse Assembler will write over any previously stored configurations. Therefore, it is recommended that one back up all of the stored configurations by copying them to a backup diskette.

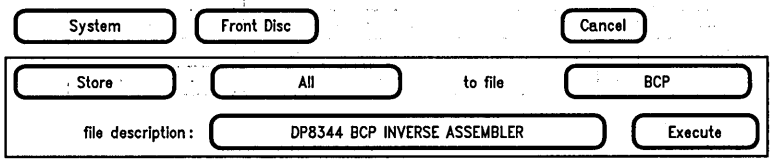
The system is now set to capture the BCP op-codes from your system and display them as mnemonics.



TL/F/10814-8

FIGURE 7. State Listing Display

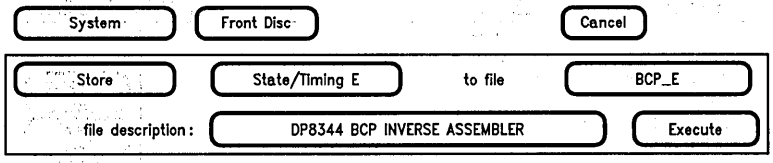
The Data Label with base Hex will display the op-codes in Hex Format. The DP8344 BCP MNEMONIC Label is generated by selecting the base type for the Label DATA to be "Invasm".



TL/F/10814-9

Filename	File Type	File Description
BCP	inverse_assem	DP8344 BCP INVERSE ASSEMBLER
BCP_D	16530A_config	DP8344 BCP INVERSE ASSEMBLER
BCP_E	16510A_config	DP8344 BCP INVERSE ASSEMBLER
BCP_	16500A_config	DP8344 BCP INVERSE ASSEMBLER

(a)



TL/F/10814-10

Filename	File Type	File Description
BCP	inverse_assem	DP8344 BCP INVERSE ASSEMBLER
BCP_D	16530A_config	DP8344 BCP INVERSE ASSEMBLER
BCP_E	16510A_config	DP8344 BCP INVERSE ASSEMBLER
BCP_	16500A_config	DP8344 BCP INVERSE ASSEMBLER

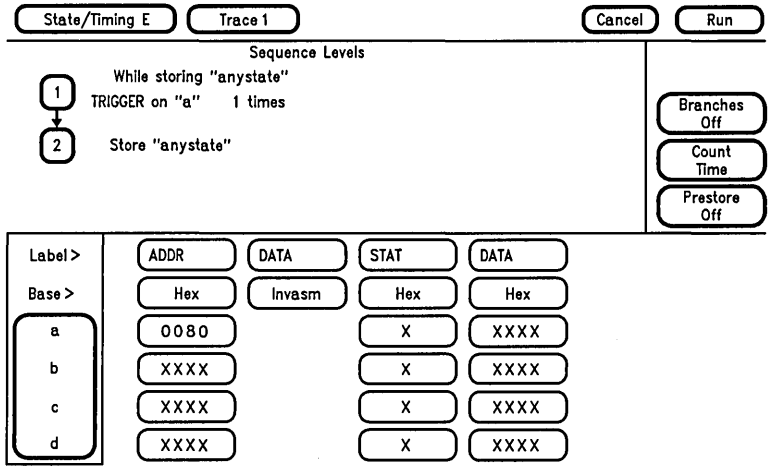
(b)

FIGURE 8. Two Methods to Store Configurations to the DP8344 BCP Inverse Assembler Software

DP8344 BCP INVERSE ASSEMBLER OPERATION

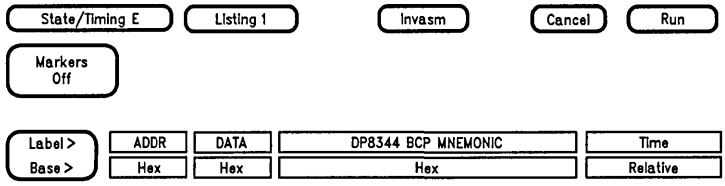
An inverse assembler converts instructions captured by the logic analyzer in binary form into mnemonics. Thus it makes it much easier to follow the program's execution flow. Furthermore, one can still use the logic analyzer to view other useful information by specifying the trace conditions, labels and channel connections in the logic analyzer's configuration file.

One needs to be aware of how the captured information is actually diassembled. The inverse assembler begins disassembling at the event which was triggered upon. Hence, any information captured prior to the trigger may not be correctly disassembled. To ensure valid disassembly of states captured prior to the trigger, one must scroll the display so the first instruction one wants disassembled is the first line on the display. Then select the "Invasm" pop-up on the top line of the State Listing Display. This causes the inverse assembler to disassemble the code from the first line on the display. For an example, refer to *Figures 9* through *12*. The inverse assembler was set to trigger when the Instruction Address Bus was 80 hex, as in *Figures 9* and *10*. The two byte instructions captured prior to the trigger were not correctly disassembled. Referring to *Figure 11*, one observes that line −10 is disassembled as an ADD Instruction rather than as the second byte of the LJMP instruction from line −11. To correct this, one must select "Invasm" from the top line of the State Listing Menu. The inverse assembler immediately disassembles the code from the first line on the screen. The correctly disassembled code is shown in *Figure 12*.



TL/F/10814–11

FIGURE 9. Triggering Event



TL/F/10814–12

−7	005C	DD01	JMP RE, S, 005EH	120 ns
−6	005D	D501	JMP RE, NS, 005FH	80 ns
−5	005F	CF40	ILLEGAL OPCODE	160 ns
−4	0060	CF00	ILLEGAL OPCODE	120 ns
−3	0061	4FE2	AND FEH, ICR/ATR	120 ns
−2	0062	C508	ILLEGAL OPCODE	80 ns
−1	0063	CC1C	CALL 0080H	120 ns
0	0080	AE80	EXX MA, MB, NCHG	160 ns
1	0081	C343	MOVE ACR/FBR, [IV+]	80 ns
2	0082	AE90	EXX AA, MB, NCHG	160 ns
3	0083	AEC8	EXX MA, AB, EI	120 ns
4	0084	C343	MOVE ACR/FBR, [IV+]	120 ns
5	0085	AEE0	EXX MA, BM, DI	160 ns
6	0104	AFF0	RET DI, RFB	80 ns
7	0085	AEE0	EXX MA, MB, DI	120 ns
8	0086	C343	MOVE ACR/FBR, [IV+]	120 ns

FIGURE 10. Triggered Event as Shown in the State Listing

State/Timing E | Listing 1 | Invasm | Cancel | Run

Markers Off

Label > / Base >	ADDR Hex	DATA Hex	DP8344 BCP MNEMONIC Hex	Time Relative
−19	0050	0051	ADD 05H, NCF/IBR	120 ns
−18	0051	4009	AND 00H, GP5/GP5′	80 ns
−17	0052	C340	MOVE CCR/DCR, [IY+]	120 ns
−16	0053	CD89	JMP GP5/GP5′	160 ns
−15	0054	B00D	MOVE 00H, IWHI	200 ns
−14	0055	B57C	MOVE 57H, IWLO	120 ns
−13	0056	CD00	LJMP [IW]	120 ns
−12	0057	8009	JRMK GP5/GP5′, 0H, 0H	80 ns
−11	0058	8D09	LJMP GP5/GP5′, 0H, S, 0058H	200 ns
−10	0059	0058	ADD 05H, GP4/GP4′	120 ns
−9	005A	8C09	LJMP GP5/GP5′, 0H, NS, 005CH	120 ns
−8	005B	005C	ADD 05H, IWLO	80 ns
−7	005C	DD01	JMP RE, S, 005EH	120 ns
−6	005D	D501	JMP RE, NS, 005FH	120 ns
−5	005F	CF40	ILLEGAL OPCODE	160 ns
−4	0060	CF00	ILLEGAL OPCODE	80 ns

TL/F/10814–13

FIGURE 11. Incorrectly Disassembled Instructions Prior to Triggered Event

State/Timing E | Listing 1 | Invasm | Cancel | Run

Markers Off

Label > / Base >	ADDR Hex	DATA Hex	DP8344 BCP MNEMONIC Hex	Time Relative
−19	0050	0051	ADD 05H, NCF/IBR	120 ns
−18	0051	4009	AND 00H, GP5/GP5′	80 ns
−17	0052	C340	MOVE CCR/DCR, [IY+]	120 ns
−16	0053	CD89	JMP GP5/GP5′	160 ns
−15	0054	B00D	MOVE 00H, IWHI	200 ns
−14	0055	B57C	MOVE 57H, IWLO	120 ns
−13	0056	CD00	LJMP [IW]	120 ns
−12	0057	8009	JRMK GP5/GP5′, 0H, 0H	80 ns
−11	0058	8D09	LJMP GP5/GP5′, 0H, S, 0058H	200 ns
−10	0059	0058		120 ns
−9	005A	8C09	LJMP GP5/GP5′, 0H, NS, 005CH	120 ns
−8	005B	005C		80 ns
−7	005C	DD01	JMP RE, S, 005EH	120 ns
−6	005D	D501	JMP RE, NS, 005FH	120 ns
−5	005F	CF40	ILLEGAL OPCODE	160 ns
−4	0060	CF00	ILLEGAL OPCODE	80 ns

TL/F/10814–14

FIGURE 12. Instructions Prior to Triggered Event, Correctly Disassembled after Choosing the "Invasm" Pop-Out

This same technique must be applied if one jumps ahead in the display and then scrolls backwards to view a certain state; in other words, you do not scroll forward through every line to reach the desired state. For example, if one manually selected the line number −12 in *Figure 12* and entered line 226, the screen would display lines 219 through 234. Now if one rolls the screen backwards to display lines 199 through 214 as in *Figure 13*, the two byte instruction, LJMP, is once again not correctly disassembled. Therefore, select the "Invasm" pop-out and the display is correctly disassembled as shown in *Figure 14*.

One of the features of the BCP is that it uses register banks. However, there is no external indication of the bank's state. The name of a register therefore depends upon which bank one is in, as in *Figure 15*. Due to the manner in which the inverse assembler disassembles the captured data, keeping track of the correct register name meant that one would constantly have to scroll the screen back to the last EXX statement and hit the "Invasm" pop-out to ensure that the displayed register names are correct. Hence, to avoid this inconvenience, the register names for both banks are displayed at all times. Refer to line 45 of *Figure 16* for an example. The op-code decodes to MOVE where the source register is R0. Therefore, the register names for R0 in both banks: Main Bank A — CCR, and Alternate Bank A — DCR, are displayed.

To view the op-code in both mnemonic form and hex form, as in *Figure 16*, define the DATA label twice in the Format Menu, as in *Figure 4*. Then, select the base label to be "Hex" for one and "Invasm" for the other in the State Listing.

OBTAINING THE DP8344 BCP INVERSE ASSEMBLER

The DP8344 BCP Inverse Assembler package for use in a Hewlett Packard Logic Analyzer can be obtained from National Semiconductor. Included in the Inverse Assembler Package is the DP8344 BCP Inverse Assembler software, including configuration files as described in this application note. These will be on a 3½" diskette formatted for use in the HP Logic Analyzer. Additionally, a 5¼" diskette formatted for use on an IBM personal computer or compatible, containing the DP8344 Inverse Assembler source code can be obtained upon a request from National Semiconductor.

If one owns the HP 10391A Inverse Assembler Development Package, the source code can be modified to make any improvements one wishes to make to the DP8344 BCP Inverse Assembler. Note that it is not necessary to have the HP 10391A Inverse Assembler Development Package to use the DP8344 BCP Inverse Assembler.

State/Timing E | Listing 1 | Invasm | Cancel | Run

Markers Off

Label > Base >	ADDR Hex	DATA Hex	DP8344 BCP MNEMONIC Hex	Time Relative
199	006A	FD08	MOVE GP4/GP4', GP4/GP4'	120 ns
200	006B	CE00	LJMP 006AH	80 ns
201	006C	006A	ADD 06H, GP6/GP6'	120 ns
202	006A	FD08	MOVE GP4/GP4', GP4/GP4'	120 ns
203	006B	CE00	LJMP 006AH	80 ns
204	006C	006A	ADD 06H, GP6/GP6'	120 ns
205	006A	FD08	MOVE GP4/GP4', GP4/GP4'	80 ns
206	006B	CE00	LJMP 006AH	120 ns
207	006C	006A	ADD 06H, GP6/GP6'	120 ns
208	006A	FD08	MOVE GP4/GP4', GP4/GP4'	80 ns
209	006B	CE00	LJMP 006AH	120 ns
210	006C	006A	ADD 06H, GP6/GP6'	120 ns
211	006A	FD08	MOVE GP4/GP4', GP4/GP4'	80 ns
212	006B	CE00	LJMP 006AH	120 ns
213	006C	006A	ADD 06H, GP6/GP6'	120 ns
214	006A	FD08	MOVE GP4/GP4', GP4/GP4'	80 ns

TL/F/10814-15

FIGURE 13. Incorrectly Disassembled Instructions Produced by Jumping Ahead in Display

State/Timing E | Listing 1 | Invasm | Cancel | Run

Markers Off

Label > Base >	ADDR Hex	DATA Hex	DP8344 BCP MNEMONIC Hex	Time Relative
199	006A	FD08	MOVE GP4/GP4', GP4/GP4'	120 ns
200	006B	CE00	LJMP 006AH	80 ns
201	006C	006A		120 ns
202	006A	FD08	MOVE GP4/GP4', GP4/GP4'	120 ns
203	006B	CE00	LJMP 006AH	80 ns
204	006C	006A		120 ns
205	006A	FD08	MOVE GP4/GP4', GP4/GP4'	80 ns
206	006B	CE00	LJMP 006AH	120 ns
207	006C	006A		120 ns
208	006A	FD08	MOVE GP4/GP4', GP4/GP4'	80 ns
209	006B	CE00	LJMP 006AH	120 ns
210	006C	006A		120 ns
211	006A	FD08	MOVE GP4/GP4', GP4/GP4'	80 ns
212	006B	CE00	LJMP 006AH	120 ns
213	006C	006A		120 ns
214	006A	FD08	MOVE GP4/GP4', GP4/GP4'	80 ns

FIGURE 14. Instructions from *Figure 13* Correctly Disassembled after Choosing the "Invasm" Pop-Out

	Alternate	Main	
A:	DCR	CCR	R0
	IBR	NCF	R1
	ATR	ICR	R2
	FBR	ACR	R3
B:	RTR	GP0	R4
	TSR	GP1	R5
	TCR	GP2	R6
	TMR	GP3	R7
	GP4'	GP4 (accumulator)	R8
	GP5'	GP5	R9
	GP6'	GP6	R10
	GP7'	GP7	R11
Index Registers (pointers)		W (low byte)	R12
		W (high byte)	R13
		X (low byte)	R14
		X (high byte)	R15
		Y (low byte)	R16
		Y (high byte)	R17
		Z (low byte)	R18
		Z (high byte)	R19
		GP8	R20
		GP9	R21
		GP10	R22
		GP11	R23
		GP12	R24
		GP13	R25
		GP14	R26
		GP15	R27
Timer		TRL	R28
		TRH	R29
Stacks		ISP	R30
		DS	R31

FIGURE 15. Register Map

State/Timing E | Listing 1 | Invasm | Cancel | Run

Markers Off

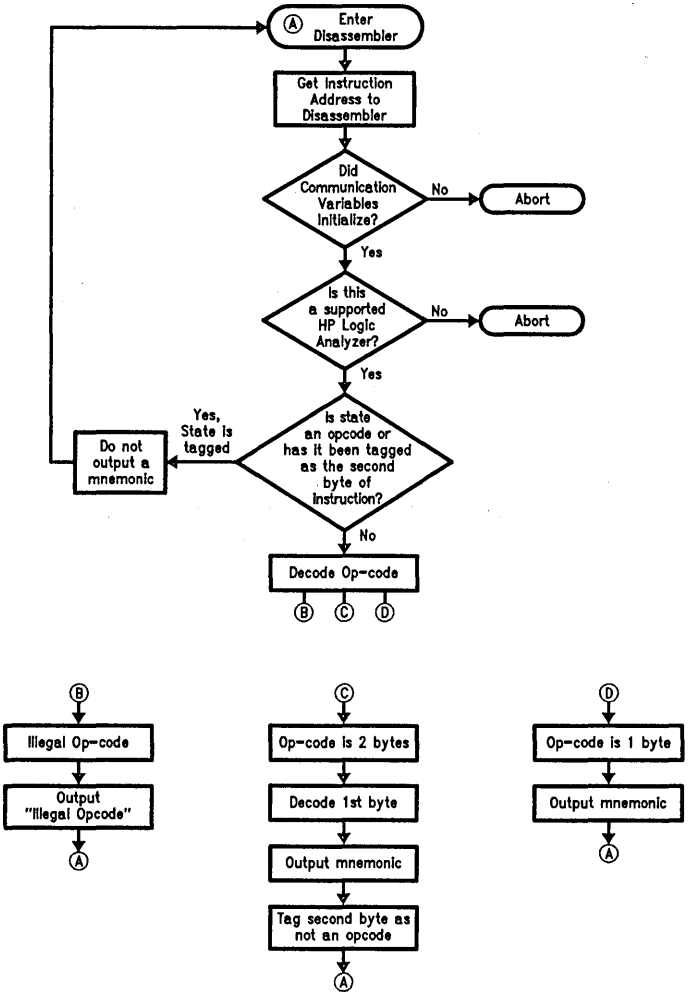
Label > Base >	ADDR Hex	DATA Hex	DP8344 BCP MNEMONIC Hex	Time Relative
45	002B	C340	MOVE CCR/DCR, [IY+]	160 ns
46	002C	E968	SUBA GP4/GP4', GP7/GP7'	160 ns
47	002D	C340	MOVE CCR/DCR, [IY+]	120 ns
48	002E	A52B	SUBA GP7/GP7', [IX+]	160 ns
49	002F	C340	MOVE CCR/DCR, [IX+]	160 ns
50	0030	207B	SUB 07H, GP7/GP7'	160 ns
51	0031	C340	MOVE CCR/DCR [IY+]	80 ns
52	0032	ED6B	SBCA GP7/GP7', GP7/GP7'	160 ns
53	0033	C340	MOVE CCR/DCR [IY+]	120 ns
54	0034	A72B	SBCA GP7/GP7' [IX+]	160 ns
55	0035	C340	MOVE CCR/DCR [IY+]	160 ns
56	0036	A92B	ANDA GP7/GP7', [IX+]	160 ns
57	0037	C340	MOVE CCR/DCR, [IY+]	160 ns
58	0038	F573	ORA IZHI, GP7/GP7'	160 ns
59	0039	C340	MOVE CCR/DCR, [IY+]	80 ns
60	003A	AB2B	ORA GP7/GP7', [IX+]	160 ns

TL/F/10814-17

FIGURE 16. Listing of Inverse Assembler on Logic Analyzer Demonstrating the Display of Both Register Bank Names

APPENDIX A

Flow Chart of DP8344 Inverse Assembler Source Code



TL/F/10814-18

Section 3

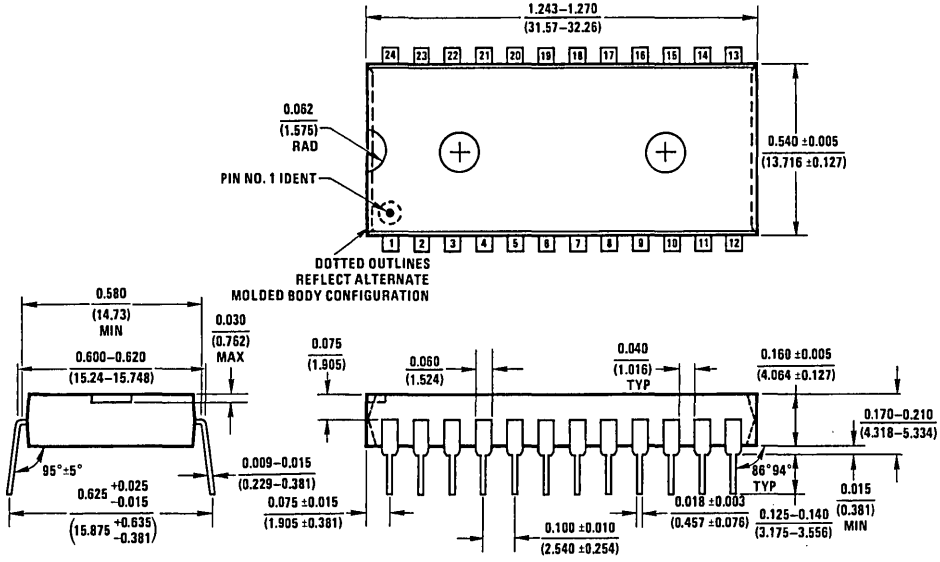
Physical Dimensions

Section 3 Contents

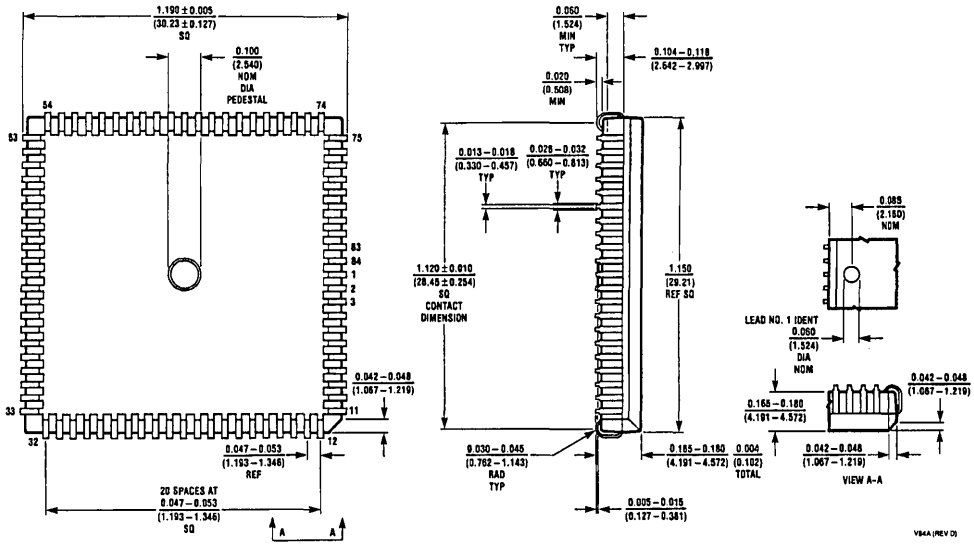
National Semiconductor

All dimensions are in inches (millimeters)

24 Lead Molded Dual-In-Line Package (N)
NS Package Number N24A



84 Lead Plastic Chip Carrier (V)
NS Package Number V84A



NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

National Semiconductor

Bookshelf of Technical Support Information

National Semiconductor Corporation recognizes the need to keep you informed about the availability of current technical literature.

This bookshelf is a compilation of books that are currently available. The listing that follows shows the publication year and section contents for each book.

Please contact your local National sales office for possible complimentary copies. A listing of sales offices follows this bookshelf.

We are interested in your comments on our technical literature and your suggestions for improvement.

Please send them to:

Technical Communications Dept. M/S 16-300
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090

ALS/AS LOGIC DATABOOK—1990

Introduction to Advanced Bipolar Logic • Advanced Low Power Schottky • Advanced Schottky

ASIC DESIGN MANUAL/GATE ARRAYS & STANDARD CELLS—1987

SSI/MSI Functions • Peripheral Functions • LSI/VLSI Functions • Design Guidelines • Packaging

CMOS LOGIC DATABOOK—1988

CMOS AC Switching Test Circuits and Timing Waveforms • CMOS Application Notes • MM54HC/MM74HC MM54HCT/MM74HCT • CD4XXX • MM54CXXX/MM74CXXX • Surface Mount

DATA ACQUISITION LINEAR DEVICES—1989

Active Filters • Analog Switches/Multiplexers • Analog-to-Digital Converters • Digital-to-Analog Converters Sample and Hold • Temperature Sensors • Voltage Regulators • Surface Mount

DISCRETE SEMICONDUCTOR PRODUCTS DATABOOK—1989

Selection Guide and Cross Reference Guides • Diodes • Bipolar NPN Transistors
Bipolar PNP Transistors • JFET Transistors • Surface Mount Products • Pro-Electron Series
Consumer Series • Power Components • Transistor Datasheets • Process Characteristics

DRAM MANAGEMENT HANDBOOK—1991

Dynamic Memory Control • Error Detection and Correction • Microprocessor Applications for the DP8408A/09A/17/18/19/28/29 • Microprocessor Applications for the DP8420A/21A/22A Microprocessor Applications for the NS32CG821

EMBEDDED SYSTEM PROCESSOR DATABOOK—1989

Embedded System Processor Overview • Central Processing Units • Slave Processors • Peripherals Development Systems and Software Tools

FDDI DATABOOK—1991

FDDI Overview • DP83200 FDDI Chip Set • Development Support • Application Notes and System Briefs

F100K ECL LOGIC DATABOOK & DESIGN GUIDE—1990

Family Overview • 300 Series (Low-Power) Datasheets • 100 Series Datasheets • 11C Datasheets
ECL BiCMOS SRAM, ECL PAL, and ECL ASIC Datasheets • Design Guide • Circuit Basics • Logic Design
Transmission Line Concepts • System Considerations • Power Distribution and Thermal Considerations
Testing Techniques • Quality Assurance and Reliability • Application Notes

FACT™ ADVANCED CMOS LOGIC DATABOOK—1990

Description and Family Characteristics • Ratings, Specifications and Waveforms
Design Considerations • 54AC/74ACXXX • 54ACT/74ACTXXX • Quiet Series: 54ACQ/74ACQXXX
Quiet Series: 54ACTQ/74ACTQXXX • 54FCT/74FCTXXX • FCTA: 54FCTXXXA/74FCTXXXA

FAST® ADVANCED SCHOTTKY TTL LOGIC DATABOOK—1990

Circuit Characteristics • Ratings, Specifications and Waveforms • Design Considerations • 54F/74FXXX

FAST® APPLICATIONS HANDBOOK—1990

Reprint of 1987 Fairchild FAST Applications Handbook
Contains application information on the FAST family: Introduction • Multiplexers • Decoders • Encoders
Operators • FIFOs • Counters • TTL Small Scale Integration • Line Driving and System Design
FAST Characteristics and Testing • Packaging Characteristics

GENERAL PURPOSE LINEAR DEVICES DATABOOK—1989

Continuous Voltage Regulators • Switching Voltage Regulators • Operational Amplifiers • Buffers • Voltage Comparators
Instrumentation Amplifiers • Surface Mount

GRAPHICS HANDBOOK—1989

Advanced Graphics Chipset • DP8500 Development Tools • Application Notes

IBM DATA COMMUNICATIONS HANDBOOK—1992

IBM Data Communications • Application Notes

INTERFACE DATABOOK—1990

Transmission Line Drivers/Receivers • Bus Transceivers • Peripheral Power Drivers • Display Drivers
Memory Support • Microprocessor Support • Level Translators and Buffers • Frequency Synthesis • Hi-Rel Interface

LINEAR APPLICATIONS HANDBOOK—1991

The purpose of this handbook is to provide a fully indexed and cross-referenced collection of linear integrated circuit applications using both monolithic and hybrid circuits from National Semiconductor.

Individual application notes are normally written to explain the operation and use of one particular device or to detail various methods of accomplishing a given function. The organization of this handbook takes advantage of this innate coherence by keeping each application note intact, arranging them in numerical order, and providing a detailed Subject Index.

LOCAL AREA NETWORK DATABOOK—1992

Integrated Ethernet Network Interface Controller Products • Ethernet Physical Layer Transceivers
Ethernet Repeater Interface Controller Products • Hardware and Software Support Products • FDDI Products • Glossary

LS/S/TTL DATABOOK—1989

Contains former Fairchild Products
Introduction to Bipolar Logic • Low Power Schottky • Schottky • TTL • TTL—Low Power

MASS STORAGE HANDBOOK—1989

Rigid Disk Pulse Detectors • Rigid Disk Data Separators/Synchronizers and ENDECs
Rigid Disk Data Controller • SCSI Bus Interface Circuits • Floppy Disk Controllers • Disk Drive Interface Circuits
Rigid Disk Preamplifiers and Servo Control Circuits • Rigid Disk Microcontroller Circuits • Disk Interface Design Guide

MEMORY DATABOOK—1990

PROMs, EPROMs, EEPROMs • TTL I/O SRAMs • ECL I/O SRAMs

MICROCONTROLLER DATABOOK—1989

COP400 Family • COP800 Family • COPS Applications • HPC Family • HPC Applications
MICROWIRE and MICROWIRE/PLUS Peripherals • Microcontroller Development Tools

MICROPROCESSOR DATABOOK—1989

Series 32000 Overview • Central Processing Units • Slave Processors • Peripherals
Development Systems and Software Tools • Application Notes • NSC800 Family

PROGRAMMABLE LOGIC DATABOOK & DESIGN MANUAL—1990

Product Line Overview • Datasheets • Designing with PLDs • PLD Design Methodology • PLD Design Development Tools
Fabrication of Programmable Logic • Application Examples

REAL TIME CLOCK HANDBOOK—1991

Real Time Clocks and Timer Clock Peripherals • Application Notes

RELIABILITY HANDBOOK—1986

Reliability and the Die • Internal Construction • Finished Package • MIL-STD-883 • MIL-M-38510
The Specification Development Process • Reliability and the Hybrid Device • VLSI/VHSIC Devices
Radiation Environment • Electrostatic Discharge • Discrete Device • Standardization
Quality Assurance and Reliability Engineering • Reliability and Documentation • Commercial Grade Device
European Reliability Programs • Reliability and the Cost of Semiconductor Ownership
Reliability Testing at National Semiconductor • The Total Military/Aerospace Standardization Program
883B/RETS™ Products • MILS/RETS™ Products • 883/RETS™ Hybrids • MIL-M-38510 Class B Products
Radiation Hardened Technology • Wafer Fabrication • Semiconductor Assembly and Packaging
Semiconductor Packages • Glossary of Terms • Key Government Agencies • AN/ Numbers and Acronyms
Bibliography • MIL-M-38510 and DESC Drawing Cross Listing

SPECIAL PURPOSE LINEAR DEVICES DATABOOK—1989

Audio Circuits • Radio Circuits • Video Circuits • Motion Control Circuits • Special Function Circuits
Surface Mount

TELECOMMUNICATIONS—1990

Line Card Components • Integrated Services Digital Network Components • Analog Telephone Components
Application Notes

NATIONAL SEMICONDUCTOR CORPORATION DISTRIBUTORS

ALABAMA
Huntsville
Arrow Electronics
(205) 837-6955
Bell Industries
(205) 837-1074
Hamilton/Avnet
(205) 837-7210
Pioneer Technology
(205) 837-9300
Time Electronics
(205) 721-1133

ARIZONA
Chandler
Hamilton/Avnet
(602) 961-1211
Phoenix
Arrow Electronics
(602) 437-0750
Tempe
Anthem Electronics
(602) 966-6600
Bell Industries
(602) 966-7800
Time Electronics
(602) 967-2000

CALIFORNIA
Agora Hills
Bell Industries
(818) 706-2608
Time Electronics
(818) 707-2890
Zeus Components
(818) 889-3838
Burbank
Elmo Semiconductor
(818) 768-7400
Calabasas
F/X Electronics
(818) 592-0120
Chatsworth
Anthem Electronics
(818) 700-1000
Arrow Electronics
(818) 701-7500
Time Electronics
(818) 998-7200
Costa Mesa
Avnet Electronics
(714) 754-6050
Hamilton Electro Sales
(714) 641-4100
Cypress
Bell Industries
(714) 895-7801
Gardena
Hamilton/Avnet
(213) 516-8600
Irvine
Anthem Electronics
(714) 768-4444
Rocklin
Anthem Electronics
(916) 624-9744
Bell Industries
(916) 652-0414
Roseville
Hamilton/Avnet
(916) 925-2216
San Diego
Anthem Electronics
(619) 453-9005
Arrow Electronics
(619) 565-4800
Hamilton/Avnet
(619) 571-1900
Time Electronics
(619) 586-1331
Zeus Components
(619) 277-9681
San Jose
Anthem Electronics
(408) 453-1200
Arrow Electronics
(408) 441-9700
Pioneer Technology
(408) 954-9100
Zeus Components
(408) 629-4789
Sunnyvale
Bell Industries
(408) 734-8570
Hamilton/Avnet
(408) 743-3300
Time Electronics
(408) 734-9888
Torrance
Time Electronics
(213) 320-0880
Tustin
Arrow Electronics
(714) 838-5422
Time Electronics
(714) 937-0911
Woodland Hills
Hamilton/Avnet
(818) 594-0404
Yorba Linda
Zeus Components
(714) 921-9000

COLORADO
Aurora
Arrow Electronics
(303) 373-5616
Englewood
Anthem Electronics
(303) 790-4500
Hamilton/Avnet
(303) 799-7800
Time Electronics
(303) 721-8882
Wheatridge
Bell Industries
(303) 424-1985

CONNECTICUT
Danbury
Hamilton/Avnet
(203) 743-6077
Shelton
Pioneer Standard
(203) 929-5600
Wallingford
Arrow Electronics
(203) 265-7741
Waterbury
Anthem Electronics
(203) 575-1575

FLORIDA
Altamonte Springs
Bell Industries
(407) 339-0078
Pioneer Technology
(407) 834-9090
Zeus Components
(407) 788-9100
Clearwater
Pioneer Technology
(813) 536-0445
Deerfield Beach
Arrow Electronics
(305) 429-8200
Bell Industries
(305) 421-1997
Pioneer Technology
(305) 428-8877
Fort Lauderdale
Hamilton/Avnet
(305) 767-6377
Time Electronics
(305) 484-7778
Lake Mary
Arrow Electronics
(407) 333-9300
Orlando
Chip Supply
(407) 298-7100
Time Electronics
(407) 841-6565
St. Petersburg
Hamilton/Avnet
(813) 572-4329
Winter Park
Hamilton/Avnet
(407) 657-3300

GEORGIA
Duluth
Arrow Electronics
(404) 497-1300
Hamilton/Avnet
(404) 446-0611
Pioneer Technology
(404) 623-1003
Norcross
Bell Industries
(404) 662-0923
Time Electronics
(404) 368-0969

ILLINOIS
Addison
Pioneer Electronics
(708) 495-9680
Bensenville
Hamilton/Avnet
(708) 860-7700
Elk Grove Village
Bell Industries
(708) 640-1910
Itasca
Arrow Electronics
(708) 250-0500
Schaumburg
Anthem Electronics
(708) 884-0200
Time Electronics
(708) 303-3000

INDIANA
Carmel
Hamilton/Avnet
(317) 844-9333
Fort Wayne
Bell Industries
(219) 423-3422
Indianapolis
Advent Electronics Inc.
(317) 872-4910
Arrow Electronics
(317) 299-2071
Bell Industries
(317) 875-8200
Pioneer Standard
(317) 573-0880

IOWA
Cedar Rapids
Advent Electronics
(319) 363-0221
Arrow Electronics
(319) 395-7230
Hamilton/Avnet
(319) 362-4757

KANSAS
Lenexa
Arrow Electronics
(913) 541-9542
Hamilton/Avnet
(913) 888-8900

MARYLAND
Columbia
Anthem Electronics
(301) 995-6840
Arrow Electronics
(301) 995-6002
Hamilton/Avnet
(301) 995-3500
Time Electronics
(301) 964-3090
Zeus Components
(301) 997-1118
Gaithersburg
Pioneer Technology
(301) 921-0660

MASSACHUSETTS
Andover
Bell Industries
(508) 474-8880
Beverly
Sertech Laboratories
(508) 927-5820
Lexington
Pioneer Standard
(617) 861-9200
Norwood
Gerber Electronics
(617) 769-6000
Peabody
Hamilton/Avnet
(508) 531-7430
Time Electronics
(508) 532-9900
Tyngsboro
Port Electronics
(508) 649-4880
Wakefield
Zeus Components
(617) 246-8200
Wilmington
Anthem Electronics
(508) 657-5170
Arrow Electronics
(508) 658-0900

MICHIGAN
Grand Rapids
Arrow Electronics
(616) 243-0912
Pioneer Standard
(616) 698-1800
Grandville
Hamilton/Avnet
(616) 243-8805
Livonia
Arrow Electronics
(313) 665-4100
Pioneer Standard
(313) 525-1800
Novi
Hamilton/Avnet
(313) 347-4720
Wyoming
R. M. Electronics, Inc.
(616) 531-9300

MINNESOTA
Eden Prairie
Anthem Electronics
(612) 944-5454
Arrow Electronics
(612) 828-7140
Pioneer Standard
(612) 944-3355
Edina
Arrow Electronics
(612) 830-1800
Time Electronics
(612) 943-2433
Minnetonka
Hamilton/Avnet
(612) 932-0600

NATIONAL SEMICONDUCTOR CORPORATION DISTRIBUTORS (Continued)

MISSOURI
Chesterfield
Hamilton/Avnet
(314) 537-1600
St. Louis
Arrow Electronics
(314) 567-6888
Time Electronics
(314) 391-6444

NEW JERSEY
Cherry Hill
Hamilton/Avnet
(609) 424-0100
Fairfield
Hamilton/Avnet
(201) 575-3390
Pioneer Standard
(201) 575-3510
Marlton
Arrow Electronics
(609) 596-8000
Time Electronics
(609) 596-6700
Parsippany
Arrow Electronics
(201) 538-0900
Pine Brook
Anthem Electronics
(201) 227-7960
Wayne
Time Electronics
(201) 758-8250

NEW MEXICO
Albuquerque
Alliance Electronics Inc.
(505) 292-3360
Bell Industries
(505) 292-2700
Hamilton/Avnet
(505) 345-0001

NEW YORK
Binghamton
Pioneer
(607) 722-9300
Buffalo
Summit Electronics
(716) 887-2800
Commack
Anthem Electronics
(516) 864-6600
Fairport
Pioneer Standard
(716) 381-7070
Hauppauge
Arrow Electronics
(516) 231-1000
Hamilton/Avnet
(516) 231-9444
Time Electronics
(516) 273-0100
Port Chester
Zeus Components
(914) 937-7400
Rochester
Arrow Electronics
(716) 427-0300
Hamilton/Avnet
(716) 292-0730
Summit Electronics
(716) 334-8110
Ronkonkoma
Zeus Components
(516) 737-4500
Syracuse
Hamilton/Avnet
(315) 437-2641
Time Electronics
(315) 432-0355
Westbury
Hamilton/Avnet Export Div.
(516) 997-6868
Woodbury
Pioneer Electronics
(516) 921-8700

NORTH CAROLINA
Charlotte
Hamilton/Avnet
(704) 527-2485
Pioneer Technology
(704) 527-8188
Durham
Pioneer Technology
(919) 544-5400
Raleigh
Arrow Electronics
(919) 876-3132
Hamilton/Avnet
(919) 878-0810
Time Electronics
(919) 874-9650

OHIO
Centerville
Arrow Electronics
(513) 435-5563
Cleveland
Pioneer
(216) 587-3600
Columbus
Time Electronics
(614) 794-3301
Dayton
Bell Industries
(513) 435-8660
Bell Industries-Military
(513) 434-8231
Hamilton/Avnet
(513) 439-6700
Pioneer Standard
(513) 236-9900
Zeus Components
(513) 937-7400
Solon
Arrow Electronics
(216) 248-3990
Hamilton/Avnet
(216) 349-5100
Westerville
Hamilton/Avnet
(614) 882-7004

OKLAHOMA
Tulsa
Arrow Electronics
(918) 252-7537
Hamilton/Avnet
(918) 664-0444
Pioneer Standard
(918) 492-7840
Radio Inc.
(918) 587-9123

OREGON
Beaverton
Anthem Electronics
(503) 643-1114
Arrow Electronics
(503) 626-7667
Hamilton/Avnet
(503) 627-0201
Lake Oswego
Bell Industries
(503) 635-6500
Portland
Time Electronics
(503) 684-3780

PENNSYLVANIA
Horsham
Anthem Electronics
(215) 443-5150
Pioneer Technology
(215) 674-4000
Mars
Hamilton/Avnet
(412) 281-4150
Pittsburgh
Pioneer
(412) 782-2300

TEXAS
Austin
Arrow Electronics
(512) 835-4180
Hamilton/Avnet
(512) 837-8911
Minco Technology Labs.
(512) 834-2022
Pioneer Standard
(512) 835-4000
Time Electronics
(512) 399-3051
Carrollton
Arrow Electronics
(214) 380-6464
Dallas
Hamilton/Avnet
(214) 308-8111
Pioneer Standard
(214) 386-7300
Houston
Arrow Electronics
(713) 530-4700
Hamilton/Avnet
(713) 240-7733
Pioneer Standard
(713) 495-4700
Richardson
Anthem Electronics
(214) 238-7100
Time Electronics
(214) 644-4644
Zeus Components
(214) 783-7010

UTAH
Midvale
Bell Industries
(801) 255-9611
Salt Lake City
Anthem Electronics
(801) 973-8555
Arrow Electronics
(801) 973-6913
Hamilton/Avnet
(801) 972-2800
West Valley
Time Electronics
(801) 973-8494

WASHINGTON
Bellevue
Arrow Electronics
(206) 643-4800
Bothell
Anthem Electronics
(206) 483-1700
Kirkland
Time Electronics
(206) 820-1525
Redmond
Bell Industries
(206) 867-5410
Hamilton/Avnet
(206) 241-8555

WISCONSIN
Brookfield
Arrow Electronics
(414) 792-0150
Pioneer Electronics
(414) 784-3480
Mequon
Taylor Electric
(414) 241-4321
Waukesha
Bell Industries
(414) 547-8879
Hamilton/Avnet
(414) 784-8205

CANADA
WESTERN PROVINCES
Burnaby
Hamilton/Avnet
(604) 420-4101
Semad Electronics
(604) 420-9889
Calgary
Electro Sonic Inc.
(403) 255-9550
Semad Electronics
(403) 252-5664
Zentronics
(403) 295-8838
Edmonton
Zentronics
(403) 468-9306
Markham
Semad Electronics Ltd.
(416) 475-3922
Richmond
Electro Sonic Inc.
(604) 273-2911
Zentronics
(604) 273-5575
Saskatoon
Zentronics
(306) 955-2207
Winnipeg
Zentronics
(204) 694-1957

EASTERN PROVINCES
Mississauga
Hamilton/Avnet
(416) 795-3825
Time Electronics
(416) 672-5300
Zentronics
(416) 564-9600
Nepean
Hamilton/Avnet
(613) 226-1700
Zentronics
(613) 226-8840
Ottawa
Electro Sonic Inc.
(613) 728-8333
Semad Electronics
(613) 727-8325
Pointe Claire
Semad Electronics
(514) 694-0860
St. Laurent
Hamilton/Avnet
(514) 335-1000
Zentronics
(514) 737-9700
Willowdale
ElectroSonic Inc.
(416) 494-1666
Winnipeg
Electro Sonic Inc.
(204) 783-3105

National Semiconductor

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1-800-272-9959
TWX: (910) 339-9240

SALES OFFICES (Continued)

INTERNATIONAL OFFICES

Electronica NSC de Mexico SA
Juventino Rosas No. 118-2
Col Guadalupe Inn
Mexico, 01020 D.F. Mexico
Tel: 52-5-524-9402
Fax: 52-5-524-9342

National Semicondutores Do Brasil Ltda.
Av. Brig. Faria Lima, 1409
6.0 Andar
Cep. 01451 J. Paulistano
Sao Paulo, SP, Brasil
Tel: (55/11) 212-5066
Telex: 391 1131931
Fax: (55/11) 212-1181 NSBR BR

National Semiconductor GmbH
Eschborner Lanstr. 130-132
D-6000 Frankfurt 90
Germany
Tel: (069) 78 91 09-0
Fax: (069) 7 89 43 83

National Semiconductor GmbH
Industriestrasse 10
D-8080 Furstenfeldbruck
Germany
Tel: (0-81-41) 103-0
Telex: 527-649
Fax: (08141) 103554

National Semiconductor GmbH
Misburger Strasse 81D
D3000 Hannover 61
Germany
Tel: (0511) 560040
Fax: (0511) 561740

National Semiconductor GmbH
Untere Waldplatze 37
D-7000 Stuttgart 80
Germany
Tel: 711 686 511
Fax: 711 686 5260

National Semiconductor (UK) Ltd.
The Maple, Kembrey Park
Swindon, Wiltshire SN2 6UT
United Kingdom
Tel: (07-93) 61-41-41
Telex: 444-674
Fax: (07-93) 69-75-22

National Semiconductor Benelux
Vorstlaan 100
B-1170 Brussels
Belgium
Tel: (02) 6-61-06-80
Telex: 61007
Fax: (02) 6-60-23-95

National Semiconductor (UK) Ltd.
Ringager 4A, 3
DK-2605 Brandy
Denmark
Tel: (02) 43-32-11
Telex: 15-179
Fax: (02) 43-31-11

National Semiconductor S.A.
Centre d'Affaires-La Boursidiere
Bâtiment Champagne, B.P. 90
Route Nationale 186
F-92357 Le Plessis Robinson
Paris, France
Tel: (1) 40-94-88-88
Telex: 631065
Fax: (1) 40-94-88-11

National Semiconductor (UK) Ltd.
Unit 2A
Clonskeagh Square
Clonskeagh Road
Dublin 14
Ireland
Tel: (01) 269-55-89
Telex: 91047
Fax: (01) 2830650

National Semiconductor S.p.A.
Strada 7, Palazzo R/3
I-20089 Rozzano
Milanofiori
Italy
Tel: (02) 57 50 03 00
Twx: 352647
Fax: (02) 57 50 04 00

National Semiconductor S.p.A.
Via del Cararaggio, 107
I-00147 Rome
Italy
Tel: (06) 5-13-48-80
Fax: (06) 5-13-79-47

National Semiconductor (UK) Ltd.
Isveien 45
Postboks 57
N-1393 Ostenstad
Norway
Tel: (2) 796500
Fax: (2) 796040

National Semiconductor AB
P.O. Box 1009
Grosshandlarvaegen 7
S-121 23 Johanneshov
Sweden
Tel: 46-8-7228050
Fax: 46-8-7229095
Telex: 10731 NSC S

National Semiconductor GmbH
Calle Agustin de Foxa, 27 (9°D)
E-28036 Madrid
Spain
Tel: (01) 733-2958
Telex: 46133
Fax: (01) 733-8018

National Semiconductor Switzerland
Alte Winterthurerstrasse 53
Postfach 567
Ch-8304 Wallisellen-Zurich
Switzerland
Tel: (01) 830-2727
Telex: 828-444
Fax: (01) 830-1900

National Semiconductor
Kauppakartanonkatu 7 A22
SF-00930 Helsinki
Finland
Tel: (90) 33-80-33
Telex: 126116
Fax: (90) 33-81-30

National Semiconductor
Postbus 90
NL1380 AB Weesp
The Netherlands
Tel: (0-29-40) 3-04-48
Telex: 10-956
Fax: (0-29-40) 3-04-30

National Semiconductor Japan Ltd.
Sanseido Bldg. 5F
4-15-3 Nishi Shinjuku
Shinjuku-ku
Tokyo 160 Japan
Tel: (03) 3299-7001
Fax: (03) 3299-7000

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block
Ocean Centre
5 Canton Road, Tsimshatsui East,
Kowloon, Hong Kong
Tel: (852) 737-1600
Telex: 51292 NSHKL
Fax: (852) 736-9960

National Semiconductor (Australia) PTY, Ltd.
Bldg. 16, Business Park Dr.
Melbourne, 3168
Victoria, Australia
Tel: (03) 558-9999
Fax: 61-3-558-9998

National Semiconductor (PTE), Ltd.
200 Cantonment Road 13-02
Southpoint 200
Singapore 0208
Tel: 2252229
Telex: RS 50808
Fax: (65) 225-7080

National Semiconductor (Far East) Ltd.
Taiwan Branch
9th Floor, No. 18
Sec. 1, Chang An East Road,
Taipei, Taiwan R.O.C.
Tel: (86) 521-3288
Telex: 22837 NSTW
Fax: 02 561-3054

National Semiconductor (Far East) Ltd.
Korea Branch
13th Floor, Dai Han Life Insurance
63 Building,
60, Yoido-dong, Youngdeungpo-ku,
Seoul, Korea 150-763
Tel: (02) 784-8051
Telex: 24942 NSPKLO
Fax: (02) 784-8054

 TL3564 RRD/RRD10M121/Printed in U.S.A.