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
April 1995

Table of Contents

1.0 INTRODUCTION
About This System User Guide
Contents of the MPA-II Design/Evaluation Kit
MPA-II Description
DP8344B BCP

2.0 OPERATION
System Requirements
Requirements for Design Development
Useful Tools
MPA-II Installation
Running Emulation: A Quick Start

3.0 DEVELOPMENT ENVIRONMENT

4.0 SOFTWARE OVERVIEW
IBM 3270 and 5250 Environments
3270 Data Stream Architecture
5250 Data Stream Architecture
Terminal Emulation
DCA
IBM
Screen Presentation
MPA-II

5.0 HARDWARE ARCHITECTURE
Architectural Overview
BCP Minimum System Core
PC Interface
Front End Interfaces
Miscellaneous Support

6.0 SOFTWARE ARCHITECTURE
Kernel
System Initialization
Coax Task
Coax Interrupt Handlers
IRMA Interface
IBM Interface
Twinax Task
Twinax Interrupt Handlers
Smart Alec Interface

7.0 LOADER AND MPA-II DIAGNOSTICS
Soft-Loading Instruction Memory
Configuring the MPA-II
MPA-II Diagnostics

APPENDIX A HARDWARE REFERENCE
MPA-II Schematic
MPA-II Layout
MPA-II Assembly Drawing
PAL Equations

APPENDIX B TIMING ANALYSIS

APPENDIX C FILTER EQUATIONS

APPENDIX D REFERENCES

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.

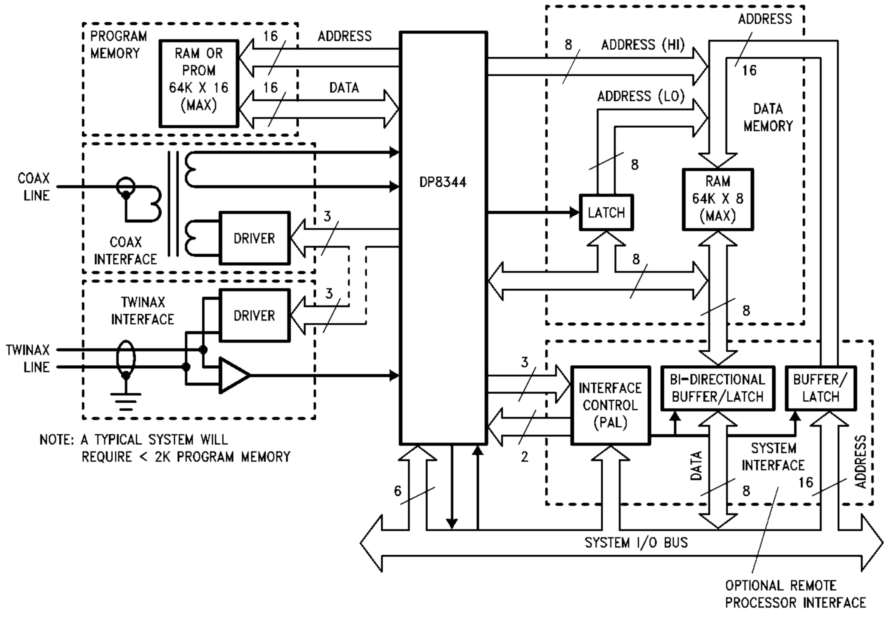


FIGURE 1-1. Block Diagram of Typical BCP System

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.

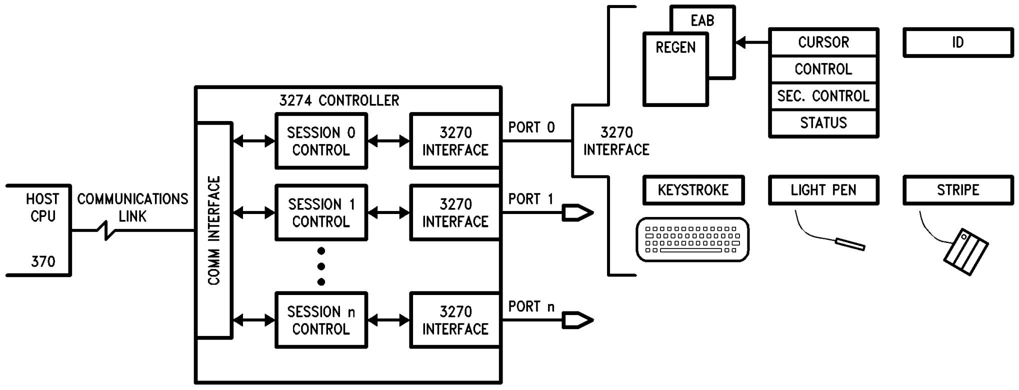
IBM 3270 and 5250 Environments

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 communcations 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

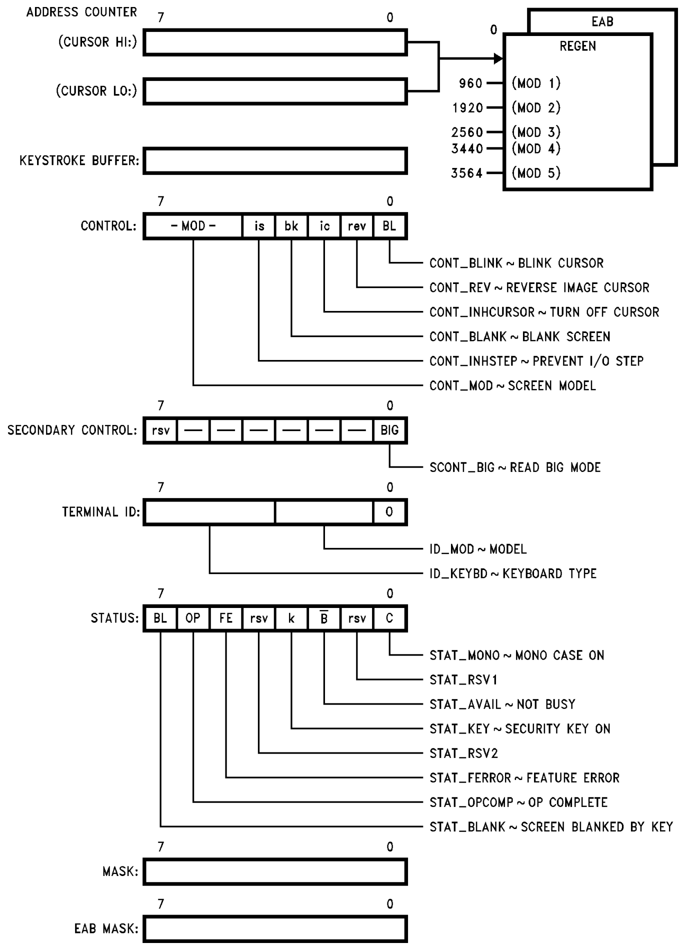
*Denotes foreground task

**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.



TL/F/10488-3

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.)

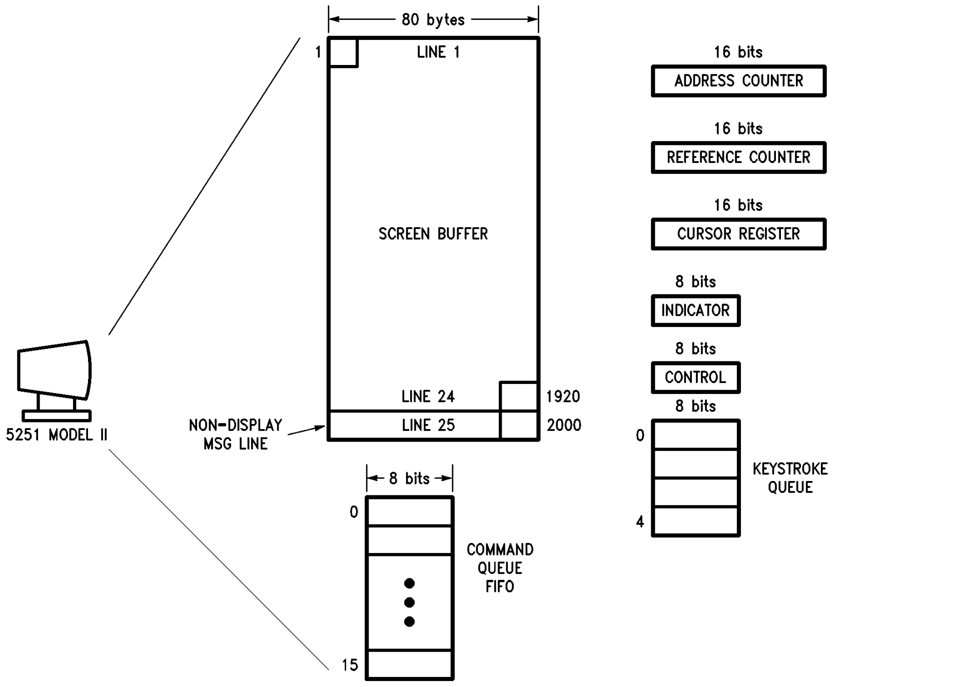
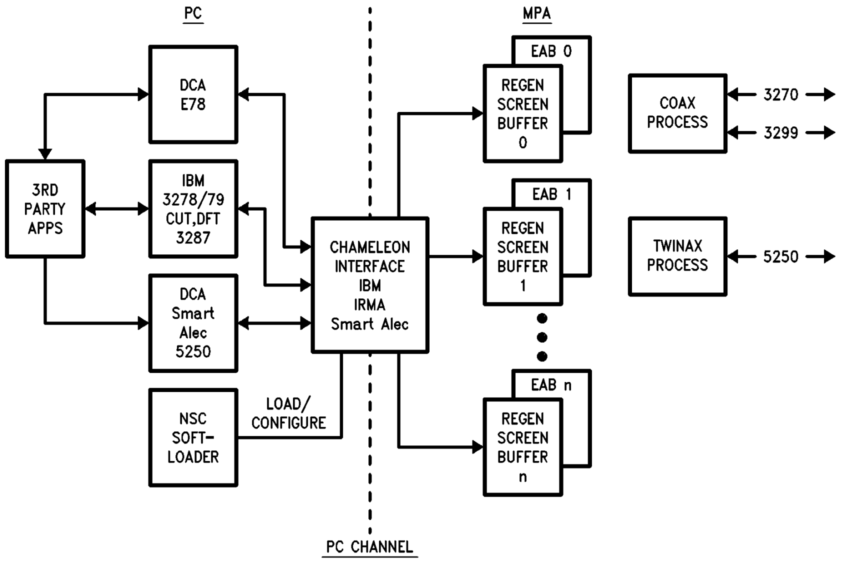


FIGURE 4-3. 5251 Terminal Architecture



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.

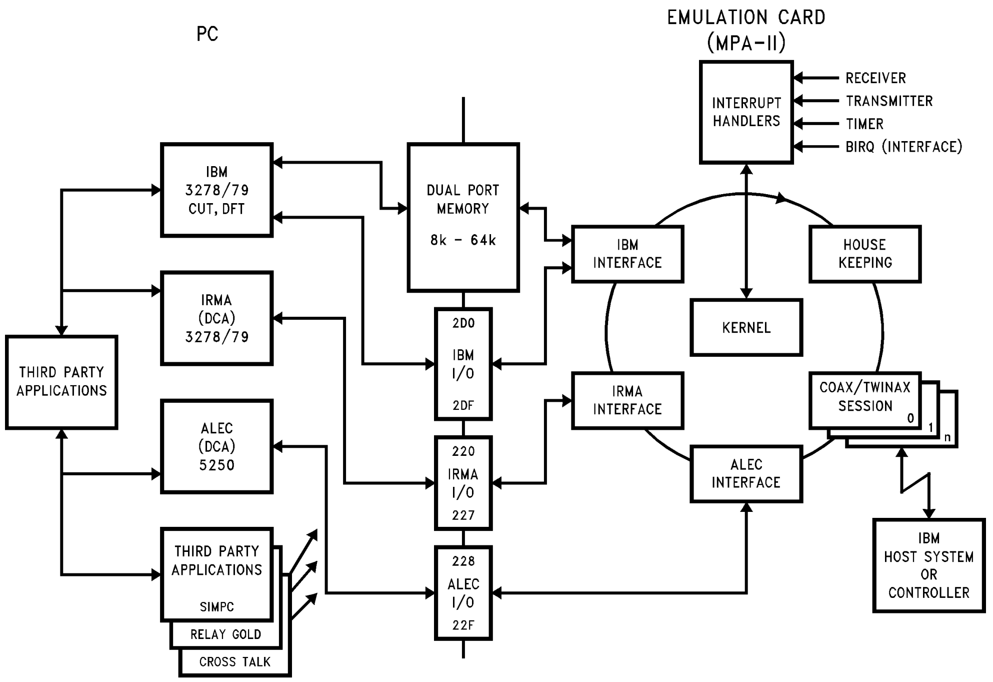
When either of the DCA modes are enabled, the I/O block 220h–22Fh is decoded, split into read and write banks, and mapped into the BCP's data memory. For the IBM mode, the I/O block from 2D0h–2DAh is decoded and the Write-Under-Mask function is enabled. In addition, the 8k of dual-port RAM is defined 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:

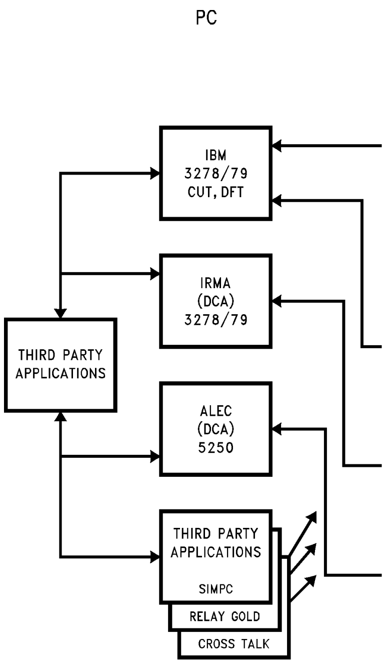
TABLE 4-3. MPA-II PC I/O Map

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



PC SOFTWARE

- REFORMATS AND TRANSLATES REGEN BUFFER TO ASCII FOR PC SCREEN*
- ACCEPTS KEY STROKES FROM USER
- PERFORMS FILE AND DATA TRANSFER

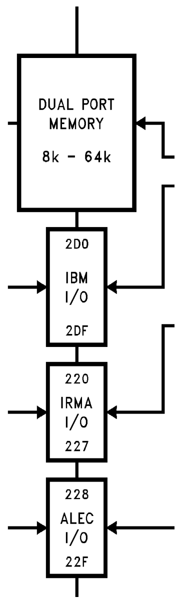
* THIS COULD BE DONE IN THE BCP MICROCODE

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

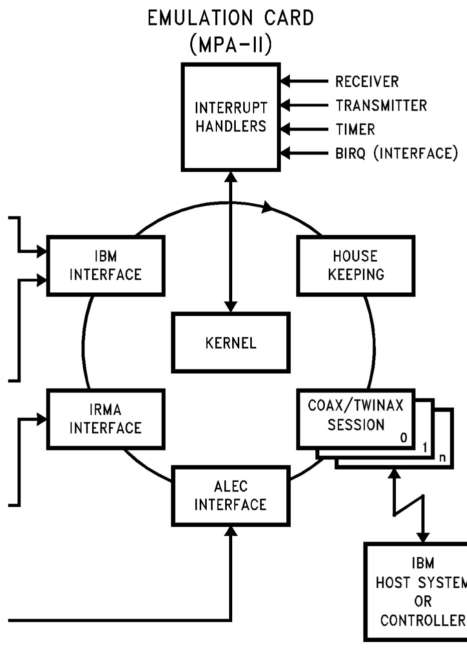


TL/F/10488–10

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

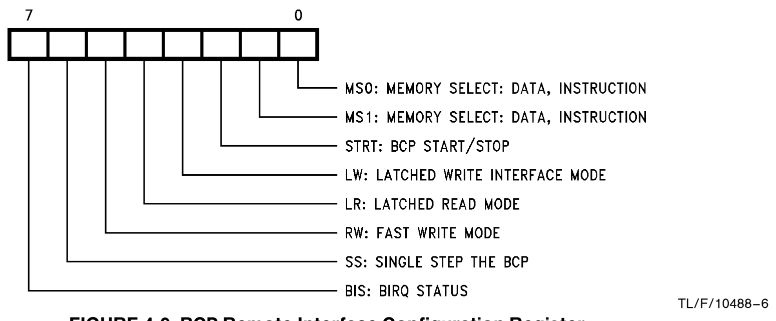


FIGURE 4-9. BCP Remote Interface Configuration Register

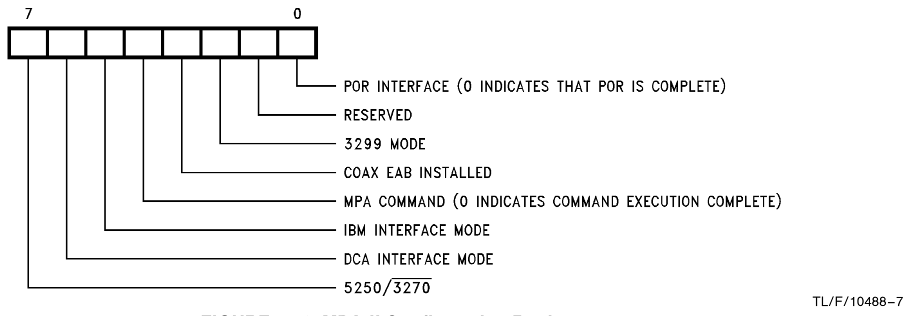


FIGURE 4-10. MPA-II Configuration Register

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.

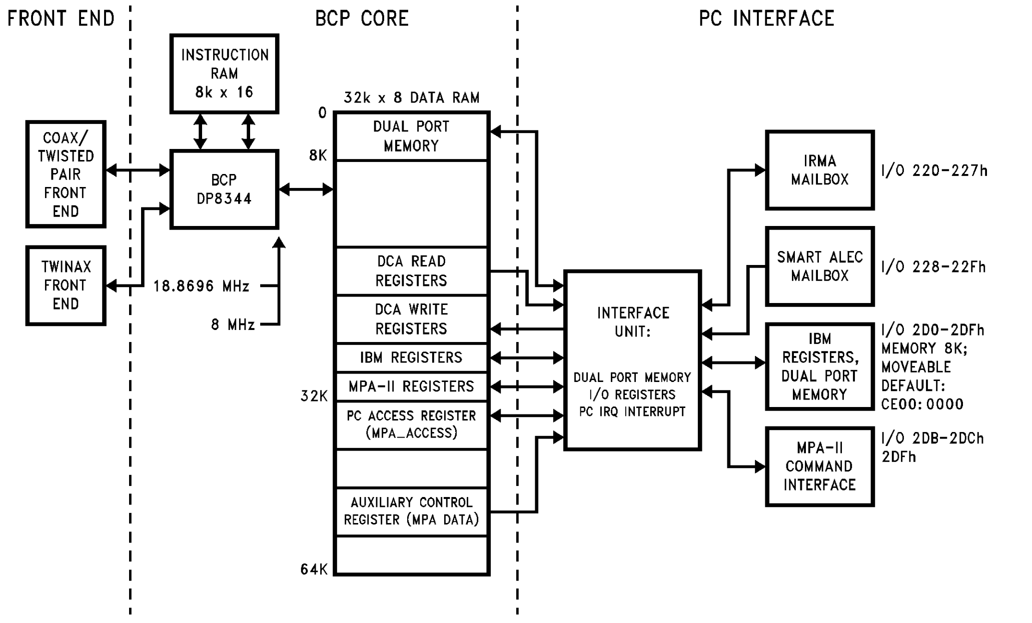
Architectural Overview

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

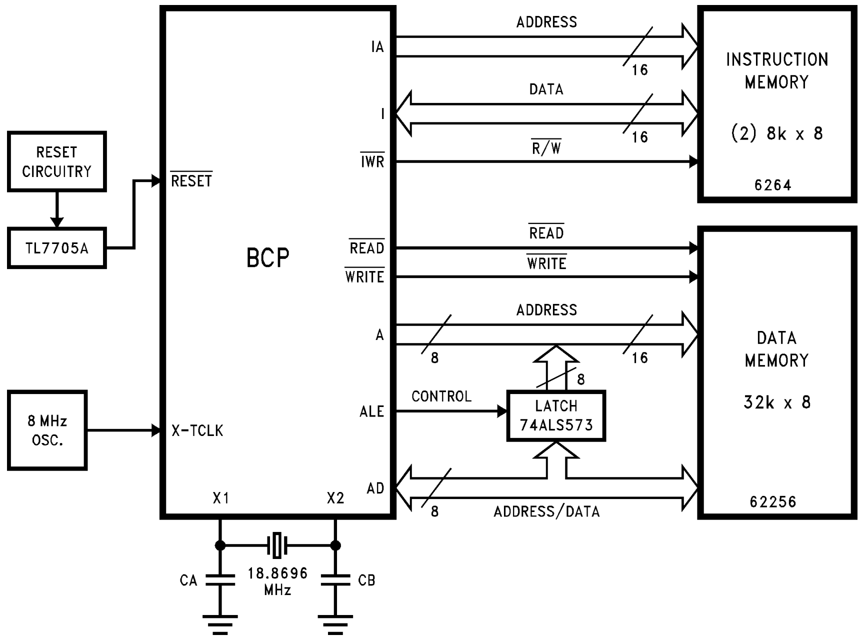
The coax and twinax protocols operate at substantially different clock frequencies (2.3587 MHz and 1 MHz, respectively), therefore two clock sources are required. The BCP has the software-programmable flexibility to drive both the CPU and transceiver in the following ways: the clock independently divided 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:

T-state = 1/(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 $\overline{\text{IWR}}$ (BCP Instruction WRite output, active low) minimum pulse width is calculated from Parameter 20 ($\overline{\text{IWR}}$ 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, $\overline{\text{IWR}}$ 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 $\overline{\text{IWR}}$ 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 $\overline{\text{IWR}}$. The BCP will hold that data on the bus for a minimum of 22 ns afterward (see Parameter 18, $\overline{\text{IWR}}$ 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 $\overline{\text{LCL}}$ (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.

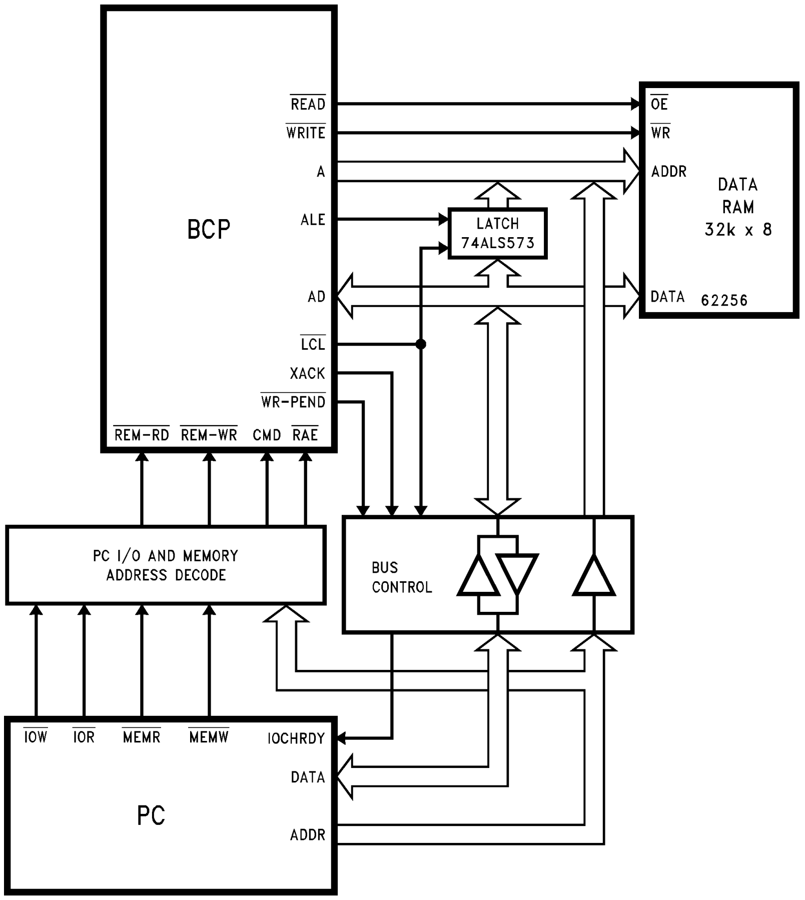
The PC address decoding is partitioned into sections that first check for accesses to the relocatable memory block and accesses to the I/O register addresses of the different interfaces. These addresses are then translated into the proper area of the BCP data memory. The BCP data memory map is divided 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 $\overline{\text{REMRD/REMWR}}$ 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 $\overline{\text{PC\_\_RDY}}$ is driven low whenever $\overline{\text{RAE}}$ = 0. After one CLK-OUT cycle the state machine moves to WAIT2.

STATE: WAIT2

In this state, REM__ENABLE = 0 and $\overline{\text{PC\_\_RDY}}$ is driven low whenever $\overline{\text{RAE}}$ = 0. After another CLK-OUT cycle the state machine moves to WAIT3.

STATE: WAIT3

In this state, REM__ENABLE = 0 and $\overline{\text{PC\_\_RDY}}$ is driven low whenever $\overline{\text{RAE}}$ = 0. After another CLK-OUT cycle the state machine moves to WAIT4.

STATE: WAIT4

In this state, REM__ENABLE = 0 and $\overline{\text{PC\_\_RDY}}$ is driven low whenever $\overline{\text{RAE}}$ = 0. After another CLK-OUT cycle the state machine moves to WAIT5.

STATE: WAIT5

In this state, REM__ENABLE = 0 and $\overline{\text{PC\_\_RDY}}$ is driven low whenever $\overline{\text{RAE}}$ = 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 $\overline{\text{PC\_\_RDY}}$ is driven low whenever $\overline{\text{RAE}}$ = 0. If a remote access has started (i.e., $\overline{\text{RAE}}$ = 1) the next state will be RESUME. Otherwise, the next state is WAIT7.

STATE: WAIT7

In this state, REM__ENABLE = 0 and $\overline{\text{PC\_\_RDY}}$ is driven low whenever $\overline{\text{RAE}}$ = 0. If a remote access has started (i.e., $\overline{\text{RAE}}$ = 1) the next state will be RESUME. Otherwise, the next state is WAIT8.

STATE: WAIT8

In this state, REM__ENABLE = 1 (allowing accesses) and $\overline{\text{PC\_\_RDY}}$ is driven low whenever $\overline{\text{RAE}}$ = 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 $\overline{\text{RAE}}$ = 1 (no access in progress). If $\overline{\text{RAE}}$ = 0, then the next state will be RESUME. Otherwise, the state machine returns to IDLE.

STATE: RESUME

In this state, REM__ENABLE = 1 and $\overline{\text{PC\_\_RDY}}$ is driven low while $\overline{\text{RAE}}$ = 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 $\overline{\text{PC\_\_RDY}}$. 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 $\overline{\text{PC\_\_RDY}}$ is driven low whenever $\overline{\text{RAE}}$ = 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 $\overline{\text{PC\_\_RDY}}$.

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 $\overline{\text{IO\_\_ACCESS}}$ 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. $\overline{\text{DATA-OUT}}$, 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. TX-ACT functions 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.

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.

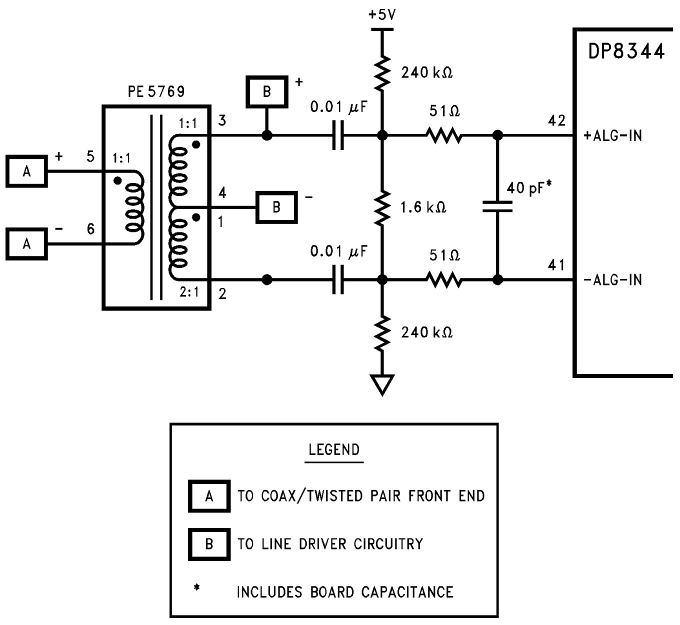
The interface design takes into account the common comparator attributes of power supply rejection, variable switching offset, finite voltage sensitivity, and fast edge rate sensitivity. V_{CC} noise can effect the comparator output when the inputs are biased to the same voltage.

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 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 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

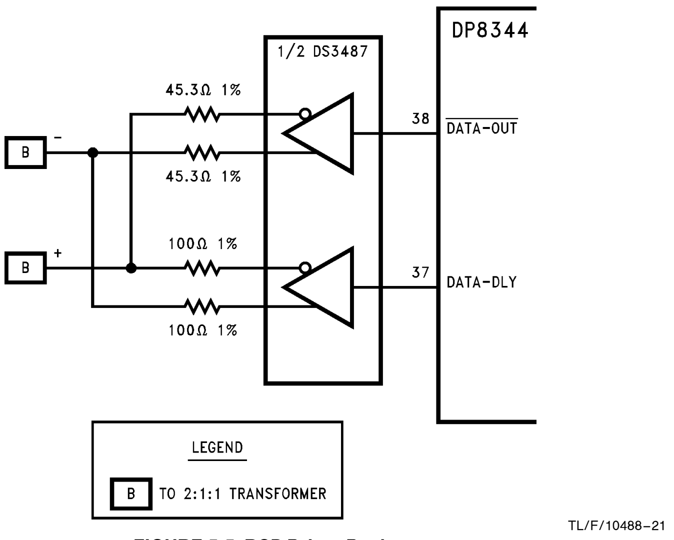


FIGURE 5-5. BCP Driver Design

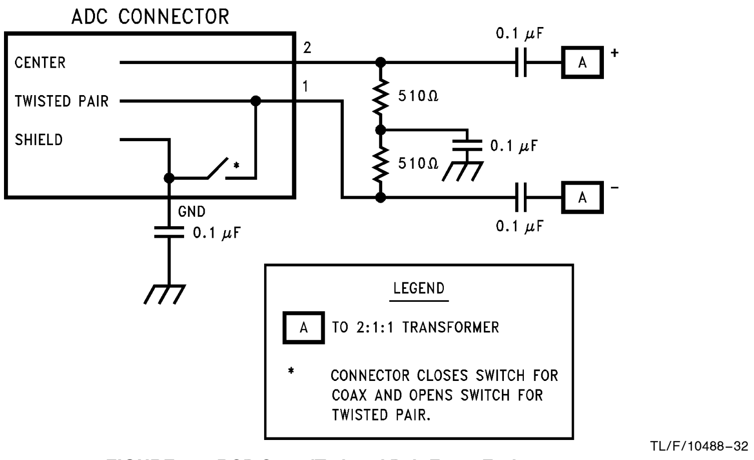


FIGURE 5-6. BCP Coax/Twisted Pair Front End

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.

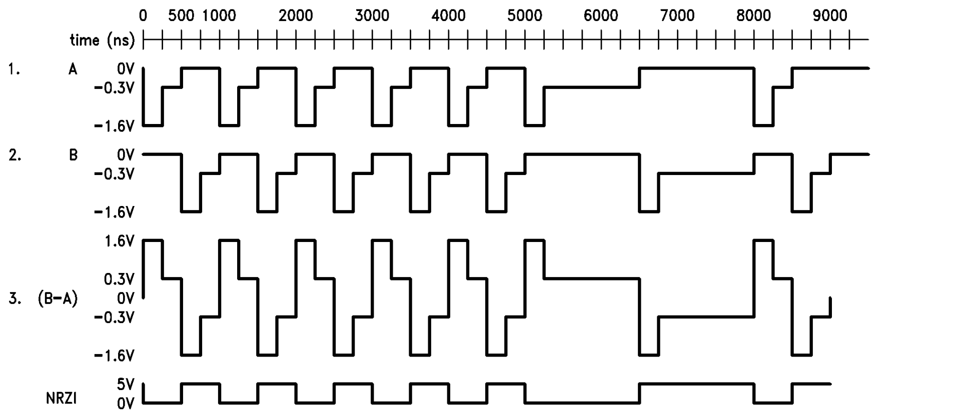
Twinax (5250) Interface

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.



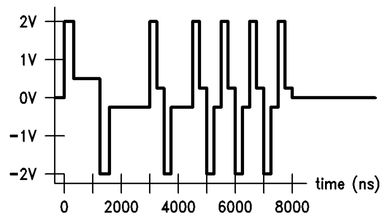
TL/F/10488–12

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.

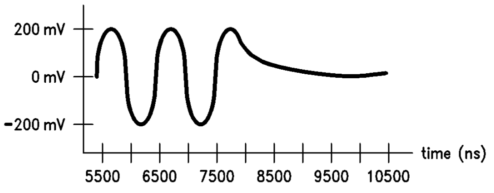
FIGURE 5-7. Twinax Waveforms



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.

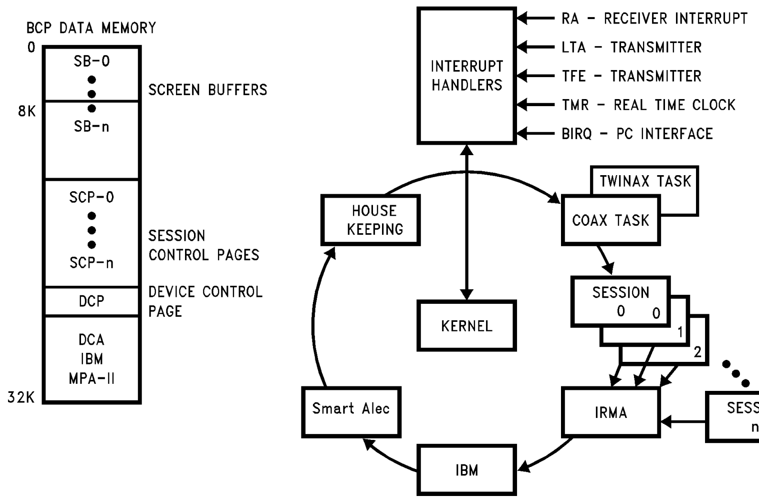
Each device has one 0.1 μF decoupling capacitor located as close as possible to the chip. These are chip capacitors (0.3 spacing, DIP configuration) to minimize lead length inductance and facilitate placement. The +5V supply line has two 22 μF electrolytic capacitors, one at each end of the board. The 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.

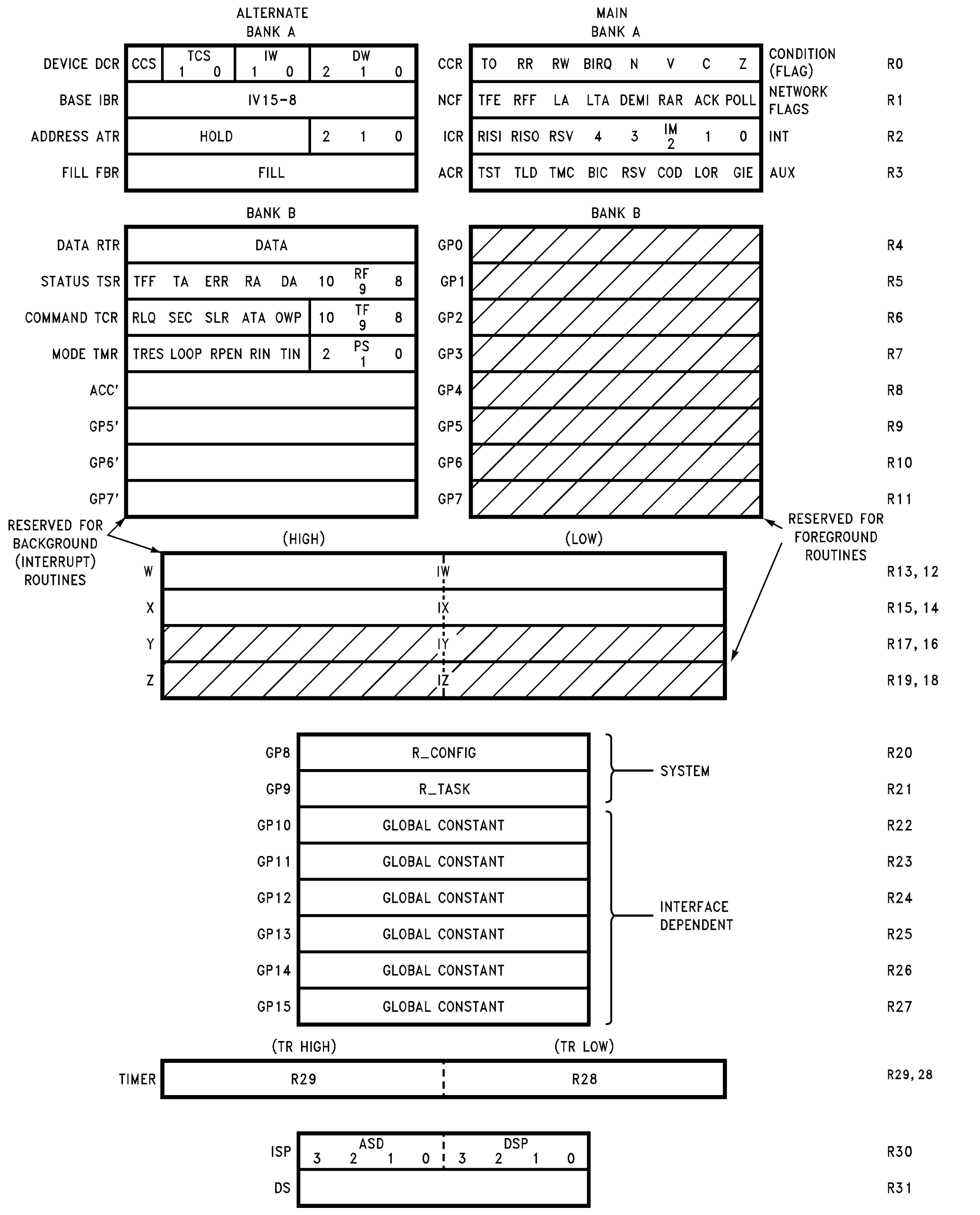
The Kernel consists of tasker, schedule__task, and desch__task routines. These three combine to allow tasks to be added or removed from the active task list, providing orderly execution of tasks. All tasks are scheduled by calling schedule__task with the task's identification byte in the selected accumulator. 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

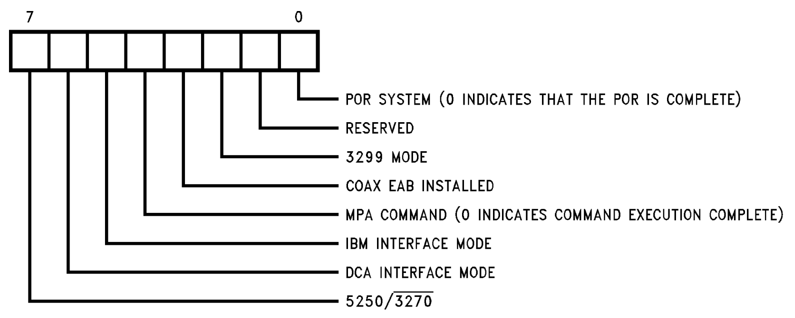
System Initialization

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.



© 1988, 1990 National Semiconductor Corporation

FIGURE 6-2. DP8344 MPA-II Register Map©


7
0
POR SYSTEM (0 INDICATES THAT THE POR IS COMPLETE)
RESERVED
3299 MODE
COAX EAB INSTALLED
MPA COMMAND (0 INDICATES COMMAND EXECUTION COMPLETE)
IBM INTERFACE MODE
DCA INTERFACE MODE
5250/$\overline{3270}$

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 intertask 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.

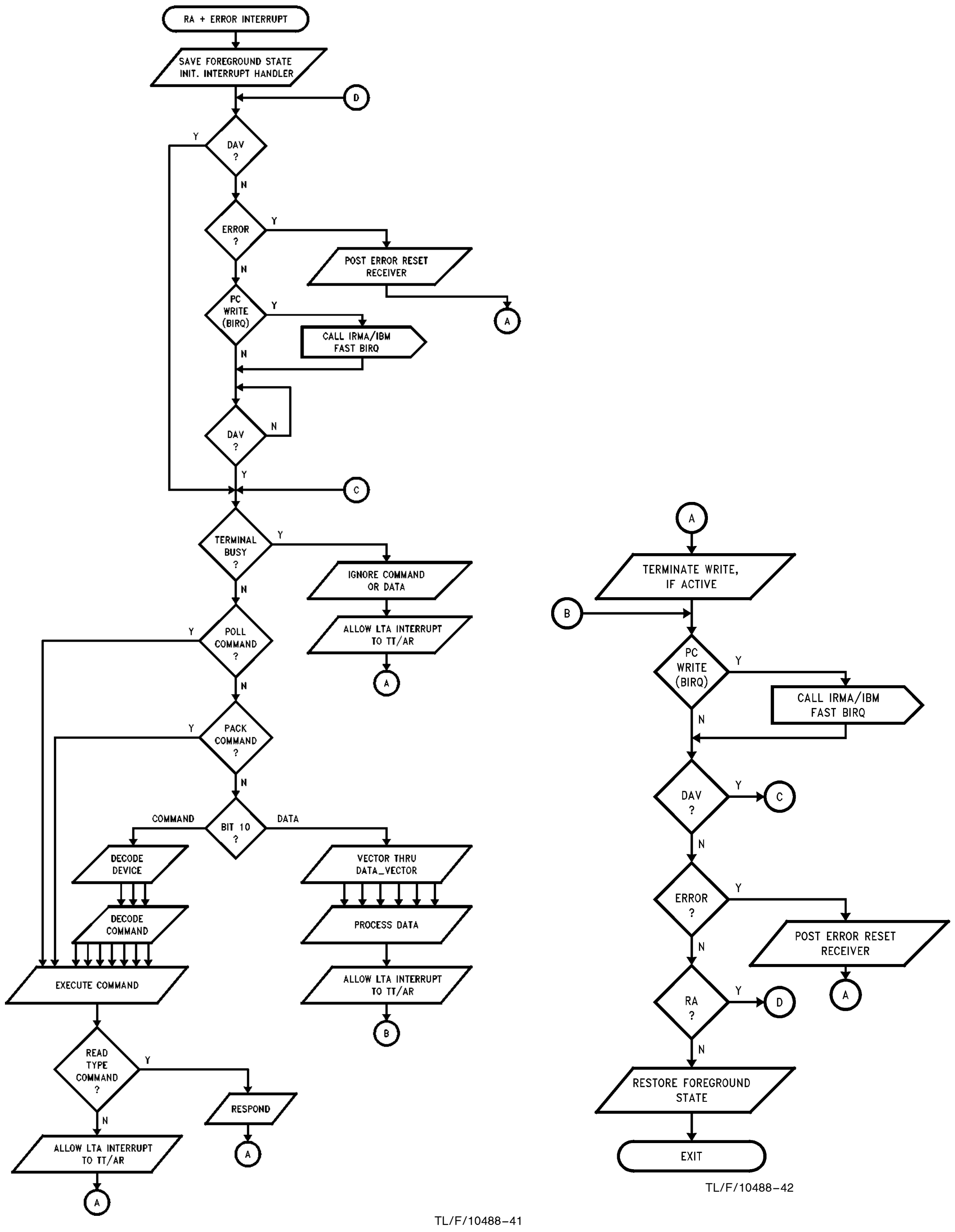
Coax Interrupt Handlers

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/10488–41

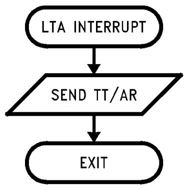
FIGURE 6-4. 3270 Coax Receiver Handler

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*.

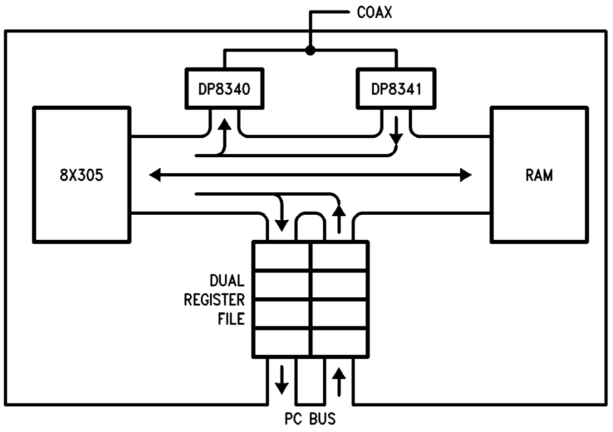


FIGURE 6-6. IRMA Hardware Block Diagram

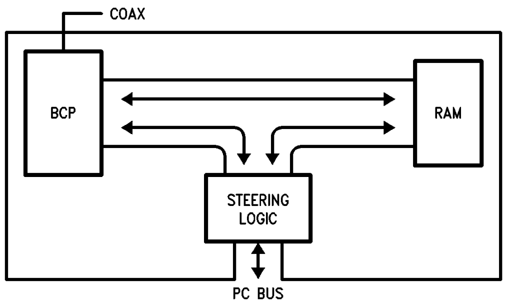


FIGURE 6-7. MPA-II Hardware Block Diagram

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*.

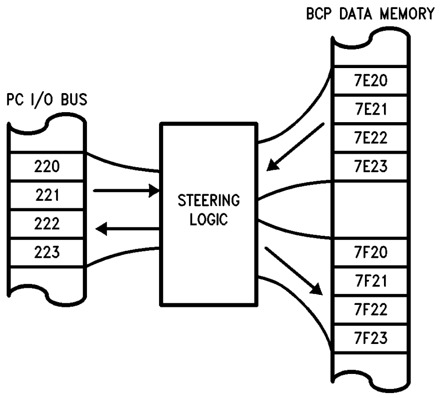


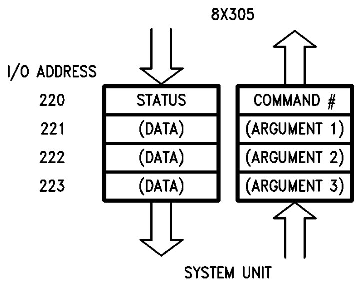
FIGURE 6-8. MPA-II Register Array Implementation

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.



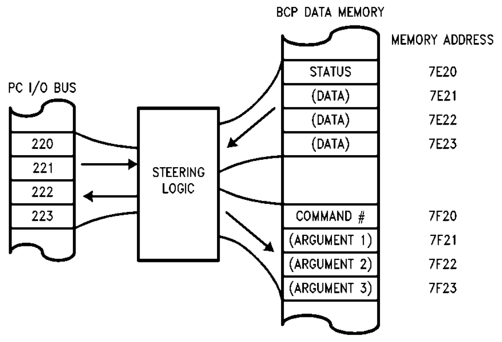
TL/F/10488–25

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

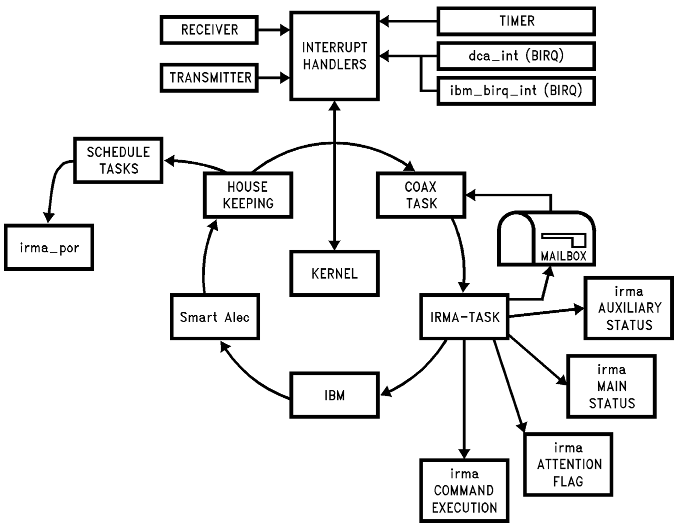
The Command Request and Attention Request flags are implemented using 74LS74's on the IRMA card, hence the setting and clearing by writing to 226h and 227h (this clocks or clears the associated flip-flop). This function is implemented on the MPA-II using an external PAL and the bi-directional interrupt 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

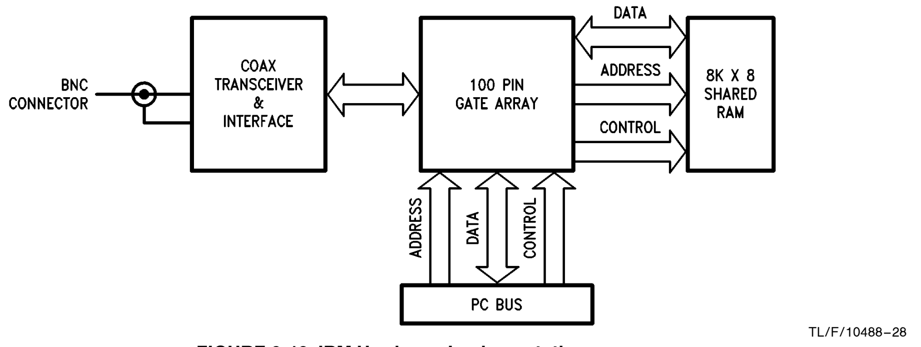


FIGURE 6-12. IBM Hardware Implementation

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.

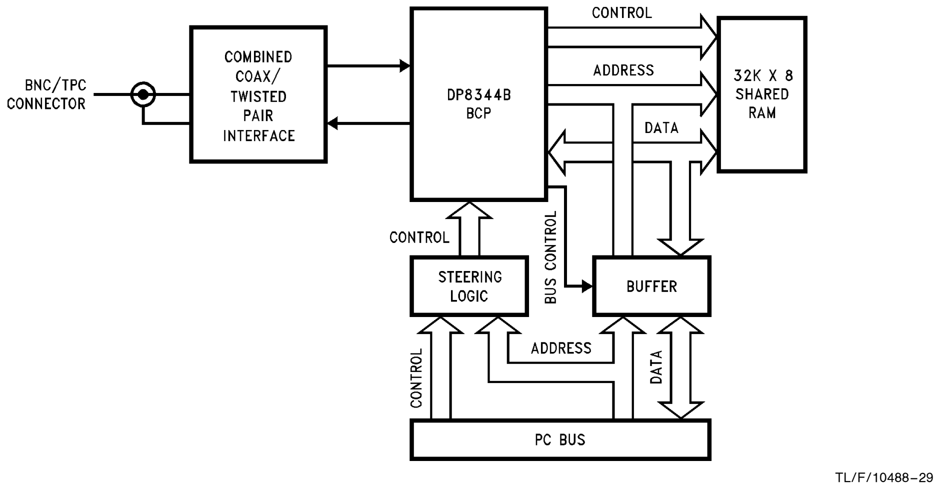


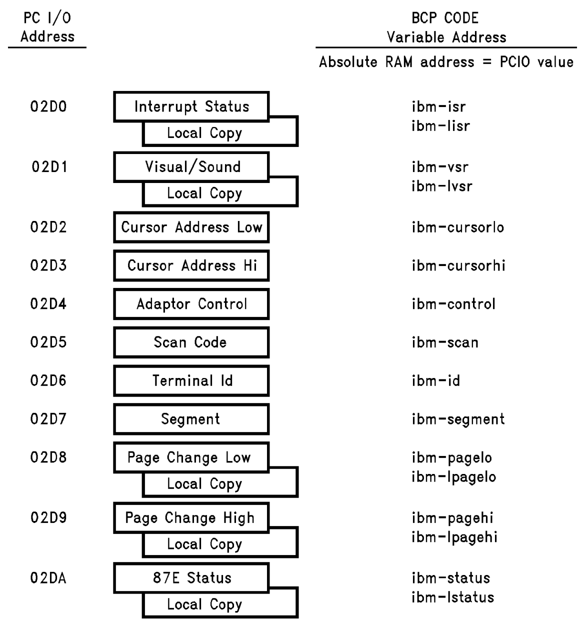
FIGURE 6-13. MPA-II Implementation of IBM Emulation Card

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.

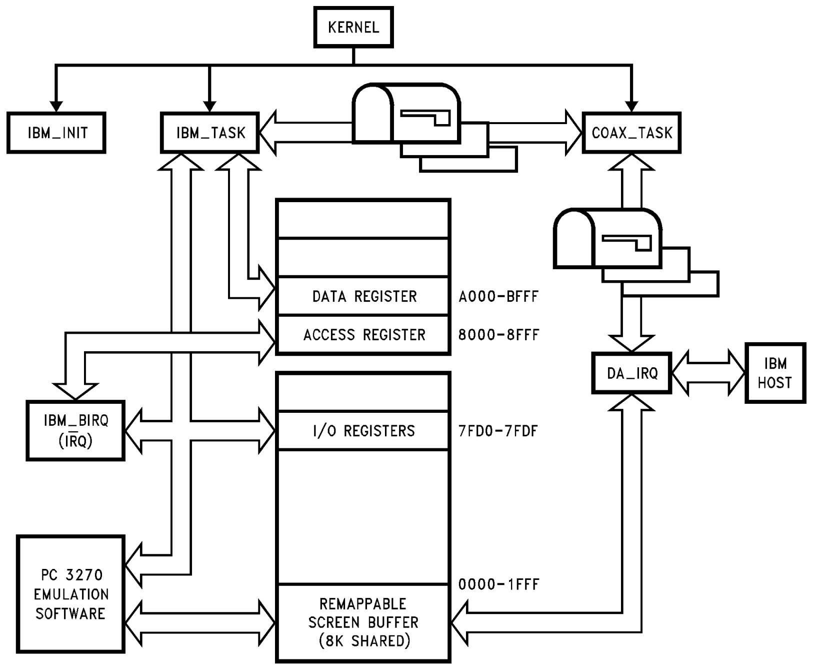
The BCP software, to handle the interface and coax routine, contains interrupt driven routines as well as foreground routines. A block diagram showing the code arrangement used to handle the IBM interface and coax session is shown in *Figure 6-15*. Four blocks run as tasks while the interrupt sources 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.

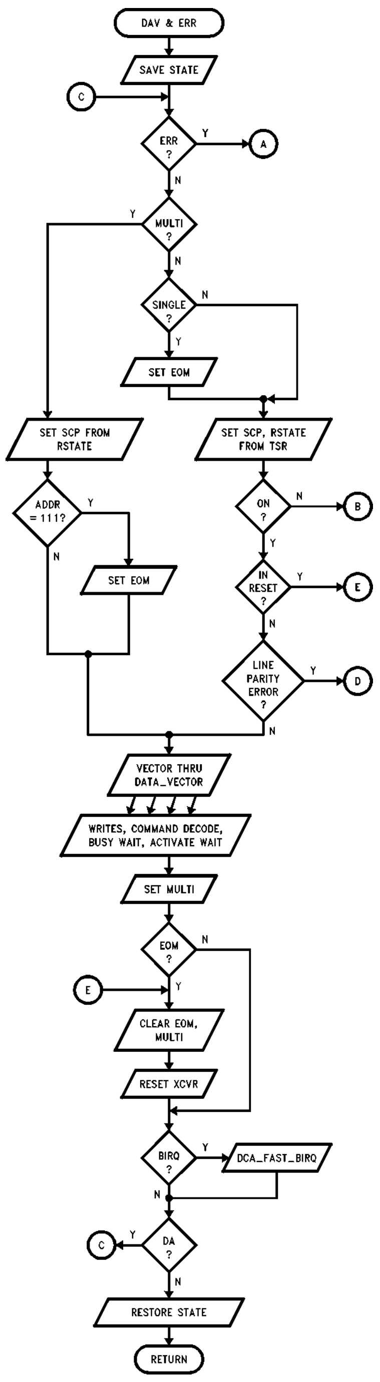
When all the sessions have been checked and all the scheduled sessions have been processed by the command processor once, tw__task returns control to the Kernel's tasker.

Twinax Interrupt Handlers

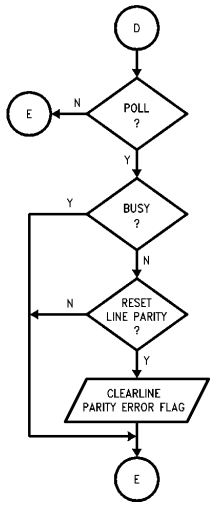
The twinax mode uses four interrupts: DAV, Data Available, for handling receiver data; TFE, Transmitter FIFO Empty, for all responses; TIMER for handling response window timing and as a real time clock for 5250 protocol requirements; and BIRQ for host interface accesses. All interrupts except BIRQ are 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

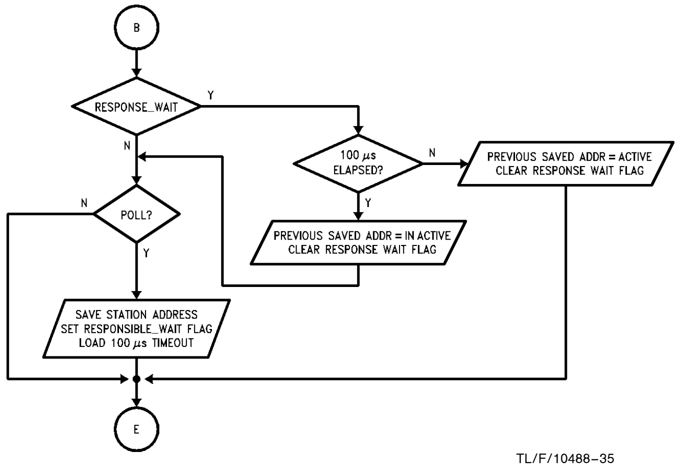
The DAV interrupt routine is responsible for decoding the commands sent by the controller, loading commands on the internal processing queue, stuffing data in to the regen buffer, "OFFLINE" address activity determination, maintaining protocol related real time status bits, and supporting all seven station addresses if necessary. A flow diagram of the DAV interrupt routine is shown in *Figure 6-16*.



TL/F/10488-44

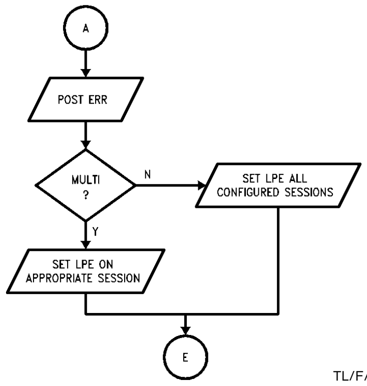


TL/F/10488-34



B
RESPONSE_WAIT
Y
N
100 μs ELAPSED?
N
PREVIOUS SAVED ADDR = ACTIVE CLEAR RESPONSE WAIT FLAG
N
POLL?
Y
PREVIOUS SAVED ADDR = IN ACTIVE CLEAR RESPONSE WAIT FLAG
Y
SAVE STATION ADDRESS SET RESPONSIBLE_WAIT FLAG LOAD 100 μs TIMEOUT
E

TL/F/10488-35



A
POST ERR
MULTI ?
N
SET LPE ALL CONFIGURED SESSIONS
Y
SET LPE ON APPROPRIATE SESSION
E

TL/F/10488-36

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 interrupt 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 detecting 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.

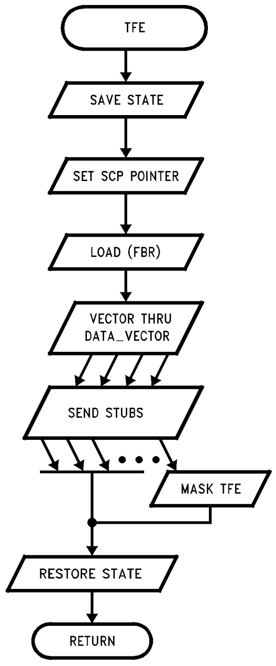
POLLs to nonconfigured station addresses do not result in a response but are used in monitoring activity on station addresses for Smart Alec address bidding purposes. When a frame to an OFFLINE address (i.e., not configured by Smart Alec) is received, the OFFLINE activity monitoring routine is responsible 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 ON-LINE 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 addressed 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 terminal. 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.

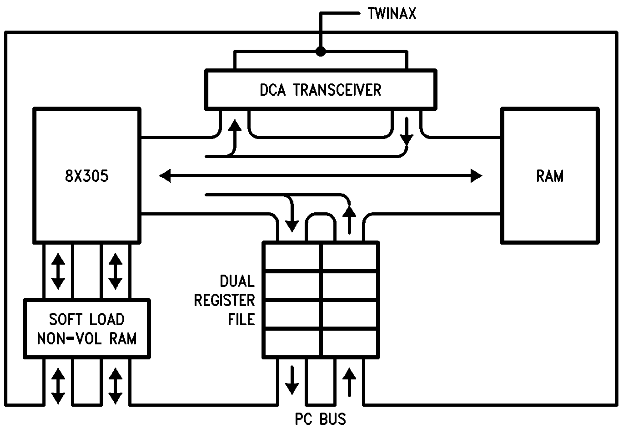
The card contains 16K of data memory for the screen buffers and temporary storage. Each session can require up to 2K of data memory for its associated screen buffer, accounting for a total of 14K. The remaining memory space is used by the 8X305 for local storage.

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

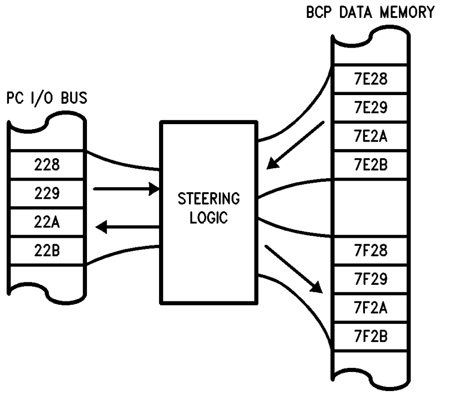
FIGURE 6-18. Smart Alec Hardware Block Diagram

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

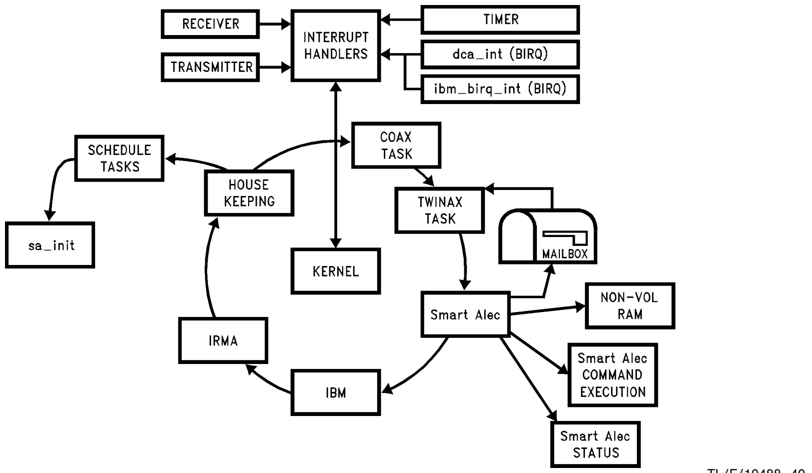


FIGURE 6-20. MPA-II Software Block Diagram in Smart Alec Mode

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).

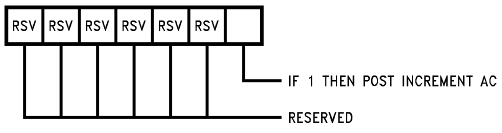
TABLE 6-9. MPA-II Command Set

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
```

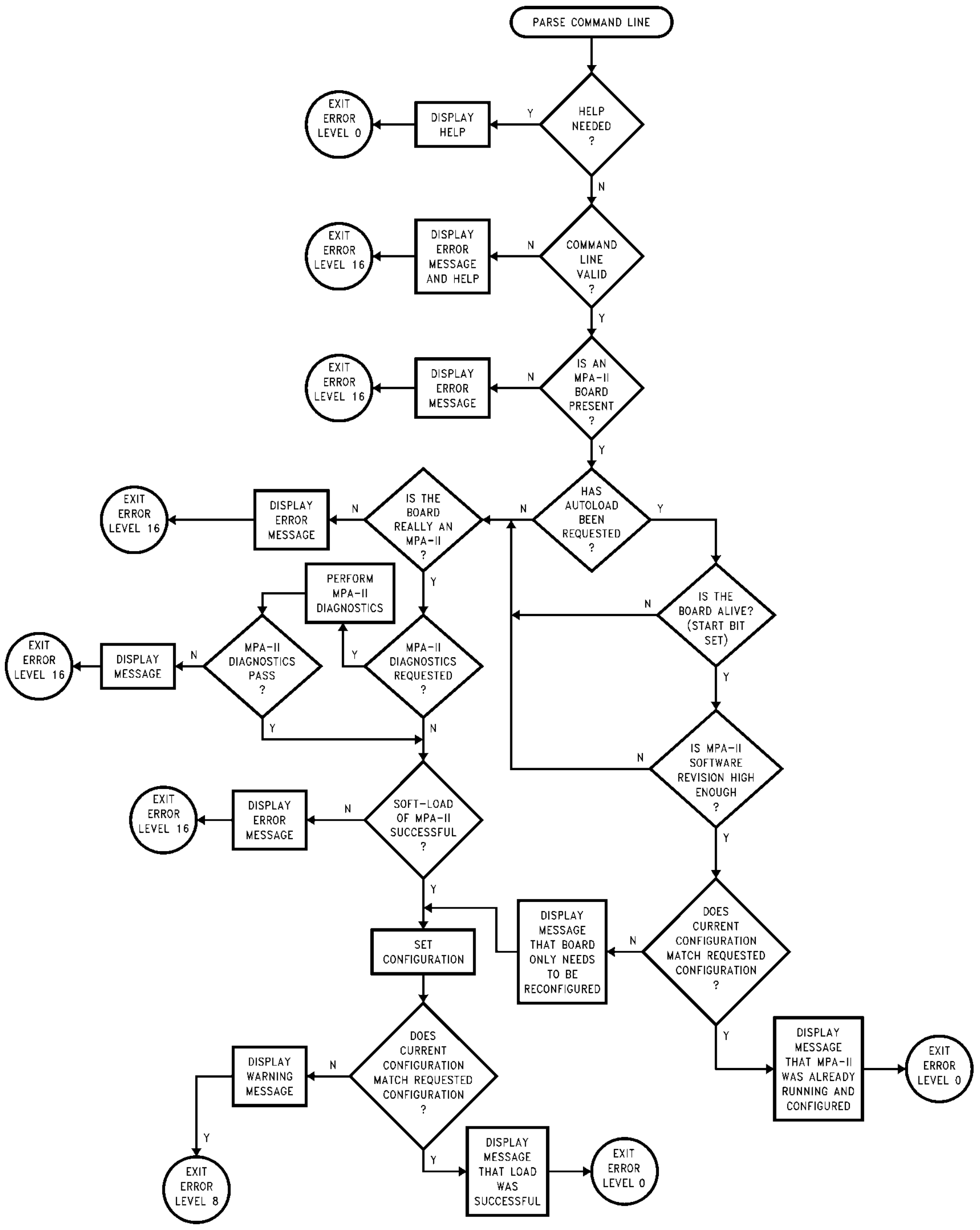
The Loader provides the following return values which are useful when using the Loader in a batch file:

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-intrusive 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



PARSE COMMAND LINE
HELP NEEDED ?
Y
N
DISPLAY HELP
EXIT ERROR LEVEL 0
COMMAND LINE VALID ?
DISPLAY ERROR MESSAGE AND HELP
EXIT ERROR LEVEL 16
IS AN MPA-II BOARD PRESENT ?
DISPLAY ERROR MESSAGE
EXIT ERROR LEVEL 16
HAS AUTOLOAD BEEN REQUESTED ?
IS THE BOARD REALLY AN MPA-II ?
DISPLAY ERROR MESSAGE
EXIT ERROR LEVEL 16
PERFORM MPA-II DIAGNOSTICS
MPA-II DIAGNOSTICS REQUESTED ?
MPA-II DIAGNOSTICS PASS ?
DISPLAY MESSAGE
EXIT ERROR LEVEL 16
IS THE BOARD ALIVE? (START BIT SET)
IS MPA-II SOFTWARE REVISION HIGH ENOUGH ?
SOFT-LOAD OF MPA-II SUCCESSFUL ?
DISPLAY ERROR MESSAGE
EXIT ERROR LEVEL 16
DOES CURRENT CONFIGURATION MATCH REQUESTED CONFIGURATION ?
DISPLAY MESSAGE THAT BOARD ONLY NEEDS TO BE RECONFIGURED
SET CONFIGURATION
DISPLAY MESSAGE THAT MPA-II WAS ALREADY RUNNING AND CONFIGURED
EXIT ERROR LEVEL 0
DOES CURRENT CONFIGURATION MATCH REQUESTED CONFIGURATION ?
DISPLAY WARNING MESSAGE
EXIT ERROR LEVEL 8
DISPLAY MESSAGE THAT LOAD WAS SUCCESSFUL
EXIT ERROR LEVEL 0

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[=][I_addr]—** Soft-loads the file into instruction memory beginning at the hex address, I_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, I_addr, defaults to the hex address 0000.

−**R[=][addr][,r_addr]—** Soft-loads the file into instruction memory beginning at the hex address I_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

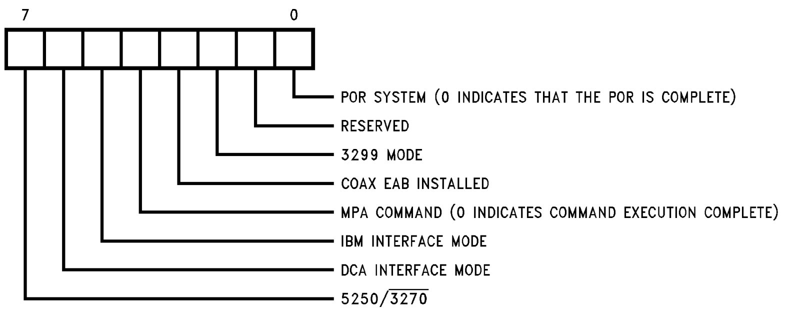
The Loader configures the MPA-II terminal emulation interface mode as requested by the user through the Configuration Options. Configuring the MPA-II interface mode enables the MPA-II to emulate the standard PC terminal emulation interfaces including DCA's IRMA and Smart Alec interface modes; and IBM's 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>
```



7
0
POR SYSTEM (0 INDICATES THAT THE POR IS COMPLETE)
RESERVED
3299 MODE
COAX EAB INSTALLED
MPA COMMAND (0 INDICATES COMMAND EXECUTION COMPLETE)
IBM INTERFACE MODE
DCA INTERFACE MODE
5250/$\overline{3270}$

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>
```

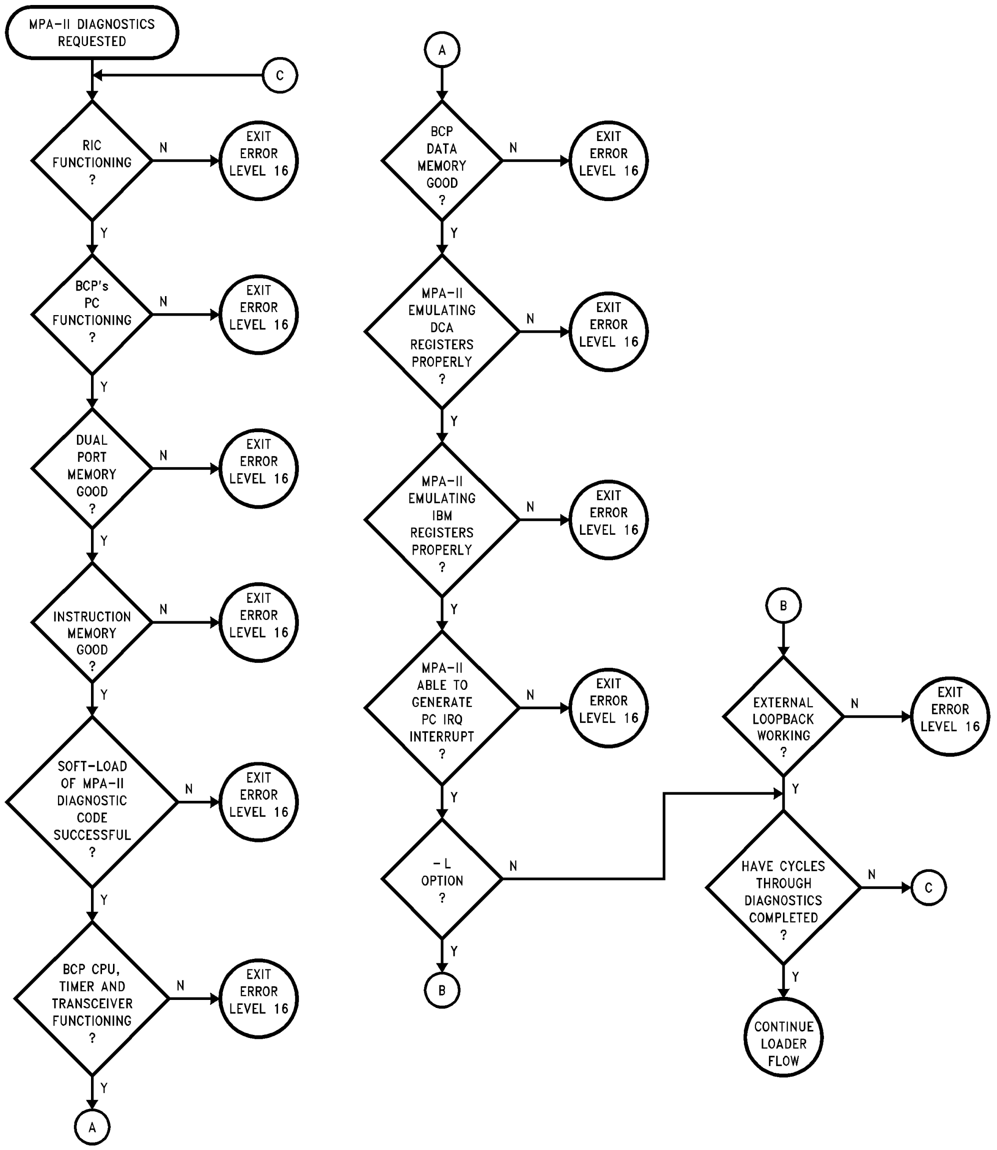
Options:

-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.



MPA-II DIAGNOSTICS REQUESTED
C
RIC FUNCTIONING ?
N
EXIT ERROR LEVEL 16
Y
BCP's PC FUNCTIONING ?
DUAL PORT MEMORY GOOD ?
INSTRUCTION MEMORY GOOD ?
SOFT-LOAD OF MPA-II DIAGNOSTIC CODE SUCCESSFUL ?
BCP CPU, TIMER AND TRANSCEIVER FUNCTIONING ?
A
BCP DATA MEMORY GOOD ?
MPA-II EMULATING DCA REGISTERS PROPERLY ?
MPA-II EMULATING IBM REGISTERS PROPERLY ?
MPA-II ABLE TO GENERATE PC IRQ INTERRUPT ?
-L OPTION ?
B
EXTERNAL LOOPBACK WORKING ?
HAVE CYCLES THROUGH DIAGNOSTICS COMPLETED ?
CONTINUE LOADER FLOW

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    = ^b11111111;
 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 seleting 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

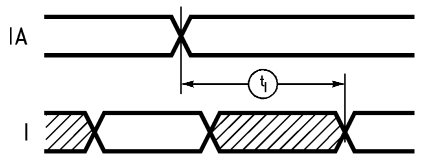
As with most other aspects of a design, choosing memory is a cost vs. performance trade off. Maximum performance is achieved running no wait-states with fast, expensive memory. Slower, less expensive memory can be used, but wait-states must be added, slowing down the BCP. Therefore one needs to choose 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

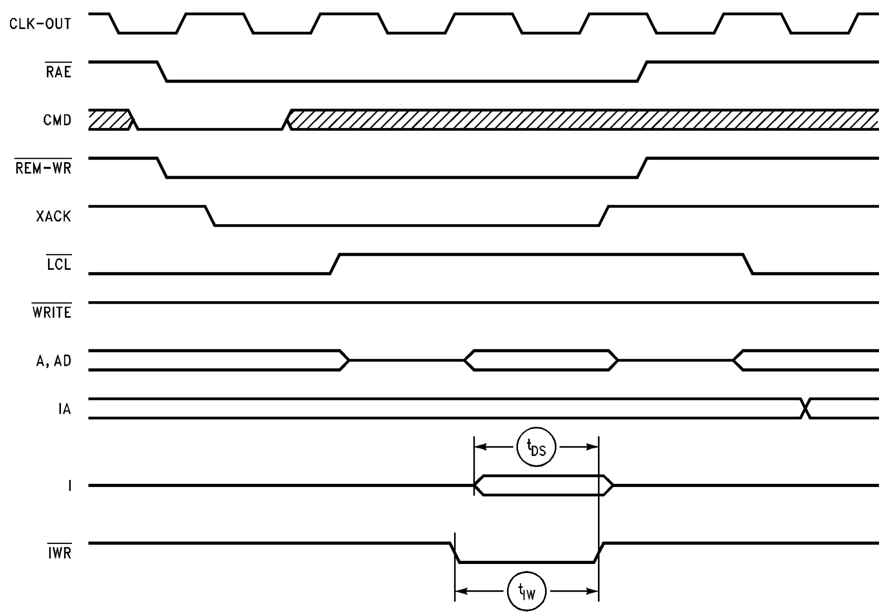
However, instruction read time is not the only timing consideration when choosing instruction RAM. If the BCP is used in an application which requires full speed softloading of instruction RAM, there are two other timing relationships which require evaluation. These are data setup time and write pulse 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 ($\overline{\text{IWR}}$) 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:

$$t_D = (2.5 + \text{MAX}\,(n_{DW}, n_{IW} - 1))T - 40$$

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

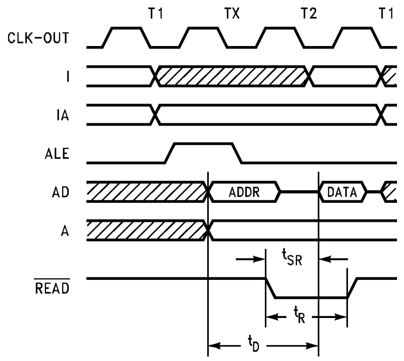
Parameter	Fujitsu RAM (Minimum) (55 ns)	DP8344B BCP (Minimum)					
		Read			Write		
		Full Speed	Half Speed	POR	Full Speed	Half Speed	POR
Access Time (t_I)	55	60.5	140	458			
Write Pulse Width (t_W)	40				43	96	414
Data Setup (t_{DS})	25				35	88	406

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. $\overline{\text{READ}}$ and $\overline{\text{WRITE}}$ 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 ($\overline{\text{IOR}}$, $\overline{\text{IOW}}$, $\overline{\text{MEMR}}$ or $\overline{\text{MEMW}}$) 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, $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ pass through a 74ALS244 before reaching the expansion bus.

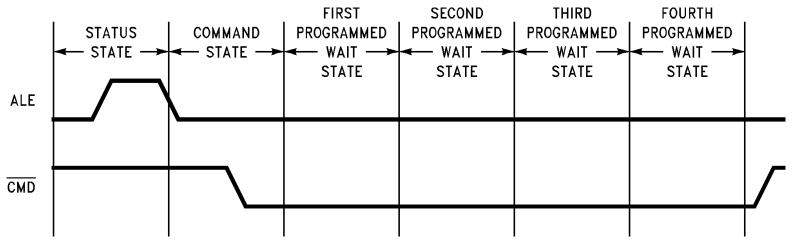
TABLE B-8. Data Memory Read and Write Parameters

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 $n_{DW} = 0$.


STATUS STATE
COMMAND STATE
FIRST PROGRAMMED WAIT STATE
SECOND PROGRAMMED WAIT STATE
THIRD PROGRAMMED WAIT STATE
FOURTH PROGRAMMED WAIT STATE
ALE
$\overline{\text{CMD}}$

TL/F/10488–63

FIGURE B-5. Relationship of ALE, $\overline{\text{CMD}}$, 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.

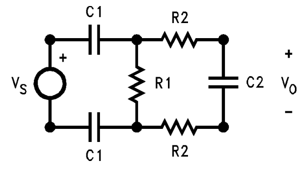
As a final note, the reader should be aware that most faster PC-AT's still run their expansion buses at 8 MHz to remain compatible. This means that the timing on these expansion buses should remain the same as those on any other PC-AT no matter how fast the CPU runs. Thus, the MPA-II will run in all 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/10488–64

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:

$$|f| = \frac{\frac{1}{2R_2C_2} \pm \sqrt{\frac{1}{4R_2^2\,C_2^2} - 4\left(\frac{1}{R_1R_2C_1C_2}\right)}}{4\pi}$$

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/TTl 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

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

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998