SLAVE EMULATOR HARDWARE REFERENCE MANUAL

2302-5003-01

KONTRON ELECTRONICS

ADVANCED ELECTRONIC INSTRUMENTATION

SLAVE EMULATOR HARDWARE REFERENCE MANUAL

2302-5003-01

October 1982

Kontron Electronics, Inc.
5730 Buckingham Parkway
Culver City, CA 90230

REVISION HISTORY

Title	Number	Date	Notes
Slave Emulator Hardware Reference Manual	2302-5003	10/82	2nd Edition

RELATED PUBLICATIONS

ADS Installation And Maintenance	2300-5003
Floppy Disk Drives	2300-5007
EPROM Programmer	2300-5035
Slave Logic Analyzer	2302-5012
Slave Emulator Reference Manual	2302-5000

CORRECTION

In our current system configuration, the Kontron Logic Analyzer (KLA) unit or the Kontron Development System (KDS) unit replaces the ADS console as the host unit. The Kontron Slave Emulator (KSE) replaces the Slave Emulator.

Figures 1-1 and 1-3 of this document still show the older ADS console, and the Slave Emulator chassis. These illustrations will be updated in the next revision of this manual. Until then, please substitute the new equipment designations for the old names. The informational content of the illustrations is otherwise accurate.

PREFACE

The scope of this manual is limited to the basic Slave Emulator. Microprocessor specific systems are defined in the Supplements to this manual.

Chapter 1, Introduction, describes the Emulator Control Unit hardware and specifications.

Chapter 2, Emulator Setup And Checkout, describes preliminary checks, system interconnections, and limited Emulator Control Unit system test.

Chapter 3, Maintenance, characterizes preventive maintenance, troubleshooting information, and disassembly instructions. Diagrams and supporting data are included.

Chapter 4, Theory Of Operation, narrates the function of Emulator boards such as the interface processor, breakpoint, and memory boards.

Please note that a **Documentation Reply Card** is inserted at the back of this manual. When you complete and return it, you help us produce better documentation for you.

A **User Registration Card** is included in the set of manuals you receive with your Kontron system. When you complete and return the User Registration Card, you ensure that you will receive all updates and new information for your configuration.

WARNING

This equipment generates, uses and can radiate radio frequency energy. If not installed and used in accordance with this instruction manual, it may cause interference to radio communications. As temporarily permitted by regulation, it has not been tested for compliance with the limits for Class A computing devices pursuant to Subpart J of Part 15 of the FCC rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference, in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER 1

INTRODUCTION

1.1 PURPOSE AND SCOPE

The purpose of this manual is to acquaint the user with Kontron's Slave Emulator. Information is given on installation, checkout, maintenance and theory of operation.

1.2 OVERVIEW

1.2.1 SLAVE VERSUS TRADITIONAL IN-CIRCUIT EMULATION

Traditional in-circuit emulation is less than ideal. Often, the processor under test is restrained from running at full speed, or some of its memory is tied up by the emulation hardware, or the system under test must be halted to conduct debugging operations. These limitations are removed when using slave in-circuit emulation, because the multiprocessor, and the multiple bus organization of the Kontron Slave Emulator allows full-speed emulation of 8-bit, or 16-bit microprocessors.

Note that from this point on, throughout this manual, the Kontron Slave Emulator is referred to as the Emulator.

1.2.2 EMULATION OBJECTIVES

The two main objectives of the emulation process are:

a. Substitution of an Emulation Processor for the target system's processor.

b. Enabling the user to monitor and control the target system. This is accomplished by surrounding the Emulation Processor with controlling circuitry.

1.2.3 TARGET SYSTEM SUBSTITUTION

The Emulator substitutes an identical processor for the target system's processor. To eliminate critical physical distances, this Emulation Processor is housed in a small portable chassis called a Probe, which is connected by a short cable to the target system's vacant microprocessor socket. The signals into and out of the Emulation Processor can be easily relayed from the Probe to the target system's socket.

1.2.4 THE EMULATION SYSTEM

The Development System consists of a 2300 Console with keyboard and CRT display, a dual floppy disk system, supporting software/firmware, and other parts.

One of the advantages of Slave Emulation is the ability to emulate up to four microprocessors simultaneously through one Development System. This provides a low-cost and efficient means to emulate a multiprocessor environment. Up to four processors can be monitored, and controlled from one station. All Emulators can be synchronously halted from a single event in any of the Emulators or a combination of several events in one or more of the Emulators. This is a powerful debugging tool in a multiprocessor environment. See **Figure 1-1, Multi-Emulation Station.**

1.3 HARDWARE DESCRIPTION

1.3.1 EMULATOR CABINET

Within the Emulator cabinet is an eight slot card cage which must have a minimum of three boards installed: an Interface Processor Board, a Breakpoint Board, and a Personality Board. The maximum number of boards is eight, adding an optional Logic Analyzer, and up to four Simulation Memory boards. When changing to a different Emulator configuration, access to the boards is through the rear panel.

1.3.2 EMULATOR PROBE

The probe is a buffer device which brings the Emulation Processor as close as possible to the target system to prevent loss of signal integrity. It is microprocessor dependent, and serves as the interface between the microprocessor being emulated, and the Emulator Personality Board. A more detailed explanation of the Emulator Probe is given in Chapter 4, and also in each of the microprocessor specific supplements to this manual.

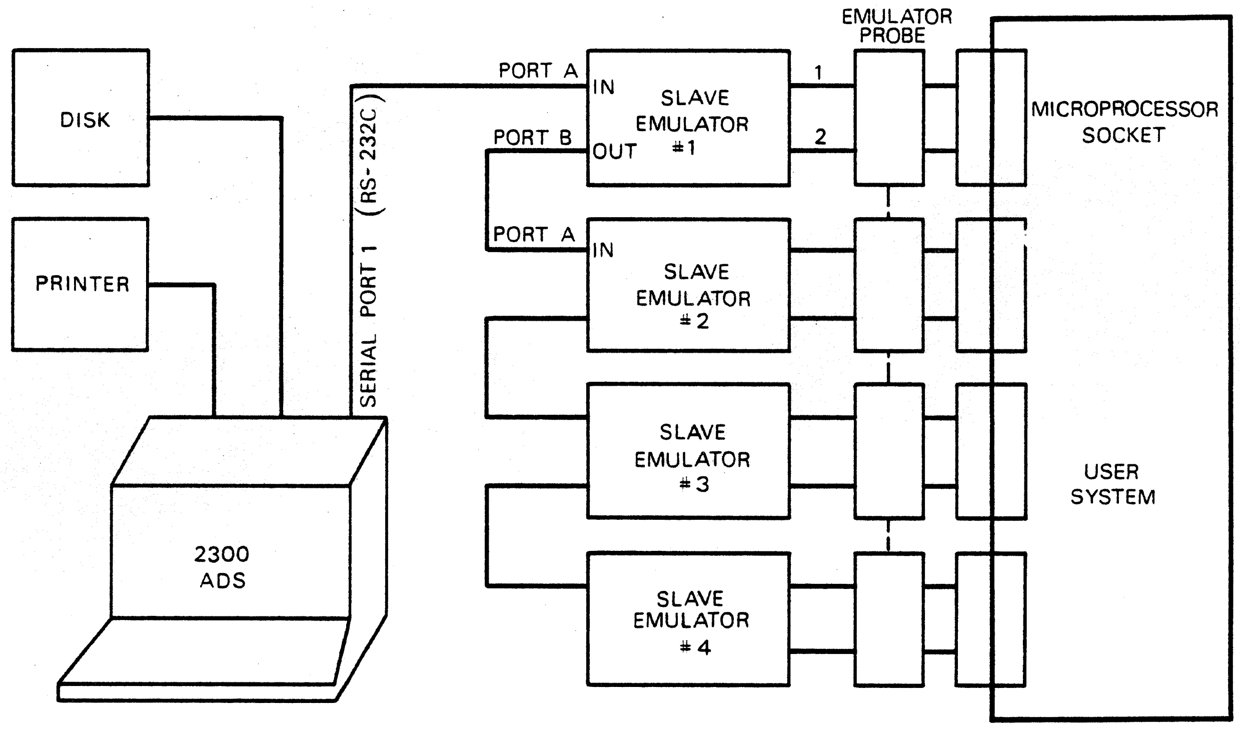


Figure 1-1. Multi-Emulation Station

1.3.3 EMULATOR BOARDS

The Emulator holds a maximum of eight boards. Each board, with its component side up, can be inserted into any one of the eight interchangeable slots in the card cage. The backplane of the card cage is a parallel bus system, designed to bring information from the microprocessor and the target system to the Emulator.
The basic boards and minimum configuration shipped with the Emulator include:

1. **Interface Processor Board** The IP responds to commands from the console, controls the Emulation Processor and returns data to the user.

2. **Breakpoint Board** The Breakpoint board has four hardware breakpoints that can break on parameters such as address, data, memory, I/O, execution/fetch, and a pass counter.

3. **Personality Board (Microprocessor specific)** This board is the interface between the emulated processor's bus and the Slave Emulator. In addition, it directly controls all emulation processes.

Optional boards are Simulation Memory, Dynamic Memory when supported, and a Logic Analyzer. Simulation Memory is available from 16K to 512K bytes of memory. The functions of these boards, and the entire system are described in **Chapter 4.**

The addition of the Simulation Memory Board and the Logic Analyzer board to the Slave Emulator configuration allows debugging without involving the target system memory. Simulation Memory board can simulate ROM, without having to program ROMs.

Logic Analyzer simplifies troubleshooting by providing detailed waveform display and cycle data display. This allows the user to view bus data during program execution.

1.3.4 EXTERNAL PROBE

The External Probe, see **Figure 1-2, External Probe,** is a multifunction device that works together with the Breakpoint, and optionally with the Logic Analyzer. It brings in four external signals, called external lines in the breakpoint parameters, from any TTL compatible source. These external lines can be used as breakpoint qualifiers or they can be traced by the Logic Analyzer.

Breakpoint triggers for breakpoints 0, 1, and 2 are brought out to the External Probe for use by another External Probe from another Emulator. These triggers are used for synchronous breakpoints in a multi-processor environment or as a complex trigger for an oscilloscope or other test instrument. **Chapter 4 Section 4.9** details the functions of the External Probe.

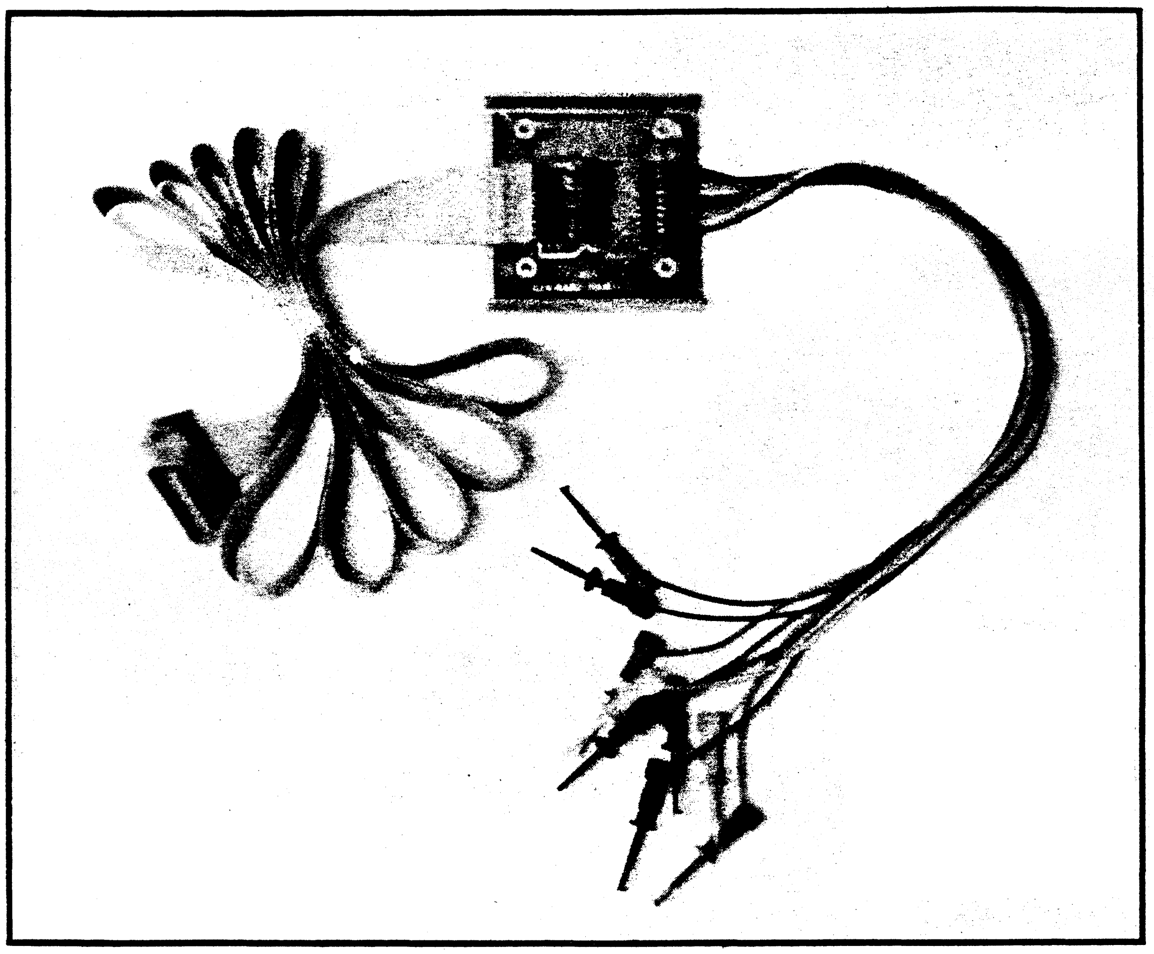


Figure 1-2. External Probe

1.3.5 INTERNAL EMULATOR CABLING

The internal Emulator cabling system is shown in **Figure 1-3, Internal Cabling.** This is a view of the Emulator with the top enclosure cover and rear panel removed. Five boards are shown with six sets of separate cables. These boards are, from top to bottom, the Breakpoint, the Logic Analyzer, the Personality, the Interface Processor, and the Memory.

Table 1-1 identifies the internal cables used in the Emulator.

Table 1-1. Internal Cables

Cable	Part Number	Usage
A	2302-0225	Breakpoint board to Logic Analyzer board to LOGIC ANALYZER connector.
B	2302-0221	Personality board to PROBE I connector.
C	2302-0222	Personality board to PROBE II connector.
D	2302-0226	Interface Processor to PORT B connector.
E	2302-0026	Interface Processor board to PORT A connector.
F	2302-0234	Logic Analyzer board to Personality board.

Figure 1-3. Internal Cabling

1.4 SPECIFICATIONS

1.4.1 EMULATOR CABINET

Outline Dimensions:

Width:	16.1" (40.64cm)
Height:	7.6" (19.3cm)
Length:	18.2" (46.23cm)

Weight: 40 lbs (18kg)

Cooling: Forced air

Operating Ambient Temperature: 0 to 40 C (32 to 104 F)

Storage Ambient Temperature: -50 to 125 C (-58 to 257 F)

Power system:

Input AC line voltage:

a. 90-130 VAC
b. 200-260 VAC

Input line voltages are a factory option.

Input current:

Less than 5 amps @ 115vac
Less than 2.5 amps @ 230vac

Single phase 3-wire system

47 - 440 Hz

Note that the Emulator's power system is preset, at the factory, to your specifications.

Power Dissipation: 350 watts maximum

Circuit Protection:

110 vac, 5 amps circuit breaker (switch on front panel)
220 vac, 2.5 amps circuit breaker (switch on front panel)
3 amps fuse (internal power supply)

1.4.2 EXTERNAL PROBE

Outline Dimensions:

Width:	2.75" (6.985cm)
Height:	.625" (1.5875cm)
Length:	2.75" (6.985cm)

Weight: 7 ounces (198gm)

Cooling: Convection air

Operating Ambient Temperature: 0 to 40 C (32 to 104 F)

Storage Ambient Temperature: -50 to 125 C (-58 to 257 F)

All inputs and outputs are TTL compatible.

1.4.3 PERFORMANCE PROLOGUE

The most important link in any Emulator is the interface between the user's target system and the Emulator. Systems are designed with the assumption that a microprocessor will reside on board. The introduction of a length of cable and buffer circuitry (the Emulator) between the microprocessor and its socket may create problems. Wirewrapped prototypes, if not carefully laid out, can be very noisy. The introduction of a length of cable may make the system inoperable. It is important therefore that prototypes have a good power and ground grid. Wirewrapped boards should have the power and ground system soldered down. Systems operating within the guidelines of standard TTL logic will not encounter problems with an Emulator. Problems may be encountered with noisy systems.

CAUTION: The Emulator may not be used with a target which is floating with respect to ground. This float is measured as the voltage between target ground and chassis ground. The Emulator is referenced to standard earth ground. The Emulator power supply returns are attached to the chassis ground. Thus the microprocessor plug ground pins are referenced to chassis ground.

IF THE TARGET IS FLOATING WITH RESPECT TO EARTH GROUND, SEVERE DAMAGE TO THE EMULATOR PROBE AND TARGET MAY OCCUR.

Signal Pulse Delays

Each signal at the Emulator Plug is buffered in both directions in the Emulator Probe. A signal such as A0 in **Figure 1-4, Buffer Circuit,** delivered to the Emulator Plug through the buffer tends to arrive typically 9 to 10 nanoseconds later. There is a similar delay in the signal shown as D0 in **Figure 1-4,**, going to the microprocessor from the target system. For request/response operations such as memory reads, these delays are additive,and can result in a total delay of 9 to 18 nanoseconds. Detailed timing specifications are given in the supplement for each subsystem.

Microprocessor signal specifications are commonly referenced to clock edges. The clock is also delayed in passing from the target system to the microprocessor. To compensate for these delays, some Emulators, such as the 8086, have an adjustable clock delay generator designed into the Emulator Probe.

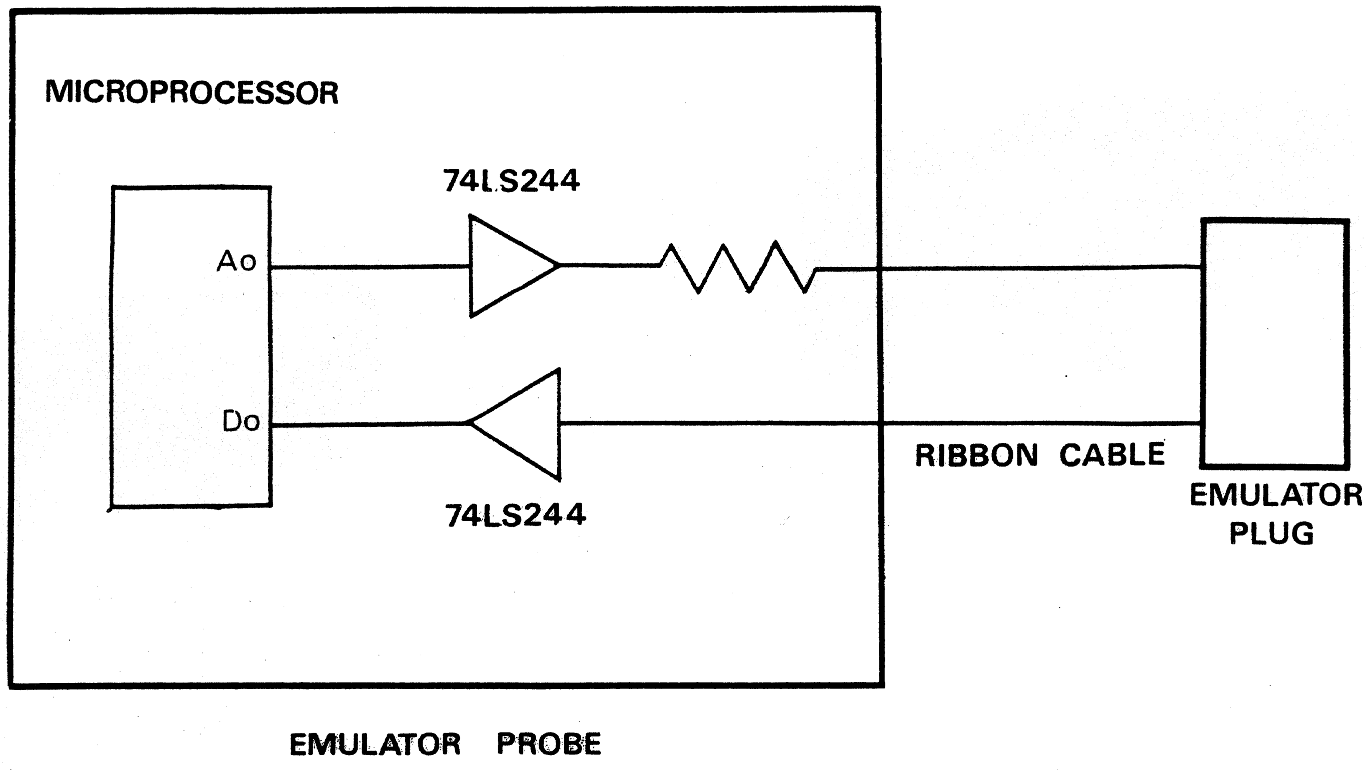


Figure 1-4. Buffer Circuit

CHAPTER 2

INSTALLATION AND CHECKOUT

2.1 INSTALLATION DOCUMENTS

Included in the shipment are four important installation documents:

Cover Letter
Installation/Quality Report
Configuration Checklist
User Registration Card

The top half of the Installation/Quality Report is filled in by Kontron before the equipment leaves the factory. The bottom half of this form is filled in by Kontron installation personnel at the customer site. Additional comments can be made by the customer in the spaces provided at the bottom of the form, and then signed off.

The configuration checklist describes all the equipment included with each shipment from Kontron. This form is signed and dated by factory representatives. A complete list of the items sent in the shipment is provided for the customer to check off when the system is received. In case of any discrepancy, contact your Kontron representative.

2.2 INSPECTING AND SETTING UP THE SYSTEM

After removing the Emulator from the carton, save all packaging materials in case the system requires shipping at a later date. An inventory of the shipment received should be taken immediately after all parts are removed from their shipping containers. Inspect the equipment for evidence of shipping damage to housing, panel controls, or cables.

Notify your Kontron representative of any problems found during the receiving inspection.

2.3 CONFIGURATION REQUIREMENTS

If the Emulator being installed will be used in an already operational Development System that was previously set up for the Emulator, or has been just received from the factory, the configuration of the boards need not checked. If the Development System has not been used for emulation previously, it should be setup according to the following requirements.

2.3.1 ADVANCED DEVELOPMENT SYSTEM CONFIGURATION

For Emulator operation, the system configuration must include, as a minimum, 64K bytes of memory, a Z80 board, and an MPIO board.

2.3.2 EMULATOR CONFIGURATION REQUIREMENTS

The Emulator boards must be configured for the microprocessor being emulated. These are set correctly at the factory for the Emulator Subsystem. However, they will require reconfiguration if the the Personality board in the Emulator is changed. Correct configuration for each microprocessor is given in the specific Hardware Supplement for that processor.

2.4 SYSTEM INTERCONNECTIONS

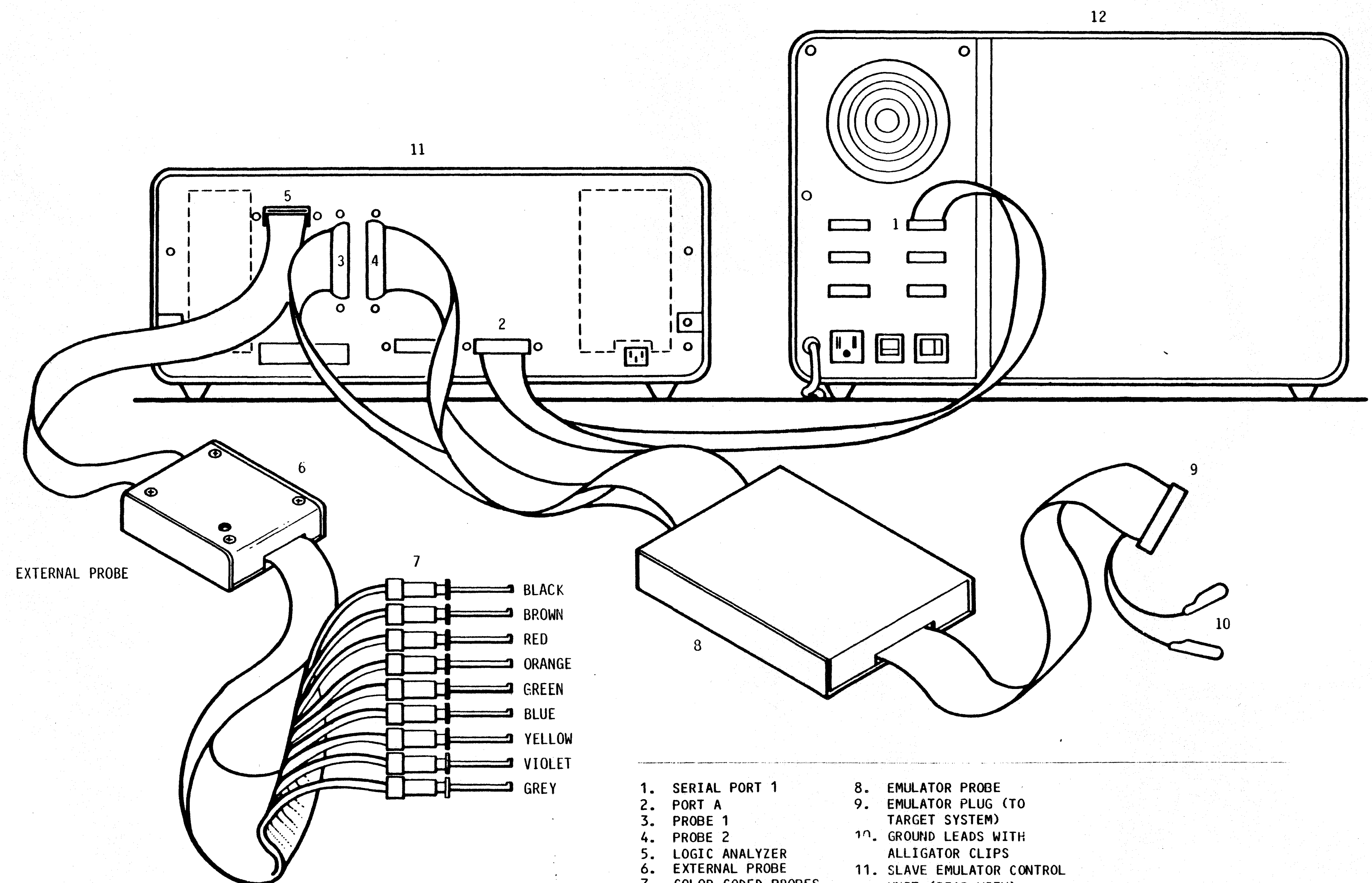
To install a single Emulator refer to **Figure 2-1, System Interconnections**, and connect the subsystems together as follows:

a. RS-232 Cable. Connect one end to Port A (input) on the Emulator; the other to Serial 1 on the Developement System. Screw the connectors in place.

b. Probe. Connect the two 50-pin connectors (the 72-inch cables) from the Emulator Probe box to Probe 1 and Probe 2 keyed connectors, on the Emulator.

Note that if more than one Emulator is being installed on the same console, the Emulators must be daisy chained as follows:

Output Port A of Emulator 1 to input Port B of Emulator 2, and Output Port A of Emulator 2 into Port B of Emulator 3 etc., up to four Emulators.



1. SERIAL PORT 1
2. PORT A
3. PROBE 1
4. PROBE 2
5. LOGIC ANALYZER
6. EXTERNAL PROBE
7. COLOR-CODED PROBES & LINES
8. EMULATOR PROBE
9. EMULATOR PLUG (TO TARGET SYSTEM)
10. GROUND LEADS WITH ALLIGATOR CLIPS
11. SLAVE EMULATOR CONTROL UNIT (REAR VIEW)
12. CONSOLE (REAR VIEW)

Figure 2-1. System Interconnections

2.5 GETTING STARTED

The following checkout procedures which are performed without a target system, will acquaint you with some of the most commonly used Emulator commands. If any problems are found while executing this procedure, contact your local representative for further assistance.

2.5.1 CONVENTIONS

The following conventions are used to key-in characters, statements, and commands.

LETTERS	Commands and statements are composed of letters and special characters. They are typed in uppercase and must be exactly as shown.
SPECIAL KEYS	If a special character key is required, it will be enclosed in angle brackets. Example 1: <RETURN> means press the RETURN key on the keyboard. Example 2: JM<RETURN> means key-in the letters JM, then press the RETURN key.

Note that the angle brackets should not be typed. They only show syntax and are not part of any command or required input.

BLANKS	No BLANKS are allowed within system commands.

2.5.2 POWERING UP THE SYSTEM

a. Set the ON/OFF power switch at the rear of the system to ON.

b. Set the ON/OFF power switch on the front panel of the disk drive to ON.

c. Set the ON/OFF power switch on the front panel of the Emulator to ON.

If the following procedure, or any portion of it fails, refer to **Chapter 3, MAINTENANCE.**

2.5.3 INITIALIZING THE EMULATOR

The Emulator must be set to a starting position in every case, regardless of processor type. The response to the initializing procedure is the same, except the type of microprocessor displayed changes according to the Emulator personality used. An example of the procedure, using the 8086, follows:

a. Press the <RESET> button on the Emulator.

b. Insert the 8086 Emulator System File diskette (Part No. 2300-6081-02) into Drive 0.

c. Enter JS<RETURN>.

This command invokes the Emulator Debugger and displays the Executive on the console screen. See **Figure 2-2, Emulator Executive Display.** The status of each Emulator attached to the system is displayed. In this example, one Emulator is attached. It is identified as 8086v1 and is shown in a halted state. Two Emulator executive commands are displayed. Notice that the first two letters are highlighted. These two letters are the only ones which must be keyed in when entering commands.

```
          Emulator Executive Vn.n

Emulator 1   8086v1   Halted

Emulator Executive Commands are:

          SWitch display to Emulator (0-8)

          SEt dialog modes

          Specify screen write options

          ; (display comment)










Vn.n = The software/hardware version number now being tested,
           and changes with new software releases.
```

Figure 2-2. Emulator Executive Display

2.5.4 SWITCH EMULATOR

The switch command **SW** activates the specified Emulator and causes it to display its status on the console.

n = 0 reactivates the Emulator Executive.

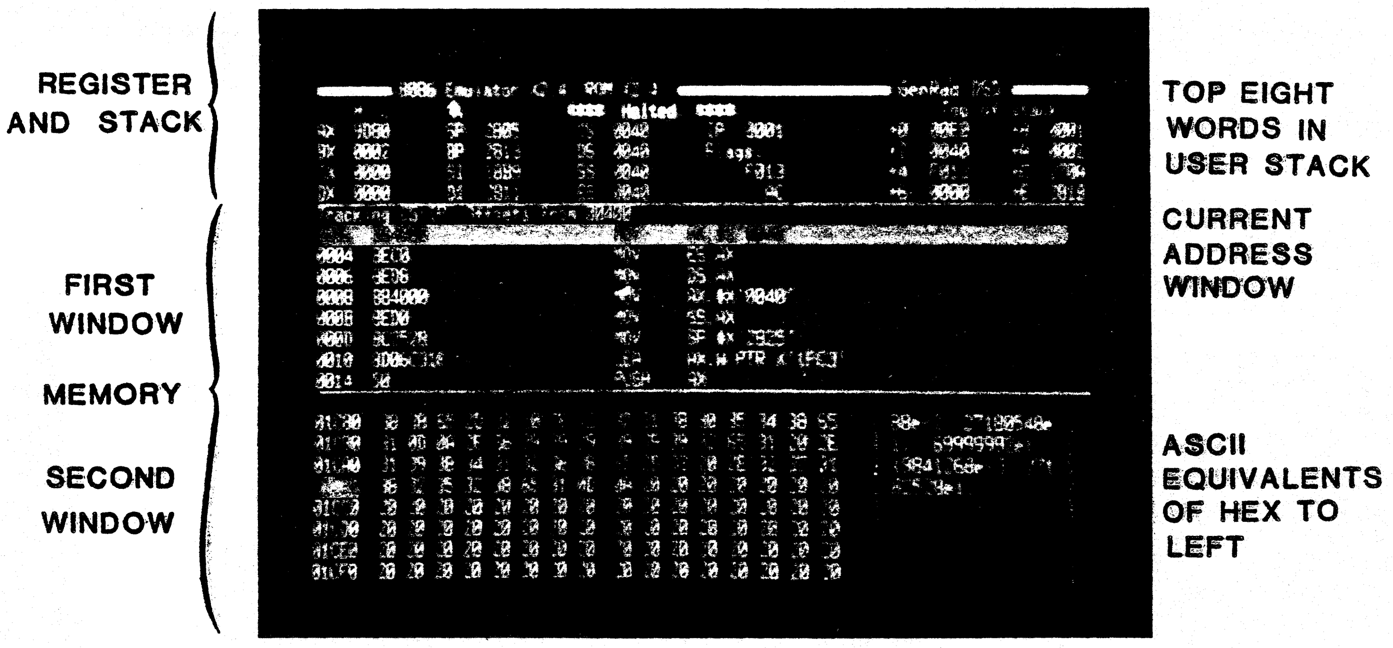
n from 1 through 8 activates the specified Emulator to be interrogated.

1. Enter SW1. **Figure 2-3** shows the correct display for this command.

```
                Emulator Executive Vn.n

      Switch display to Emulator (0-8) 1 [Confirm]

      Emulator 1   8086v1   Halted

      Emulator Executive commands are:

                   SWitch display to Emulator (0-8)

                   SEt dialog modes

                   Specify screen write options

                   ; (display comment)




     Vn.n = The software/hardware version number now being tested,
            and changes with new software releases.
```

Figure 2-3. Enter SW1

2. Enter <RETURN>. **Figure 2-4, Switch Command Display** shows the correct display for this command. In this display, the screen is divided into two major parts, REGISTERS AND STACK and MEMORY. In the REGISTERS AND STACK, all registers and a portion of the stack are displayed. The MEMORY portion is subdivided into CURRENT ADDRESS WINDOW, FIRST WINDOW, SECOND WINDOW, and an ASCII display of hexadecimal equivalents.



```
      8086 Emulator [illegible]                          GenRad [illegible]
                          **** Halted ****                  Top of stack
AX 0080   SP 2805   CS 0042   IP 0001           +0 [illegible]  +8 0001
BX 0002   BP [illegible]   DS 0040   Flags                   +2 0040  +A 0001
CX 0000   SI [illegible]   SS 0042       F013              +4 [illegible]  +C [illegible]
DX 0000   DI [illegible]   ES 0042         AC              +6 0000  +E [illegible]
[illegible]
0004  8EC0                 MOV   ES AX
0006  8ED8                 MOV   DS AX
0008  B84000               MOV   AX 0040
000B  8ED0                 MOV   SS AX
000D  BC[illegible]        MOV   SP [illegible]
0010  [illegible]          [illegible]  AX W PTR [illegible]
0014  50                   PUSH  AX
```

REGISTER DESIGNATIONS

AX – Accumulator	SP – Stack Pointer	CS – Code Segment
BX – Base	BP – Base Pointer	DS – Data Segment
CX – Count	SI – Source Index	SS – Stack Segment
DX – Data	DI – Destination Index	ES – Extra Segment
		IP – INSTRUCTION POINTER

Figure 2-4. Switch Command Display

2.5.5 DEBUGGER COMMANDS

When in the debugger mode, commands can be entered in any order. The commands available can be referenced at any time by entering a <?> or <HELP>. Either command will display the help file, which lists all commands. In addition, once a command has been entered, a <?> can be entered again for additional information on the current command.

1. To begin, enter the Edit Simulation Memory Command EDS<RETURN>. The display will switch to the Simulation Memory configuration map. If Simulation Memory was not ordered or included in your unit, this procedure cannot be executed. The Simulation Memory Map indicates the number of blocks available, the size of the blocks (8K, 32K or 64K), and where they are presently mapped. When the system is initialized, all blocks are mapped contiguously from 0. Make sure that there is at least 16K bytes of Simulation Memory available. The Debugger prompts the block which you choose to relocate. The next prompt would be what starting address you want to relocate it to. In this manner, Simulation Memory blocks can be relocated to any 8K or 32K byte block boundary within the microprocessor's address space. In this checkout procedure the aim is to check on the current status of Simulation Memory without altering it. Press the <CAN> key to return to the Debugger display.

2. The counterpart to Edit Simulation Memory command is Edit Target Memory. Enter **EDT<RETURN>**. The display will switch to the Target Memory map. This map indicates which memory spaces are internal and which are external. Blocks of memory as small as 256 bytes can be mapped internally or externally by answering the series of prompts. When the system is initialized, it maps all of the memory spaces internally. If there is no target system connected, all of the memory space would be mapped internally. Press the **<CAN>** key to return to the Debugger.

3. All of the registers in the top window can be altered by the **SEt** command. Enter **SEt**, then the desired register mnemonic, and then the value. In this case set the program counter or instruction pointer to 500 (The program counter/instruction pointer is hereafter referred to as program counter). Notice that as you select the register, it is highlighted in reverse video, and then is altered to the value entered. At the same time, the upper memory window is updated to display the values at the program counter. When the Emulator is initialized, the top window will track the program counter.

4. One of the memory window options is to display from an offset or absolute address. The Emulator, when emulating segmented microprocessors such as the 8086, will initialize with the offset mode enabled in the upper window. To enable the offset mode the command WIndow is used and the option OFfset is selected. The offset value for the memory window is set using the OFfset command.

5. Programs can be stored into memory in two ways. They can be loaded down from disk or they can be entered via the keyboard. In this example we will store a short loop by using the STore command. The STore command will store data in bytes starting at the current memory location. The current memory location is indicated in one of the memory windows by the reverse video display across the entire memory window. In this case data will be stored at 500. In the next step, proceed to the section for the microprocessor you are emulating.

6. A. 8086

   The 8086 has a code segment register. The upper memory window will track the instruction pointer and code segment registers. Set the code segment register **CS** to 0000. To do this use the **SEt** register command. Enter **SEt** register **CS=0000.**

   Store the following bytes.

   Enter **SToreFF C0 A3 50 05 EB F9 <RETURN><RETURN>**

   Notice that the upper memory window now has the above opcode displayed in both machine code and disassembled shown below.

```
500     FFC0      INC     AX
502     A35005    MOV     W PTR X`0550',AX
505     EBF9      JMP     SHORT X`0500'
```

   Proceed to step 7.

   B. 68000

   The 68000 has a Trace bit that will generate an interrupt after every instruction if it is set in the Status Register **SR.** Set the Status Register to 2700 by entering **SEt** register 2700 <RETURN> **to disable the Trace bit.**

   Store the followinſ ⁽vtes.

Enter STore52 80 31 C0 05 50 60 **F8 <RETURN> <RETURN>**

Notice that the upper memory window now has the above opcode displayed in both machine code and disassembled as shown below.

```
500     5280      ADDQ.L    #1,D0
502     31C00550  MOVE.W    D0,000500
506     60F8 BRA  000500
```

Proceed to step 7.

C. Z8000

Set the Fcw by entering **SEt register Fcw** 4000.

Store the following bytes.

Enter **STore**A9 10 6F 01 05 50 5E 08 05 00 **<RETURN> <RETURN>**

Notice that the upper memory window now displays the machine code and its disassembly as shown below.

```
500     A910 INC  R1,#1
502     6F010550  LD        0550,R1
506     5E080500  JP        UN,0500
```

Proceed to step 7.

D. 6502

Store the following bytes.

Enter **STore**EE 05 50 AD 05 50 4C 00 05 **<RETURN> <RETURN>**

Notice that the upper memory window displays the machine code and its disassembly as shown below.

```
500     EE0550    INC       5005
503     AD0550    LDA       5005
506     4C0005    JMP       0500
```

Proceed to step 7.

E. 6809

Store the following bytes.

Enter **STore4C B7 05 50 20 FA <RETURN> <RETURN>**

Notice that the upper memory window now displays the machine code and its disassembly as shown below.

```
500   4C       INC   A
501   B70550   STA   0550
504   20FA     BRA   0500
```

Proceed to step 7.

F. Z80

Store the following bytes.

Enter **STore3C 3A 50 05 C3 00 05 <RETURN> <RETURN>**

Notice that the upper memory window displays the machine code and its disassembly as shown below.

Show disassembled Z80 code.

Proceed to step 7.

7. When executing a program, several memory windows can be tracked. The number of memory windows (1 to 4), and their orientation is selected by the SCreen command. This command selects 8 memory window display formats. To find out what the optional formats are, enter **SC?**. The display will switch to the screen format help display. Press the **<CAN>** key to return to the Debugger display. When initialized the Emulator selects format 2.

8. While the Emulator is executing, the memory windows are updated every time a command string is completed or a breakpoint is reached. To update the screen periodically a **<RETURN>** can be entered to force updating. Memory windows can track a specific location or track at a register. By entering **Display expression,** the memory window will track a specific memory space. **Display @ expression** is used to track a register. These commands are implemented on the current memory window. The current memory window is selected by pressing the **<TAB>** key. In this example the top memory window is tracking the program counter and the bottom window is displaying above and below 0000.

9. Each memory window can display memory data in two formats: hexadecimal or symbolic. This is selected by the **WIndow** command. In this example the upper window is displaying in the symbolic format and the lower window is in the hexadecimal format.

10. The program counter is now at 500 and the upper memory window is tracking the program counter at 500. The **EXecute** command can now be invoked and the Emulator will begin execution at the program counter. Enter **EX**ecute <RETURN>. The message **Running** will appear at the message line on the display. Press the **<RETURN>** key several times and notice that the program counter, the incremented register, and the upper memory window are updated with each stroke of the return key.

11. The **Halt** command forces the Emulator to cease execution of user's code and updates the Debugger display to the point where the processor is halted. Enter **Halt<RETURN>** Notice that the message line now reads **Halted** and that the program counter and upper window display are now updated to the next instruction to be executed.

13. The **<STEP>** key will execute one instruction at the current program counter. Press the **<STEP>** key and notice the message **Completed Single Step** appears at the message line. Notice also that the program counter and upper mem[illegible] window have advanced to the next instruction.

The four hardware breakpoints give complex breaking options. Enter **(Breakpoint 1,)**. The screen will switch to the breakpoint 1 status display. On the left hand side of the screen are the options and on the right hand side of the screen are the current parameter settings for breakpoint 1. Count indicates the number of passes over the breakpoint before a break occurs. Set Count to 32767 by entering **(Count 32767)**.

Address and data can be set to a specific value or **X** for don't care. Set address to 500 by entering **(Address 500)**. Data should be left at don't care for this example.

External lines are the four external lines from the External Probe. In this example they are set at don't care.

Instruction/Data indicates whether the breakpoint should break on the fetch or the execution of the specified address. For prefetching processors like the 8086, this makes a significant difference since the microprocessor can fetch ahead as many as six bytes. When breaking on an instruction, it should be set to Instruction. When breaking on a data move as in a peripheral access, it should be set to Data. In this example it is set to Instruction.

Read/Write specifies the direction of data flow for the breakpoint.

Memory/IO indicates that the break will occur during a memory or I/O cycle. For some microprocessors like the 68000, which has no I/O cycles, this parameter is meaningless, and it is ignored. For this example, it is left at Memory.

Halt/Snapshot indicates whether the processor will halt or continue execution after the breakpoint is reached. In this example, it is left at Halt.

Breakpoint 1 should now be set as follows:

Count	= 32767
Add	= 00500
Data	= XXXX
Ext Line	= XXXX
Instruction/Data	= I
Read/Write	= R
Memory/IO	= M
Halt/Snapshot	= H

14. Execute the program by entering the **EXecute** command. Notice the message **Stopped at Breakpoint 1** appears on the message line. The program counter and the upper memory window will display the next instruction to be executed by the Emulator. Since prefetch and breakpoint characteristics of different microprocessors are different, each microprocessor will halt at a different point.

15. The breakpoints can be cleared individually using the **RESEt** command or all breakpoints can be cleared using the **CLear** command.

The preceding 15 steps have been an introduction to some of the commands as they might be used without a target system. Three more steps are required to begin emulation with a target system.

1. The Emulator Plug is inserted in place of the microprocessor on the target system. Use caution when plugging the Emulator probe into the target system. Placing the plug in backwards can damage the probe, or cause the target power supply to short to the probe's ground. Care should be taken that pins are not bent and that the Plug is inserted in the correct orientation.

2. Some or all of the microprocessor's address space must be mapped externally to gain access to the desired portions of target memory. This is done using the **EDit target memory** command.

3. Each Emulator Subsystem has target system control lines to the Emulation Processor. These can be enabled, or disabled under the **ENable control lines command.** It may be necessary to enable some or all of the control lines to the target system depending on the microprocessor and the target system. For example, if the target memory to be accessed is slow, and requires the use of a wait line, this line should be enabled under the **ENable control lines** command. When you choose to ignore all interrupts, and DMA processing, leave the **ENable control lines** disabled.

Additional processor specific information is given in the Supplement for each of the Emulator Subsytems. The Debugger chapter of the Development Manual provides detailed descriptions of all commands.

2.6 <u>EXTERNAL PROBE FUNCTIONAL DESCRIPTION</u>

The External Probe performs two functions. First function is to provide external breakpoint triggers. These triggers can then be used as a complex trigger for oscilloscopes, or passed on to another Emulator to provide synchronous breakpoints in a multiprocessor emulation environment. The second function of the External Probe is to input four external, TTL compatible conditions to the Breakpoint Board for external break conditioning, and to the Logic Analyzer to be traced with the other Emulation Processor functions.

The External Probe is described in complete detail in Section 4.9.

CHAPTER 3

MAINTENANCE

3.1 SYSTEM MAINTENANCE AND TROUBLESHOOTING

Preventive maintenance is not required on the Emulator as a whole, or on any subsystem unit. When a malfunction does occur, troubleshooting and analysis should first be attempted on a system level. This can best be done by using the system software to examine or interrogate the various units, observing the resulting console displays (and/or hard copy if available), then proceeding to correct the operational defect. There are several operational checks to determine if proper procedure has been violated resulting in an error. In addition to the procedures following, please refer to Section 2 in the Supplement for possible errors in configuration or application. Emulator troubleshooting requires the user to be familiar with total system software and ADS Console programming operations.

3.1.1 EMULATOR MALFUNCTIONS

The Emulator unit requires no operational mechanical or electrical preventive maintenance. If trouble develops immediately after any part of the Emulator has been disconnected, or disassembled, recheck the work done.

Error symptoms may have causes outside the Emulator unit. These could be hardware-originated, software-originated, or both. Check to see that the Emulator Probe and Emulator Personality boards are compatible. The software revision number on the Emulator Executive must match the Interface Processor board PROMs. Check the Hardware Supplement Section 2 for correct board configurations.

If any board is suspected as the source of trouble, check first to make sure that all IC's are snug in their sockets.

Check all boards, cables and connectors for a tight fit.

Check that the individual straps on each of the boards is correct.

3.1.2 FLOPPY DISK UNIT MALFUNCTIONS

Preventive Maintenance. Procedures for preventive maintenance are detailed in the FutureData **INSTALLATION AND MAINTENANCE MANUAL** (2300-5003-01).

3.1.3 ADS MALFUNCTIONS

ADS malfunctions, maintenance techniques, and troubleshooting information are found in the FutureData **INSTALLATION AND MAINTENANCE MANUAL** (2300-5003-01).

3.2 EMULATOR TROUBLESHOOTING CHARTS

Each of the following generalized charts is correlated with the Emulator checks performed in **Chapter 2**.

PROBABLE CAUSE

A	I/F PROCESSOR PWB
B	MEMORY PWB
C	PERSONALITY PWB
D	BREAKPOINT PWB
E	LOGIC ANALYZER PWB
F	RS-232 INTERFACE
G	EMULATOR PROBE
H	SYSTEM CABLING
I	DISKETTE/DISK UNIT
J	CONSOLE HARDWARE
K	CONSOLE CONFIG.
L	DOCUMENTATION
M	INTERNAL CABLING
N	EMULATOR NOT RESET
O	CONSOLE NOT RESET
P	MPIO STRAPPING
Q	WRONG S/W VERSION
R	OPERATOR ERROR
S	WRONG F/W VERSION
T	
U	

EMULATOR
EXECUTIVE TEST
ENTER JS<RETURN>

ERROR SYMPTOM	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1. No 'Halted' Display	X					X				X			X	X	X	X					
2. No 'Commands' Display			X						X					X	X						
3. Blank Screen										X											
4. Target System Check	X		X				X						X					X			
5. Logic Error 18			X				X						X								
6. Cannot Run Command File														X	X		X	X	X		
7. Cannot Enter Executive									X	X	X										
8.																					
9.																					
10.																					
11.																					
12.																					
13.																					
14.																					
15.																					

Figure 3-1. Executive Test Symptoms

PROBABLE CAUSE

A	I/F PROCESSOR PWB
B	MEMORY PWB
C	PERSONALITY PWB
D	BREAKPOINT PWB
E	LOGIC ANALYZER PWB
F	RS-232 INTERFACE
G	EMULATOR PROBE
H	SYSTEM CABLING
I	DISKETTE/DISK UNIT
J	CONSOLE HARDWARE
K	CONSOLE CONFIG.
L	DOCUMENTATION
M	INTERNAL CABLING
N	EMULATOR NOT RESET
O	CONSOLE NOT RESET
P	MPIO STRAPPING
Q	WRONG S/W VERSION
R	OPERATOR ERROR
S	WRONG F/W VERSION
T	
U	

EMULATOR
SWITCH DISPLAY
ENTER SW[n]
n=[0-8]

ERROR SYMPTOM	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1. Wrong Status Event																					
2. Emulator Will Not Switch										X							X		X		
3. Register/Stack Not Shown	X		X														X		X		
4. No Memory Displayed	X		X														X		X		
5. Current Window Not Shown																					
6. No ASCII Display						X															
7. No Tracking Message																					
8. Cannot Store Data		X																			
9.																					
10.																					
11.																					
12.																					
13.																					
14.																					
15.																					

Figure 3-2. Switch Display Symptoms

3.3 DISASSEMBLY INSTRUCTIONS

Major subassemblies can be readily removed from the main frame. These subassemblies are:

a. rear panel
b. enclosure cover
c. front panel with power supply

The customer should avoid disassembly beyond that described in this manual, especially during the warranty period. Only Kontron Service Representatives should attempt further detailed disassembly of this equipment. **Figure 3-3, Exploded View**, depicts the Emulator subassembly.

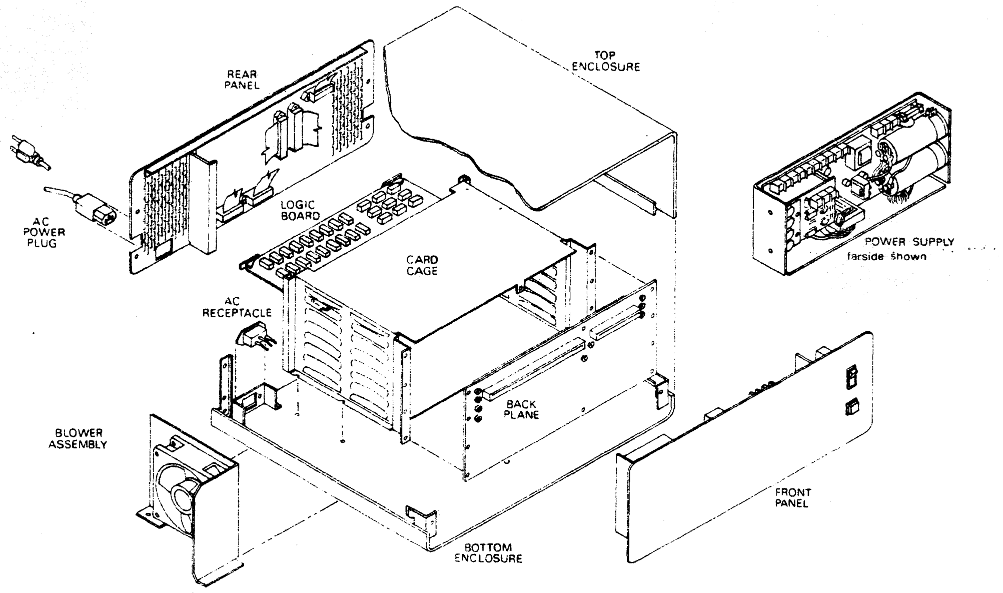


Figure 3-3. Exploded View

3.3.1 REAR PANEL REMOVAL

To reconfigure the Emulator, it is necessary to remove the rear panel. Refer to **Figure 3-4, Rear Panel Removal.**

a. Set the ac ON/OFF power switch to OFF and unplug the ac line cord at both ends.

b. Remove all flat cables from their rear panel connectors. When unscrewing jack-screws, alternate between screws so that both connectors stay parallel.

c. Remove the four screws shown in the illustration. The rear panel can now be removed.

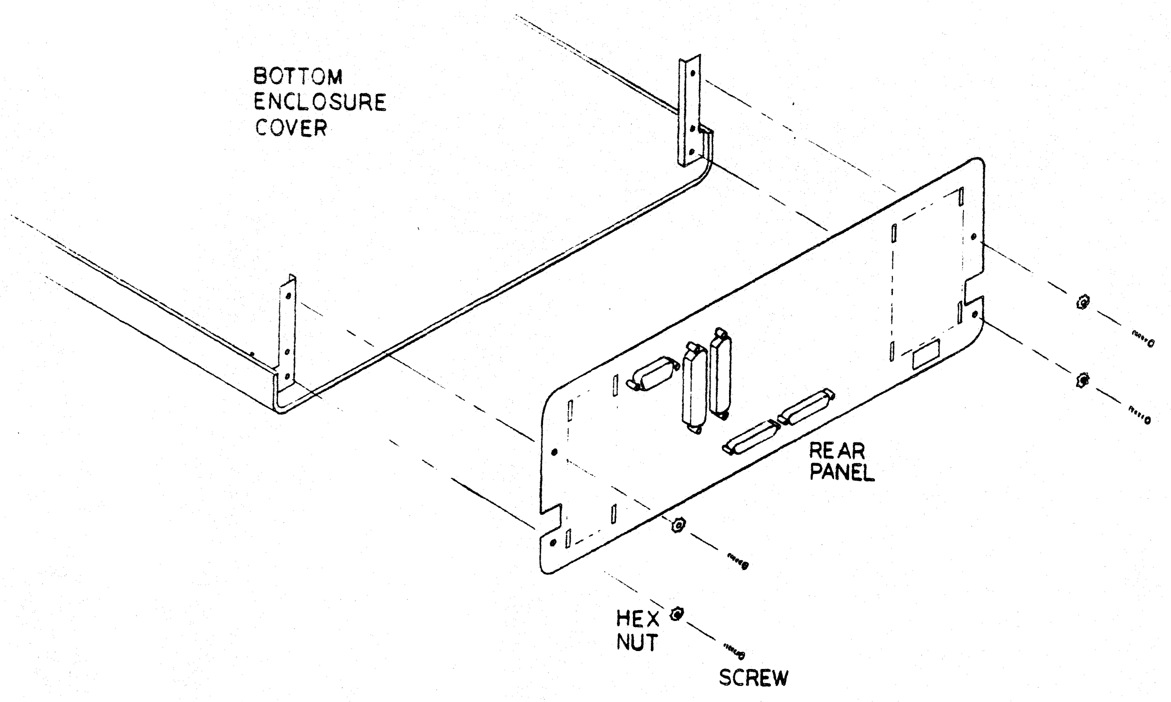


Figure 3-4. Rear Panel Removal

3.3.2 BOARD REMOVAL

Removing logic boards is a simple operation. Refer to **Figure 3-5, Board Removal.**

Position thumbs on the board ejectors and pull back, rotating the ejectors outward until the board is pulled away from the Emulator backplane connector. Gently slide the board all the way out.

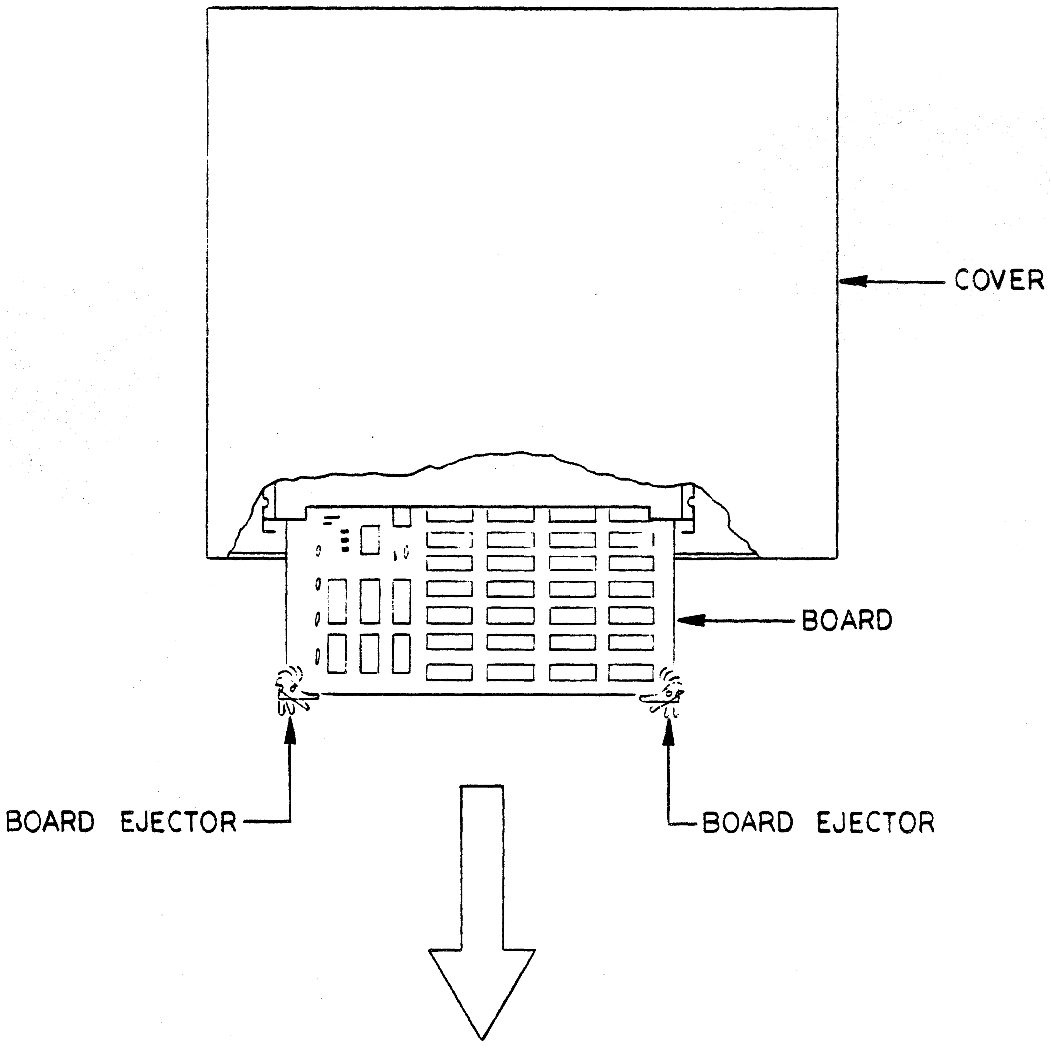


Figure 3-5. Board Removal

3.3.3 ENCLOSURE COVER REMOVAL

Refer to **Figure 3-6, Enclosure Cover Removal.** It is not usually necessary to open up the interior of the Emulator except to replace the power supply fuse.
First, the rear panel must be removed as shown in Section 3.3.1. Then, remove the two screws holding the top cover to the corner brackets as shown in the illustration, Lift and pull back from the rear as shown. Notice the cover-locking guides at the front sides of the chassis.

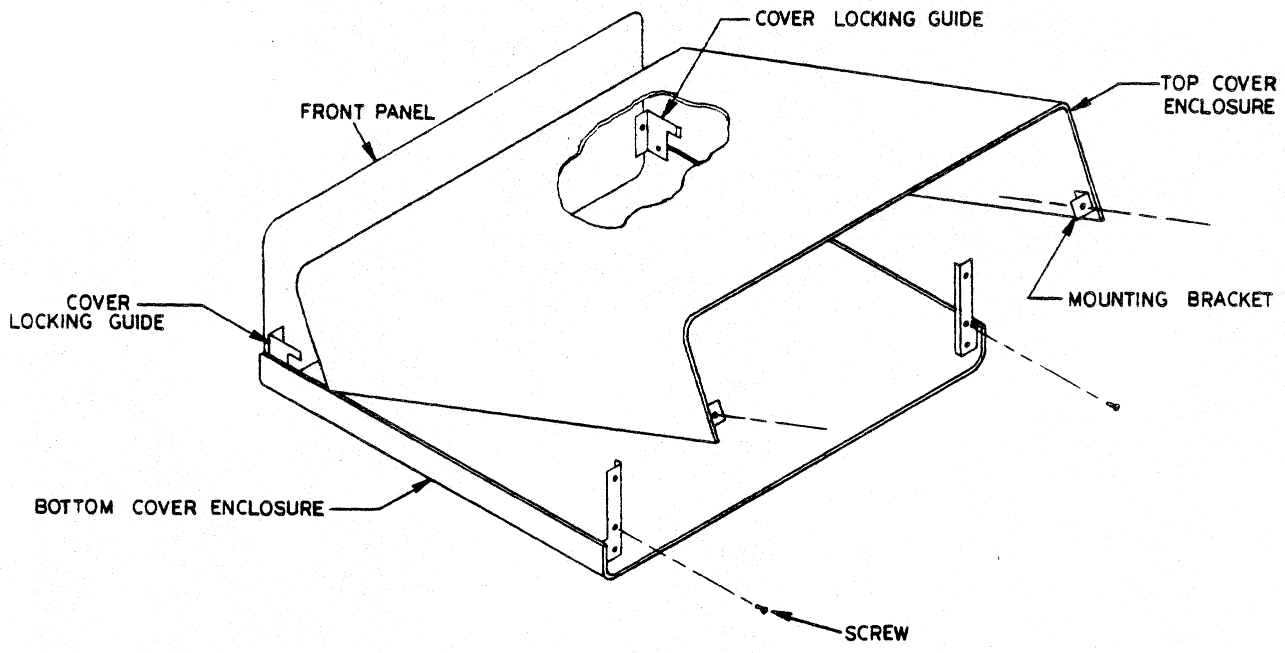


Figure 3-6. Enclosure Cover Removal

3.3.4 FRONT PANEL/POWER SUPPLY REMOVAL

Refer to **Figure 3-7, Front Panel Details.**

a. Label or mark all wires and wire cables on strip terminals, ON/OFF switch, and RESET switch. Remove the wires and wire cables.

b. Remove the two hexagonal nuts, shown in the illustration, from the front mounting flange on the bottom enclosure. Remove the front panel.

c. Remove the four hexagonal nuts holding the power supply mounting flange to the front panel. Remove the power supply assembly from the front panel.

d. Remove the hexagonal nuts holding the power supply to the flange.

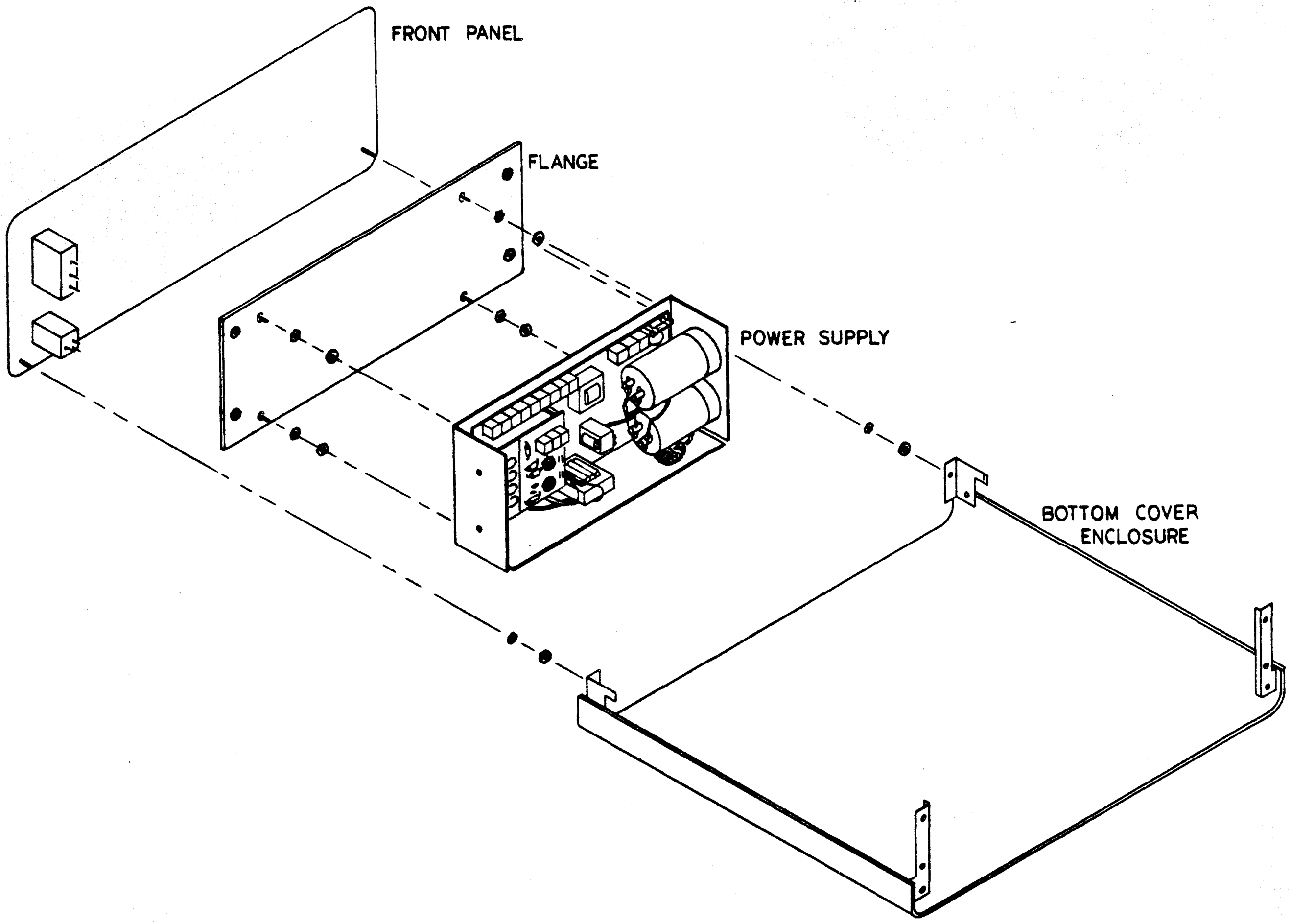


Figure 3-7. Front Panel Details

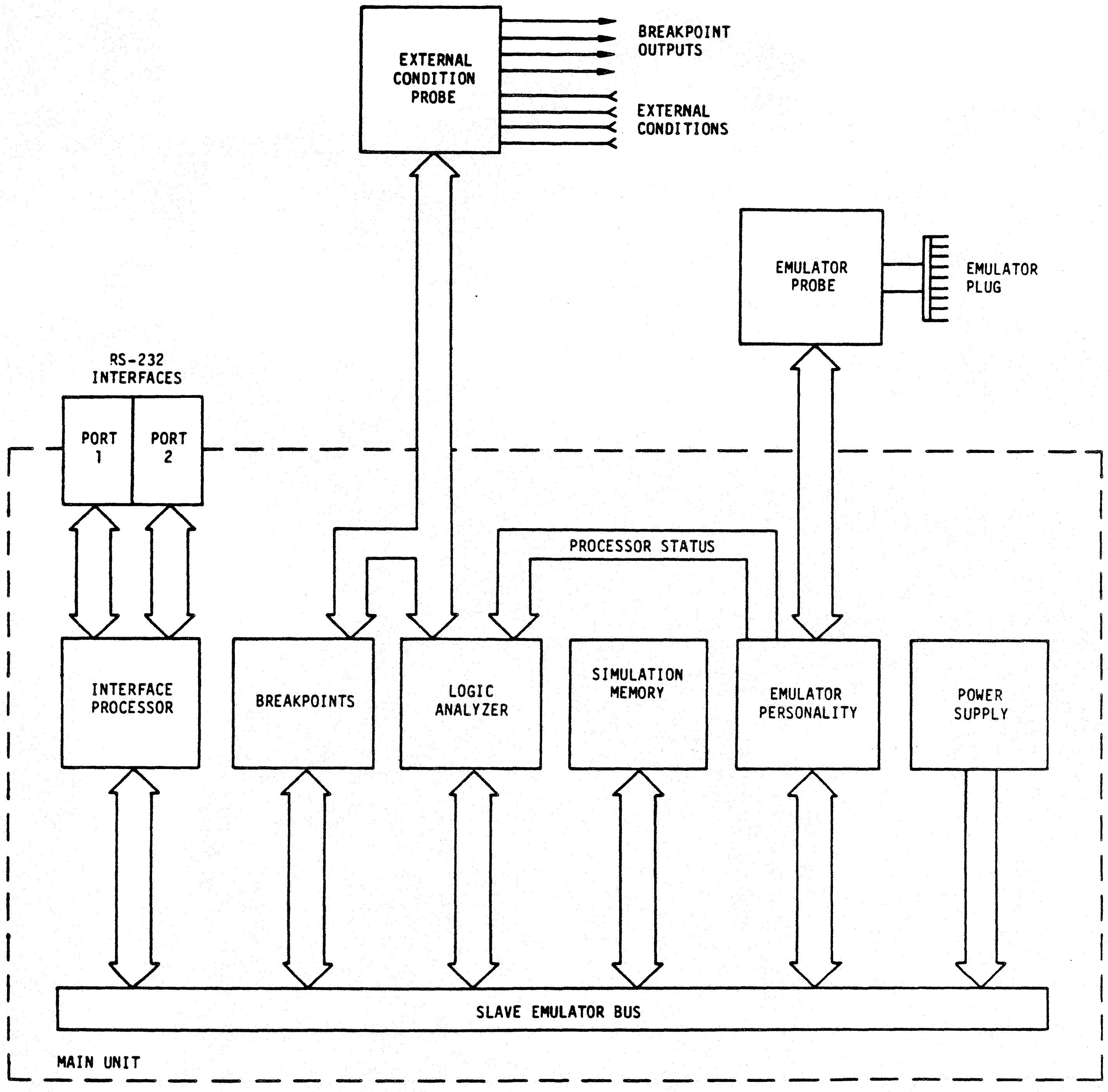


Figure 4-0 Slave Emulator Block Diagram

CHAPTER 4

THEORY OF OPERATION

4.1 SCOPE

This chapter describes all of the major functions of the Emulator and each logic board.

4.2 GENERAL DESCRIPTION

In the past, practical in-circuit emulation equipment did not allow the processor under test to run at full speed, or it partially limited the use of the microprocessor memory by the emulation equipment. Also, the system under test had to be halted to conduct debugging operations. These limitations are solved with the 2302 Emulator. Rather than using a single bus, the Emulator employs a multiprocessor, multiple bus organization that permits full-speed emulation of 8-bit, or 16-bit microprocessors.

The Emulator works in conjunction with Kontron's 2300 Series Advanced Development System (ADS). Communication takes place over a 19.2-kb/s RS-232C serial interface. Up to four Emulators may be daisy-chained to a single development system. Each Emulator includes a 4MHz Z80-based Interface Processor as the main processing unit.

Every Emulator has a separate Probe which contains the processor to be emulated. The user can add optional boards to get additional memory, or a Logic Analyzer. Each Emulator may uses a different Emulator subsystem, which has a different Personality Board that plugs into the Emulator back plane cardcage, and an external Emulator Probe which places critical components close to the user's prototype.

The target system's microprocessor is removed and the Emulator Probe is plugged into the vacant socket. This Probe module contains a processor of the same type as the one just removed from the target system. The Emulator substitutes this proxy processor -- called an Emulation Processor -- for the target system's processor.

Thus the Emulator's first function is to imitate the target system's processor. Its second function is to provide the user control over the emulation. This is accomplished by surrounding the Emulation Processor with controlling circuitry which is used to monitor and control the Emulation Processor.

The Emulator therefore performs these three functions:

1. Substitute an Emulation Processor for the target system's processor.

2. Provide the user the ability to monitor the Emulation Processor.

3. Provide the user the ability to control the Emulation Processor.

4.3 SUBSTITUTING THE EMULATION PROCESSOR FOR THE TARGET SYSTEM'S PROCESSOR

The Emulator Subsystem -- consisting of Emulator Personality board and Probe board -- substitutes an identical processor for the target system's processor. To eliminate critical physical distances, this Emulation Processor is housed in the Probe, which is connected by a short cable to the target system's vacant microprocessor socket. The signals into and out of the Emulation Processor are relayed from the Probe to the target system's socket.

4.4 MONITORING THE EMULATION PROCESSOR

Controlling the Emulation Processor requires the ability to monitor it. The Emulator contains three boards concerned with this task. They are:

1. The Logic Analyzer Board (optional).
2. The Breakpoint Board.
3. The Emulator Personality Board.

4.4.1 LOGIC ANALYZER BOARD

The Logic Analyzer works together with the Breakpoint Board to provide up to 256 bus cycles of information. This information includes all address and data lines, four external lines from the External Probe which the user selects, and control lines from the microprocessor. The Logic Analyzer is always in operation, recording all bus cycles even when not in use. The Logic Analyzer is halted by breakpoints defined by the user. The Analyzer also contains a timer which can be activated by Breakpoint 1 and halted by Breakpoint 2. This timer can record elapsed time between two events in 100's of nanoseconds, microseconds, or number of bus cycles.

4.4.2 **BREAKPOINT BOARD**

The Breakpoint Board provides four hardware breakpoints. Hardware breakpoints are used so that the user's code is not altered as would be necessary with software breakpoints. Four hardware points also provide the user with increased versatility in specifying the number of passes over a breakpoint, whether to break on a read or write cycle, and in specifying the address or the data on which to break, without interfering with the user's code

4.4.3 **EMULATOR PERSONALITY BOARD**

This board examines the internal registers of the Emulation Processor, records the contents of these registers in Test Memory area, and allows user access to this information.

4.4.4 **SIMULATION MEMORY**

Simulation Memory is provided for use in target system development or debugging. The Emulator does not use the Simulation Memory to perform any of its functions. Depending on the memory options selected, up to four Simulation Memory Boards may be placed in the Emulator, to provide from 16-kilobytes to 512-kilobytes of memory. This memory can be edited into 8-kilobyte blocks and mapped internally into any area of the emulated processor's address space, or externally to the target memory, down to 256-kilobyte boundaries. Simulation memory is provided to be used together with, or in place of the user's target memory. Simulation memory may also be write-protected to simulate ROM.

4.5 **CONTROLLING THE EMULATION PROCESSOR**

Although there are three boards in the Emulator concerned with monitoring the Emulation Processor, only the Personality Board has direct control over it. The Emulator interface is separated into two sections to minimize the physical and electrical distances between the Emulator and the target being developed. The bulk of the Emulator interface is housed on the Personality Board. The rest of the interface is in the Emulation Probe which is attached to the Personality Board by two 50-connector cables. The probe is plugged into the microprocessor socket of the system being emulated.

4.5.1 EMULATOR PERSONALITY BOARD

The Emulator Personality board is designed to perform the following functions:

1. Examine registers of the Emulation Processor.
2. Change the contents of registers.
3. Read and write to memory locations.
4. Read and write to I/O ports.
5. Enable or disable target control of the microprocessor.
6. Run or halt the microprocessor.

This last function adds to the ability of the Emulator to imitate the target system's microprocessor.

4.5.2 INTERFACE PROCESSOR BOARD

The Interface Processor board controls all the other boards in the Emulator, and communicates with the ADS controlling the Emulator. This board contains a Z80 microprocessor running at 4MHz, 16K of dynamic RAM, two serial I/O ports, and 16K of ROM.

All communication with the user comes from the ADS into Serial Port 1. For example, when the user examines the contents of the Logic Analyzer, the command to do so, and the information retrieved from the Logic Analyzer are relayed to the user through the Interface Processor. When communicating with any one board, the information is transmitted from the user through the Interface Processor to the specific board.

One of the important functions of the Interface Processor is the handling of "test" programs. These are the short programs which the Emulation Processor executes allowing user control over the Emulation Processor. When the Emulation processor is asked to execute a test program, the information picked up by the test program is relayed to the user through the Interface Processor. The test programs are stored in the 16K of dynamic RAM on the Interface Processor. When the Emulation Processor is interrupted, the appropriate test programs are loaded down into a small memory space on the Personality board called the Test Memory. If a data gathering test program is executed, the results of the program are stored in Test Memory. When requested by the user, information in Test Memory is picked up, and relayed to the ADS by the Interface Processor.

4.6 EMULATION TECHNIQUES

4.6.1 INITIALIZATION

Initialization is the process which prepares the Emulator hardware for emulation. This includes clearing all of the breakpoints, disabling the Logic Analyzer, mapping the Emulation Processor's address space internally to the Emulator, disabling all external Emulation Processor control lines, and "capturing" the Emulation Processor.

Capturing the Emulation Processor means to stop it from whatever it is doing (usually executing user code) and to give control of it to the Emulator. The Interface Processor (IP) commands the Emulator Personality (EP) to capture the Emulation Processor through one of its I/O control lines. The EP then begins the capture sequence through the use of a sequencer (state machine). Each Personality Board has to utilize a different technique, depending on the microprocessor involved, for capturing the Emulation processor. Some Personality Boards use a non-maskable interrupt and others substitute an instruction onto the data bus.

When execution of the interrupt or the substituted instruction begins, a small block of memory, called Test Memory (TM) on the EP, is mapped into the address space of the Emulation Processor. Execution then begins within the TM which has been previously loaded with a program by the IP.

TM can be various sizes and locations depending on the microprocessor under emulation. There are several small programs run by the Emulation Processor while executing out of TM. Each program gathers a specific type of information about the Emulation Processor or the target system.

As each test program is completed in TM, the IP is notified by an Emulator Personality Interrupt (EPI) which is generated by hardware as the Emulation Processor executes a specific portion of Test Memory code. The Emulation Processor is then halted using a jump to self or HALT instruction. The IP can then load additional programs into TM. Once the Test Memory is loaded, the Emulation Processor is forced to jump to the beginning of the test program. The Emulation Processor executes the program, another EPI is sent to the IP, and the Emulation Processor executes a jump to self. This process continues until all initial information is received by the IP. The IP sends this information back to the ADS and waits for additional commands.

4.6.2 THE EXECUTE COMMAND

Once the Emulator is initialized, and all emulation parameters have been set, the user is ready to execute. Before execution begins, the Emulation Processor is at a jump to self instruction as it does at the completion of each test program from TM.

When the user issues an execute command, the IP changes the jump to self instruction that the Emulation Processor is executing to a jump relative. It then executes one or two instructions that are another jump relative or a return from interrupt. The sequencer on the EP detects this operation and tracks the Emulation Processor as it begins to fetch the code pointed to by the jump relative or the return from interrupt. When the first op-fetch of user code is made, the sequencer disables Test Memory out of the Emulation Processor's address space and execution of user's code begins.

4.6.3 THE HALT COMMAND

When the user issues a halt command, the Emulation Processor is not actually halted. It is captured by the EP. During the capture sequence, Emulation Processor's status is saved either by stacking it in Test Memory, or by hardware found on the EP. At the same time some, external control lines (interrupts) are disabled. This prevents loss of control by the Emulator. Once the Emulation Processor has been captured by the EP, an EPI is sent to the IP.

If the EP fails to capture the Emulation Processor from user control, a target system check message is displayed on the screen. Failure to capture the processor is usually caused by an Emulation Processor that has crashed. The causes of target system check messages are hard to determine, and often microprocessor specific. These are discussed in the Supplements for each microprocessor. Lack of power on the target, or the failure of target execution because of a bad code are a couple of possibilities.

The IP then loads Test Memory with test programs for the Emulation Processor to execute. These test programs are used to update the information displayed on the screen. When all test programs are completed, the Emulation Processor remains in the halt loop until another command is received from the user.

4.6.4 STORING DATA AND SCREEN UPDATING

Storing data and updating the screen while the Emulation Processor is executing user code are accomplished in the same manner as halting and then executing. If a user desires to alter memory, or to change a register, or to update the screen by pressing the return key, or to use snapshot breakpoints, the same technique is used. The Emulation Processor is captured, test programs are run in Test Memory by the Emulation Processor to alter registers, and to change or read memory, then execution is continued.

4.6.5 **SINGLE STEP**

Single stepping is essentially an execute command followed immediately by a halt command. The sequencer on the EP tracks the Emulation Processor as it exits TM code and begins to execute user code. As the first instruction is completed, the Emulation Processor is forced to execute from TM by the sequencer and effectively halts.

4.6.6 **MEMORY MAPPING**

There are two types of memory mapping used for emulation: (A) Simulation Memory Mapping and (B) Internal/External Address Space Mapping.

A. Simulation Memory Mapping

Simulation memory is optional memory within the Emulator which can be used in place of the user's target memory or by itself when executing code entirely within the Emulator. Memory is available on 16 Kbyte static, 32 Kbyte static, 64 Kbyte dynamic, and 128 Kbyte dynamic boards. Both of the static memories can be further divided into 8 Kbyte blocks and the 128 Kbyte dynamic memory into 32 Kbyte blocks. The 64 Kbyte memory is designed to work with some 8-bit 64 Kbyte microprocessors, and is therefore one contiguous memory. The blocks of Simulation Memory can be mapped anywhere within the address space of the microprocessor. This mapping function is accomplished by a high speed RAM on each of the memory boards. As mapping commands are given by the user, the IP configures the mapping RAM on the Simulation Memory boards. When emulation is in progress, each of the mapping RAMs monitors the Emulator address bus. When a bus cycle is executed within a memory block's mapped space, it is enabled for that cycle.

B. Internal/External Address Space Mapping

Internal refers to the mapping function of the address space within the Emulator. External refers to the mapping function of the address space within the target system. This address space mapping function can be on the IP, or on the Simulation Memory board depending on the configuration. Mapping is controlled by high speed RAM that is configured by the IP as the user edits the target system's address space. The internal/external address space mapper, which is called the target system map on the Emulator, can define whether an address is internal or external down to 256-byte blocks.

An example of how the mappers work together is shown in **Figure 4-1, Memory Mapping.** Let us assume the user wants to simulate target memory from 1C000 to 1C7FF and from 28000 to 28FFF and has one 16-kilobyte memory board. All other memory is to be mapped to the target system. First, Simulation Memory must be mapped to the appropriate areas. Since there is 16 Kbyte of Simulation Memory, both 8 Kbyte blocks are used. One block is placed at 1C000 and the other at 28000. Then target memory is mapped externally from 00000 to 1BFFF, 1C800 to 27FFF and 29000 to FFFFF. The target system mapper has priority over the simulation mapper and makes the decision to fetch from external or internal memory regardless of how Simulation Memory is mapped. If the target is mapped internally where there is no Simulation Memory, a floating bus will be read.

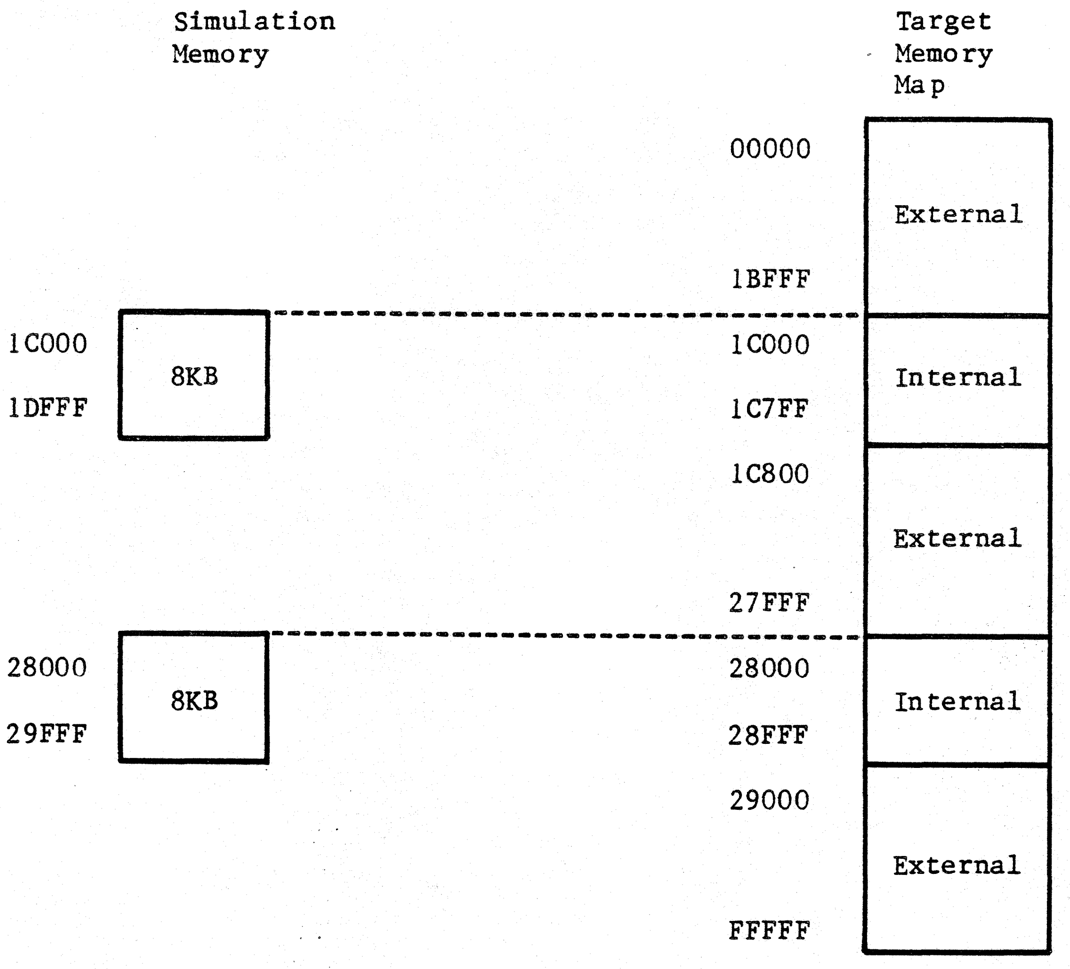


Figure 4-1. Memory Mapping

The decision to access data internally, or externally is made on a cycle by cycle basis. As the Emulation Processor puts out the address, the target memory mapper sends a signal to the EP and to the Emulator probe to switch all transceivers and buffers in the appropriate direction for the Emulation Processor to transmit or receive data. **Figure 4-2, Buffer Control** indicates the data flow and buffer control for each type of access.

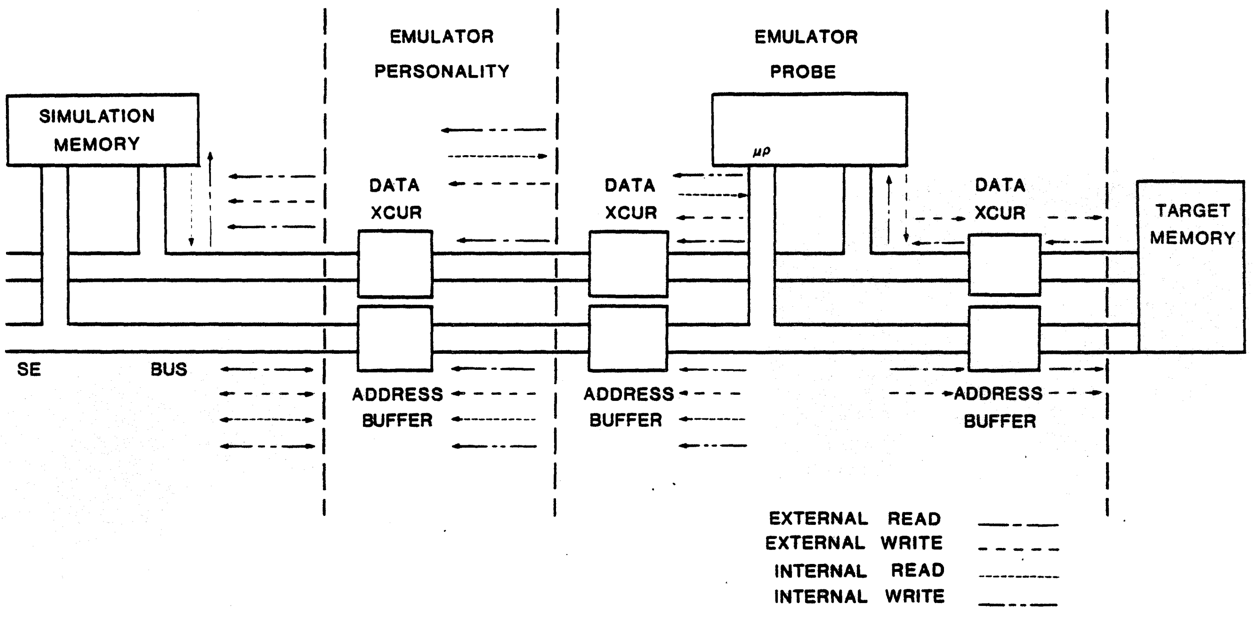


Figure 4-2. Buffer Control

4.6.7 EXTERNAL CONTROL LINES

All input control lines from the target system to the Emulation Processor on the Emulator Probe are gated through an enable/disable control. These are inputs such as interrupt lines, halt, reset, and bus arbitration. The user can control these lines while the target is under emulation using the enable command. When the Emulation Processor is executing from Test Memory, the Emulator controls and overrides the user's definitions in order to prevent the target sytem from affecting the Emulation Processor. Bus arbitration control lines are not disabled by the Emulator during Test Memory execution if the user has enabled them.

4.6.8 BREAKPOINTS

Breakpoints function in the same manner as the halt instruction. When a breakpoint is reached, an interrupt is sent to the IP (breakpoint) and to the Emulator Personality board. The Personality board interrupts execution of user code as it does when halting and then issues an interrupt (EPI) to the IP. The IP, receiving both the EPI and breakpoint, forces the execution of test programs by the Emulation Processor to update the screen. If the breakpoint was specified as halt after break, the Emulation Processor remains in a halt loop. If a snapshot breakpoint is specified, user execution continues as with the execute command.

4.7 BACKPLANE CONNECTIONS TO THE EMULATOR BUS

The backplane of the Emulator cardcage consists of eight pairs of connectors; P1 with 86 pins, and P2 with 60 pins. The designation A indicates the component side of the board and the designation B the solder side. The eight slots on the cardcage backplane are identical, and any board can be plugged into any slot on the backplane, within the internal cabling limitations of the Emulator as described in Section 1.3.5.

Figure **4-3, Emulator Bus** shows the bus system found on the backplane. +5 volts, and $\pm$ 12 volts is supplied on the bus with grounds distributed on approximately every fifth pin on both sides of each board.

P1A		P1B		P2A		P2B	
GND	1	GND	1	GND	1	GND	1
+5V	2	+5V	2	SA0+	2	SA1+	2
+5V	3	+5V	3	SA2+	3	SA3+	3
+12V	4	+12V	4	BP1+	4	BP2+	4
LAS-	5	BA25	5	BP3+	5	BP4+	5
GND	6	GND	6	BP1OUT-	6		6
BA24	7		7	EPI-	7		7
BA22	8	BA23	8	MAP-	8		8
BA20	9	BA21	9		9	BP3OUT-	9
BA18	10	BA19	10	SYS+	10	RSTPB-	10
BA16	11	BA17	11	BP2OUT-	11	BP3IN-	11
GND	12	GND	12	BP1IN-	12	BP4OUT-	12
BA14	13	BA15	13	BP2IN-	13	BP4IN-	13
BA12	14	BA13	14	IO00	14	IO01	14
BA10	15	BA11	15	IO02	15	IO03	15
BA08	16	BA09	16	IO04	16	IO05	16
GND	17	GND	17		17	IO07	17
BA06	18	BA07	18	IO08	18		18
BA04	19	BA05	19	IO10	19	IO11	19
BA02	20	BA03	20	IO12	20	IO13	20
BA00	21	BA01	21	IO14	21	IO15	21
GND	22	GND	22	IO16	22	IO17	22
BD14	23	BD15	23	IO18	23	IO19	23
BD12	24	BD13	24	IO20	24	IO21	24
BD10	25	BD11	25	IO22	25	IO23	25
BD08	26	BD09	26	IO24	26	IO25	26
GND	27	GND	27	IO26	27	IO27	27
BD06	28	BD07	28	IO28	28	IO29	28
BD04	29	BD05	29	IO30	29	IO31	29
BD02	30	BD03	30	GND	30	GND	30
BD00	31	BD01	31				
GND	32	GND	32				
BHE+	33	BYTE+	33				
EPCYC-	34	R/W-	34				
	35	HLDA-	35				
GND	36	MRDY+	36				
MEM-	37	GND	37				
GND	38	GND	38				
IPWAIT-	39		39				
	40		40				
+5V	41	+5V	41				
+5V	42	+5V	42				
GND	43	GND	43				

Figure 4-3. Emulator Bus

Signal Descriptions

BA00 to BA25	This is the address bus. Both the interface processor and the Emulation Processor under emulation (on the Emulator probe) have access to this bus. The Emulation Processor has priority on the bus. The IP uses the bus for all transactions concerning the Test Memory, Breakpoint, Bus Analyzer trace setup and data, and memory mapping. The Emulation Processor uses the bus to access Simulation Memory. In the descriptions below, a "-" indicates that the signal is active low.
BD00 to BD15	This is the data bus. Its use is the same as the address bus described above.
LAS-	Logic Analyzer Strobe. The negative edge of this signal strobes trace information into the Logic Analyzer, and is also the strobe for breakpoint comparisons on the Breakpoint Board. Sourced from the EP board.
BHE	Byte High Enable. Enables the upper byte from Simulation or Test Memory onto data lines BD8-BD15. Sourced from the IP or EP.
EPCYC-	Emulator Personality Cycle. Indicates that the Emulation Processor is the bus master. This signal is asserted high for IP cycles. Sourced from the EP board.
MEM-	Memory Cycle. Indicates a memory cycle is in progress on the bus. Sourced from the IP or EP.
IPWAIT	Interface Processor Waiting. Indicates to the Emulator Personality Board that the IP is requesting the bus. The EP will hold off the bus to the Emulation Processor and give it to the IP by asserting EPCYC high for one IP bus cycle. Sourced from the IP.
BYTE	Indicates that the bus cycle in progress is a byte cycle and data should be placed on the BD00-BD07 lines. Sourced from the IP or the EP.
R/W-	Indicates the type of cycle being conducted by the IP or the Emulation Processor. Low indicates a write cycle. Sourced from the IP or EP.
HLDA-	Unused; reserved.
PRTY-	Indicates a parity error in Simulation Memory. Generates an interrupt to the IP.

RFSH-	Refresh indicates a refresh cycle is in progress. Generated by the IP to refresh Simulation Memory.
MRDY	Memory Ready indicates memory access time is satisfied. Sourced from any memory device on the bus to the IP or the EP.
SA0 to SA3	System address used by the Interface Processor to access Emulator system functions such as the Logic Analyzer trace buffers.
BP1 to BP4	Breakpoint interrupts from the Breakpoint board these signals indicate that a breakpoint has been reached. This signal triggers the EP's sequencer to capture the Emulation Processor.
EPI-	Emulator Personality Interrupt. Generated by the EP hardware to indicate the Emulation Processor has reached a specified point in test memory execution.
MAP-	From the IP or memory board depending on configuration. This signal indicates, when low, that the present bus cycle should reference Simulation Memory.
TME-	Test Memory Enable. A mapping signal used by some EP's to indicate that the address on the bus is a valid test memory address.
SYS	System from the IP. Indicates that the ongoing bus cycle is a system function access. This signal enables recognition by system devices.
BP1OUT- to BP4OUT-	Breakpoint out. This is the breakpoint match pulse from the Breakpoint Board prior to the breakpoint counter conditioning. It is made available to the EP for further processing if required.
BP1IN to BP4IN	Breakpoint in. This is the return of the conditioned BPXOUT signal from the EP back to the Breakpoint Board.
BPINT-	Breakpoint interrupt. From the Breakpoint Board, this signal indicates to the IP that a breakpoint has been reached.
WPROT-	Write-protect. From the write-protect mapper RAM, this signal indicates to the Simulation Memory that the ongoing write cycle should not take effect.

RSTPB-	Reset push button. Active low when the front panel reset button is pressed.
IO00 to IO31	From the IP these I/O lines are defined by the EP board.

4.8 INTERFACE PROCESSOR BOARD

The Interface Processor is the most important board in the Emulator. It is a single board computer by itself, with 16K dynamic RAM, 16K ROM, and a Z80 microprocessor.

The IP board is connected directly to the ADS, through an RS-232C link. All user communication with the Emulator goes through the IP board.

The IP board is responsible for initializing the system, and then responding to the commands given by the user at the ADS. Information is passed to and from the Emulation Processor through the Test Memory on the Emulator Personality board (EP). Each EP board is different, but the basic interaction between the IP board and any EP board is the same.

The IP board is responsible for setting the match conditions into the Breakpoint Board. (Breakpoint Board is described in Section 4.9). Once the match conditions have been set, the IP does not again interact with the Breakpoint Board until it is time to reset the match conditions.

The interaction between the Interface Processor and the Logic Analyzer boards is complex. The IP board is responsible for initializing the Logic Analyzer, for reading out the contents of the Logic Analyzer and for relaying this information to the ADS. (Logic Analyzer board is described in Section 4.10)
The logic sections of the IP board are described in detail in the following paragraphs.

4.8.1. Z80 PROCESSOR

The IP board is a single board computer controlled by a Z80 processor. All of the needed support circuitry for the operation of a Z80 processor has been included.

4.8.2. BUS INTERFACE UNIT

The backplane bus of the Emulator interconnects the target system (via the Probe board and the Emulator Personality board), Simulation Memory, the Breakpoint Board, the Logic Analyzer Board, and the Interface Processor. Either the Emulator Personality board or the Interface Processor board may control the bus. Most of the time, the bus is controlled by the Emulator Personality board. This allows the Emulation Processor to access mapped Simulation Memory with no speed loss for most types of microprocessors.

Occasionally, the Interface Processor requires control of the bus to perform one of two functions:

1. Refresh dynamic simulation memory (if equipped). The dynamic simulation memory refresh function, once enabled by Z80 software on the Interface Processor, is performed automatically by Interface Processor hardware.

2. Allow access to the Emulator bus resources by the Interface Processor's Z80 microprocessor.

The Bus Interface Unit controls the Interface Processor board's access to the Emulator bus. The Bus Interface Unit has five functional divisons:

1. Request and receive permission to control the bus by a handshake sequence with the Emulator Personality board.
2. Allow Interface Processor Z80 software to control the 24 bit Emulator bus address, and additional device selection signals.

3. Provide timing and control for refreshing dynamic simulation memory.

4. Provide timing and control for Z80 read and write cycles to the Emulator bus.

5. Protect system integrity by detecting the condition where bus access, when requested, has not been granted within a reasonable interval (10 microseconds).

4.8.3. EMULATOR BUS ADDRESSING FOR INTERFACE PROCESSOR Z80 ACCESSES

The Z80 microprocessor on the Interface Processor board accesses resources on the Emulator bus by selectively mapping those resources into a 4096 byte window in the Z80's memory address space. This window is located at addresses 4000-4FFF (locations 5000-5FFF, 6000-6FFF, and 7000-7FFF contain identical images of this area due to the decoder implementation). The Z80 software is thus able to manipulate the resources connected to the Emulator bus with fast, efficient instruction sequences.

A total of 29 address and control signals are used to select the Emulator bus resource to be accessed. The first signal, SYS, separates accesses to Simulation Memory from accesses to other devices, such as the breakpoint logic. When SYS is false, Simulation Memory is accessed according to the 24 bit address presented to the Emulator bus by the Bus Interface Unit. When SYS is true, other devices are accessed, the particular device being determined by the System Address signals SA3-SA0, and by the other 24 bus address lines.

The 24 bit bus address is assembled from three components. The least significant 12 bits (BA11-BA00) are taken directly from the least significant 12 bits of the Z80 memory address (within the range 4000-4FFF). The next most significant 8 bits (BA19-BA12) are taken from an 8-bit register. The contents of this register are set when the Z80 writes to its memory address 8000.

The four most significant bits of the bus address (BA23-BA20) are taken from the four least significant bits of another 8-bit register, at Z80 memory address 8008. The most significant four bits of this register are routed to the System Address signals SA3-SA0 mentioned above. Finally, the SYS signal is controlled by the least significant bit of the value written to the register at Z80 memory address 8030. If this bit is zero, SYS will be false, and Simulation Memory will be accessed. If this bit is one, SYS will be true, and the device selected by SA3-SA0 will be accessed.

When the Interface Processor's Z80 software needs to read or write to a device on the Emulator bus, it first presets the high order address bits and System Address by storing into the registers at Z80 addresses 8000 and 8008. It then selects the desired state of SYS by storing into the register at Z80 address 8030. Subsequent accesses to Z80 memory addresses 4000-4FFF will read or write Simulation Memory or device registers.

4.8.4. **EMULATOR BUS ARBITRATION**

Bus arbitration is controlled by two signals: IPWAIT* and EPCYC*. EPCYC* is driven by the Emulator Personality board and is normally asserted, indicating that the Emulator Personality board is in control of the bus. When the Interface Processor board desires a bus access (for either a refresh cycle or a Z80 read or write access within the address range 4000-4FFF), it asserts IPWAIT* and waits for the Emulator Personality board to negate EPCYC*. The Z80 processor's WAIT* signal is also asserted during this waiting period, maintaining the Z80 in wait states until bus access is granted.

The Emulator Personality board, upon seeing the asserted state of IPWAIT*, will, as soon as feasible, place the Emulation Processor in a wait or hold state, then negate EPCYC*. This grants bus control to the Interface Processor, which then negates the Z80 WAIT* and completes the Z80 read or write cycle, interfacing Z80 timing and control to that of the Emulator bus.

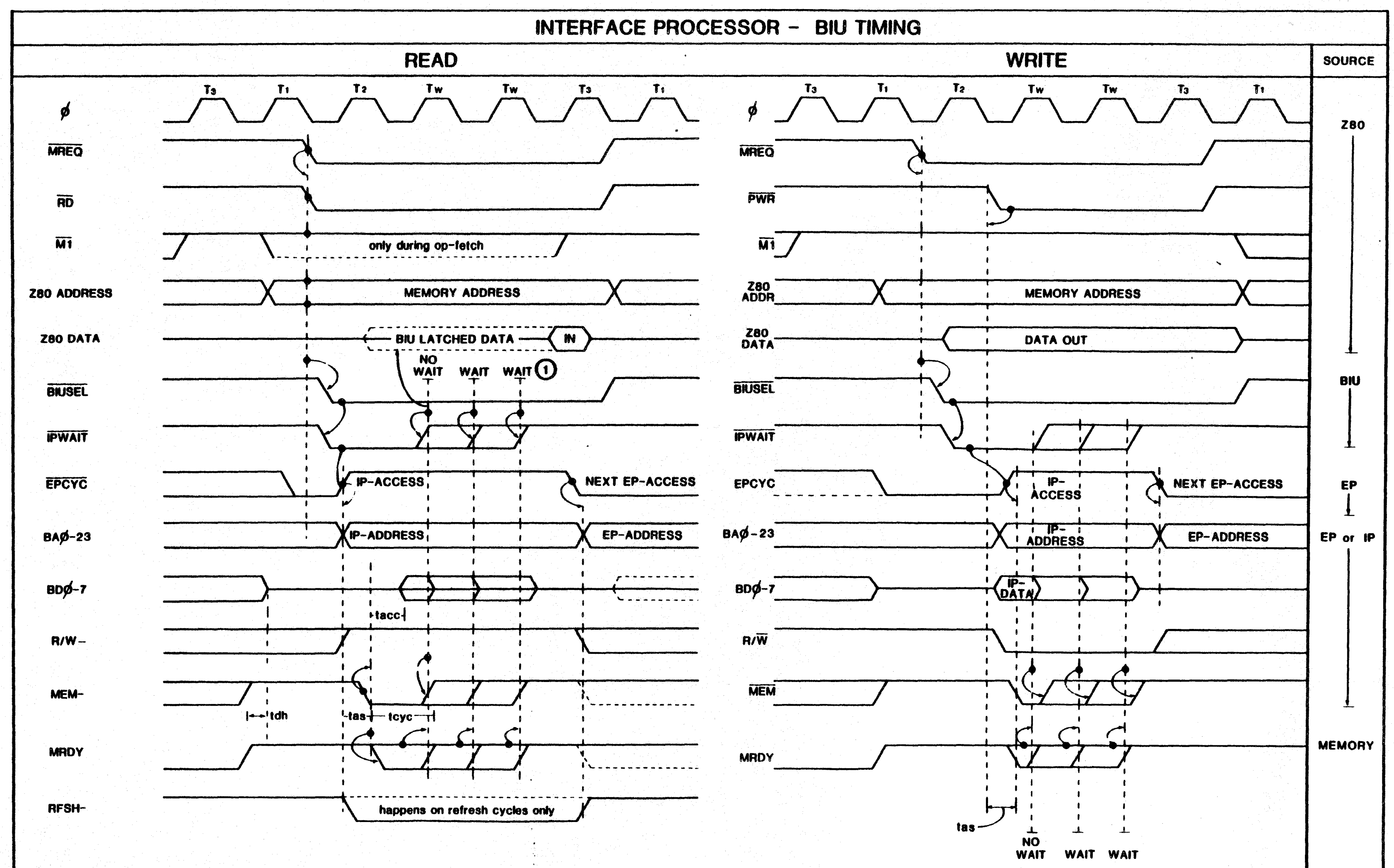
Bus control is arbitrated on a cycle-by-cycle basis. After each read, write, or refresh cycle is completed, the Bus Interface Unit relinquishes control of the Emulator bus to the Emulator Personality board by negating IPWAIT*. The Emulator Personality then reasserts EPCYC* and removes the Emulation Processor from its hold or wait state, allowing it to resume normal operation. This allows breakpoint parameters to be changed and the Logic Analyzer trace buffer to be accessed concurrently with normal operation of the Emulation Processor, while impacting its performance only slightly.

4.8.5. BUS FAULT TIMER

Accesses (read, write, and refresh) by the Interface Processor to the Emulator bus are monitored by the Bus Fault Timer. If the Interface Processor requests access to the Emulator bus, and access is not granted within 10 microseconds by the Emulator Personality board, the Bus Fault Timer triggers an interrupt to the Interface Processor Z80. This interrupt allows the Interface Processor software to report the bus fault condition to the user. This is important not only to avoid unexplained hang-up conditions, but also to warn that the integrity of dynamic simulation memory has been compromised by the lack of timely refresh. The bus fault condition is caused by malfunctioning or improperly connected hardware, either in the target system or within the emulator itself. Hardware changes which can be made on the I.P. to correct the condition where timely dynamic memory refresh fails to occur, because the target system ties up the bus for too long are explained in the Appendix.

4.8.6. DIRECT INTERFACE PROCESSOR ACCESS TO SIMULATION MEMORY

Accesses made by the Interface Processor's Z80 to Simulation Memory are subject to the address mapping previously programmed into the Simulation Memory Block Mappers (see section 4.4.6). Simulation Memories are capable of operating in either a byte or a word mode, and are byte-addressable. Accesses by the Interface processor Z80 are performed in byte mode, with transfers taking place on the lower 8 bits of the 16-bit Emulator data bus. The Interface Processor software, in fact, directly accesses Simulation Memory only during the power on initialization sequence. Subsequent accesses, for purposes of displaying or storing memory contents are made via Test Programs executed on the target processor. This is done to simplify mapped access to external (target system) or Simulation Memory.



SYMBOL	CHARACTERISTIC	MIN	TYP	MAX	UNIT
tacc	BIU Read access (NO WAIT +)	125	150	186	N sec
tas	Address set up time to MEM-	100	120	150	N sec
tdh	Data hold time from MRDY	130	-	-	-

① Two wait states are shown for illustrative purposes only In actuality there can be a maximum of 48 wait states before the IP will abort the BIU access.

Figure 4-4. Interface Processor -BIU Timing

4.8.7 **MEMORY MAPPERS**

There are three memory mappers in the Emulator, all residing on the IP board. Each map controls one attribute of memory.

The mappers are 4K by 1 RAMs connected to the upper 12-bits of the Emulator Personality Board's address space. There is one RAM chip for each of the attributes.

MAP#1	WPROT- (write-protect). This mapper allows the user to protect memory areas from write cycles. This function can also be provided by the memory board.
MAP#2	MAP- (map). This mapper allows the user to choose between internal (active low) and external memory. This function can also be supplied by the memory board.
MAP#3	TME- (test memory enable). This mapper positions the Test Memory in the address space.

The memory mappers are software controlled. The Z80 on the IP board loads the appropriate bit into the mappers. When the corresponding address appears on the Emulator address bus, the three mapper control lines appear on the bus and can be read by the other boards (the SM board or the EP board).

4.8.8 **DYNAMIC RAM REFRESH**

The IP board refreshes both its own dynamic RAM and the dynamic RAM on the Simulation Memory board if the Simulation Memory board is a dynamic memory board, and not the standard static memory board.

Every 224 usecs, a pulse is produced called REFSHON*. This signal causes a BUSRQ* to be sent to the Z80. When the Z80 responds with a BUSAK*, the RMREQ* signal is enabled. This signal, when enabled is a series of pulses approximately 372 nsecs long.

After each set of 16 counts, the BUSRQ* signal is cleared. This ends the refresh cycle until the next RFSHON* signal is produced.

The IP, therefore, performs bursts of 16 row refreshes every 224 usecs. Each row access takes 312.5 nsecs which means each burst of 16 row refreshes takes approximately 8 microseconds.

Once off board refresh has been enabled, every refresh cycle causes the BIU to be activated.

The refresh cycle must wait for the EP to respond with EPCYC* = 1.

The RAMRQ* signal is synchronized with EPCYC* through the WAIT signal which is produced on the IP board by the EPCYC* signal. When a refresh cycle begins and if the EPCYC* signal is high (indicating that the IP board may drive the Emulator bus) off board refresh is enabled by activating a latch which connects the refresh addresses to the Emulator backplane.

4.8.9 **PROMS**

The IP board has four sockets which are used for PROMs. Each socket can be configured to hold either a 2716, a 2732, or a 2532. This allows flexibility in the amount of ROM avaiable for the IP operating system.

The ROM on the IP board appears at addresses 0000 to 3FFF in the Z80 address space. This means the present address paging provides 16K of PROM, of which the IP uses only 12K.

The processor-independent software resides in PROM and is used by the IP to control its internal operations, and its communication with the other boards in the Emulator.

4.8.10 **I/O LINES**

Numbered 0 to 31, these lines are general purpose, and will have different designations with each different EP board. They are written into by software through the Z80 on the IP board, and control hardware of the particular EP board in question.

For example, the 8086 EP board uses I/O 14 as a control line which allows the user to choose either internal or external clock signal. Software sets the bit in the latch corresponding to this port, and the EP board then uses this signal to select either the target system's clock or the EP board's clock.

Each of the ports appears at a specific address location in the IP's address space. They are located in the IP address space at address A100, A101, A102, A103. They are read at A000, A001, A002, and A003 respectively.

4.8.11 **SERIAL PORTS**

The IP board contains two USARTs. The first communicates with the ADS. The second allows another Emulator to be daisy-chained to the first. The two USARTs then are in series allowing the user to communicate with the daisy-chain of Emulators.

4.8.12 **SYSTEM INTERRUPTS**

There are eight separate interrupts to which the IP board must respond. Four of these are concerned with the USARTs on the IP board and are given highest priority. The other four are: TIMOL* (TIMEOUT signal latched); PRTY (PARITY signal from the Simulation Memory board); EPI* (EP interrupt from the EP board); LAI* (Logic Analyzer interrrupt from the Logic Analyzer board). These four interrupts can be disabled by software. TIMOL* and PRTY are ORd together for the RST X`38′ response.

4.8.13 **SYSTEM ADDRESS LATCHES**

Throughout the Emulator there are several sections of circuitry which require direct access by the IP board. These circuits are called system devices.

To access a system device the IP board uses four system address bits (SA0-SA3) and a control signal SYS. These five signals go to the backplane where they can be decoded by the other boards in the Emulator.

The four system address lines are programmed by performing a write to memory address location 8008H with the data bits D4-D7 corresponding to system address bits SA0-SA3 respectively.

Once these signals have been set to the correct value, the Z80 performs an access through the BIU. Some special system devices decode the Emulator backplane address lines as well as the system address lines. For example, the memory mappers are enabled by the correct system access, but the individual bit within the memory mapper is selected by the Emulator backplane address bits.

4.8.14 PIO LATCHES

The 32 PIO lines are set by accesses to the latches which appear in the following address locations:

Address Write only	PIO Line Number (in order corresponding to data bit order D0-D7).
X'A100'	00 - 07
X'A101'	08 - 15
X'A102'	16 - 23
X'A103'	24 - 31

To set the PIO lines, writes are performed to the correct latch with the data bits set accordingly.

The contents of these latches can be read by the Z80 processor by reading the buffers connected to these latches. To find the address needed to read a particular latch, subtract 100H from the latch address and perform a read to this location. This access will turn on both the latch outputs and the buffers which feed this data to the Z80.

4.8.15 BACKPLANE ADDRESS BITS

The BIU drives the lower 12 address bits on the backplane. However, the Emulator backplane contains, 24 possible address bits which are needed by larger microprocessors. To drive these higher order bits, the IP board sets latches which then drive these address bits.

To set backplane address bits BA12-BA19, a write is performed to address 8000H with data bits D0-D7 corresponding to address bits BA12-BA19.

To set backplane address bits BA20-BA23, a write is performed to address 8008H with data bits D0-D3 corresponding to backplane address lines BA20-BA23.

4.9 EXTERNAL PROBE

The External Probe is a simple buffer/driver that provides four inputs and four outputs from the Emulator. The four inputs are user defined signals that are routed to the Logic Analyzer and Breakpoint Boards. They are the external lines of the breakpoint parameters. This gives the user the ability to break on an event occurring externally to the Emulator and the capability to trace external events.

The four outputs are trigger pulses from the Breakpoint Board. Three of them are the triggers of breakpoint 0, 1, and 2 and the fourth one is called SYNC*. They can be passed on to the input of another External Probe to provide synchronous breaking of two or more Emulators in a multiprocessor environment. They can also be used as a complex trigger to give oscilloscopes very flexible triggering capability. **Figure 4-5, SYNC*/Breakpoint 1** shows the relationship between the Breakpoint 1 output and the SYNC* output. If Breakpoint 1's count is set to 1, there will be one SYNC* pulse for every Breakpoint 1 pulse, if Breakpoint 1's count is set to 10000, there will be 10000 SYNC* pulses for one Breakpoint 1 pulse.

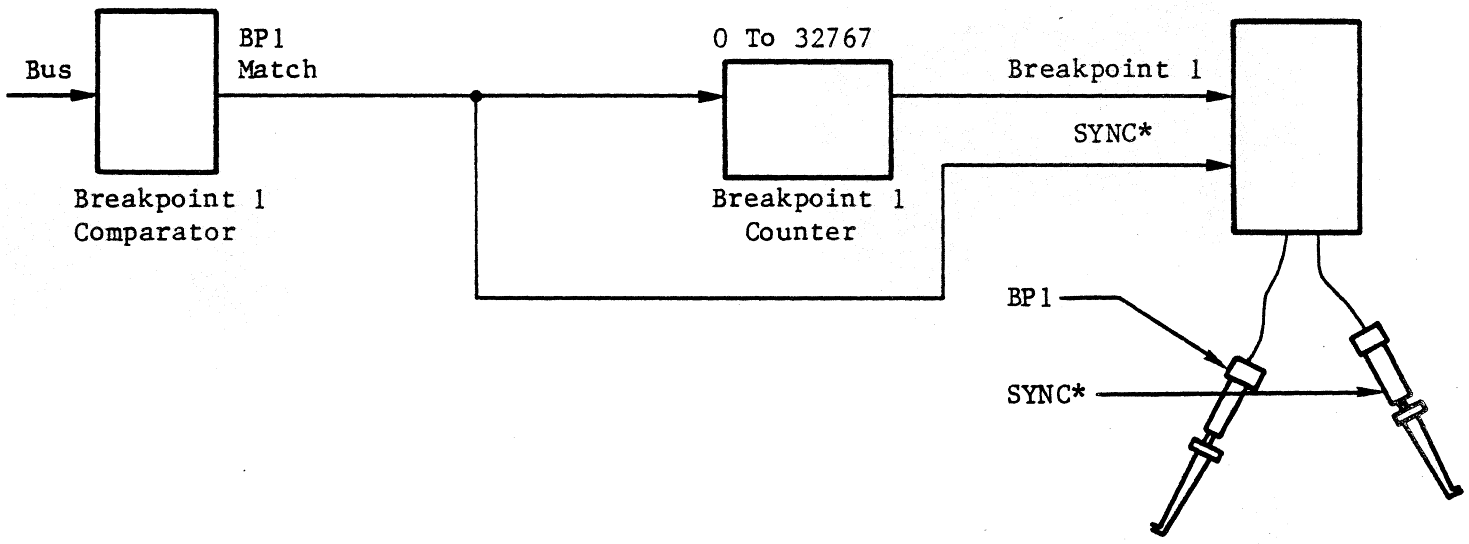


Figure 4-5. SYNC*/Breakpoint 1

The functions of each of the nine color coded probes of the External Probe are given below. All are TTL compatible.

Brown - This input is the least significant bit of the External lines in the breakpoint parameters. It is designated EC0 in the Logic Analyzer display.

Red - This input is the second least significant bit of the External lines in the breakpoint parameters. It is designated EC1 in the Logic Analyzer display.

Orange - This input is the third least significant bit of the External lines in the breakpoint parameters. It is designated EC2 in the Logic Analyzer display.

Yellow - This input is the most significant bit of the External lines in the breakpoint parameters. It is designated EC3 in the Logic Analyzer display.

The following four outputs are active low, and will generate a 20ms pulse when the designated trigger is reached.

Green	- Trigger for breakpoint 0.
Blue	- Trigger for breakpoint 1.
Violet	- Trigger for breakpoint 2.
Gray	- Trigger for breakpoint 1's match condition as described in **Figure** 4-5.
Black	- Ground.

The External Probe is connected to a 24-pin, "D" connector on the back panel of the Emulator. Inside the Emulator, the probe is connected to the Breakpoint Board, and if present, to the optional Logic Analyzer. The Probe has a six foot ribbon cable separating it from the Emulator.

The External Probe allows the user to define four external conditions on which the four breakpoints can be set. The first four clips, color coded brown, red, orange, and yellow respectively, can be used to probe four points on the user's target system. These four probes are monitored by each of the four Emulator breakpoints and can be set in the breakpoint dialog for a break on any combination of "1", "0", or "X" (don't care).

The user has the option to record the functions of these first four probes in the Logic Analyzer, and can display them along with other microprocessor functions.

The next three probes, green, blue, and violet respectively, represent the first three breakpoints (breakpoints 0 to 2). These probes are normally high (+4v). When a breakpoint is reached, the associated probe will go low (.8v). It will remain low for about 2ms. These probes are used to pass on the breakpoints to another Emulator for multiprocessor emulation or as a complex trigger for other instrumentation.

The eighth probe (grey), is a synchronization pulse whose source is the match condition of breakpoint 1. This means anytime a breakpoint match is reached by breakpoint 1, a 20ns pulse appears on the grey probe. By setting breakpoint 1's count to the maximum and putting it in the snapshot mode, the user can obtain a complex trigger to synchronize an oscilloscope or other instruments.

The LED in the probe box is lit when the Emulator is running emulation, and has one or more active breakpoints. The LED is extinguished when the emulation is halted or no breakpoints are set. The black probe is ground.

4.10 BREAKPOINT BOARD

This section familiarizes the reader with the Breakpoint Board. A brief general description of the Board's purpose and function is followed by detailed explanations of its functions.

4.10.1 GENERAL DESCRIPTION OF THE BREAKPOINT BOARD

The Breakpoint Board allows the user to set up one to four breakpoints at any point in the software. These breakpoints allow the software to be halted or interrupted briefly for information collection, while running in a full speed, real-time regime.

Breakpoints can be programmed to trigger on a number of specific conditions: the placing of a specific address or data pattern on the target system bus, an instruction or data fetch, a read or write cycle by the Emulation Processor, a memory or I/O reference by the Emulation Processor, or the occurrence of a specific pattern on four external lines. These four lines allow the user to monitor events of interest on the target system. Breakpoint conditions can be either combined into a compound condition, or used singly. In addition, the user may specify a certain condition as a "don't care" condition, and that condition is ignored.

A breakpoint can be set to halt the emulation processor, allowing the user to request additional information about the state of his program at that point in its execution, or to update the "snapshot" of the data currently displayed on the ADS, with only a brief period in which the real-time behavior of the program is degraded.

Additional qualification for the triggering of breakpoints is possible. A breakpoint may be set so that its conditions must be fulfilled a certain number of times before any action is taken. The group of four breakpoints can be ANDed or ORed together, so that all the individiual breakpoint conditions must be fulfilled in order, from the lowest active breakpoint to the highest, before any action is taken.

Breakpoints may also be used to trigger the Slave Logic Analyzer. This device allows the user to record all transactions that take place on the target system bus, with a recording occurring every bus cycle. When the Logic Analyzer is enabled with the Mode command, a breakpoint will not interrupt the emulation processor. Instead, the Logic Analyzer is signaled to begin recording the bus data. In this case, the fourth breakpoint (breakpoint 3) cannot be used, since the Logic Analyzer uses it as a trace qualifier for selecting the kind of information to be recorded.

The functions of the Breakpoint Board will now be described in detail.

4.10.2 **REGISTERS AND PORTS**

There are two registers and 14 ports on the Breakpoint board. They are accessed using seven system addresses, 8H through EH. To utilize these registers and ports, the desired system address is placed in the system address register at 8008H on the IP. Only the upper four-bits apply to the system address. The lower four are used as BA20-23. See **Figure 4-6, System Address Register.**

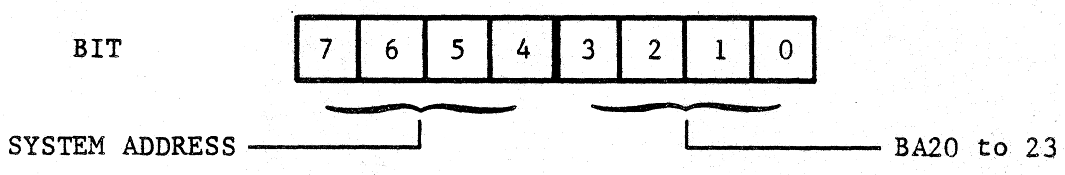


Figure 4-6. System Address Register

The system address is enabled by setting bit 0 of the system enable register 8030H on the IP. Once enabled, operation can be performed on the registers and ports. Read and write cycles perform unrelated functions and caution must be used in their use.

The following six IP system addresses give access to set the breakpoint parameters in RAM. Each system address gives access to up to 16 parameters for all four breakpoints simultaneously. Parameters are "set" by placing a zero in the location in RAM equal to the parameters on which the break is desired. Following are the ports, the parameters they access, and the address lines to be used for the port. These ports cannot be read. Reading with these system addresses active will return the breakpoint status register (BSR). The breakpoint status register contains the status of each of the four breakpoints. It indicates which breakpoint was reached during the last breakpoint interrupt. The BSR is shown in **Figure 4-7, Breakpoint Status Register.**

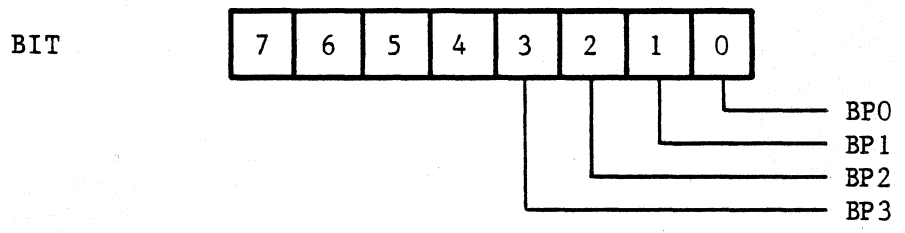


Figure 4-7. Breakpoint Status Register

Figure 4-8, Address And Data Lines shows how the address and data lines are used to set the breakpoint in RAM. Shown below are the addresses used, and what they represent for each port.

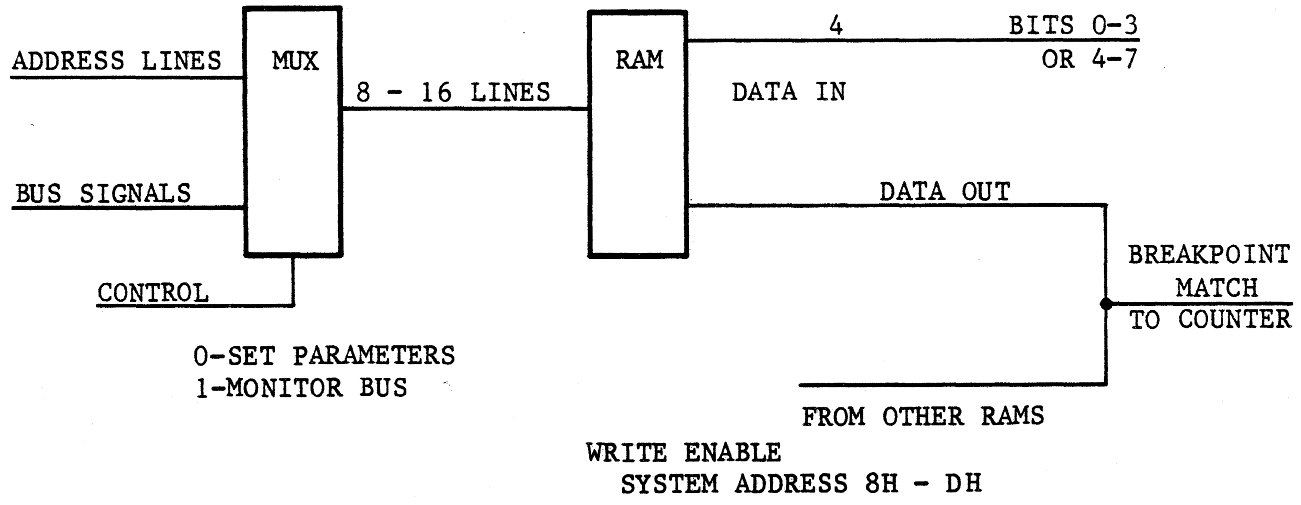


Figure 4-8. Address And Data Lines

4.10.3 SYSTEM ADDRESSES

8H: Bus addresses used: 0-15
Represents: Themselves for "equal to" breakpoint
Data bits 0-3 represent breakpoints 0-3 respectively for BA8-15
Data bits 4-7 represent breakpoints 0-3 respectively for BA0-7

9H: Bus addresses used: 8-23
Represent: Themselves: 8-15 for >< breakpoint
16-23 for = breakpoint
Data bits 0-3 represent breakpoints 0-3 respectively for BA8-15
Data bits 4-7 represent breakpoints 0-3 respectively for BA16-23

AH: Bus addresses used: 16-23
Represent: Themselves for >< breakpoint
Data bits 4-7 represent breakpoints 0-3 respectively

BH: Bus addresses used/represent:

BA4	BHE
BA5	IO-
BA6	R/W-
BA7	MEM-
BA8	DATA BIT 0
BA9	DATA BIT 1
BA10	DATA BIT 2
BA11	DATA BIT 3

Data bits 0-3 represent breakpoints 0-3 respectively for BHE, IO-, R/W-, MEM-
Data bits 4-7 represent breakpoints 0-3 respectively for data bits 0-3

CH: Bus addresses used: 12-15, 20-23 represent data bits 4-7, 12-15
Data bits 0-3 represent breakpoints 0-3 respectively for data bits 12-15
Data bits 4-7 represent breakpoints 0-3 respectively for data bits 4-7

DH: Addresses used: 0-3, 16-19
Represent external data bits 8-11, conditions 0-3
Data bits 0-3 represent breakpoints 0-3 respectively for EC0-3
Data bits 4-7 represent breakpoints 0-3 respectively for data bits 8-11

System address EH is to be used for write cycles only. Read cycles are unpredictable. With system address EH enabled and address bit 3 set low, a write cycle to the BIU will load the breakpoint mode register. The mode parameter is defined in **Figure 4-9, Breakpoint Mode Register.**

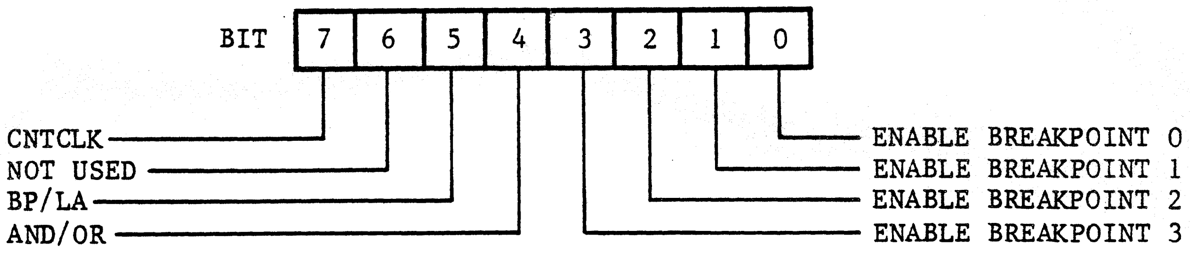


Figure 4-9. Breakpoint Mode Register

If the enable bit is set to zero, the breakpoints continue to function normally, and will cause Logic Analyzer interrupts in the normal manner, but do not allow the breakpoint to be passed on to the bus. The enable bits, when set to 1, allow the breakpoints to be passed on to the bus, causing Emulator interrupts.

The AND/OR bit controls whether breakpoints 0-2 are ANDed or ORed together.

BP/LA sets the Breakpoint Boards mode to either (1) breakpoint or (0) Logic Analyzer.

Bit 6 of the mode register is not used, and must be set to 1.

CNTCLK is used to clock the state of I06 onto the Breakpoint Board. To do this, I06 is set to the desired state. Then set CNTCLK to zero, then back to 1.

With the system address EH enabled, writing to the addresses indicated below, loads the breakpoint counters as indicated.

Address	
4008	LSB of BP0 Counter
4009	MSB of BP0 Counter
400A	LSB of BP1 Counter
400B	MSB of BP1 Counter
400C	LSB of BP2 Counter
400D	MSB of BP2 Counter
400E	LSB of BP3 Counter
400F	MSB of BP3 Counter

I06 (Interface Processor I/O, bit 6) is used to enable/disable the breakpoint counters and disable the Logic Analyzer trace qualifier. This, in effect, freezes the state of the breakpoints and Logic Analyzer memory.

When the IP is loading the breakpoint counters, the counters are disabled until they are ready for use. The breakpoints also are frozen when using the timer/counter on the Logic Analyzer, since they directly control the enable/disable function of the counter. The trace qualifier is frozen to prevent memory from being altered on the Logic Analyzer once a trace is complete.

Setting I06 to 1 will automatically disable the counters and trace qualifier. When an Interface Processor breakpoint interrupt occurs, it automatically disables the counters and trace qualifier.

To enable the counters and trace qualifier, I06 is set to zero and clocked in.

4.10.4 OPERATION OF BREAKPOINT RAMs

There are 49 parameters on which the breakpoint is set. Forty-eight (48) of these are defined in the breakpoint RAMs. The delay count is set in the counter. See **Figure 4-8, Address And Data Lines.** When an Emulator bus cycle occurs, the RAMs are directly addressed by the lines they are to monitor. When the lines equal the breakpoint parameters, a zero is output from the RAM. When all 48 lines match, the breakpoint counter is incremented. To set the RAMs for a breakpoint, the RAMs are set to zero at the address equal to the desired breakpoint parameters for each RAM.

Example

Breakpoint 0 is desired on address 2F80H when the four external condition bits from the External Probe are 0101 or 5H. System address 8H gives access to the bus addresses 0-15. Enabling 8H, ones are written to all locations except 2F80H. At 2F80H, BP0 is desired. Since data bits 0 and 4 represent BP0 in this RAM, 11101110 or EEH is written in to 2F80H.

DH gives access to the external conditions RAM system address DH is enabled and all addresses are filled with ones except address 5H. Since data bit 0 represents breakpoint 0, 11111110 or FEH is written into 5H. All other RAMs at all locations are filled with ones.

In addition to being able to set a breakpoint on a given set of parameters, the board has the ability to set breakpoints for addresses greater than or less than a specified address. This is done using two additional breakpoint RAMs which monitor address bits 8-23. **Figure 4-10, Breakpoint Address RAMs** shows the breakpoint address RAMs for 1 breakpoint in the greater than or equal to configuration. **Figure 4-11, Breakpoint Address RAMs** shows a less than or equal to configuration. It can be seen from these examples that the equal to RAMs are set to the equal to value and the greater than and less than RAMs are set such that all values greater than or less than equal to value are set to zero as required.

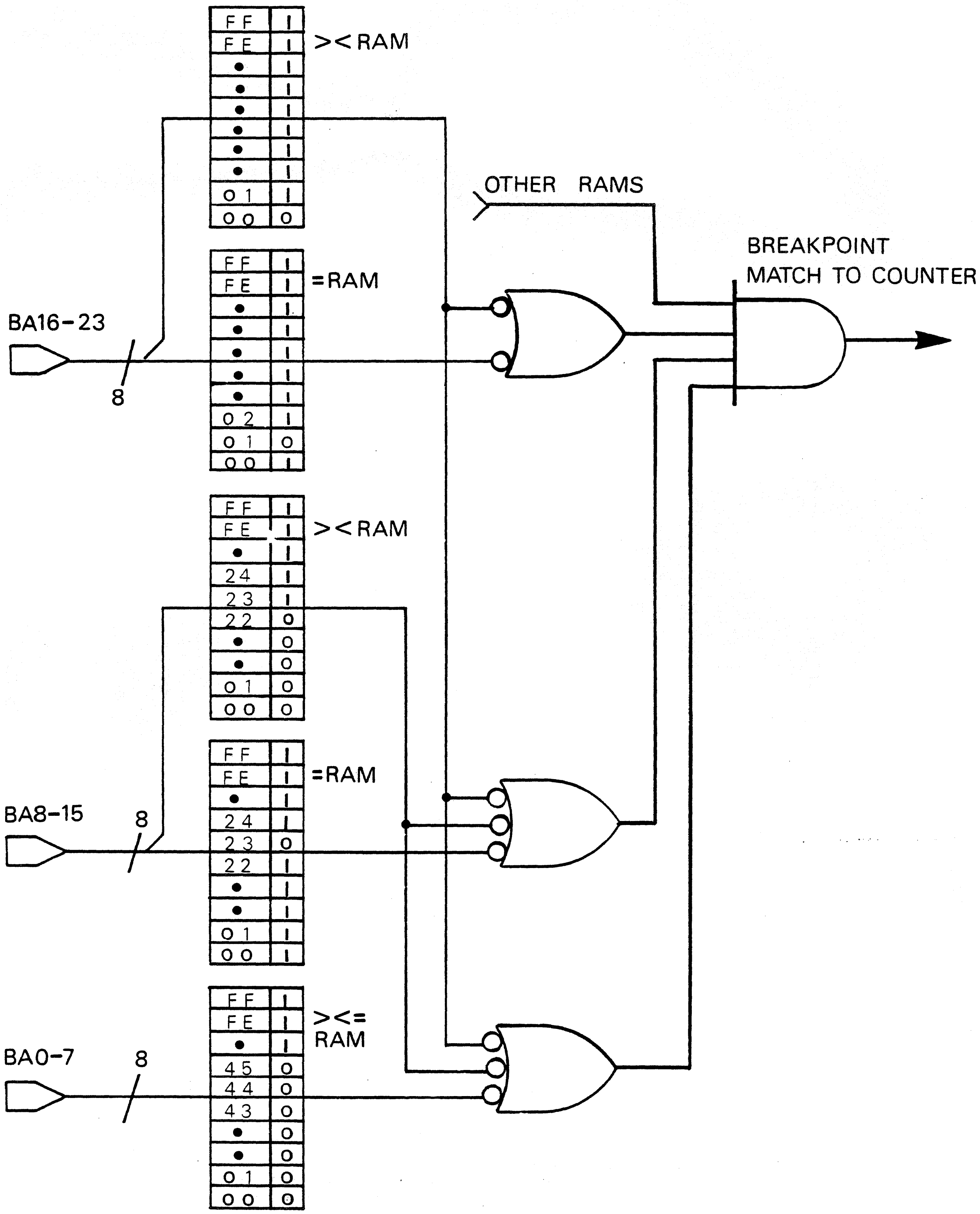


Figure 4-10. Breakpoint Address RAMs
(Greater Than Or Equal To)

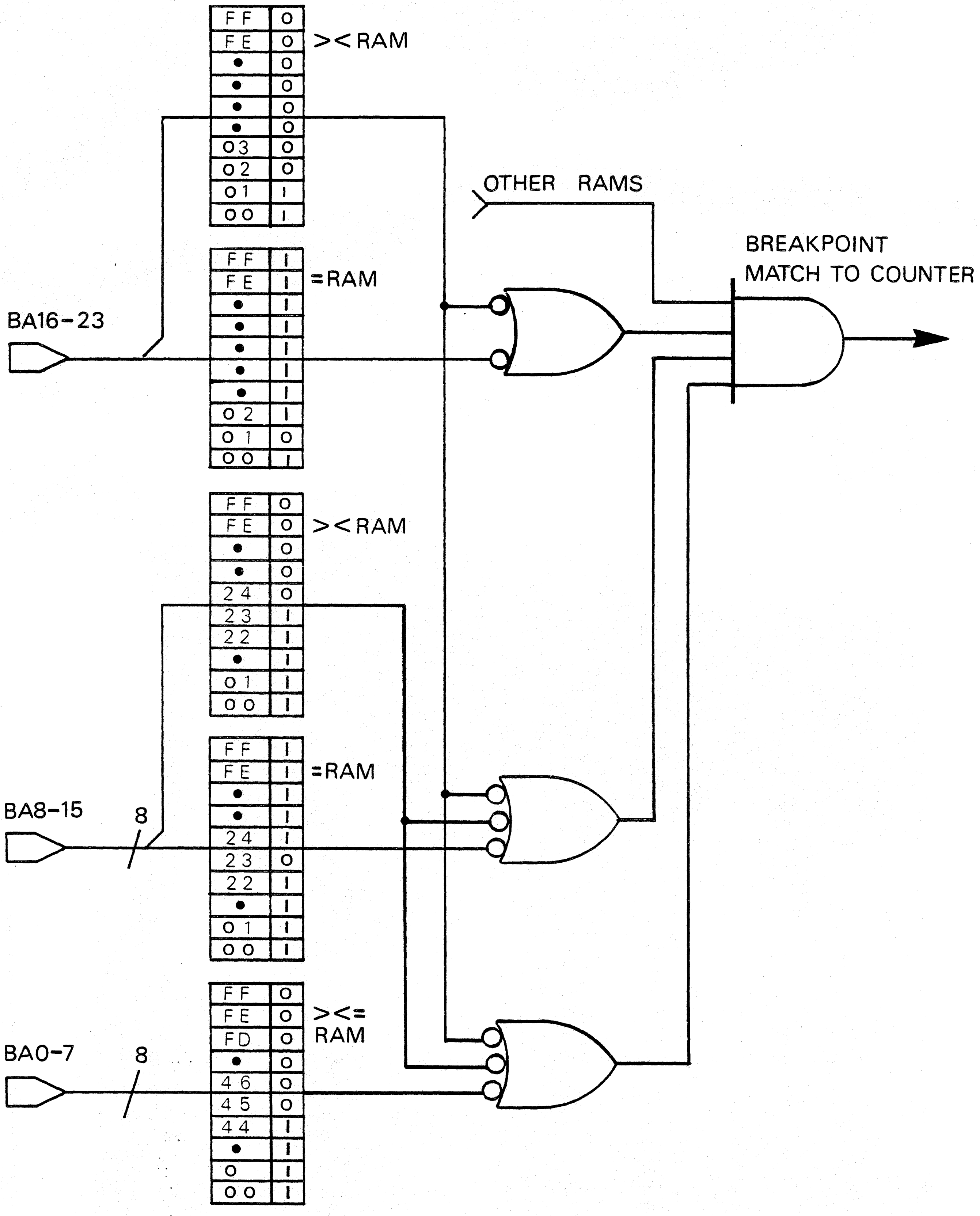


Figure 4-11. Breakpoint Address RAMs
(Less Than Or Equal To)

4.10.5 **BREAKPOINT COUNTERS**

The breakpoint counters allow the user to pass through a breakpoint up to 8000H times before the breakpoint interrupts. When the breakpoint parameters are met, the counter may be incremented.

When the counter reaches 8000H, the breakpoint has been reached, and if enabled it will interrupt.

To set the counter for n passes through the breakpoint, the value 8000H minus n is loaded into the counter.

4.10.6 **BREAKPOINT MODE REGISTER**

The only function of the enable bit is to allow the breakpoint to be passed on to the bus, which may generate an Emulation Processor interrupt. All other functions of the breakpoint continue to operate, regardless of the state of the enable bits.

Figure 4-12, Breakpoint Control Logic shows the relationships between the mode register and the breakpoints. BP/LA and AND/OR work together, defining how the breakpoints, breakpoint counters, and breakpoint interrupts to the interface processor function.

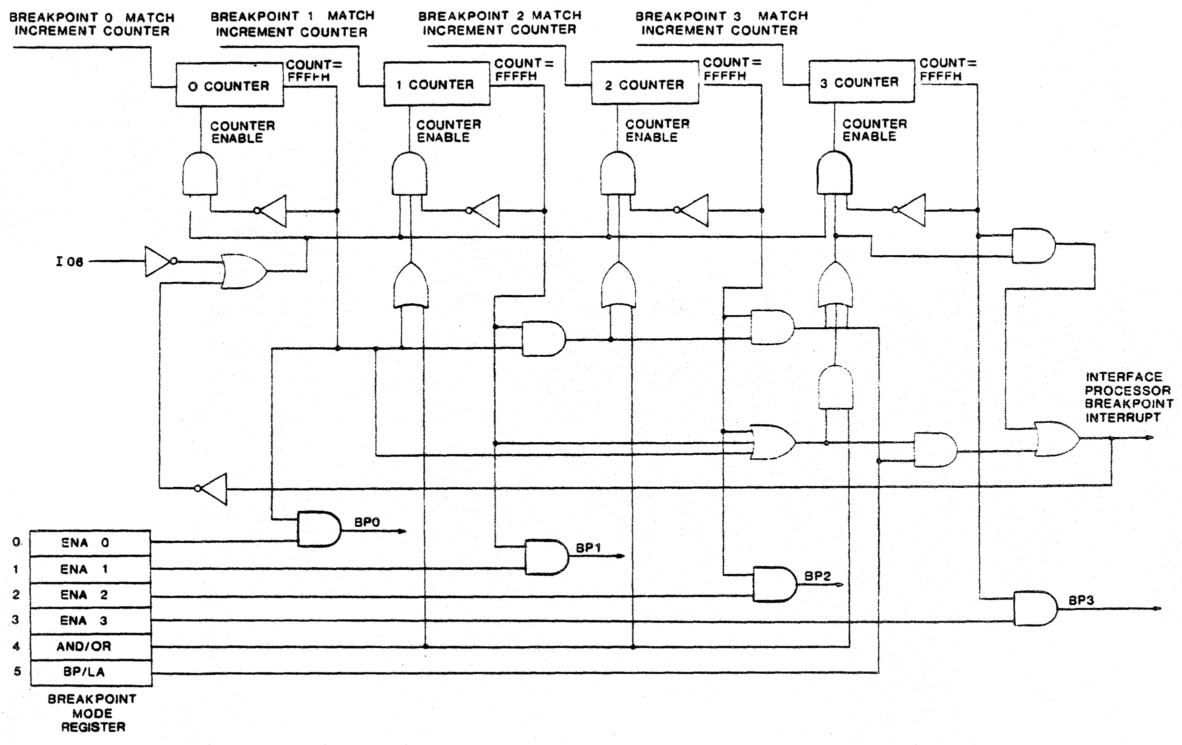

BREAKPOINT 0 MATCH INCREMENT COUNTER
BREAKPOINT 1 MATCH INCREMENT COUNTER
BREAKPOINT 2 MATCH INCREMENT COUNTER
BREAKPOINT 3 MATCH INCREMENT COUNTER
0 COUNTER
1 COUNTER
2 COUNTER
3 COUNTER
COUNT= FFFFH
COUNTER ENABLE
I 06
INTERFACE PROCESSOR BREAKPOINT INTERRUPT
BP0
BP1
BP2
BP3
0 ENA 0
1 ENA 1
2 ENA 2
3 ENA 3
4 AND/OR
5 BP/LA
BREAKPOINT MODE REGISTER

Figure 4-12. Breakpoint Control Logic

If ANDing is selected (bit 4 of mode register not set), breakpoint 3's counter will not begin counting until breakpoint 2 has been reached. Breakpoint 2's counter will not be enabled until breakpoint 1 has been reached. Breakpoint 1's counter will not be enabled until breakpoint 0 has been reached. Breakpoint 0's counter will be enabled by IO6. Each breakpoint's requirements must be reached in succession; first breakpoint 0, then breakpoint 1, and so on. Each breakpoint, if enabled, will cause the Emulation Processor to be interrupted and forced to execute Test Memory programs. In order to use the AND function, only the last breakpoint in the succession is enabled. Example:

> If BP0 and BP1 are to be ANDed, bit 4 is not set, and only BP1's enable bit is set in the mode register. If BP0's enable bit were set, an interrupt would occur before breakpoint 1 was reached.

The following are definitions of how the breakpoints interact with BP/LA and AND/OR.

Breakpoint 0 - unaffected. Counter is always enabled if IO6 is clocked to zero.

Breakpoint 1 - counter inhibited until BP0 is reached if AND/OR is zero. Not affected by BP/LA.

Breakpoint 2 - counter inhibited until BP1 is reached if AND/OR is zero. Not affected by BP/LA.

Breakpoint 3 - Counter enabled when:

a. BP/LA is set, regardless of other breakpoints or AND/OR.

b. Breakpoint 0, 1 and 2 have been reached regardless of BP/LA or AND/OR.

c. AND/OR is set and breakpoint 0 or 1 or 2 is reached.

4.10.7 INTERFACE PROCESSOR BREAKPOINT INTERRUPT

The Interface Processor breakpoint interrupt will occur when:

1. BP/LA is set and breakpoint 0 or 1 or 2 or 3 is reached.
2. Breakpoint 3 and 2 and 1 and 0 are reached.
3. AND/OR is set and breakpoint 3 is reached and breakpoint 0 or 1 or 2 is reached.

4.11 LOGIC ANALYZER

The Logic Analyzer board is an optional feature whose main function is to record bus signaling during emulation, beginning at a specific event which is predetermined on the Breakpoint Board. Sixty-four (64) signals may be recorded. They are:

- 24 Bus Addresses (BA0-23)
- 16 Bus Data (BD0-15)
- 4 Control Lines (MEM, R/W, I/O, BHE)
- 16 Optional Inputs
- 4 External Conditions (from External Probe)

The Analyzer has the ability to trigger on three sets of breakpoint conditions which can be ANDed/ORed. The trace can also be pretriggered or delayed and a set of parameters can be given for what is to be traced (defined by Breakpoint 3). Once captured, the data is read through eight-bit buffers to the Interface Processor for processing.

24 Address Lines	- Of course, not all microprocessors use all of these lines. Only those used are displayed on the screen.
16 Data Lines	- Address and data are demultiplexed from the microprocessor if necessary.
4 Control Lines	- These are standard Emulator control lines. They may be used or substituted with another signal depending on the processor.
16 Optional Inputs	- Up to 16 processor dependent signals can be traced by the Logic Analyzer. These come from the Emulator Personality Board.
4 External Conditions	- From the External Probe. The Logic Analyzer has the capacity to record 256 traces. Once the breakpoints and trace qualifier (breakpoint 3) have been set, and emulation begun, all cycles are latched onto the Logic Analyzer using LAS*. (See **Section 4.7, Emulator Bus** for Bus signal descriptions.) The Breakpoint Board then waits for the trigger to occur. The trigger is the current breakpoint definition except for breakpoint 3 which is disabled, and used as the trace qualifier. At the same time the trace qualifier from the Breakpoint Board determines if the trace should be recorded in the buffer. The trace qualifier signal from the Breakpoint Board, TQ-, is passed to the Logic Analyzer through P3, the front edge connector between the Logic Analyzer and Breakpoint boards. The RAM on the Logic Analyzer acts as a circular buffer and continues to record until the trigger is reached.

The trace qualifier then counts down the post-trigger delay count specified and stops tracing when the count is reached. This generates an interrupt, BPINT-, to the IP. The IP then reads the trace from the Analyzer for display on the CRT.

On board is a 24-bit counter capable of recording time or bus cycles elapsed between two events. The two events are detected by breakpoint 1 and 2. When breakpoint 1 occurs, the counting begins and is halted by breakpoint 2. The counter runs in three modes; microseconds, 100 nanoseconds, or Emulator bus cycles. The results are read from the Analyzer through three 8-bit buffers.

4.11.1 **ANALYZER OPERATION**

All Logic Analyzer access is performed with system address 7H at 8008H. As with other system devices, bit 0 of 8030H must be set to access the Logic Analyzer. Any other procedure may modify the Logic Analyzer during operation.

Registers And Ports

With the Analyzer selected, the following addresses perform the functions indicated. Read and write cycles to the specified address perform completely separate functions. Caution must be used in accessing these.

The following eight addresses read bus data stored in the Logic Analyzer at the location pointed to by the address pointer. Read at:

4003	Bus address 0 through 7 (BA0-7)
4013	Bus address 8 through 15 (BA8-15)
4023	Bus address 16 through 23 (BA16-23)
4033	Bus data 0 through 7 (BD0-7)
4043	Bus data 8 through 15 (BD8-15)
4053	Bit 0 - Memory cycle (MEM) active low
	Bit 1 - Read/Write (R/W) low equals write
	Bit 2 - Input/Output (I/O) active low
	Bit 3 - Byte high enable (BHE) active high
	Bit 4-7 - Four least significant bits of 16 optional inputs
4063	Mid byte of 16 optional inputs
4073	Bit 0-3 - Four most significant bits of 16 optional inputs
	Bit 4-7 - Four external condition inputs

4103H reads address Overflow Status Register. Within the Logic Analyzer is an 8-bit address pointer used to point to the next location trace data is to be recorded. When the address pointer wraps around from FFH to 0, bit [illegible] of 4103H is set (1).

4113H reads the eight least significant bits of the 24-bit time/cycle counter (Ref. 2.2).

4123H reads the most significant byte of the 24- bit time/cycle counter (Ref. 2.2).

4133H reads the most significant byte of the 24-bit time/cycle counter (Ref. 2.2).

Write at:

4003 - 4013	These have no function. They will not affect the system.
4103H	Increments the address pointer. Data written is insignificant.
4113H	Clears the address pointer. Data written is insignificant.
4123H	24-bit time/cycle counter mode register. The time/cycle counter has three operating modes. These modes are set by writing the following to this register. 0 - 1 microsecond counter 20H - 100 nanoseconds counter 40H - Bus cycle counter
4133H	Clears the 24-bit time/cycle counter. Data written is insignificant.

4.11.2 EXTERNAL CONTROLS

This section describes external controls, not located directly on the Logic Analyzer, which control its operation.

Logic Analyzer Strobe (LAS) originates from the Emulator Personality board. Its falling edge should be synchronous with valid address, data, and control lines on the bus and should occur only once per Emulation Processor cycle.

Trace qualifier originates from the Breakpoint Board. A bus cycle is recorded on every occurrence of TQ. TQ is enabled by LAS, and bit 7 of system address E0 and will occur when all parameters (except the count) of breakpoint 3 are met. Breakpoint 3's counter determines the trigger's position in the field of 256 traces.

Breakpoints 0-2 are used as the trigger for breakpoint 3's counter.

4.11.3 **LOGIC ANALYZER INITIALIZATION**

Logic Analyzer initialization requires only that the address pointer be cleared. The address pointer is used to point to the address at which the bus cycle is to be recorded. After each trace qualifier, the address counter is auto-incremented, ready for the next cycle. The only software control is to clear it, increment it, or read the overflow status.

The trace qualifier is initialized by setting breakpoint 3's parameters, bit 7 of system address E0H and LAS.

The trigger (breakpoints 0-2) is initialized as required per the Breakpoint Board specification. When these requirements are met, breakpoint 3's counter is enabled. Breakpoint 3's counter determines the end of the trace.

The Analyzer is initialized when breakpoints have been set.

Data Capture

Once initialization is complete and the Analyzer is armed, tracing is automatic. The tracing is complete, and frozen in memory, when breakpoint 3 occurs and LAS is disabled.

Data Retrieval

Once the trace is complete and breakpoint 3 interrupt is complete, the trace qualifier is immediately disabled by setting IO6 high and/or setting bit 7 of system address B0H low.

Data is retrieved from the Analyzer by selecting the Analyzer on the system bus and reading the data buffers 4003H to 4073H. When reading the buffers, the address pointer is auto-incremented.

4.11.4 **TIME/CYCLE COUNTER OPERATION**

The time/cycle counter is 24-bits wide, and counts the time in microseconds or 100 nanosecond increments or bus cycles, which occur between breakpoint 1 and breakpoint 2.

Initialization of the counter is accomplished by clearing the counter by writing to 4113H when the Analyzer is enabled.

The mode of the counter is selected by setting the mode register at 4123H. This register cannot be read. The following describes the mode writes to 4123H:

0	- 1 microsecond counter
20H	- 100 nanosecond counter
40H	- Bus cycle counter

In the bus cycle mode the count will equal all bus cycles executed by the Emulation Processor. The timer is armed by enabling breakpoints 1 and 2. When breakpoint 1 occurs, the count will begin. When breakpoint 2 occurs, the count will halt. The counter is read in three registers. With the Analyzer enabled reading:

4113H	- Least significant byte
4123H	- Mid byte
4133H	- Most significant byte

4.12 <u>SIMULATION MEMORY</u>

<u>26-Bit (32K or 16K) Static Memory Board</u>

The name "26-Bit Static Memory Board" is a slight misnomer in this case as the "26-Bit" refers to the 64-megabyte mapping range (or positioning) of all or part of the 32K on board memory within this mapping range and not the quantity of memory on each board. The memory board is configurable as a maximum of four 8K-byte blocks. Each 8K-byte block of memory is relocatable (mappable) anywhere in a 64-megabyte address range. Write-protect and internal/external mapping is provided on 256-byte address boundaries within this address range on all MAPPED memory. Memory can be reconfigured as words or bytes with odd parity checking on all read cycles. Each memory board can be strapped as one of four possible memory boards in a multiple memory board system. In addition, each memory board is strappable for the old configuration static memory board (1-megabyte mapping range).

4.12.1 BLOCK MAPPERS

The basic function of the block mappers is to generate the chip selects for the 2147 memory array. The block mappers are software programmable as a system device (i.e., system address 3X). From a software point of view, the block mappers appear as a 256 X 8 read/write memory with 14 address inputs (A12 through A25). This is necessary to generate a mapper with a mapping range of 64-megabytes. That is, in order to generate a mapping range of 64-megabytes in 4K-byte increments, a 16K mapper is required (16K blocks X 4K-bytes/block = 64-megabyte). By using four 256 X 4 RAMs and 8 OR gates, the equivalent of a 16K X 8 mapper is generated. The block mappers are programmed by writing binary zeros into those bit locations where mapped memory is desired on 4K-byte block boundaries. Because of the HIGH BYTE - LOW BYTE configuration of memory array, it is necessary to write two binary zeros (in two adjacent bit locations) into each desired location of the block mapper to map 4K-bytes. To map an 8K block, the same pattern is repeated twice. As an example, the following block mapper pattern would appear in the first eight locations of the block mapper in order to map 32K of memory from address X'0000' to X'7FFF'. All subsequent block mapper locations would contain X'FF' (deasserted form).

BA25 - BA12 (address inputs to mapper)	Data Pattern
X'00'	X'FC'
X'01'	X'FC'
X'02'	X'F3'
X'03'	X'F3'
X'04'	X'CF'
X'05'	X'CF'
X'06'	X'3F'
X'07'	X'3F'

The write-protect and internal/external mappers are a direct extension of the block mappers. That is, the block mappers decide where each 8K block of Simulation Memory is mapped and the write-protect and internal/external mappers decide if each 256-byte block of memory within this mapped block of Simulation Memory is write-protected or internally or externally mapped. This is done by encoding the eight block mapper outputs to generate two additional address lines to be qualified with BA8 through BA12 as address inputs to the write-protect and internal/external mappers.

Like the block mappers, the write-protect and internal/external mappers are programmed as a system device (i.e., system address FX). The write-protect mapper is written by BD0 and the internal/external mapper is written by BD1. BD4 and BD5 must be set to zero in order to write to these mappers. All other data bits are don't cares.

4.12.2 WORD/BYTE MEMORY CONFIGURATION

Simulation memory is configurable as words or bytes by the following buffer control logic.

BYTE	BHE	A0	OPERATION
0	1	0	word
0	1	1	high byte (odd byte)
0	0	0	low byte (even byte)
0	0	1	no operation
1	x	0	even byte (8-bit processor)
1	x	1	odd byte (8-bit processor)

where x = don't care

MEM- is the memory control signal which validates any Simulation Memory cycle.

4.13 DYNAMIC MEMORY

The 128K dynamic memory board contains four 32K blocks of memory which can be mapped anywhere within a maximum address range of 64-megabytes. These memory blocks are configurable as words (16-bits) or bytes (8-bits) by asserting the proper combination of signals A0, BHE, and BYTE. Odd parity checking is provided for the purpose of an interface processor memory diagnostic after Emulator power-up. In this way, defective block of memory will not be offered as available Simulation Memory. Write-protection and internal/external memory mapping is provided on 256-byte boundaries on all MAPPED Simulation Memory. Each memory board is strappable as one of eight memory boards in a multiple Simulation Memory board configuration. In addition, each memory board is strappable as a 20-bit memory board (1-megabyte mapping range) as well as the 26-bit memory.

4.13.1 BLOCK MAPPERS

The block mapper's sole function is to select 4K blocks of memory by decoding bus addresses A12 through A25 for the 26-bit memory configuration or A12 through A19 for the 20-bit memory configuration. Jumper configurations for 20-bit, or 26-bit memory selection is described in each of the microprocessor specific Supplements. Decoding is accomplished by programming 256 X 8 RAMs with the desired decoding pattern. This decoding pattern then generates the proper column address strobe (CAS) signals to the memory array.

4.13.2 MEMORY ARRAY

The memory array is composed of four rows of 16K X 1 dynamic memory chips of type 4116. Each row is composed of 18 memory chips with 2 chips used for parity checking (one parity chip for each 8-bits of memory). Bus addresses A1 through A14 are multiplexed to each memory chip. Memory is divided into an even bank (A0=0) and an odd bank (A0=1). This means that half the memory is reserved for even byte accesses and the other half is reserved for odd byte accesses. All word accesses will use both even and odd banks. What this all means is that buffer control logic through the use of A0, BHE, and BYTE is used to channel the memory data to or from the appropriate data bus (D0-D7 or D8-D15 or both). See **Table 4-1. Buffer Control** for more details.

Table 4-1. Buffer Control

BYTE	BHE	A0	OPERATION
0	0	0	low byte
0	0	1	no operation
0	1	0	word
0	1	1	high byte
1	x	0	even byte (8-bit operation)
1	x	1	odd byte (8-bit operation)

4.13.3 CONTROL LOGIC

Control Logic

1. RAS and CAS generation. MEM- (an externally generated signal from either the IP or EP) initiates the memory cycle and immediately generates a RAS- to all the memory chips. Meanwhile, MEM- activates an internal clock synchronizing circuit which synchronizes MEM- to the next occurring edge on a 20MHz crystal oscillator clock. Once this clock is synchronized, the next occurring edge of the clock (25 nanoseconds later) enables the column addresses to the memory chips. The next edge of the clock 25 nanoseconds later enables the CAS- lines (generated from the block mapper) to the memory chips and also activates a shift register (used as a delay generator) which raises the memory ready line and enables the parity error line after a determinate number of clock cycles. The number of clock cycles used to clock memory ready and parity error is determined by the speed of the memory chips used (200 nanoseconds in this case).

2. Buffer control logic. (See also Section 4.13.1 on Block Mappers). The block mapper generates the CAS- signals to the memory array. However, whatever data gets enabled on the data bus is determined by the three signals A0, BHE (byte high enable), and BYTE (8-bit operation). These three signals decide through buffer control whether to channel odd bank data to D0-D7 (odd-byte access) or D8-D15 (word access). All even bank data is channeled to D0-D7.

4.13.4 PARITY GENERATION AND CHECKING

One parity bit is generated on each write operation (two parity bits on each word operation). The stored parity bit is determined by whether a "1" is required to generate a 9-bit "odd parity". If not, then a "0" is stored. A subsequent read of this memory location should generate an "odd parity" output if the memory has not deviated by one-bit per byte of data. If the memory has deviated by one-bit then a parity error interrupt is fired to the IP which in turn can relocate the bad block of memory.

4.13.5 REFRESHING

The dynamic memory refresh cycles are generated from the IP as a sequence (burst) of 16 IP cycles. Each of these IP cycles represents one refresh cycle as the IP generates one row address per IP cycle. This burst of 16 refresh cycles is repeated 8 times every 224 microseconds to give a worst case refresh rate of 128 refresh cycles in 1.792 milliseconds (4116 specification calls for 128 refresh cycles every 2 milliseconds). The delay between each refresh cycle in the 16 cycle burst is dependent on how quickly the EP can grant IP cycles as the IP can generate IPWAIT- at a maximum rate of approximately 3MHz during the 16 cycle refresh burst. The IP will wait no longer than 12 microseconds before timing out and terminating the IPWAIT-.

4.13.6 WRITE PROTECT AND INTERNAL/EXTERNAL MEMORY MAPPERS

Simulation memory can be write-protected or internally mapped on 256-byte boundaries within a 64-megabyte range. Usually, 256K X 1 mappers would be required to accomplish this mapping. However, by encoding the memory array chip-select lines (CAS lines), and using the CAS lines as address inputs to 512 X 1 mappers (along with A8-A14), equivalent mappers with 64-megabyte mapping range are generated. Only mapped blocks of Simulation Memory can generate unique encoded addresses to the write-protect, and internal/external memory mappers. Note that A12-A14 must be input to these mappers to determine which 4K block within each 32K block of mapped Simulation Memory must be write-protected or internally mapped.

4.14. DYNAMIC MEMORY INTERFACE WITH IP BOARD

Both the block mappers and write-protect and internal/external mappers are programmable from the IP as a SYSTEM device. This means that these devices can only be written to by the IP. However, the IP is also capable of accessing the memory array.

4.14.1 PROGRAMMING THE BLOCK MAPPERS

Before the block mappers are actually programmed, certain IP address latches must be set up. The IP can directly drive 24-bits of addresses on the Emulator bus. The extra 2-bits required for a 26-bit memory must be driven by the IP I/O. The following steps should be followed before actually programming the block mappers.

1. Store a X`01' in location X'8030'. This sets up the Emulator bus for a system device access by setting the SYS line to "1".

2. Store a X`00' in location X'8000'. This sets up BA12-BA19 to all zeros. Note that D0 corresponds to BA12 and D7 corresponds to BA19 when storing to the X'8000' latch.

3. Store a X`30' in location X'8008'. The upper nibble of data represents the system address of the system device to be accessed. In this case, the block mappers have a system address of 3. The lower nibble represents BA20-BA23 with BA20 being the least significant bit in the X'8008' latch.

Note that setting up an address to the block mappers is at least a three step process. First BA12-BA19 must be set up in the X'8000' latch, MEM- should fall (negative true) when addresses are valid and set up by approximately 100 nanoseconds. The dynamic memory board generates data (read cycles) memory ready, write-protect (WPROT-), internal/external mapping (MAP-) and parity error (intended for IP use only). See EP memory timing interface.

4.14.2 WRITE PROTECT AND INTERNAL/EXTERNAL MEMORY MAPPERS

Programming these two mappers is very similar to programming the block mappers except for three important differences:

1. The write-protect and internal/external mappers have a system address of 15. Therefore, we would store a X`F0' into X`8008' when first initializing.

2. The block mappers are programmed first since they generate addresses (from chip selects) to the write-protect and internal/external mappers.

3. The write-protect attribute is written on BD0 and the internal/external attribute is written on BD1. BD4 and BD5 must be set to zero when writing to these mappers. BD2 is the test memory enable attribute (TME-) and resides on the IP board. BD6 and BD7 are "don't cares".

The write-protect and internal/external mappers are a direct extension of the block mappers. This is, the block mappers decide where each 32K block of Simulation Memory is mapped and the write-protect and internal/external mappers decide if each 256-byte block of memory within this mapped block of Simulation Memory is write-protected or internally mapped.

4.15 EMULATOR PERSONALITY

The Emulator Personality is discussed in general, since each Personality board uses different techniques to suit a specific microprocessor.

4.15.1 TEST MEMORY

The most involved task is that of controlling what code the Emulation Processor is to execute. While emulating, it is executing user code in Simulation or target system memory. When not executing user code, the Emulation Processor is under the Emulator's control executing out of Test Memory. Test Memory, located on the Personality Board, is transparent to the user. It is 256 to 2K-bytes and is overlayed in a convenient location for the microprocessor being emulated.

When the execute command is given, the Emulation Processor exits execution from Test Memory and begins execution of user code. Simultaneously, Test Memory disappears and all emulation functions are enabled. When a breakpoint is reached or the halt command is given, the Personality Board must force the Emulation Processor to execute Test Memory code, overlay Test Memory into the Emulation Processors address space, disable external control lines, and save all of the user's processor status without modifying user code. Specific techniques used to accomplish these tasks are discussed in the Supplements.

4.15.2 EMULATOR BUS ARBITRATION

Since the Emulation Processor is the master of the Emulator bus, the Personality Board must provide bus arbitration for the Interface Processor. When the IP requests the bus by asserting IPWAIT*, the Personality Board gives the bus to the IP at the appropriate time by negating EPCYC*. The appropriate time is when the Personality Board has assured that the Emulation Processor will not access the bus. The technique for this varies depending on the characteristics of the microprocessor being emulated. The bus is given up for only one bus cycle, and then returned to the Emulation Processor.

4.15.3 **PRE-FETCH TRACKING**

Several microprocessors have a pre-fetch queue. Without a pre-fetch, a breakpoint is reached when all break parameters appear on the bus. This is not true for pre-fetching. If the Emulator is to halt execution when an instruction has completed execution, the Breakpoint Board must not signal a breakpoint when the op-fetch appears on the bus but rather several cycles later when the op-code has passed through the queue and has been executed. The Personality board processes the breakpoint signals without halting emulation until a breakpoint is actually reached.

4.15.4 **TIMING SIGNALS**

The EP is required to generate valid addresses, R/W- (advance read/write), data (write cycles), and MEM- then BA20-BA23 must be set in the lower nibble of the X`8008′ latch and BA24-BA25 set up at the IP I/O. Once this is done, then a mapping pattern can be stored in one location in the block mappers, the address latches incremented, and a new mapping pattern stored.

4.15.5. **MAPPING**

The memory mapping RAMs provide the signal MAP* to the Personality Board. The Personality Board then uses MAP together with test memory control signals to control all address and data buffers within the Personality and Probe. All data address, and control information is placed on the Emulator bus for trace and breakpoint operations. The mapping determines where the Emulation Processor is to receive/transmit data. **Figure 4-13, Buffers/Transceivers** shows the bus and buffer control.

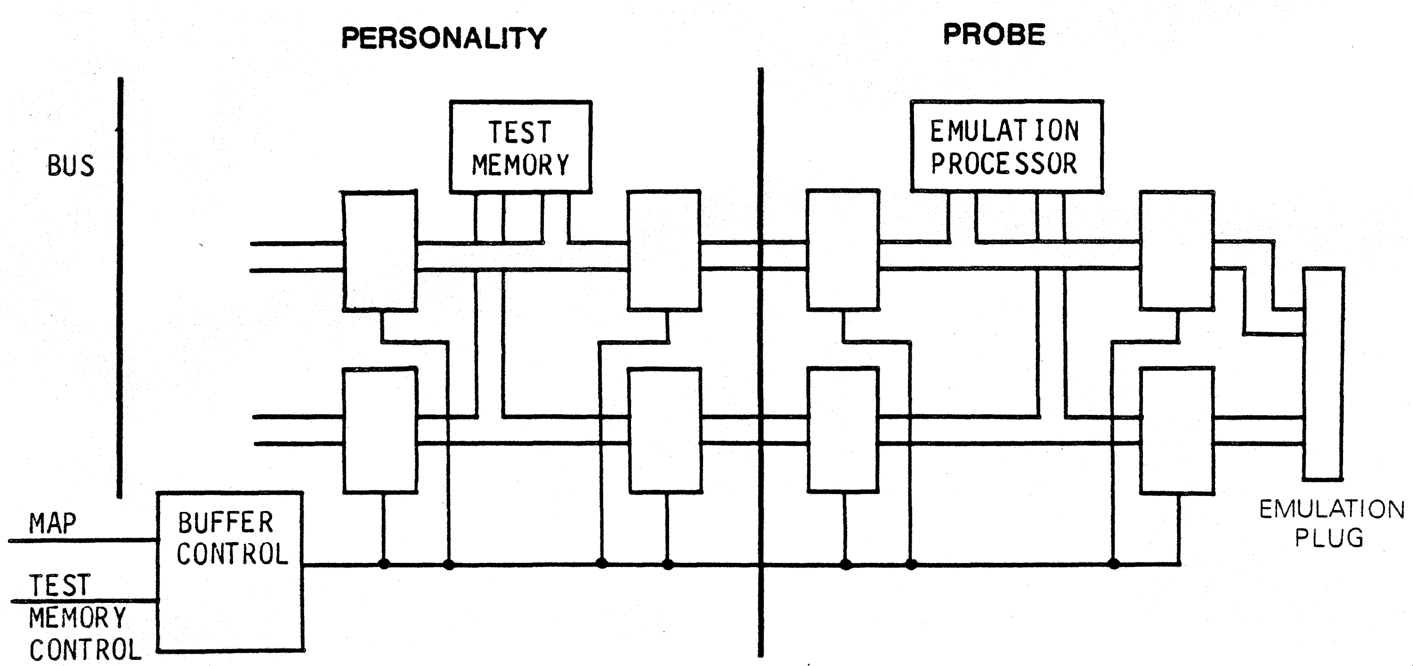


Figure 4-13. Buffers/Transceivers

The Personality Board must provide the interface of control signals between the Emulation Processor and the Emulator bus. This is generally a simple matter since only a R/W, I/O, and strobe are required.

APPENDIX

A piggy-back board (2302-4768) has been designed to provide flexible bus time-out control for the Interface Processor board. The existing dynamic RAM's on the 2302-4722 IP board are replaced with new static RAM's located on the new piggy-back board, and a strap-selectable timer is provided for the user. The piggy-back board eliminates a contention problem between the IP's need to referesh its internal dynamic memory, and external target processes. In addition, the the bus time-out control circuitry gives the user the option to extend the Target System bus access time, or inhibit it completely, just by changing the RC value of the timer.

Dynamic simulation memory is compatible only when the original 10 microsecond bus timeout is strap selected.

Older systems with IP boards at revision B levels can be retrofit with the piggy-back board modification. Please contact your Kontron Service Representative, or the factory.

STRAPPING:

The strap ER 1 located on the piggy-back board provides the following timing options:

	3&4	Original Bus Time-Out Duration. (Required With Dynamic Simulation Memory)
ER 1	3&1	Bus Time-Out Inhibited At All Times
	3&5	Extended Bus Time-Out (About 1 Second)

NOTE:

The extended bus time-out period of 1 second is recommended for normal system operation. The inhibited selection can be used for periods greater than 1 second, however, a target system check will occur after 4 seconds.

When ER1 is strapped at 3 and 5, the duration of the bus time-out can be altered by changing the value of C23 as shown in the following table. R1 is set at 1 megohm.

BUS TIME-OUT TABLE

BUS TIME-OUT DURATION	C23
30 ms	0.010 microfarad
100 ms	0.068 microfarad
1 sec	0.680 microfarad

INDEX

KONTRON SERVICE NUMBERS

Kontron Customer Service is available from the following phone numbers during regular business hours:

All areas except California	call (800) 227-5416
Northern California	call (415) 330-7962
Southern California	call (714) 730-6238

Calls received after business hours (8:00 a.m. to 5:00 p.m., Monday through Friday) will be answered the next working day.

5730 Buckingham Parkway, Culver City, CA 90230

(213) 641-7700/641-7200
TWX: 910-328-7202

ADVANCED ELECTRONIC INSTRUMENTATION KONTRON ELECTRONICS

SERVICE LOCATIONS

BOSTON	Conn., Maine, Massachusetts, New Hampshire, Rhode Island, Vermont
CHICAGO	Illinois, Iowa, Minnesota, Ohio Missouri, North Dakota, South Dakota, Kentucky, Indiana, Michigan, West PA., Wisconsin Nebraska
SAN FRANCISCO	Colorado, Montana, Wyoming, New Mexico, Utah, No. California, Idaho, No. Nevada, Washington, Oregon
LOS ANGELES	So. California, Arizonia, So. Nevada
WASHINGTON D.C.	Maryland, Virginia, West Virginia, D.C.
ORLANDO	Alabama, Georgia, Florida, North Carolina, South Carolina, Tennessee, Mississippi
CULVER CITY	Factory Support (213) 641-7200 Ask for Field Service

Please note the following numbers for service:

All areas except California	800/227-5416
Northern California	415/330-7962
Southern California	714/730-6238.

KONTRON ELECTRONICS

Technical Publications Remarks Form

Please use this form to submit your suggestions for revisions, corrections, or additions to this publication. Your comments will be promptly investigated by appropriate technical personnel, and action will be taken as required. If your answer to any of the questions is 'NO' or requires qualification, please include any additional comments on a separate sheet and enclose it inside this pre-addressed form.

TITLE:

ORDER NO.

DATED

	YES	NO
DOES THIS DOCUMENT MEET YOUR NEEDS?	____	____
IS THE MATERIAL CONTAINED IN THIS DOCUMENT:		
ACCURATE?	____	____
EASY TO READ AND UNDERSTAND?	____	____
ORGANIZED FOR CONVENIENT REFERENCING?	____	____
WELL ILLUSTRATED WITH USEFUL EXAMPLES?	____	____
COMPLETE?	____	____

CUT ALONG DOTTED LINE

ERRORS IN THE PUBLICATION

SUGGESTIONS FOR IMPROVEMENT

(Please Print)

FROM: NAME ________________________ **DATE** ________

ORGANIZATION ________________________

ADDRESS ________________________

TITLE ________________________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 608 CULVER CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

ADVANCED
ELECTRONIC
INSTRUMENTATION

KONTRON
ELECTRONICS

Technical Publications
18001 Cowan
Irvine, CA 92714