User’s Guide for the Terminal Interface

HP 64744 68000/HC001/EC000 Emulation/Analysis

Notice

Hewlett-Packard
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1	64744-97000, March 1992
Edition 2	**64744-97002, December 1993**

Safety, and Certification and Warranty

Safety and certification and warranty information can be found at the end of this manual on the pages before the back cover.

68000/HC001/EC000 Emulation and Analysis

The HP 64744 68000/HC001/EC000 Emulator replaces the microprocessor in your embedded microprocessor system, also called the *target system*, so that you can control execution and view or modify processor and target system resources.

The emulator requires an *emulation analyzer* that captures 48 channels of emulation processor bus cycle information synchronously with the processor's clock signal. The HP 64706 Emulation Bus Analyzer meets this requirement.

You can also use the HP 64703 Emulation Bus Analyzer which has 64 channels and an *external analyzer* that captures up to 16 channels of data external to the emulator.

And, you can use the HP 64704 or HP 64794 Emulation Bus Analyzers which have 80 channels; however, these analyzers do not have external analysis channels.

With the Emulator, You Can ...

- Plug into 68000/HC000 target systems with Dual In-line Package (DIP) sockets, 68HC000/001 target systems with Pin Grid Array (PGA) or Plastic Leaded Chip Carrier (PLCC) sockets, or 68EC000 target systems with PLCC sockets.
- Download programs into emulation memory or target system RAM.
- Display or modify the contents of processor registers and memory resources.
- Run programs at clock speeds up to 16 MHz (with no wait-states from emulation memory), set up software breakpoints, step through programs, and reset the emulation processor.

With the Analyzer, You Can ...

- Trigger the analyzer when a particular bus cycle state is captured. States are stored relative to the trigger state.
- Qualify which states that get stored in the trace.
- Prestore certain states that occur before each normal store state.
- Trigger the analyzer after a sequence of up to 8 events have occurred.
- Capture data on signals of interest in the target system.
- Cause emulator execution to break when the analyzer finds its trigger condition.

With the HP 64700 Card Cage, You Can ...

- Use the RS-422 capability of the serial port and an RS-422 interface card on the host computer (for example, the HP 98659 for the HP 9000 or the HP 64037 for the PC) to provide upload/download rates of up to 230.4K baud.
- Easily upgrade HP 64700 firmware by downloading to flash memory.

With Multiple HP 64700s, You Can ...

- Start and stop up to 16 emulators at the same time (up to 32 if modifications are made).
- Use the analyzer in one HP 64700 to arm (that is, activate) the analyzers in other HP 64700 card cages or to cause emulator execution in other HP 64700 card cages to break.
- Use the HP 64700's BNC connector to trigger an external instrument (for example, a logic analyzer or oscilloscope) when the analyzer finds its trigger condition, or you can allow an external instrument to arm the analyzer or break emulator execution.

In This Book

This book documents the HP 64744 68000/HC001/EC000 Emulator and the HP 64706/3 Analyzer. It is organized into five parts whose chapters are described below.

Part 1. Quick Start Guide

Chapter 1 presents an overview of emulation and analysis and quickly shows you how to use the emulator and analyzer.

Part 2. User's Guide

Chapter 2 shows you how to plug the emulator into target systems.
Chapter 3 shows you how to enter Terminal Interface commands and display HP 64700 system information.
Chapter 4 shows how to use the emulator.
Chapter 5 shows how to use the analyzer in the "easy" configuration.
Chapter 6 shows how to use the analyzer in the "complex" configuration.
Chapter 7 shows how to use external state analyzer.
Chapter 8 shows how to make coordinated measurements.

Part 3. Reference

Chapter 9 describes Terminal Interface commands.
Chapter 10 describes the error messages that can occur while using the Terminal Interface and provides recovery information.
Chapter 11 lists the emulator and external analyzer specifications and characteristics.

Part 4. Concept Guide

Chapter 12 contains conceptual (and more detailed) information on various topics.

Part 5. Installation Guide

Chapter 13 shows you how to install emulator and analyzer boards into the HP 64700 Card Cage and how to connect the HP 64700 to a host computer or terminal.

Contents

Part 1 Quick Start Guide

1 Quick Start

Part 2 User's Guide

Part 5 Installation Guide

13 Installation

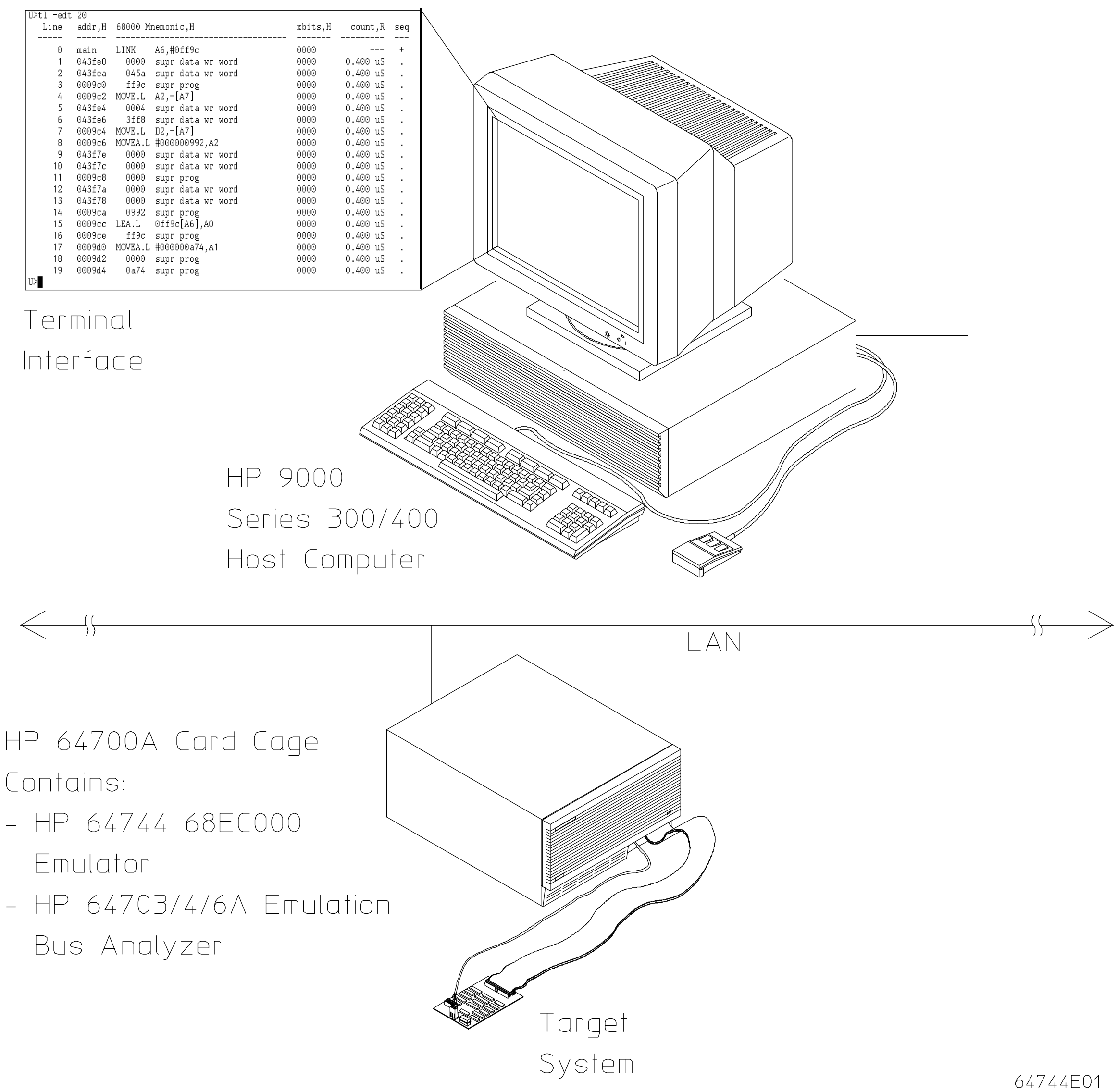
Part 1

Quick Start Guide

1

Quick Start

The 68000/HC001/EC000 Emulator — At a Glance



The tutorial examples presented in this chapter make the following assumptions:

- The HP 64700 is connected to the same LAN as an HP 9000 Series 300 host computer (refer to the "Installation" chapter).
- Networking software is installed on your HP 9000 Series 300 host computer (primarily telnet and ftp software).
- The emulator demo program (see the description in the "Concepts" chapter) is compiled, assembled, and linked and an HP 64000 format absolute file is created.
- A symbols file has been created.

Step 1. Log in to the emulator

- Use the **telnet** command on the host computer to connect to the HP 64700.

```
$ telnet hostname
```

Where "hostname" is the name of the emulator. Or, you could use the Internet Protocol (IP) address (or internet address) in place of the hostname:

```
$ telnet 15.35.226.210
```

You should see messages similar to:

```
Trying...
Connected to 15.35.226.210
Escape character is '^]'.
```

After you connect to the emulator, you should see a prompt similar to:

```
R>
```

Step 2. Initialize the emulator

Make sure you begin this tutorial with the emulator in its default, power-up state by initializing the emulator.

- Initialize the emulator by entering the **init** command.

```
R>init
  # Limited initialization completed
```

Step 3. Map memory for the demo program

Because the emulator can use target system memory or emulation memory (or both), it is necessary to map ranges of memory so that the emulator knows where to direct its accesses. You can map up to 16 memory ranges with 4 Kbyte resolution (beginning on 4 Kbyte boundaries and at least 4 Kbytes in length).

You can characterize memory ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or as guarded memory.

The "cmd_rdr" demo program occupies ROM locations from 0 through 0FE3H and RAM locations from 40000H through 43FFFH and from 60000H through 63129H. (You can tell this by looking at the linker load map output listing that is generated when compiling the demo program.)

1 Map emulation memory for the demo program by entering the following **map** commands.

```
R>map 0..0fff erom
R>map 40000..43fff eram
R>map 60000..63fff eram
```

2 View the resulting memory map by entering the **map** command with no parameters.

```
R>map
  # remaining number of terms   : 13
  # remaining emulation memory  : 6000h bytes
  map  0000000..0000fff     erom          # term 1
  map  0040000..0043fff     eram          # term 2
  map  0060000..0063fff     eram          # term 3
  map  other tram
```

The "other" term in the memory map specifies that unmapped memory ranges are treated as target system RAM by default.

Step 4. Set up the supervisor stack pointer value

After emulator initialization, the "reset supervisor stack pointer" configuration item (**cf rssp**) and the supervisor stack pointer are set to an odd value. Since you cannot run the emulator when the supervisor stack pointer is odd, you must set up the supervisor stack pointer register to contain an even value. You can do this by using the **reg** command to modify the supervisor stack pointer, or you can change the reset supervisor stack pointer configuration item (as shown below). The value you assign to the **rssp** configuration item is placed into the supervisor stack pointer on the entrance to the emulation monitor from an emulation initiated RESET state (the R> prompt).

- Set the stack pointer to address 44000H by entering the following command.

```
R>cf rssp=44000
```

Upon the first transition from emulation reset into the emulation monitor, the supervisor stack pointer register (A7) is set to 44000H.

Step 5. Load the demo program absolute file and symbols

The HP AxLS software development tools generate IEEE-695 format or HP format absolute files. However, the Terminal Interface's **load** command only supports the following formats: HP absolute, Intel hexadecimal, Extended Tektronix hexadecimal, and Motorola S-records. So, when using the HP AxLS tools, be sure to generate HP format absolute files.

You can typically create an ASCII symbol file using information from a linker load map output file; however, the ASCII symbol file must be in the proper format.

Suppose the following "cmd_rdr.sym" file exists on the HP 9000 host computer.

```
#
:Cmd_Input                    00060056
:Msg_Dest                     00060057
:Write_Msg                    00000992
:main                         000009BE
atexit:__exec_funcs           00000E6C
atexit:__top_of_func_stack    000600FC
atexit:_atexit                00000E3A
cmd_rdr:_Cmd_Input            00060056
cmd_rdr:_Msg_Dest             00060057
cmd_rdr:_Write_Msg            00000992
cmd_rdr:_main                 000009BE
crt1:__exit                   0000047E
crt1:_exit                    0000046A
crt1:entry                    00000400
data_gen:__infinity           00000E92
data_gen:__malloc_init        00060106
data_gen:__rand_seed          000600FE
data_gen:_errno               00060102
disp_msg:XEnv_68k_except      00060034
disp_msg:__display_message    00000534
disp_msg:end_of_program       00000562
fperror:__fp_control          0006007A
fperror:__fp_error            00000AD8
fperror:__fp_errorf           00000AD8
fperror:__fp_errori           00000AD8
fperror:__fp_status           00060078
getmem:__getmem               0000050A
getmem:nextblk                00060030
mon_stub:JSR_ENTRY            00000FE0
mon_stub:MONITOR_MESSAGE      00060128
sysheap:TopOfHeap             0006312A
sysheap:heap                  0006012C
sysstack:TopOfStack           00044000
sysstack:stack                00040000
systrap:trap                  000005F2
#
```

1 Change to the directory that contains the "cmd_rdr.X" absolute file and the "cmd_rdr.sym" symbol file. For example:

```
$ cd 68ec000/demo <RETURN>
```

2 Connect to the emulator's ftp interface by entering the **ftp** command on your local host computer (use any name and password).

Note The "ftp" capability of the HP 64700 is unsupported. It is provided at no cost. Hewlett-Packard makes no warranty on its quality or fitness for a particular purpose.

When connecting to the HP 64700's ftp interface, you can use either the HP 64700's hostname or the Internet Protocol (IP) address (or internet address). When you use the HP 64700's hostname, the ftp software on your computer will look up the internet address in the hosts table, or perhaps a name server will return the internet address.

```
$ ftp 15.35.226.210
Connected to 15.35.226.210.
220 User connected to HP64700
Name (15.35.226.210:guest): <RETURN>
Password (15.35.226.210:guest): <RETURN>
```

3 Set up ftp for binary file transfers.

```
ftp> binary
200 Type set to I
```

4 Download the HP 64000 format absolute file into the emulator.

```
ftp> put cmd_rdr.X -h
200 Port      ok
150
226-
R>
226 Transfer completed
3332 bytes sent in 0.20 seconds (16.27 Kbytes/sec)
```

5 Download the symbol file into the emulator.

```
ftp> put cmd_rdr.sym -S
200 Port      ok
150
226-
R>
226 Transfer completed
1789 bytes sent in 4.78 seconds (0.37 Kbytes/sec)
```

6 Exit out of the ftp interface:

```
ftp> quit
221 Goodbye
$
```

Step 6. Display the demo program symbols

1 Display the symbols with the **sym** command.

```
R>sym
  sym Cmd_Input=00060056
  sym Msg_Dest=00060057
  sym Write_Msg=00000992
  sym main=000009be
  sym atexit:__exec_funcs=00000e6C
  sym atexit:__top_of_func_stack=000600fc
  sym atexit:_atexit=00000e3a
  sym cmd_rdr:_Cmd_Input=00060056
  sym cmd_rdr:_Msg_Dest=00060057
  sym cmd_rdr:_Write_Msg=00000992
  sym cmd_rdr:_main=000009be
  sym crt1:__exit=0000047e
  sym crt1:_exit=0000046A
  sym crt1:entry=00000400
  sym data_gen:__infinity=00000e92
  sym data_gen:__malloc_init=00060106
  sym data_gen:__rand_seed=000600fe
  sym data_gen:_errno=00060102
  sym disp_msg:XEnv_68k_except=00060034
  sym disp_msg:__display_message=00000534
  sym disp_msg:end_of_program=00000562
  sym fperror:__fp_control=0006007a
  sym fperror:__fp_error=00000ad8
  sym fperror:__fp_errorf=00000ad8
  sym fperror:__fp_errori=00000ad8
  sym fperror:__fp_status=00060078
  sym getmem:__getmem=0000050a
  sym getmem:nextblk=00060030
  sym mon_stub:JSR_ENTRY=00000fe0
  sym mon_stub:MONITOR_MESSAGE=00060128
  sym sysheap:TopOfHeap=0006312a
  sym sysheap:heap=0006012c
  sym sysstack:TopOfStack=00044000
  sym sysstack:stack=00040000
  sym systrap:trap=000005f2
```

Step 7. Display the demo program in memory

The **m** command lets you display and modify memory locations. When displaying memory, the **-dm** option causes the contents of memory locations to be disassembled and displayed in assembly language mnemonic format.

- Display the demo program in memory by entering the following **m -dm** command.

```
R>m -dm main..
  00009be  main             LINK     A6,#0ff9c
  00009c2  -                MOVE.L   A2,-[A7]
  00009c4  -                MOVE.L   D2,-[A7]
  00009c6  -                MOVEA.L #000000992,A2
  00009cc  -                LEA.L    0ff9c[A6],A0
  00009d0  -                MOVEA.L #000000a74,A1
  00009d6  -                MOVEQ.L #000000020,D1
  00009d8  -                MOVE.B   [A1]+,[A0]+
  00009da  -                DBF      D1,00009d8
  00009de  -                LEA.L    0ffbd[A6],A0
  00009e2  -                MOVEA.L #000000a95,A1
  00009e8  -                MOVEQ.L #000000020,D1
  00009ea  -                MOVE.B   [A1]+,[A0]+
  00009ec  -                DBF      D1,00009ea
  00009f0  -                LEA.L    0ffde[A6],A0
  00009f4  -                MOVEA.L #000000ab6,A1
  00009fa  -                MOVEQ.L #000000020,D1
  00009fc  -                MOVE.B   [A1]+,[A0]+
  00009fe  -                DBF      D1,00009fc
  0000a02  -                NOP
  0000a04  -                MOVE.B   #000,Cmd_Input
  0000a0c  -                BRA.B    0000a12
  0000a0e  -                NOP
  0000a10  -                NOP
  0000a12  -                MOVE.B   Cmd_Input,D2
  0000a18  -                BEQ.B    0000a10
  0000a1a  -                NOP
  0000a1c  -                MOVE.B   D2,D0
  0000a1e  -                EXT.W    D0
  0000a20  -                EXT.L    D0
  0000a22  -                SUBI.L   #000000041,D0
  0000a28  -                SUBQ.L   #1,D0
  0000a2a  -                BHI.W    0000a56
  0000a2e  -                ADD.W    D0,D0
  0000a30  -                MOVE.W   0000a3a[PC,D0.W],D0
  0000a34  -                JMP      0000a38[PC,D0.W]
  0000a38  -                ORI.B    #012,D6
  0000a3c  -                NOP
```

Step 8. Execute the demo program

The **r <addr>** command causes the emulator to run from a particular address. The entry address of the demo program is 400H and is specified by the symbol "crt1:entry".

- Execute the demo program by entering the **r <addr>** command.

```
R>r crt1:entry
U>
```

Before the **r** command, the emulation status character (in the Terminal Interface prompt) was "R" indicating that the emulation processor was being held in reset. After the **r** command, the emulation status character is "U" which indicates the emulator is executing the user program.

Step 9. Trace demo program execution

The **t** (trace) command tells the analyzer to look at the data on the emulation processor's bus and control signals at each clock cycle. The information seen at a particular clock cycle is called a state.

When one of these states matches the "trigger state" you specify, the analyzer stores states in trace memory. When trace memory is filled, the trace is said to be "complete."

The default trigger state specification is any state, so the **t** command will cause the analyzer to "trigger" on the first state it sees and store the following states in trace memory.

1 Specify the trigger state as the starting address (main) of the demo program by entering the following **tg** command.

```
U>tg addr=main
```

2 Start the trace by entering the **t** command.

```
U>t
  Emulation trace started
```

3 Run the demo program from the demo program's entry address by entering the following run command.

```
U>r crt1:entry
```

4 View the status of the trace by entering the **ts** command.

```
U>ts
  --- Emulation Trace Status ---
  NEW User trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 512 (512) 0..511
  Sequence term 2
  Occurrence left 1
```

Notice that the trace is complete and that 512 states have been stored.

5 List the first twenty states stored in the trace (-t 20), with instructions disassembled (-d) and symbols and addresses stored in the addr column (-e), by entering the following **tl** command.

```
U>tl -d -e -t 20
```

```
 Line   addr,H  68000 Mnemonic,H                     xbits,H    count,R  seq
-----   ------  ----------------------------------   -------   ---------  ---
    0   main    LINK    A6,#0ff9c                    0000            ---   +
    1   043fe8    0000  supr data wr word            0000      0.200 uS   .
    2   043fea    045a  supr data wr word            0000      0.200 uS   .
    3   0009c0    ff9c  supr prog                    0000      0.200 uS   .
    4   0009c2  MOVE.L  A2,-[A7]                     0000      0.200 uS   .
    5   043fe4    0004  supr data wr word            0000      0.200 uS   .
    6   043fe6    3ff8  supr data wr word            0000      0.200 uS   .
    7   0009c4  MOVE.L  D2,-[A7]                     0000      0.200 uS   .
    8   0009c6  MOVEA.L #000000992,A2                0000      0.200 uS   .
    9   043f7e    0992  supr data wr word            0000      0.200 uS   .
   10   043f7c    0000  supr data wr word            0000      0.200 uS   .
   11   0009c8    0000  supr prog                    0000      0.200 uS   .
   12   043f7a    0000  supr data wr word            0000      0.200 uS   .
   13   043f78    0000  supr data wr word            0000      0.200 uS   .
   14   0009ca    0992  supr prog                    0000      0.200 uS   .
   15   0009cc  LEA.L   0ff9c[A6],A0                 0000      0.200 uS   .
   16   0009ce    ff9c  supr prog                    0000      0.200 uS   .
   17   0009d0  MOVEA.L #000000a74,A1                0000      0.200 uS   .
   18   0009d2    0000  supr prog                    0000      0.200 uS   .
   19   0009d4    0a74  supr prog                    0000      0.200 uS   .
```

The first column in the trace list contains the line number. The trigger state is always on line number 0.

The second column contains the address information associated with the trace states. Addresses in this column may be locations of instruction opcodes on fetch cycles, or they may be sources or destinations of operand cycles. The **-e** option in the **tl** command causes both addresses and symbols to appear in this column.

The third column shows mnemonic information about the emulation bus cycle. The **-d** option in the **tl** command causes instructions to be disassembled.

If your analyzer card contains external analysis (for example, HP 64703), the next column shows the data captured on the external trace signals.

The next column shows the count information (**time** is counted by default). The "R" indicates that each count is relative to the previous state.

The last column contains information about the analyzer's sequencer. Whenever a "+" appears in this column, it means the state caused a sequencer branch.

Step 10. Stop (break from) program execution

The **b** command causes emulator execution to break from the user program into the emulation monitor program.

The emulation monitor program is a program that is executed by the emulation processor that allows the emulator to access target system resources. For example, when you display target system memory locations, the monitor program executes 68000 instructions that read the target memory locations and send their contents to the emulator.

When the emulator is running the user program, commands that require access to target system resources will cause temporary breaks to the monitor program (unless the emulator is restricted to real time execution).

When the emulator is running in the monitor program, it executes in a loop that waits for commands that require access to target system resources.

- Break emulator execution out of the demo program and into the monitor program by entering the **b** command.

```
U>b
M>
```

Notice that the emulation status character becomes "M" which indicates that the emulator is running in the monitor program.

Step 11. Display processor registers

- Display the contents of the basic processor registers by entering the **reg** command.

```
M>reg

reg pc=00000a12 st=2704 d0=00000000 d1=0000ffff d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00043fe3 a1=00000ad7
reg a2=00000992 a3=00000000 a4=00000000 a5=00068056 a6=00043fe4 a7=00043f78
reg usp=00000000 ssp=00043f78
```

Step 12. Step through program execution

The **s** command lets you step through user program execution. You can step single instructions or a number of instructions at a time.

1 Step one instruction in the user program by entering the **s** command.

```
M>s
  0000a12@sp  -                 MOVE.B  Cmd_Input,D2
  PC = 0000a18@sp
```

2 Step eight instructions in the user program by entering the **s 8** command.

```
M>s 8
  0000a18@sp  -                 BEQ.B   0000a10
  0000a10@sp  -                 NOP
  0000a12@sp  -                 MOVE.B  Cmd_Input,D2
  0000a18@sp  -                 BEQ.B   0000a10
  0000a10@sp  -                 NOP
  0000a12@sp  -                 MOVE.B  Cmd_Input,D2
  0000a18@sp  -                 BEQ.B   0000a10
  0000a10@sp  -                 NOP
  PC = 0000a12@sp
```

Step 13. Reset the emulator

- Reset the emulator by entering the **rst** command.

```
M>rst
R>
```

Notice that the emulation status character is "R" which shows that the emulator is being held in a reset state.

If the emulator status character is unfamiliar

The "R", "U", and "M" emulation prompt status characters are described in this chapter. If you see other emulation status characters, enter the **es** command for more information about the emulator status.

- Display the emulator status information by entering the **es** command.

```
R>es
  M68000--Emulation reset
```

Part 2

User’s Guide

2

Plugging into a Target System

Plugging the Emulator into a Target System

This chapter describes the tasks you must perform when plugging the emulator into a target system. These tasks are grouped into the following sections:

- Connecting the emulator to the target system.
- Configuring the emulator for operation with your target system.
- Selecting the emulation monitor program.

Connecting the Emulator to the Target System

This section describes the steps you must perform when connecting the emulator to a target system:

1 Turn OFF power.

2 If the emulator is currently connected to a different target system, unplug the probe adapter; otherwise, disconnect the emulator probe cables from the demo board.

3 Connect the emulator probe cables to the emulator probe adapter.

4 Plug the emulator probe adapter into the target system.

5 Turn ON power.

CAUTION

Possible Damage to the Emulator Probe. The emulation probe contains devices that are susceptible to damage by static discharge. Therefore, precautionary measures should be taken before handling the microprocessor connector attached to the end of the probe cable to avoid damaging the internal components of the probe by static electricity.

We STRONGLY suggest using a ground strap when handling the emulator probe. A ground strap is provided with the emulator.

Step 1. Turn OFF power

CAUTION

Possible Damage to the Emulator. Make sure target system power is OFF and make sure HP 64700 power is OFF before removing or installing the emulator probe adapter into the target system.

Do not turn HP 64700 power OFF while the emulator is plugged into a target system whose power is ON.

1 If the emulator is currently plugged into a different target system, turn that target system's power OFF.

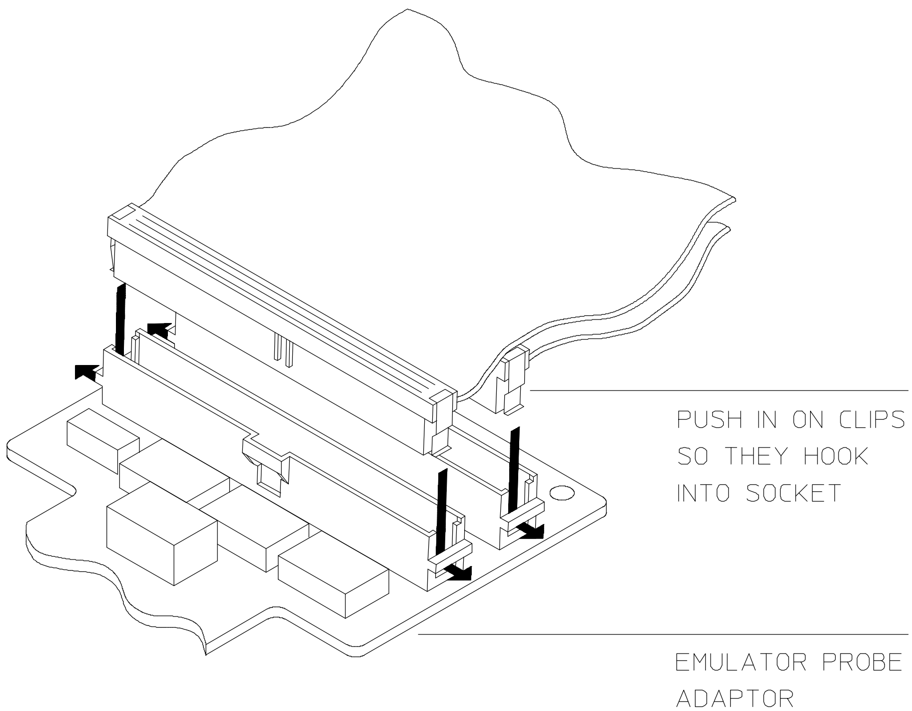
2 Turn emulator power OFF.

Step 2. Unplug probe and/or disconnect probe cables

1 If the emulator is currently connected to a different target system, unplug the probe adapter; otherwise, disconnect the emulator probe cables from the demo board.

Step 3. Attach the emulator probe adapter

1 If the emulator probe cables were disconnected from the demo board, you need to connect them to the emulator probe adapter.



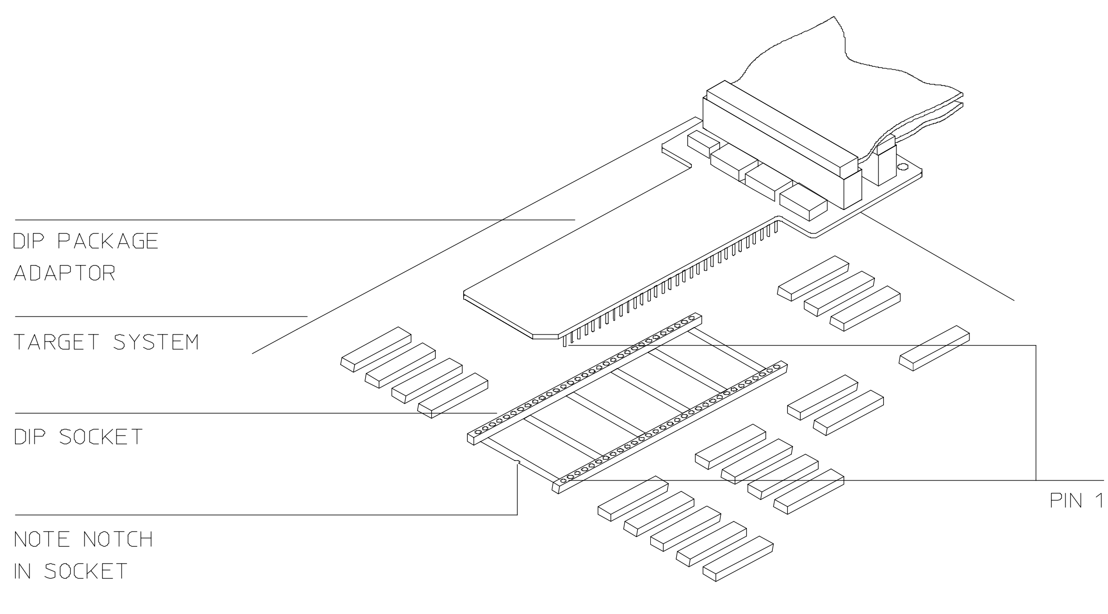
Step 4. Connect the probe to the target system

CAUTION

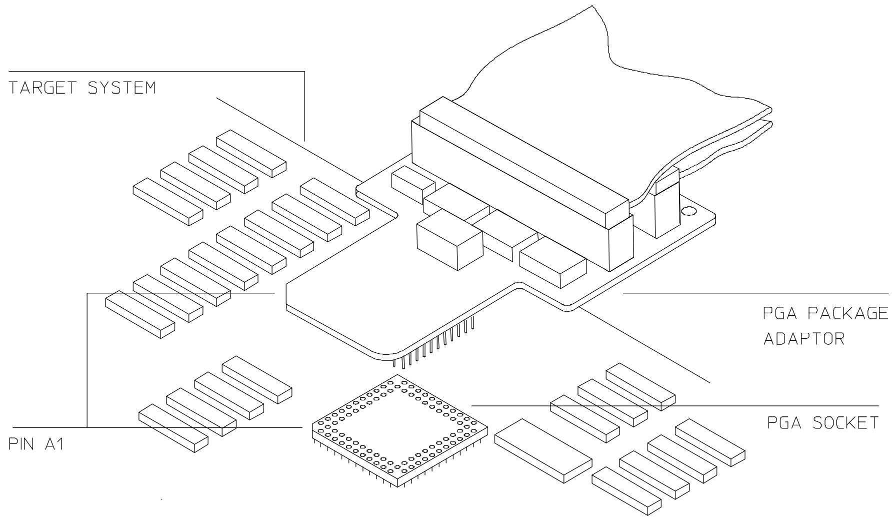
Possible Damage to the Emulator Probe. The DIP and PGA package adapters include pin extenders. **Do not use the probe without a pin extender installed**. Replacing a broken pin extender is much less expensive than replacing the emulator probe adapter.

The use of more than one pin extender is discouraged, unless it is necessary for mechanical clearance reasons, because pin extenders cause signal quality degradation.

To connect the DIP package adapter: Install the emulator probe adapter into the target system socket. Make sure that pin 1 of the connector aligns with pin 1 of the socket. **Damage to the emulator will result if the probe adapter is incorrectly installed.**



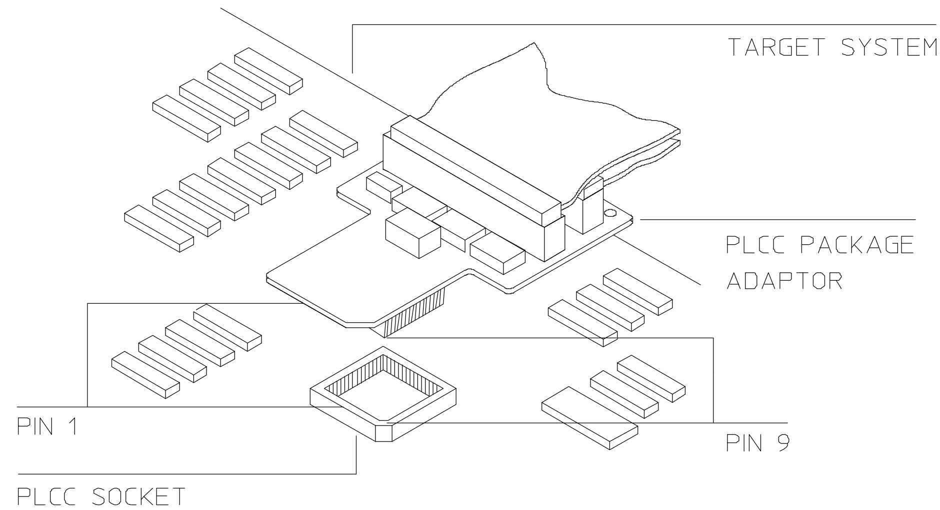
To connect the PGA package adapter: Install the emulator probe adapter into the target system socket. Make sure that pin 1 of the connector aligns with pin 1 of the socket. **Damage to the emulator will result if the probe adapter is incorrectly installed.**



CAUTION: MAKE SURE PIN 1 OF THE PROBE CONNECTOR IS ALIGNED WITH PIN 1 OF THE SOCKET. DAMAGE TO THE EMULATOR PROBE WILL RESULT IF THE PROBE IS INCORRECTLY INSTALLED!

To connect the PLCC package adapter: Install the emulator probe adapter into the target system socket. Make sure that pin 1 of the connector aligns with pin 1 of the socket. **Damage to the emulator will result if the probe adapter is incorrectly installed.**

Hewlett-Packard recommends that you do not use an additional adapter with the 68000/HC001 PLCC emulator probe.



Step 5. Turn ON power

1 Turn emulator power ON.

2 Turn target system power ON.

Configuring for Operation with Your Target System

After you plug the emulator into a target system and turn on power to the HP 64700, you need to configure the emulator so that it operates properly with your target system.

Before the emulator can operate in your target system, you must:

Map memory. Because the emulator can use target system memory or emulation memory (or both), it is necessary to map ranges of memory so that the emulator knows where to direct its accesses. You can synchronize emulation memory accesses to the target system in order to more closely imitate target system memory. For example, if emulation memory replaces slower target system memory that requires wait states, synchronizing emulation memory to the target system causes wait states to be inserted on emulation memory accesses as they would be on target system memory accesses. Refer to the "Mapping Memory" section in the "Using the Emulator" chapter.

Set the reset value of the supervisor stack pointer register. The supervisor stack pointer must point to RAM in order for the emulator to transition into the run state, to step, or to perform other functions after emulation reset.

Also, the emulator needs to know the following things:

If you're emulating a 68HC001 or 68EC000 microprocessor, should the emulator operate in 16-bit mode or 8-bit mode? In other words, what is the target system data bus width?

Is there circuitry in the target system that requires programs to run in real-time? Some emulator commands cause temporary breaks to the monitor program, typically to access microprocessor register values or target system memory. If the target system requires that programs run in real-time, you must restrict the emulator to real-time runs.

Should the emulator respond to target system interrupts when running programs? If so, you must enable the emulator's response to target system interrupts; otherwise, you must disable the emulator's response to target system interrupts.

Should the emulator respond to target system interrupts when running in the monitor program? If so, you must use a foreground monitor program since target system interrupts are always ignored during background operation (refer to the "Selecting the Emulation Monitor Program" section later in this chapter). You must also enable the emulator's response to target system interrupts. If it's not important that the emulator respond to target system interrupts when running in the monitor, you can use the background monitor.

Is there circuitry in the target system that constantly monitors bus cycle execution (for example, memory refresh circuitry or a watchdog timer)? If so, you should drive background cycles to the target system.

Should monitor cycles be synchronized to the target system (just as emulation memory accesses can be)?

Should bus arbitration be allowed? Generally, the answer to this question will be "yes". However, you may disable bus arbitration in order to isolate target system problems. For example, if you have a situation where the processor never seems to execute any code, you can disable bus arbitration to check and see if arbitration circuitry in your target system might be contributing to the problem.

This section shows you how to:

- Set the reset value of the Supervisor Stack Pointer.
- Set the processor data bus width.
- Restrict to real-time runs.
- Turn OFF the restriction to real-time runs.
- Enable/disable response to target system interrupts.
- Drive background cycles to the target system.
- Stop driving background cycles to the target system.
- Synchronize monitor cycles to the target system.
- Disable the synchronization of monitor cycles.
- Enable/disable bus arbitration.

To set the reset value of the Supervisor Stack Pointer

- Enter the **cf rssp=<value>** command.

In order for the emulator to transition into the run state, to step, or to perform other functions after emulation reset, the supervisor stack pointer must be set to an appropriate value.

The value specified must be a 32-bit hexadecimal even address. This address should reside in an otherwise unused emulation or target system RAM area.

Upon the first transition from emulation reset into the emulation monitor, the supervisor stack pointer register is set to the value specified.

To set the processor data bus width (68HC001/68EC000 only)

- Enter the **cf pdw=8** to set the emulator to 8-bit mode.

- Enter the **cf pdw=16** to set the emulator to 16-bit mode.

This configuration option is only valid for 68HC001/68EC000 emulation. When emulation the 68000, the 16-bit mode is the only valid selection.

To restrict to real-time runs

- Enter the **cf rrt=en** command.

While running programs, temporary breaks to the monitor program are not allowed. The emulator refuses the following commands:

- **reg** (register display/modification).
- **m** (memory display/modification) commands that access target system memory.

  (Because the emulator contains dual-port emulation memory, commands that access emulation memory are allowed while runs are restricted to real-time.)

CAUTION

Target system damage could occur! If your target system circuitry is dependent on constant execution of program code, the following commands still cause breaks from running programs even when you have restricted the emulator to real-time runs:

- **rst** (reset).
- **r** (run).
- **b** (break to monitor).
- **s** (step).

Use caution in executing these commands.

To turn OFF the restriction to real-time runs

- Enter the **cf rrt=dis** command.

Temporary breaks to the monitor while running programs are allowed, and the emulator accepts commands normally.

To enable/disable response to target system interrupts

- Enter the **cf ti=en** command to enable the emulator to respond to target system interrupts during foreground operation.

- Enter the **cf ti=dis** to disable emulator response to target system interrupts.

Target system interrupts are always ignored during background operation.

All 68000 systems require a vector table to process system conditions such as divide by zero or trace traps. You need to provide such a vector table to manage these conditions. Exception processing attempted without a vector table will cause unpredictable results.

To drive background cycles to the target system

1. Enter the **cf dbc=en** command.

2. Specify the A23-A8 values for background cycles driven to target by entering the **cf bbk=<value>** command.

3. Specify the function codes for background cycles driven to target by entering the **cf bfc=<function_code>** command.

When background cycles are driven to the target system, all of the emulation processor's address, data and control strobes are driven. Writes to background memory will appear as reads to the target system.

The value specified for the upper 16 bits of the address bus (A23-A8) should be a hexadecimal byte value. You should choose an address block which will not interfere with your target system circuitry such as memory management units or cache memory.

The function code specified can be:

sp (supervisor program, FC2-FC0=110).

sd (supervisor data, FC2-FC0=101).

up (user program, FC2-FC0=010).

ud (user data, FC2-FC0=001).

Choose a function code that will not cause target system hardware such as memory management units to behave in an unpredictable manner.

To stop driving background cycles to the target system

- Enter the **cf dbc=dis** command.

The emulator will appear to the target system as if it is between bus cycles while it is operating in the background monitor.

To synchronize monitor cycles to the target system

- Enter the **cf mondti=en** command.

The termination of monitor cycles, that is, accesses to the locations that the monitor program occupies, will not occur until the target system provides a /DTACK or /VPA (if the microprocessor being emulated supports /VPA).

When using the background monitor, background cycles must be driven to the target system in order for the **mondti** setting to have an effect. A /BERR signal from the target system will also terminate a monitor cycle and cause the emulator to begin execution of the bus error handler.

To disable the synchronization of monitor cycles

- Enter the **cf mondti=dis** command.

Monitor cycles, that is, accesses to the locations that the monitor program occupies, are terminated by an emulator-generated /DTACK signal.

To enable/disable bus arbitration

- Enter the **cf ba=en** command to enable target system bus arbitration.

- Enter the **cf ba=dis** command to disable target system bus arbitration.

When bus arbitration is enabled:

> The emulator responds to the /BR (bus request) and /BGACK (bus grant acknowledge) signals from the target system just like the microprocessor.
>
> The HP 64744 emulator does not support DMA (direct memory access) to emulation memory.

When bus arbitration is disabled:

> The emulator ignores the /BR and /BGACK signals from the target system.

Selecting the Emulation Monitor Program

This section shows you how to:

- Use the background monitor program.
- Use a foreground monitor program.

When you power up the emulator, or when you initialize it, the background monitor is used by default. You can also configure the emulator to use a foreground monitor. Before the background and foreground monitors are described, you should understand the foreground and background emulator modes as well as the function of the emulation monitor program.

The Background Emulator Mode

Background is the mode in which emulation processor execution does not appear normally on the emulator probe. Background cycles may be driven to the target system or hidden from the target system. When background cycles are driven, they appear as reads. When background cycles are hidden, the emulator appears to the target system to be in a suspended state. In background mode, the emulation microprocessor executes out of background memory.

The Foreground Emulator Mode

Foreground is the mode in which all emulation processor cycles appear on the emulation probe, and the emulator executes as if it were a real microprocessor. The emulator is in foreground when it executes user programs. In foreground mode, the emulation microprocessor typically executes out of target system or emulation memory.

The Function of the Monitor Program

The monitor program is the interface between the emulation system controller and the target system. The emulation system controller uses its own microprocessor to accept and execute emulation, system, and analysis commands. The monitor program is executed by the emulation microprocessor.

The monitor program makes possible emulation commands which access target system resources. (The only way to access target system resources is through the emulation processor.) For example, when you enter a command to modify target system memory, it is the execution of monitor program instructions that cause the new values to be written to target system memory.

When the emulation system controller recognizes that an emulation command needs to access target system resources, it writes a command code to a communications area and breaks the emulation processor execution into the monitor program. The monitor program reads this command (and any associated parameters) from the communications area and executes the appropriate instructions to access these target system resources.

The Background Monitor

On emulator power-up, or after initialization, the emulator uses the background monitor program. The background monitor program executes entirely in the *background* emulator mode. The background monitor does not occupy processor address space.

The Foreground Monitor

You can configure the emulator to use a foreground monitor program. When a foreground monitor is selected, it executes in the *foreground* emulator mode. The foreground monitor occupies processor memory space and executes as if it were part of the user program.

When you use a foreground monitor, breaks into the monitor still cause the emulator to execute a number of cycles in background. The difference between the foreground monitor and the background monitor is that when the background monitor is used, all monitor functions are executed in background; when the foreground monitor is used, the monitor functions are executed in foreground.

Two foreground monitor programs are included with the emulator on a floppy disk: the first can be assembled with the HP 64000 68000/10 Cross Assembler/Linker, and the second can be assembled with the HP AxLS 68000/10/20 Assembler/Linker/Librarian or with the Microtec Research 68000 assembler and linker.

You may customize the foreground monitor if necessary; however, you must maintain the basic communications protocol between the monitor and the emulation system controller. Comments in the monitor program source file detail sections that cannot be changed.

Comparison of Background and Foreground Monitor Programs

Monitor Program Characteristic	Background	Foreground
Takes up processor memory space	No	Yes, 4 Kbytes
Allows the emulator to respond to target system interrupts during monitor execution	No	Yes
Can be customized	No	Yes
Can be used when performing coordinated measurements with other emulators	Yes	No
Resident in emulator firmware	Yes	No, must be assembled, linked, and loaded

To use the background monitor program

1 Enter the **cf mon=bg** command.

2 Re-map memory.

When you select the background monitor program, a memory overlay is created and the background monitor is loaded into that area.

Changing the monitor configuration resets the memory map, so you must re-map memory.

To use a foreground monitor program

1 Edit the monitor program source file to define its base address.

2 Assemble and link the monitor program.

3 Enter the **cf mon=fg..<4K_boundary>@<function_code>** command.

4 Re-map memory.

5 Set the supervisor stack pointer value.

6 Load the monitor program absolute file.

7 Load the user program absolute file.

8 Modify the TRACE exception vector to point to the TRACE_ENTRY symbol in the monitor program.

The foreground monitor program's base address should be on any 4 Kbyte boundary (address ending in 000H) except 0H (since that's the location of the vector table). An ORG statement in the foreground monitor source file defines the base address. Also, the base address is specified when configuring the emulator to use a foreground monitor program.

You can only use function codes **x** (no function code) or **s** (supervisor space) when configuring for a foreground monitor. Remember to use the **cf lfc=<function_code>** before loading the foreground monitor program.

When you configure the emulator for a foreground monitor program, the memory map is reset, and a 4 Kbyte block of emulation memory is automatically mapped for the monitor program. You must re-map other memory ranges before loading user programs.

Because the monitor program is used when writing to target system memory, it should be loaded separately, before user programs that reside in target system memory.

In order to step through programs when using a foreground monitor, you **must** modify the TRACE vector (24H) in the processor's exception vector table. The TRACE exception vector must point to the TRACE_ENTRY label in the foreground monitor program.

Examples

The following examples of how to set up and use a foreground monitor program make the following assumptions:

- The HP 64700 is connected to the same LAN as an HP 9000 Series 300 host computer.
- The HP AxLS 68000/10/20 Assembler/Linker/Librarian and networking software are installed on your HP 9000 Series 300 host computer.
- The foreground monitor program source file exists on the host computer.

To edit the monitor program source file to specify its base address, 6000H in this example, you must modify the ORG statement near the top of the file:

```
ORG 006000H  * START MONITOR ON 4K BOUNDARY OTHER THAN ZERO
```

Notice that the ORG statement is indented from the left margin; if it is not indented, the assembler will interpret the ORG as a label and will generate an error when processing the address portion of the statement.

To assemble and link the monitor program, enter the following commands:

```
$ as68k -Lh fm64742.s > fm64742.lis <RETURN>

$ ld68k -c fm64742.k -Lh > fm64742.map <RETURN>
```

Where the "fm64742.k" linker command file contains:

```
name fm64742
load fm64742.o
end
```

To configure the emulator to use a foreground monitor program, enter the following command:

```
R>cf mon=fg..006000
```

The memory map is reset and a 4 Kbyte block of emulation memory (range 6000H through 6FFFH) is mapped for the foreground monitor program.

To map memory for the demo user program, enter the following commands:

```
R>map 0..0fff erom
R>map 40000..43fff eram
R>map 60000..63fff eram
```

To set the value of the supervisor stack pointer after emulation reset, enter the following command:

```
R>cf rssp=44000
```

To load the monitor program and user program absolute files, enter the following commands from the host computer:

```
$ ftp 15.35.226.210
Connected to 15.35.226.210.
220 User connected to HP64700
Name (15.35.226.210:guest): <RETURN>
Password (15.35.226.210:guest): <RETURN>
ftp> binary
200 Type set to I
ftp> put fm64742.X -h
200 Port      ok
150
226-
R>
226 Transfer completed
3332 bytes sent in 0.20 seconds (16.27 Kbytes/sec)
ftp> put cmd_rdr.X -h
200 Port      ok
150
226-
R>
226 Transfer completed
3332 bytes sent in 0.20 seconds (16.27 Kbytes/sec)
ftp> quit
221 Goodbye
$
```

To modify the TRACE exception vector to point to the TRACE_ENTRY symbol in the monitor program (so that the emulator can single-step), look up the address of TRACE_ENTRY in the monitor program list file and modify the TRACE exception vector (24H) with a command similar to:

```
M>m -dl 24=000064a6
```

Now, you are ready to use the emulator.

3

Using the Terminal Interface

Using the Terminal Interface

This chapter describes general tasks you may wish to perform while using the Terminal Interface, in other words, tasks that don't necessarily relate to using the emulator or the analyzer. These tasks are grouped into two sections:

- Accessing HP 64700 system information.
- Entering commands.

Accessing HP 64700 System Information

The HP 64700's Terminal Interface provides access to two types of system-wide information:

- Help information for the Terminal Interface commands.
- Software version number information for the products installed in the HP 64700 Card Cage.

To access on-line help information

- Use the **help** or **?** commands.

The HP 64700's Terminal Interface provides an on-line help command to provide you with quick information on the various commands and command syntax. From any system prompt, you can enter **help** or **?** as shown below.

Commands are grouped into various classes. To see the commands grouped into a particular class, you can use the **help** command with that group. Viewing the group help information in short form will cause the commands or the grammar to be listed without any description.

Help information exists for each command. Additionally, there is help information for each of the emulator configuration items. For example, to access the help information for the **rrt** configuration item, you can enter the **help cf rrt** command).

Examples

To display information on the help command:

```
M>help

  help  - display help information

    help <group>         - print help for desired group
    help -s <group>      - print short help for desired group
    help <command>       - print help for desired command
    help                 - print this help screen

  --- VALID <group> NAMES ---
    gram     - system grammar
    proc     - processor specific grammar

    sys      - system commands
    emul     - emulation commands
    trc      - analyzer trace commands
    xtrc     - external trace analysis commands
    *        - all command groups
```

To display information on the grammar used in the Terminal Interface:

```
M>help gram

gram - system grammar
-----------------------------------------------------------------------------
--- SPECIAL CHARACTERS ---
  #  - comment delimiter    ;  - command separator    Ctl C - abort signal
  {} - command grouping     "" - ascii string         ''    - ascii string
  Ctl R - command recall    Ctl B - recall backwards

--- EXPRESSION EVALUATOR ---
  number bases:  t-ten   y-binary   q-octal   o-octal   h-hex
  repetition and time counts default to decimal - all else default to hex
  operators:     ()  ~  *  /  %  +  -  <<  <<<  >>  >>>  &  ^  |  &&

--- PARAMETER SUBSTITUTION ---
  &token& - pseudo-parameter included in macro definition
          - cannot contain any white space between & pairs
          - performs positional substitution when macro is invoked
  Example
     Macro definition:  mac getfile={load -hbs"transfer -t &file&"}
     Macro invocation:  getfile MYFILE.o
     Expanded command:  load -hbs"transfer -t MYFILE.o"
```

To display information specific to the 68000 processor:

M>**help proc**

```
--- Address format ---
  24 bit address for memory with optional function codes
  address format is either XXXXXX or XXXXXX@fc
  where XXXXXX is a 24 bit address and
  where fc may be any of the following function codes
  x - no function codes
  sp - supervisor program                s - supervisor
  sd - supervisor data                   u - user
  up - user program                      p - program
  ud - user data                         d - data

--- Address range format ---
  24 bit address thru 24 bit address with optional function codes
  address range format is XXXXXX..XXXXXX or XXXXXX..XXXXXX@fc
  where XXXXXX is a 24 bit address and
  where fc may be any of the Address format function codes

--- Emulation Status Characters ---
  R - emulator in reset state           c - no target system clock
  U - running user program              r - target system reset active
  M - running monitor program           h - processor halted
  W - waiting for CMB to become ready   g - bus granted
  ? - unknown state                     b - no bus cycles
                                        w - cpu in wait

--- Equates for Analyzer Label stat ---
  supprog   - supervisor program        grd     - guarded memory
  supdata   - supervisor data           wrrom   - write to rom
  userprog  - user program              read    - memory read
  userdata  - user data                 write   - memory write
  cyc6800   - 6800 cycle                dma     - bus released to DMA device
  sup       - supervisor cycle          user    - user cycle
  prog      - program cycle             data    - data cycle
  byte      - byte cycle                word    - word cycle
  intack    - interrupt acknowledge
```

To display information on the emulator commands:

M>**help emul**

```
emul - emulation commands
-----------------------------------------------------------------------------
b......break to monitor  cp.....copy memory       mo.....modes
bc.....break condition   dump...dump memory       r......run user code
bp.....breakpoints       es.....emulation status  reg....registers
cf.....configuration     io.....input/output      rst....reset
cim....copy target image load...load memory       rx.....run at CMB execute
cmb....CMB interaction   m......memory            s......step
cov....coverage          map....memory mapper     ser....search memory
```

To display information on the **cf** command:

M>**help cf**

```
cf - display or set emulation configuration

  cf                - display current settings for all config items
  cf <item>         - display current setting for specified <item>
  cf <item>=<value> - set new <value> for specified <item>
  cf <item> <item>=<value> <item> - set and display can be combined

help cf <item>    - display long help for specified <item>

--- VALID CONFIGURATION <item> NAMES ---
  ba      - en/dis bus arbitration
  bat     - en/dis emulation analyzer tag of the bus arbitration handshake
  bbk     - select memory block during background operation
  be      - en/dis /BERR on non-interlocked emulation memory accesses
  bfc     - select function codes during background operation
  dbc     - en/dis drive of background cycles to the target system
  mondti  - en/dis /MONDTACK interlock
  lfc     - select function codes for file loading
  mon     - selection of a foreground or background monitor
  pdw     - 8/16 processor configuration for data bus width
  rrt     - en/dis restriction to real time runs
  rssp    - set SSP when monitor is entered from emulation reset
  swtp    - select trap for software breaks
  ti      - en/dis of target system interrupts
```

To display information on the **rrt** configuration item:

M>**help cf rrt**

```
Restrict to Real Time Runs

  cf rrt=en      enable
  cf rrt=dis     disable

When enabled and while the emulator is running the user program, any
  command that requires a break to the monitor will be rejected except
  'rst', 'b', 'r' or 's'.
When disabled, commands that require a break to the monitor will
  always be accepted.
```

To display version information

- Use the **ver** command.

The Terminal Interface provides the **ver** command if you need to check the software version numbers of the HP 64700 system and other products in the HP 64700.

Examples

To display version information:

M>**ver**

```
                  Copyright (c) Hewlett-Packard Co. 1987
All Rights Reserved.  Reproduction, adaptation, or translation without prior
written permission is prohibited, except as allowed under copyright laws.

  HP64700B Series Emulation System
    Version:   B.01.00 20Dec93
    Location:  Flash
    System RAM:1 Mbyte

  HP64744A (PPN: 64742A) Motorola 68000/68HC001/68EC000 Emulator
    Version:   A.01.00 13Mar92
    Probe:     Loopback Board
    Speed:     16 MHz
    Memory:    60 KBytes

  HP64740 Emulation Analyzer with External State/Timing Analyzer
    Version:   A.02.02 13Mar91
```

Entering Commands

This section describes tasks that are related to entering commands. Entering commands is easy: use the keyboard to type in the command and press the carriage return key. However, the Terminal Interface provides other features that make entering commands even easier. For example, you can:

- Enter multiple commands on one line.
- Recall commands.
- Edit commands.
- Repeat commands.
- Define macros, which save a set of commands for later execution.
- Use command files over LAN.

To enter multiple commands on one command line

- Separate the commands with semicolons (;).

More than one command may be entered in a single command line if the commands are separated by semicolons (;).

Examples

To step the next instruction and display the registers:

M>**s;reg**

```
0000a12@sp  -                MOVE.B  Cmd_Input,D2
PC = 0000a18@sp
reg pc=00000a18 st=2704 d0=00000000 d1=0000ffff d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00043fe3 a1=00000ad7
reg a2=00000992 a3=00000000 a4=00000000 a5=00068056 a6=00043fe4 a7=00043f78
reg usp=00000000 ssp=00043f78
```

To recall commands

- Press <CTRL>r.

You can press <CTRL>r to recall the commands that have just been entered. If you go past the command of interest, you can press <CTRL>b to move forward through the list of saved commands.

Examples

To recall and execute the last command press <CTRL>r and then press <RETURN>.

To edit commands

1 Use the **cl -e** command to enable the command line editor.

2 Use <CTRL>r to recall previous commands, or if you wish to edit the current command or search for a previous command, press <ESC> to enter the editing mode.

The Terminal Interface provides a command line editing feature. The editing mode commands are as follows.

Command	Description
<ESC>	enter command editing mode
i	insert before current character
a	insert after current character
x	delete current character
r	replace current character
dd	delete command line
D	delete to end of line
A	append to end of line
$	move cursor to end of line
0	move cursor to start of line
^	move cursor to start of line
h	move left one character
l	move right one character
k	fetch previous command
j	fetch next command
/<string>	find previous command in history matching <string>
n	fetch previous command matching <string>
N	fetch next command matching <string>

To repeat commands

- Use the **rep** command.

The **rep** command is helpful when entering commands repetitively. You can repeat the execution of macros as well as commands.

Examples

To cause the **s** and **reg** commands to be executed two times.

```
M>rep 2 {s;reg}
```

```
0000a18@sp  -                BEQ.B   0000a10
PC = 0000a10@sp
reg pc=00000a10 st=2704 d0=00000000 d1=0000ffff d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00043fe3 a1=00000ad7
reg a2=00000992 a3=00000000 a4=00000000 a5=00068056 a6=00043fe4 a7=00043f78
reg usp=00000000 ssp=00043f78
0000a10@sp  -                NOP
PC = 0000a12@sp
reg pc=00000a12 st=2704 d0=00000000 d1=0000ffff d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00043fe3 a1=00000ad7
reg a2=00000992 a3=00000000 a4=00000000 a5=00068056 a6=00043fe4 a7=00043f78
reg usp=00000000 ssp=00043f78
```

To enter multiple commands with macros

1 Define the macro with the **mac** command.

2 Execute the defined macro.

If you wish to enter the same set of commands at various times while you use the emulator, you can assign these commands to a macro and enter the macro instead of the set of commands.

Examples

To define a macro that will display registers after every step, enter the following command.

```
M>mac st={s;reg}
```

To execute the macro, enter it as you would any other command.

```
M>st
```

```
# s ; reg
0000a12@sp  -                MOVE.B  Cmd_Input,D2
PC = 0000a18@sp
reg pc=00000a18 st=2704 d0=00000000 d1=0000ffff d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00043fe3 a1=00000ad7
reg a2=00000992 a3=00000000 a4=00000000 a5=00068056 a6=00043fe4 a7=00043f78
reg usp=00000000 ssp=00043f78
```

To use command files over LAN

1 Using **ftp -in** and the ftp command "cd <parameter>", copy information from the HP 64700 to the host computer.

2 Using **ftp -in** and the ftp command "cd <parameter>", copy information from the host computer to the HP 64700.

The **ftp** software in the HP 64700 responds to the ftp command "cd <parameter>" by executing the parameter as a Terminal Interface command.

By using **ftp -in** (refer to the **ftp** documentation for option details), you can send multiple Terminal Interface commands to a HP 64700 on the LAN.

For example, when entered from a UNIX workstation on the same LAN as the HP 64700 named *hostname*, the following command will display emulator version information:

```
$ echo "open hostname\nverbose\ncd ver\nquit" | ftp -in
```

If the Terminal Interface command you wish to execute has arguments, you must enclose the command and its arguments in quotes. For example:

```
$ echo "open hostname\nverbose\ncd \"help ver\"\nquit" | ftp -in
```

In order for these commands to work properly, there must not be an open telnet connection to the HP 64700.

Examples

The following example assumes the HP 64700 is connected to the same LAN as the UNIX host computer.

To save the emulator configuration, memory map, and other emulator settings, create a configuration file by entering the following commands on the UNIX workstation:

```
$ echo "open hostname" > cfg_file
$ echo "open hostname\nverbose\ncd cf\ncd map\ncd equ\ncd mac\nquit"
| ftp -in | grep '^  ' | awk '{printf "cd \"%s\"\n", $0}' >> cfg_file
$ echo "quit" >> cfg_file
```

To restore the emulator configuration information saved in "cfg_file", enter the following command:

```
$ ftp -in < cfg_file
```

4

Using the Emulator

Using the Emulator

This chapter describes general tasks you may wish to perform while using the emulator. These tasks are grouped into the following sections:

- Initializing the emulator.
- Mapping emulation and target system memory.
- Loading absolute files.
- Loading and using symbols.
- Executing user programs (starting, stopping, stepping, and resetting the emulator).
- Using software breakpoints.
- Enabling and disabling break conditions.
- Accessing registers.
- Accessing memory.

Initializing the Emulator

This section shows you how to:

- Initialize the emulator.
- Display emulator status information.

To initialize the emulator

- To perform a limited initialization, enter the **init** command.
- To perform a complete initialization without system verification, enter the **init -c** command.
- To perform a complete initialization with system verification, enter the **init -p** command.
- To perform a complete initialization without optional product verification, enter the **init -r** command.

The **init** command with no options causes a limited initialization. A limited initialization does not affect system configuration. However, the **init** command will reset emulator and analyzer configurations. The **init** command:

- Resets the memory map.
- Resets the emulator configuration items.
- Resets the break conditions.
- Clears breakpoints.

The **init** command does not:

- Clear any macros.

- Clear any emulation memory locations; mapper terms are deleted, but if you re-specify the exact same memory map, you will find that the emulation memory contents are the same.

The **-c** option specifies a complete initialization (except system verification tests are not run).

The **-p** option specifies a complete initialization with system verification tests. The **-p** initialization sequence includes emulator, analyzer, system controller, communications port, LAN interface, and flash EPROM initialization.

The **-r** option specifies a complete initialization with system verification tests (as with **-p**), but all optional products are ignored. Do not use flash ROM.

Examples

To perform a limited initialization:

```
R>init
  # Limited initialization completed
```

To display emulator status information

- Enter the **es** command.

or

- Enter the **help proc** command.

The Terminal Interface prompt displays an emulator status character. You can find the meaning of the emulator status character in one of two ways: with the **help proc** command or with the **es** command.

Examples

To use the **help proc** command:

```
R>help proc
.
.
.

  --- Emulation Status Characters ---
    R - emulator in reset state           c - no target system clock
    U - running user program              r - target system reset active
    M - running monitor program           h - processor halted
    W - waiting for CMB to become ready   g - bus granted
    ? - unknown state                     b - no bus cycles
                                          w - cpu in wait

.
.
.
```

To use the **es** command:

```
R>es
  M68000--Emulation reset
```

Mapping Memory

Because the emulator can use target system memory or emulation memory (or both), it is necessary to map ranges of memory so that the emulator knows where to direct its accesses.

Up to 16 ranges of memory can be mapped, and the resolution of mapped ranges is 4 Kbytes bytes (that is, the memory ranges must begin on 4 Kbyte boundaries and must be at least 4 Kbytes in length).

The emulator contains 60 Kbytes of base emulation memory and provides two slots for additional emulation memory modules. The amount of emulation memory that can be mapped depends on the number, and size, of memory modules installed on the emulator board.

<table>
<tr><th colspan="2" rowspan="2">Amount of emulation memory</th><th colspan="3">Number of 256 Kbyte memory modules</th></tr>
<tr><th>0</th><th>1</th><th>2</th></tr>
<tr><th rowspan="3">Number of 1 Mbyte memory modules</th><th>0</th><td>60 Kbytes</td><td>316 Kbytes</td><td>572 Kbytes</td></tr>
<tr><th>1</th><td>1084 Kbytes</td><td>1340 Kbytes</td><td></td></tr>
<tr><th>2</th><td>2108 Kbytes</td><td></td><td></td></tr>
</table>

Direct memory access (DMA) to emulation memory is not permitted.

You should map all memory ranges used by your programs before loading programs into memory.

To map memory ranges

- Use the **map <range> <type> <attrib>** command.

The <type> parameter allows you to characterize the memory range as emulation RAM, emulation ROM, target system RAM, target system ROM, or as guarded memory.

Guarded memory accesses will cause emulator execution to break into the monitor program.

Writes to locations characterized as ROM will cause emulator execution to break into the monitor program if the **rom** break condition is enabled (**bc -e rom**).

Writes to emulation ROM will not modify memory. Writes by user code to target system memory locations that are mapped as ROM or guarded memory will result in a break to the monitor but are not inhibited (that is, the write still occurs).

Emulation memory ranges can have an <attrib> parameter. The **dti** attribute specifies that accesses in that range be synchronized with the target system. This means the termination of accesses in the range will not occur until the target system provides a /DTACK or /VPA. A /BERR signal from the target system will also terminate an emulation memory cycle and cause the emulator to begin execution of the bus error handler.

For emulation memory accesses that are not synchronized to the target system (that is, accesses to ranges that are mapped without the **dti** attribute), you can either allow the emulator to respond to target system /BERR signals by entering the **cf be=en** command, or cause the emulator to ignore target system /BERR signals by entering the **cf be=dis** command.

Examples

Consider the following section summary from the linker load map output listing.

```
SECTION SUMMARY
---------------

SECTION    ATTRIBUTE                      START      END        LENGTH     ALIGN

           ABSOLUTE DATA                  00000000   0000002F   00000030   0 (BYTE)
0          NORMAL                         00000030   00000030   00000000   2 (WORD)
env        NORMAL CODE                    00000400   0000098A   0000058B   2 (WORD)
prog       NORMAL CODE                    0000098C   00000A73   000000E8   2 (WORD)
const      NORMAL ROM                     00000A74   00000AD6   00000063   2 (WORD)
lib        NORMAL CODE                    00000AD8   00000E39   00000362   2 (WORD)
libc       NORMAL CODE                    00000E3A   00000E99   00000060   2 (WORD)
libm                                      00000E9A   00000E9A   00000000   0 (BYTE)
mon        NORMAL CODE                    00000E9A   00000FE3   0000014A   2 (WORD)
stack      NORMAL DATA                    00040000   00043FFF   00004000   4 (LONG)
envdata    NORMAL DATA                    00060000   00060055   00000056   4 (LONG)
data       NORMAL DATA                    00060056   00060076   00000021   2 (WORD)
idata                                     00060077   00060077   00000000   0 (BYTE)
udata                                     00060077   00060077   00000000   0 (BYTE)
libdata    NORMAL DATA                    00060078   0006007B   00000004   4 (LONG)
libcdata   NORMAL DATA                    0006007C   00060106   0000008B   2 (WORD)
mondata    NORMAL DATA                    00060108   0006012B   00000024   2 (WORD)
heap       NORMAL DATA                    0006012C   00063129   00002FFE   4 (LONG)
```

Notice the ABSOLUTE DATA, CODE, and ROM sections occupy locations 0 through 0FE3H. Because the contents of these sections will eventually reside in target system ROM, this area should be characterized as ROM when mapped. This will prevent these locations from being written over accidentally. If the **rom** break condition is enabled, instructions that attempt to write to these locations will cause emulator execution to break into the monitor.

Also, notice the DATA sections occupy locations 40000H through 43FFFH and 60000H through 63129H. Since these sections are written to, they should be characterized as RAM when mapped.

To map emulation memory for the above program, you would enter the following **map** commands.

R>**map 0..0fff erom**

R>**map 40000..43fff eram**

R>**map 60000..63fff eram**

To display the memory map

- Enter the **map** command with no parameters.

Examples

To view the memory map, enter the **map** command with no parameters.

```
R>map
  # remaining number of terms   : 13
  # remaining emulation memory  : 6000h bytes
  map  0000000..0000fff     erom          # term 1
  map  0040000..0043fff     eram          # term 2
  map  0060000..0063fff     eram          # term 3
  map  other tram
```

To characterize unmapped ranges

- Enter the **map other** command.

The default characterization for unmapped memory ranges are treated as target system RAM.

The "other" term cannot be mapped as emulation memory.

Examples

To characterize unmapped ranges as target RAM:

```
R>map other tram
```

To characterize unmapped ranges as guarded memory:

```
R>map other grd
```

To delete memory map ranges

- Enter the **map -d** command.

Note that programs should be reloaded after deleting mapper terms. The memory mapper may re-assign blocks of emulation memory after the insertion or deletion of mapper terms.

Examples

To delete term 1 in the memory map:

```
R>map -d 1
```

To delete all map terms:

```
R>map -d *
```

To map memory ranges that use function codes

- Specify function codes with address ranges when mapping memory.

If you specify a function code when mapping a range of memory, you must include the function code when referring to locations in that range. If you don't include the function code, an "ambiguous address" error message is displayed.

If you use different function codes, it's possible to map address ranges that overlap. When address ranges with different function codes overlap, you must load a separately linked module for the space associated with each function code. The modules are linked separately because linker errors occur when address ranges overlap.

When address ranges are mapped with different function codes, and there are no overlapping ranges, your program modules may exist in one absolute file. However, you have to use multiple load commands—one for each function code specifier. This is necessary to load the various sections of the absolute file into the appropriate function code qualified memory ranges. When you do this, be sure that all address ranges not mapped (that is, the "other" memory mapper term) are mapped as target RAM. When "other" is mapped as guarded, guarded memory access errors (from the attempt to load the absolute file sections that are outside the specified function code range) can prevent the absolute file sections that are inside the specified function range from being loaded.

Examples

Suppose you're developing a system with the following characteristics:

- Input port at 100 hex.
- Output port at 400 hex.
- Supervisor program from 1000 through 1fff hex.
- Supervisor data from 2000 through 2fff hex.
- User program from 3000 through 3fff hex.
- User data from 3000 through 3fff hex.

Notice that the last two terms have address ranges that overlap. You can use function codes to cause these terms to be mapped to different blocks of memory.

Suppose also that the only things that exist in your target system at this time are the input and output ports and some control logic; no memory is available. You can reflect this by mapping the I/O ports to target system memory space and the rest of memory to emulation memory space:

```
R>map 0..0fff tram
R>map 1000..1fff@sp erom
R>map 2000..2fff@sp eram
R>map 3000..3fff@up eram
R>map 3000..3fff@ud eram
R>map
  # remaining number of terms   : 11
  # remaining emulation memory  : b000h bytes
  map  0000000..0000fff     tram          # term 1
  map  0001000..0001fff@sp  erom          # term 2
  map  0002000..0002fff@sp  eram          # term 3
  map  0003000..0003fff@up  eram          # term 4
  map  0003000..0003fff@ud  eram          # term 5
  map  other tram
```

Notice that the mapper reserved two different spaces for the user program and user data areas even though the addresses are the same.

To display a byte of memory at 1000H:

```
R>m -db 1000
!ERROR  312! Ambiguous address: 0001000
R>m -db 1000@sp
  0001000..0001000@sp  00
```

Loading Absolute Files

This section describes the tasks related to loading absolute files into the emulator. You can load absolute files when using the serial connection or when using the LAN connection:

- When using the serial connection, the HP 64700 is connected to a host computer and accessed via a terminal emulation program. In this configuration, you can load absolute files by downloading from the same port.
- When using the LAN connection, the HP 64700 is connected to the same LAN as a computer that has the ARPA Services File Transfer Protocol (ftp) software. In this configuration, you can use the **ftp** command to load absolute files over the LAN.

The Terminal Interface's **load** command will accept absolute files in the following formats:

- HP absolute.
- Intel hexadecimal.
- Extended Tektronix hexadecimal.
- Motorola S-records.

Some 68000 software development tools generate IEEE-695 format absolute files. You can convert IEEE-695 format files to HP format files on HP 9000 Series 300 host computers with the **symconv** utility which is part of the HP 64888 File Format Conversion Utilities product.

To load absolute files over the serial port

- Load the file over the "command" port.

If the emulator is connected to a host computer, and you are accessing the emulator from the host computer via a terminal emulation program, you can also download files with the **load** command. In this configuration, files are loaded from the same port that commands are entered from.

Examples

To download a Tektronix hexadecimal file from a Vectra personal computer:

R>**load -t** <RETURN>

After you have entered the **load** command, exit from the terminal emulation program to the MS-DOS operating system. Then, copy your hexadecimal file to the port connected to the emulator, for example:

```
C:\copy thexfile com1: <RETURN>
```

Now you can return to the terminal emulation program and verify that the file was loaded (for example, by displaying memory in mnemonic format).

To load absolute files over the LAN

- Use the **ftp** command on your local host computer to transfer files to the remote HP 64700.

When connecting to the HP 64700’s ftp interface, you can use either the HP 64700’s hostname or the Internet Protocol (IP) address (or internet address). When you use the HP 64700’s hostname, the ftp software on your computer will look up the internet address in the hosts table, or perhaps a name server will return the internet address.

Examples

To connect to the emulator's ftp interface, enter the following command (use any name and password):

```
$ ftp 15.35.226.210
Connected to 15.35.226.210.
220 User connected to HP64700
Name (15.35.226.210:guest):
Password (15.35.226.210:guest):
230-
                              NOTICE

   This utility program is unsupported. It is provided at no cost.
   Hewlett-Packard makes no warranty on its quality or fitness for
   a particular purpose.

 FTP on the HP64700 serves as a means for downloading absolute files to the
 emulation environment. The file transfer can be be performed as follows:

 1. The data mode type must be set to IMAGE (binary)

 2. Store the file using options to indicate the file format.  The following
 example uses PUT as the host command for sending the file. This may be
 different for your ftp implementation.

 put <file_name> <options>
     <file_name> - host file to be loaded.
     <options>   - The options are preceeded by a minus (-). The available
     options vary for individual emulators. All support HP OLS, Intel hex,
     Motorola S-records, and Extended Tek Hex.  Emulator specific options can
     be viewed by issuing a Terminal Mode help for the load command.

     put hpfile.X  -h  #to download an HP OLS file
     put intelfile -i  #to download an Intel Hex file
     put motfile   -m  #to download a Motorola S-record file
     put tekfile   -t  #to download an Extended Tek Hex file

230
```

To set up ftp for binary file transfers:

```
ftp> binary
200 Type set to I
```

To download the HP 64000 format absolute file into the emulator:

```
ftp> put cmd_rdr.X -h
200 Port      ok
150
226-
R>
226 Transfer completed
3332 bytes sent in 0.20 seconds (16.27 Kbytes/sec)
```

To exit out of the ftp interface:

```
ftp> quit
221 Goodbye
$
```

To load absolute files into memory mapped with function codes

1 Enter the **cf lfc=<function_code>** command.

2 Load the absolute file.

3 Repeat steps 1 and 2 for each range mapped with a different function code.

Absolute files are only loaded into the address ranges that are mapped with the function code specified. For example, if you enter the **cf lfc=s** command, subsequent **load** commands only load the absolute file into memory ranges designated as "supervisor" space (that is, mapped with **map <address>..<address>@s** commands).

The function code can be:

x (no function code is used — absolute files are loaded into memory ranges that have been mapped without function codes).

s (supervisor space).

u (user space).

p (program space).

d (data space).

sp (supervisor program space).

sd (supervisor data space).

up (user program space).

ud (user data space).

To simplify loading of many modules, you can include the above sequence in macros defined with the **mac** command or in host computer resident command files.

Examples

To load the portions of the absolute file that reside in supervisor memory space:

```
M>cf lfc=s
M>load -hbs "transfer -tb absfile.X"<ESC>g
```

Loading and Using Symbols

The emulator supports the use of symbolic references. The symbols can be loaded from an ASCII text file on a host computer or defined with the **sym** command.

This section describes the tasks related to loading ASCII symbol files into the emulator. ASCII symbol files must be loaded from a host computer.

Symbols will be shown when you display memory in mnemonic format. Also, you can use the **-s** or **-e** options to the trace list command (**tl**) to have symbolic information included in the trace list.

You can typically use symbol table information from a linker map file when creating the ASCII symbol file; however, you need to make sure the information is in the following format.

```
#
:global_sybmol
module:local_symbol
.
.
.
#
```

This section describes how to:

- Load symbol files over the serial port.
- Load symbol files over the LAN.
- Define user symbols.
- Display symbols.
- Remove symbols.

To load symbol files over the serial port

- Use the **load -S** command.

ASCII symbol files are loaded into the emulator with the **load -S** command.

Examples

Suppose the "cmd_rdr.sym" file below exists on a Vectra personal computer.

```
#
:Cmd_Input                      00060056
:Msg_Dest                       00060057
:Write_Msg                      00000992
:main                           000009BE
atexit:__exec_funcs             00000E6C
atexit:__top_of_func_stack      000600FC
atexit:_atexit                  00000E3A
cmd_rdr:_Cmd_Input              00060056
cmd_rdr:_Msg_Dest               00060057
cmd_rdr:_Write_Msg              00000992
cmd_rdr:_main                   000009BE
crt1:__exit                     0000047E
crt1:_exit                      0000046A
crt1:entry                      00000400
data_gen:__infinity             00000E92
data_gen:__malloc_init          00060106
data_gen:__rand_seed            000600FE
data_gen:_errno                 00060102
disp_msg:XEnv_68k_except        00060034
disp_msg:__display_message      00000534
disp_msg:end_of_program         00000562
fperror:__fp_control            0006007A
fperror:__fp_error              00000AD8
fperror:__fp_errorf             00000AD8
fperror:__fp_errori             00000AD8
fperror:__fp_status             00060078
getmem:__getmem                 0000050A
getmem:nextblk                  00060030
mon_stub:JSR_ENTRY              00000FE0
mon_stub:MONITOR_MESSAGE        00060128
sysheap:TopOfHeap               0006312A
sysheap:heap                    0006012C
sysstack:TopOfStack             00044000
sysstack:stack                  00040000
systrap:trap                    000005F2
#
```

To load symbols from the ASCII file above:

```
R>load -S <RETURN>
```

After you have entered the **load** command, exit from the terminal emulation program to the MS-DOS operating system. Then, copy your symbols file to the port connected to the emulator, for example:

```
C:\copy cmd_rdr.sym com1: <RETURN>
```

To load symbols over the LAN

- Use the **ftp** command on your local host computer to transfer files to the remote HP 64700.

Loading symbol files over the LAN is the same as loading absolute files over the LAN, except that a different option is used with the "put" command in ftp.

Examples

To connect to the emulator's ftp interface, enter the following command (use any name and password):

```
$ ftp 15.35.226.210
Connected to 15.35.226.210.
220 User connected to HP64700
Name (15.35.226.210:guest):
Password (15.35.226.210:guest):
230-
                              NOTICE

   This utility program is unsupported. It is provided at no cost.
   Hewlett-Packard makes no warranty on its quality or fitness for
   a particular purpose.

.
.
.
```

To set up ftp for binary file transfers:

```
ftp> binary
200 Type set to I
```

To download the symbol file into the emulator:

```
ftp> put cmd_rdr.sym -S
200 Port      ok
150
226-
R>
226 Transfer completed
1789 bytes sent in 4.78 seconds (0.37 Kbytes/sec)
```

To exit out of the ftp interface:

```
ftp> quit
221 Goodbye
$
```

To define user symbols

- Use the **sym <name>=<addr>** command.

You can use the **sym** command to define new symbols.

Examples

To define the symbol "while_statement" for the address 0A10H:

```
M>sym while_statement=0a10
```

To display symbols

- Use the **sym** command.

After symbols are loaded, you can use the **sym** command to display and delete symbols, as well as to define new symbols.

Examples

To display all the symbols:

```
M>sym
  sym while_statement=0000a10
  sym Cmd_Input=0060056
  sym Msg_Dest=0060057
  sym Write_Msg=0000992
  sym main=00009be
  sym atexit:__exec_funcs=0000e6c
  sym atexit:__top_of_func_stack=00600fc
  sym atexit:_atexit=0000e3a
  sym cmd_rdr:_Cmd_Input=0060056
  sym cmd_rdr:_Msg_Dest=0060057
  sym cmd_rdr:_Write_Msg=0000992
  sym cmd_rdr:_main=00009be
```

```
  sym crt1:__exit=000047e
  sym crt1:_exit=000046a
  sym crt1:entry=0000400
  sym data_gen:__infinity=0000e92
  sym data_gen:__malloc_init=0060106
  sym data_gen:__rand_seed=00600fe
  sym data_gen:_errno=0060102
  sym disp_msg:XEnv_68k_except=0060034
  sym disp_msg:__display_message=0000534
  sym disp_msg:end_of_program=0000562
  sym fperror:__fp_control=006007a
  sym fperror:__fp_error=0000ad8
  sym fperror:__fp_errorf=0000ad8
  sym fperror:__fp_errori=0000ad8
  sym fperror:__fp_status=0060078
  sym getmem:__getmem=000050a
  sym getmem:nextblk=0060030
  sym mon_stub:JSR_ENTRY=0000fe0
  sym mon_stub:MONITOR_MESSAGE=0060128
  sym sysheap:TopOfHeap=006312a
  sym sysheap:heap=006012c
  sym sysstack:TopOfStack=0044000
  sym sysstack:stack=0040000
  sym systrap:trap=00005f2
```

To display all the global symbols:

```
M>sym -g
  sym Cmd_Input=0060056
  sym Msg_Dest=0060057
  sym Write_Msg=0000992
  sym main=00009be
```

To display all the local modules:

```
M>sym -l
  sym atexit:
  sym cmd_rdr:
  sym crt1:
  sym data_gen:
  sym disp_msg:
  sym fperror:
  sym getmem:
  sym mon_stub:
  sym sysheap:
  sym sysstack:
  sym systrap:
```

To display all the user-defined symbols:

```
M>sym -u
  sym while_statement=0000a10
```

To remove symbols

- Use the **sym -d** command.

You can use the **sym -d** command to delete symbols.

Examples

To delete all user symbols:

```
R>sym -du
```

To delete all global symbols:

```
R>sym -dg
```

To delete all local symbols in all modules:

```
R>sym -dl
```

To delete all symbols:

```
R>sym -d
```

Executing User Programs

This section describes how to:

- Start the emulator running the user (target system) program.
- Stop (break from) user program execution.
- Step through user programs.
- Reset the emulation processor.

To run (execute) user programs

- Use the **r** command.

The run command causes the emulator to execute the user program. When the emulator is executing the user program, the "U" emulator status character is shown in the Terminal Interface prompt.

The **r rst** (run from reset) command specifies a run from target system reset. It is equivalent to entering a **rst** (reset) command followed by a **r** (run) command. The processor will be reset and then allowed to run.

Examples

To run from the current program counter:

```
R>r
U>
```

To run from address 400H:

```
M>r 400
U>
```

To stop (break from) user program execution

- Use the **b** command.

You can use the break command (**b**) command to generate a break to the background monitor.

The "Using Software Breakpoints" section of this chapter describes how to stop execution at particular points in the user program.

Examples

To break execution into the monitor:

```
U>b
M>
```

To break from reset into the monitor:

```
R>b
M>
```

To step through user programs

- Use the **s** command.

The emulator allows you to step through the user program. You can step from the current program counter (in other words, instruction pointer) or from a particular address. You can step a single instruction or a number of instructions.

A step count of 0 will cause the stepping to continue "forever" (until some break condition, such as "write to ROM", is encountered, or until you enter <CTRL>c).

If a foreground monitor is selected, the target system trace vector must point to TRACE_ENTRY in the foreground monitor code for single step to function properly.

Examples

To step one instruction from the current program counter:

```
M>s
  0000a12@sp  -                    MOVE.B  Cmd_Input,D2
  PC = 0000a18@sp
```

To step a number of instructions from the current program counter:

```
M>s 8
  0000a18@sp  -                    BEQ.B   0000a10
  0000a10@sp  -                    NOP
  0000a12@sp  -                    MOVE.B  Cmd_Input,D2
  0000a18@sp  -                    BEQ.B   0000a10
  0000a10@sp  -                    NOP
  0000a12@sp  -                    MOVE.B  Cmd_Input,D2
  0000a18@sp  -                    BEQ.B   0000a10
  0000a10@sp  -                    NOP
  PC = 0000a12@sp
```

To step a number of instructions from a specified address:

```
M>s 16 main
  00009be  main                 LINK    A6,#0ff9c
  00009c2@sp  -                    MOVE.L  A2,-[A7]
  00009c4@sp  -                    MOVE.L  D2,-[A7]
  00009c6@sp  -                    MOVEA.L #000000992,A2
  00009cc@sp  -                    LEA.L   0ff9c[A6],A0
  00009d0@sp  -                    MOVEA.L #000000a74,A1
  00009d6@sp  -                    MOVEQ.L #000000020,D1
  00009d8@sp  -                    MOVE.B  [A1]+,[A0]+
  00009da@sp  -                    DBF     D1,00009d8
  00009d8@sp  -                    MOVE.B  [A1]+,[A0]+
  00009da@sp  -                    DBF     D1,00009d8
  00009d8@sp  -                    MOVE.B  [A1]+,[A0]+
  00009da@sp  -                    DBF     D1,00009d8
  00009d8@sp  -                    MOVE.B  [A1]+,[A0]+
  00009da@sp  -                    DBF     D1,00009d8
  00009d8@sp  -                    MOVE.B  [A1]+,[A0]+
  PC = 00009da@sp
```

To step until <CTRL>c:

```
M>s 0
  00009da@sp  -                DBF     D1,00009d8
  00009d8@sp  -                MOVE.B  [A1]+,[A0]+
  00009da@sp  -                DBF     D1,00009d8
  00009d8@sp  -                MOVE.B  [A1]+,[A0]+
  00009da@sp  -                DBF     D1,00009d8
  .
  .
  .
<CTRL>c
  .
  .
  .
  0000a02@sp  -                NOP
  0000a04@sp  -                MOVE.B  #000,Cmd_Input
  0000a0c@sp  -                BRA.B   0000a12
  0000a12@sp  -                MOVE.B  Cmd_Input,D2
  0000a18@sp  -                BEQ.B   0000a10
  PC = 0000a12@sp
!STATUS  685! Stepping aborted
```

To reset the emulation processor

- Use the **rst** command.

The **rst** command causes the processor to be held in a reset state until a **b** (break), **r** (run), or **s** (step) command is entered. A CMB execute signal will also cause the emulator to run if reset. Also, a request to access target memory while reset will cause a break into the monitor.

The **-m** option to the **rst** command specifies that the emulator begin executing in the monitor after reset.

Examples

To reset the emulation processor:

```
U>rst
R>
```

To reset the emulation processor and break into the monitor:

```
U>rst -m
M>
```

Using Software Breakpoints

Software breakpoints provide a way to accurately stop the execution of your program at selected locations.

When you set a software breakpoint at an address, the instruction at that address is replaced with a TRAP instruction. (You specify which TRAP instruction (0 through 15) the emulator uses by setting an emulator configuration option.) When the TRAP instruction is executed, control is passed to the emulator's monitor program, and the original instruction is restored in the user program.

In order to successfully set a software breakpoint, the emulator must be able to write to the memory location specified. Therefore, software breakpoints cannot be set in target ROM. You must use the breakpoint registers to set breakpoints in target ROM.

Another way to break user program execution at a certain point is to break on the analyzer trigger.

This section shows you how to:

- Specify which TRAP instruction is used for software breakpoints.
- Enable the breakpoints feature.
- Set software breakpoints.
- Display software breakpoints.
- Enable software breakpoints.
- Disable software breakpoints.
- Remove software breakpoints.
- Disable the breakpoints feature.

CAUTION Software breakpoints should not be set, cleared, enabled, or disabled while the emulator is running user code. If any of these commands are entered while the emulator is running user code, and the emulator is executing code in the area where the breakpoint is being modified, program execution may be unreliable.

To specify which TRAP instruction is used for software breakpoints

- Enter the **cf swtp=<number>** command.

The software trap number must be from 0 through 0FH.

When you define a software breakpoint at some address, the opcode at that address is replaced by the TRAP instruction.

When a TRAP instruction is executed, the emulator breaks into the monitor. Since the emulator knows the locations of defined software breakpoints, it can determine whether the TRAP was generated by an enabled software breakpoint or a TRAP instruction in the user program.

If the TRAP instruction was inserted as a software breakpoint, the TRAP instruction is replaced by the original opcode. A subsequent run or step command will execute from this address.

If the TRAP instruction is part of the user program, an "undefined breakpoint" message is displayed. To continue program execution, you must run or step from the user program's TRAP vector address.

When you change the value assigned to the **swtp** configuration item, any software breakpoints currently defined with the **bp** command are disabled (since the software trap instructions currently in memory may differ from the new value specified).

Examples

To specify that the TRAP #0FH instruction be used for the software breakpoint feature:

```
M>cf swtp=0f
```

To enable the breakpoints feature

- Enter the **bc -e bp** command.

Currently defined breakpoints are not automatically enabled when you enable the breakpoints feature; you must explicitly enable the software breakpoints.

To set software breakpoints

- Use the **bp <addr>** command.

Note that you must only set software breakpoints at memory locations which contain instruction opcodes (not operands or data).

Examples

To set a software breakpoint at address 0A10H:

```
M>bp 0a10
```

To display software breakpoints

- Enter the **bp** command with no options.

The **bp** command with no options displays the software breakpoints list. Also, it shows whether the breakpoint feature is enabled or disabled.

Examples

To display the software breakpoint list:

```
M>bp
  ### BREAKPOINT FEATURE IS ENABLED ###
  bp 0000a10 # enabled
```

To enable software breakpoints

- Use the **bp -e <addr>** command.

When a breakpoint is hit, it becomes disabled. You can use the **-e** option to the **bp** command to re-enable the software breakpoint.

Examples

To enable the software breakpoint at 0A10H:

```
M>bp -e 0a10
```

To enable all software breakpoints:

```
M>bp -e *
```

To disable software breakpoints

- Use the **bp -d <addr>** command.

When a breakpoint is hit, it becomes disabled. You can also disable breakpoints before they are hit (while they are enabled) by using the **-d** option to the **bp** command.

Examples

To disable the software breakpoint at 0A10H:

```
M>bp -d 0a10
```

To remove software breakpoints

- Use the **bp -r <addr>** command.

You can remove existing breakpoints by using the **-r** option to the **bp** command.

Examples

To remove the software breakpoint at 0A10H from the breakpoint list:

```
M>bp -r 0a10
```

To disable the breakpoints feature

- Enter the **bc -d bp** command.

All breakpoints are disabled, but they remain defined.

Using Break Conditions

Break conditions allow you to specify breaks in the user program when certain events occur during emulation processor execution. (You can also break user program execution when certain events occur in another HP 64700; these break conditions are described in the "Making Coordinated Measurements" chapter.)

The **bc** command lets you enable or disable breaks on the following conditions:

- Writes to ROM.
- Analyzer trigger.

To break on writes to ROM

- Enter the **bc -e rom** command.

When the **rom** break condition is enabled, the emulator will break from user program execution when a write occurs to a memory location mapped as ROM.

Examples

To enable breaks on writes to ROM:

```
M>bc -e rom
```

To disable breaks on writes to ROM:

```
M>bc -d rom
```

When disabled, the emulator will not break to the monitor upon a write to ROM; however, it will not modify the memory location if the memory at that location is actually RAM.

To break on an analyzer trigger

1 Specify internal signal for analyzer to drive.

2 Enable the emulator break on the internal signal.

Use the **tgout** (trigger output) command to specify which signal is driven when the emulation analyzer triggers.

Use the **xtgout** (external trigger output) command to specify which signal is driven when the external analyzer, configured as an independent state analyzer, triggers.

Either the "trig1" or the "trig2" internal signals can be driven on the trigger.

Note that the actual break may be several cycles after the analyzer trigger.

After the break occurs, the analyzer will stop driving the **trig** line that caused the break. Therefore, if **trig2** is used both to break and to drive the CMB TRIGGER (for example), TRIGGER will go true when the trigger is found and then will go false after the emulator breaks. However, if **trig2** is used to cause the break and **trig1** is used to drive the CMB TRIGGER, TRIGGER will stay true after the trigger until the trace is halted or until the next trace starts.

Examples

To break on the emulation analyzer trigger (over the internal trig2) signal:

```
M>tg any
M>tgout trig2
M>bc -e trig2
M>r 400
U>t
  Emulation trace started
U>es
  M68000--Running in monitor
!ASYNC_STAT  619! trig2 break
M>
```

To break on the external analyzer trigger (over the internal trig2) signal:

```
M>xtmo -s
M>xtg any
M>xtgout trig2
M>bc -e trig2
M>r 400
U>xt
  External trace started
U>es
  M68000--Running in monitor
!ASYNC_STAT  619! trig2 break
M>
```

To disable breaks on the internal trig2 signal:

```
M>bc -d trig2
```

Accessing Registers

This section describes tasks related to displaying and modifying emulation processor registers.

You can display the contents of an individual register or of all the registers.

Refer to the **reg** command description in the "Commands" chapter for a description of the 68000 registers.

To display register contents

- Use the **reg** command.

When displaying registers, you can display classes of registers and individual registers.

Examples

To display the basic register contents:

```
M>reg
```

```
reg pc=00000a12 st=2704 d0=00000000 d1=0000ffff d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00043fe3 a1=00000ad7
reg a2=00000992 a3=00000000 a4=00000000 a5=00068056 a6=00043fe4 a7=00043f78
reg usp=00000000 ssp=00043f78
```

To modify register contents

- Use the **reg <reg>=<value>** command.

Examples

To modify register d7 to contain the value 0FFFF1234H:

```
M>reg d7=0ffff1234
```

Accessing Memory

This section describes the tasks related to displaying, modifying, copying, and searching the contents of memory locations.

You can display and modify the contents of memory in byte, word, and long word lengths. You can also display the contents of memory in assembly language mnemonic format.

When displaying memory, the *display mode* specifies the format of the memory display. When modifying memory, the display mode specifies the size of the location to be modified.

When accessing target memory locations, the *access mode* specifies the type of microprocessor cycles that are used to read or write the value(s).

This section describes the following tasks:

- Setting the display and access modes.
- Displaying memory contents.
- Modifying memory contents.
- Copying memory contents.
- Searching memory for data.

To set the display and access modes

- Use the **mo** command.

When displaying memory, the display mode specifies the format of the memory display.

When modifying memory, the display mode specifies the size that the data is to be interpreted as. For example, suppose you modify a memory location with the value 41H; in the byte display mode, the value is 41H, but in the word display mode, the value is 0041H, and in the long word display mode the value is 00000041H.

When accessing target system memory locations, the access mode specifies the type of microprocessor cycles that are used to read or write the value(s). For example, when the access mode is byte and a target system location is modified to contain the value 12345678H, byte instructions are used to write the byte values 12H, 34H, 56H, and 78H to target system memory.

You can also specify the display and access modes in the **m** command, which is used to display and modify memory locations.

Examples

To display the display and access mode settings:

```
M>mo
  mo -ab -dw
```

To specify the long word display mode:

```
M>mo -dl
```

To specify the word access mode:

```
M>mo -aw
```

To display memory contents

- Use the **m** command.

The **m** command displays the contents of the address or address range specified. Also, you can specify display and access modes with the **-d** and **-a** options.

For viewing code in memory, the **m -dm** command displays memory contents in disassembled mnemonic format.

Examples

To display the byte contents of a memory location:

```
M>m -db Msg_Dest
  0060057..0060057     00
```

To display the contents of a range of memory locations:

```
M>m -dw Msg_Dest..Msg_Dest+1f
  0060057..0060066     0000 0000 0000 0000 0000 0000 0000 0000
  0060067..0060076     0000 0000 0000 0000 0000 0000 0000 0000
```

To display the range "main" through "main+7" in mnemonic format:

```
M>m -dm main..main+7
  00009be  main             LINK    A6,#0ff9c
  00009c2  -                MOVE.L  A2,-[A7]
  00009c4  -                MOVE.L  D2,-[A7]
```

To modify memory contents

- Use the **m <addr>=<value>** command.

You can modify the contents of a memory location or a range of memory locations. Also, you can specify display and access modes with the **-d** and **-a** options.

Examples

To modify the location "Cmd_Input" with a byte value of 41H:

```
M>m -db Cmd_Input=41
M>m -db Cmd_Input
  0060056..0060056     41
```

To modify the range of locations from "Msg_Dest" through "Msg_Dest+1FH" with word values of 41H, 42H, 43H, and 44H:

```
M>m -dw Msg_Dest..Msg_Dest+1f=41,42,43,44
M>m -dw Msg_Dest..Msg_Dest+1f
  0060057..0060066     0041 0042 0043 0044 0041 0042 0043 0044
  0060067..0060076     0041 0042 0043 0044 0041 0042 0043 0044
```

To copy memory contents

- Use the **cp** command.

The **cp** (copy memory) command gives you the ability to copy the contents of one range of memory to another. This is a handy feature to test whether programs are relocatable, etc.

Examples

To copy the range of memory locations from 400H through 0FE3H to 1000H:

```
M>cp 1000=400..0fe3
```

To search memory

- Use the **ser** command.

The **ser** command allows you to search for data in a range of memory locations. If any part of the data specified in the **ser** command is not found, no match is displayed.

Examples

To search the range of memory from 40000H through 43FFFH for the ASCII string "Command A Entered":

```
M>ser 40000..43fff="Command A Entered"
  pattern match at address: 0042ea0
  pattern match at address: 0042f10
  pattern match at address: 0042f80
  pattern match at address: 0043f80
M>ser 40000..43fff="Command A EntereD"
M>
```

Notice that if the string is not found, no information is returned.

5

Using the Emulation Analyzer - Easy Configuration

Using the Emulation Analyzer - Easy Configuration

This chapter describes tasks you may wish to perform while using the emulation analyzer in its "easy" configuration (the "Using the Emulation Analyzer - Complex Configuration" chapter describes how to access and use the full capability of the analyzer). These tasks are grouped into the following sections:

- Initializing the analyzer.
- Qualifying the analyzer clock.
- Starting and stopping trace measurements.
- Displaying trace lists.
- Qualifying trigger and store conditions.
- Using the sequencer.

Initializing the Analyzer

This section describes how to:

- Initialize the analyzer.
- Include bus arbitration tags in the trace.
- Exclude bus arbitration tags from the trace.
- Display trace activity.
- Arm (activate) the emulation analyzer when the external analyzer triggers.

To initialize the analyzer

- Enter the **tinit** command.

The **tinit** command initializes the analyzer to its default or power-up state.

Examples

To initialize the analyzer:

```
U>tinit
```

To include bus arbitration tags in the trace

1. Enable bus arbitration by entering the **cf ba=en** command.
2. Enable bus arbitration tagging by entering the **cf bat=en** command.

When bus arbitration tagging is enabled (and a trace has been started), the emulation analyzer will store a single trace state (and label it as a bus arbitration state) every time your target system goes through a bus arbitration sequence.

To exclude bus arbitration tags from the trace

- Entering the **cf bat=dis** command.

If any bus arbitration sequence occurs, it will be ignored by the analyzer.

To display trace activity

- Enter the **ta** command.

The **ta** (trace activity) command allows you to display the current status of the analyzer trace signals. The trace activity display shows the status of trace signals at any time, regardless of whether a pending trace is completed or not.

The trace signals are displayed in sets of sixteen. Pod 1 represents emulation analyzer trace signals 0 through 15 (the least significant bit is on the right). Pod 2 represents emulation analyzer trace signals 16 through 31, and so on. External Pod represents the external analyzer trace signals.

A trace signal is displayed as a low (0), high (1), or moving (?). For the external analyzer, low means below the threshold voltage (as specified by the **xtv** command), and high means above the threshold voltage.

Examples

To display the activity on the analyzer trace signals:

```
U>ta
  Pod 3        = ???????? ????????
  Pod 2        = 111????? 00000??0
  Pod 1        = 00?????? ???????0
  External pod = 00000000 00000000
```

To arm the emulation analyzer with the external analyzer trigger

- Use the **tarm** command.

You can arm (that is, activate) the emulation analyzer when the external analyzer finds its trigger condition. The connection between the emulation analyzer and the external analyzer is made over one of the emulator's internal trigger signals (trig1 or trig2). You set up the external analyzer to drive the internal trigger signal when it finds its trigger condition, and you use the **tarm** command to arm the emulation analyzer when the internal trigger signal appears.

Examples

To arm the emulation analyzer with the external analyzer's trigger output, over the internal trig2 signal, and trigger when the arm goes true:

```
M>xtmo -s
M>xtgout trig2
M>tarm =trig2
M>tg arm
```

Qualifying the Analyzer Clock

The emulator/analyzer interface looks at the data on the emulation processor's bus and control signals at each clock cycle. This interface generates clocks to the analyzer. Address, data, and status fields which are then clocked into the analyzer.

You can qualify the analyzer clock so that the analyzer only looks at background cycles. It's even possible to qualify the analyzer clock so that the analyzer only looks at bus cycles when some external signal is active.

This section describes how to:

- Qualify the analyzer clock to trace background execution.
- Qualify the analyzer clock to trace only when an external signal is active.

To trace background execution

- Enter the **tck -b** command.

By default, the analyzer traces user (that is, foreground) code; this is specified by the **-u** option to the **tck** command. However, it is possible to trace background code; this is specified by the **-b** option to the **tck** command.

You can trace both user and background code by specifying the **-ub** option in a single **tck** command.

Examples

To trace background execution:

```
U>tck -b
U>tck
  tck -r L -b -s S
```

Notice that the user/background option is a switch in the clock specification. Changing the option as shown above does not affect the rest of the trace clock specification.

To trace foreground and background execution:

```
U>tck -ub
U>tck
  tck -r L -ub -s S
```

To return to tracing foreground execution:

```
U>tck -u
U>tck
  tck -r L -u -s S
```

To trace execution when an external signal is active

1 Connect the external analyzer JCL or KCL line to the external signal.

2 Use the **tck** command to specify the clock qualifier.

It may occasionally be useful to use an external clock signal (either the JCL or KCL inputs to the external analyzer) to qualify the emulation analyzer clock signal. In other words, the emulation analyzer clock signal may only clock the analyzer when the qualifying clock signal is true. (This is how the analyzer provides the capability of tracing only user program execution or only background execution.)

Clock signals are qualified by using the **-l** and **-h** options to the **tck** command.

The **-l** option is used to specify a qualifying signal which only allows the trace to clock when this signal is lower than the threshold voltage.

The **-h** option is used to specify a qualifying signal which only allows the trace to clock when this signal is higher than the threshold voltage.

Note that you must specify the external analyzer threshold voltage before qualifying the emulation analyzer clock with an external signal.

Note also that if several clock qualifiers are specified, the analyzer will be clocked if any one is true. This means you must turn off the user/background qualifier; in other words, **tck -ub**.

Qualifier setup time is approximately 25 nanoseconds when the external analyzer is aligned with emulation analyzer, (**xtmo -e**). Qualifier setup time is approximately 20 nanoseconds when the external analyzer operates as an independent state analyzer (**xtmo -s**). Qualifier hold time is approximately 5 nanoseconds.

Examples

To trace execution only when there is a TTL high value on the external analyzer's J clock input:

```
U>tinit
U>tck -ub
U>xtv -l TTL
U>tck -h J
U>t
```

To trace execution only when there is a CMOS low value on the external analyzer's K clock input:

```
U>tinit
U>tck -ub
U>xtv -u CMOS
U>tck -l K
U>t
```

Starting and Stopping Traces

This section describes the tasks that relate to starting and stopping trace measurements.

When you start a trace measurement, the analyzer begins looking at the data on the emulation processor's bus and control signals on each analyzer clock signal. The information seen on a particular clock is called a state.

When one of these states matches the "trigger state" you specify, the analyzer stores states in trace memory. When trace memory is filled, the trace is said to be "complete."

The default trigger state specification is "any state," so when you start a trace measurement after initializing the analyzer, the analyzer will "trigger" on the first state it sees and store the following states in trace memory.

Once you start a trace measurement, you can view the progress of the measurement by displaying the trace status.

In some situations, for example, when the trigger state is never found or when the analyzer hasn't filled trace memory, the trace measurement does not complete. In these situations, you can halt the trace measurement.

This section describes how to:

- Start trace measurements.
- Display the trace status.
- Halt trace measurements.

To start a trace measurement

- Enter the **t** command.

The **t** (trace) command tells the analyzer to begin monitoring the states which appear on the trace signals. You will see a message which confirms that a trace is started.

After the emulator is powered-up or initialized, the analyzer is in its simplest configuration. The default condition will trigger on any state, and store all captured states. You can simply issue a trace command (**t**) to trace the states currently executing.

Examples

To start a trace measurement after analyzer initialization:

```
U>tinit
U>t
  Emulation trace started
```

To trace a program as it starts up:

```
U>rst
R>t
  Emulation trace started
R>r crt1:entry
U>
```

To display the trace status

- Enter the **ts** command.

The **ts** (trace status) command lets you view what the analyzer is doing (or what the analyzer has done if the trace has completed).

The first line of the emulation trace status display shows whether the user trace has been "completed"; other possibilities are that the trace is still "running" or that the trace has been "halted". The word "NEW" indicates that the most recent trace has not been displayed. The word "User" indicates that the trace was taken in response to a **t** command; the other possibility is that a "CMB" execute signal started the trace.

The second line of the **ts** display contains information on the arm condition. If the **tarm** condition is specified as **always**, the message "Arm ignored" is displayed. If the **tarm** condition is specified as one of the internal signals, either the message "Arm not received" or "Arm received" is displayed. The display indicates if the arm condition happened any time since the most recent trace started, even if it happened after the trace was halted or became complete.

When an arm condition has been specified with the **tarm** command, the "Arm to trigger" line displays the amount of time between the arm condition and the trigger. The time displayed will be from -0.04 microseconds to 41.943 milliseconds, less than -0.04 microseconds, or greater than 41.943 milliseconds. If the arm signal is ignored or the trigger is not in memory, a question mark (?) is displayed.

The "States" line shows the number of states that have been stored (out of the number that is possible to store) and the line numbers that the stored states occupy. (The trigger state is always stored on line 0.)

The "Sequence term" line of the trace status display shows the number of the term the sequencer was in when the trace completed. Because a branch **out of the last sequence term** constitutes the trigger, the number displayed is what would be the next term (2 in the example below) even though that term is not defined. If the trace is halted, the sequence term number just before the halt is displayed; otherwise, the current sequence term number is displayed. If the current sequence term is changing too quickly to be read, a question mark (?) is displayed.

The "Occurrence left" line of the trace status display shows the number of occurrences remaining before the primary branch can be taken out of the current sequence term. If the occurrence left is changing too quickly to be read, a question mark (?) is displayed.

Examples

To display the trace status:

```
U>ts
  --- Emulation Trace Status ---
  NEW User trace complete
  Arm ignored
  Trigger in memory
  States 512 (512) 0..511
  Sequence term 2
  Occurrence left 1
```

To halt a trace measurement

- Enter the **th** command.

The **th** (trace halt) command allows you to halt a trace measurement. When the **th** command is entered, the message "Emulation trace halted" is displayed.

Examples

To halt a trace measurement:

```
U>th
  Emulation trace halted
```

Displaying Traces

When states are stored in trace memory, you can display these states in the trace list. Also, you can change the format of the trace list. This section describes how to:

- Display the trace list.
- Change the format of the trace list.

To display the trace

- Use the **tl** command.

The **tl** (trace list) command displays the trace data.

Examples

The trace list displayed in the following examples was set up with the following commands.

```
U>rst
R>t
  Emulation trace started
R>r crt1:entry
U>ts
  --- Emulation Trace Status ---
  NEW User trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 512 (512) 0..511
  Sequence term 2
  Occurrence left 1
```

To display the trace list:

```
U>tl
```

```
 Line   addr,H  68000 Mnemonic,H                   xbits,H    count,R  seq
-----   ------  ---------------------------------  -------  ---------  ---
    0   043ffa    2700  supr data rd word          0000           ---   +
    1   043ffc    0000  supr data rd word          0000      0.200 uS   .
    2   043ffe    0400  supr data rd word          0000      0.200 uS   .
    3   000400    4ff9  supr prog                  0000      0.200 uS   .
    4   000402    0004  supr prog                  0000      0.200 uS   .
    5   000404    4000  supr prog                  0000      0.200 uS   .
    6   000406    4879  supr prog                  0000      0.200 uS   .
    7   000408    0000  supr prog                  0000      0.200 uS   .
    8   00040a    0000  supr prog                  0000      0.200 uS   .
    9   00040c    9dce  supr prog                  0000      0.200 uS   .
```

The first column in the trace list contains the line number. The trigger is always on line 0.

The second column contains the address information associated with the trace states. Addresses in this column may be locations of instruction opcodes on fetch cycles, or they may be sources or destinations of operand cycles.

The third column shows mnemonic information about the emulation bus cycle.

If your analyzer card contains external analysis (for example, HP 64703), the next column shows the data captured on the external trace signals.

The next column shows the count information (**time** is counted by default). The "R" indicates that each count is relative to the previous state.

The last column contains information about the sequencer. The "+" on line 0 indicates the state satisfied a branch condition (in this case, a trigger condition).

The default number of states to display is 10.

To display the top 10 states in disassembled format:

```
U>tl -d -t 10
```

```
 Line   addr,H  68000 Mnemonic,H                    xbits,H    count,R  seq
-----   ------  ----------------------------------  -------  ---------  ---
    0   043ffa    2700  supr data rd word            0000           ---   +
    1   043ffc    0000  supr data rd word            0000      0.200 uS   .
    2   043ffe    0400  supr data rd word            0000      0.200 uS   .
    3   000400  LEA.L   0044000,A7                   0000      0.200 uS   .
    4   000402    0004  supr prog                    0000      0.200 uS   .
    5   000404    4000  supr prog                    0000      0.200 uS   .
    6   000406  PEA.L   0000000                      0000      0.200 uS   .
    7   000408    0000  supr prog                    0000      0.200 uS   .
    8   00040a    0000  supr prog                    0000      0.200 uS   .
    9   00040c  SUBA.L  A6,A6                        0000      0.200 uS   .
```

To display the top 10 states with symbols and absolute addresses in the address column:

```
U>tl -e -t 10
```

```
 Line   addr,H  68000 Mnemonic,H                    xbits,H    count,R  seq
-----   ------  ----------------------------------  -------  ---------  ---
    0   043ffa    2700  supr data rd word            0000           ---   +
    1   043ffc    0000  supr data rd word            0000      0.200 uS   .
    2   043ffe    0400  supr data rd word            0000      0.200 uS   .
    3   :entry  LEA.L   sysstack:TopOfStack,A7       0000      0.200 uS   .
    4   000402    0004  supr prog                    0000      0.200 uS   .
    5   000404    4000  supr prog                    0000      0.200 uS   .
    6   000406  PEA.L   0000000                      0000      0.200 uS   .
    7   000408    0000  supr prog                    0000      0.200 uS   .
    8   00040a    0000  supr prog                    0000      0.200 uS   .
    9   00040c  SUBA.L  A6,A6                        0000      0.200 uS   .
```

To display the states at line 100:

```
U>tl 100
```

```
 Line   addr,H  68000 Mnemonic,H                    xbits,H    count,R  seq
-----   ------  ----------------------------------  -------  ---------  ---
  100   r_loop    2605  supr prog                    0000      0.200 uS   .
  101   000a90    2a02  supr prog                    0000      0.200 uS   .
  102   000a92    4eb9  supr prog                    0000      0.200 uS   .
  103   000a94    0000  supr prog                    0000      0.200 uS   .
  104   000a96    1224  supr prog                    0000      0.200 uS   .
  105   001224    2039  supr prog                    0000      0.200 uS   .
  106   043fd0    0000  supr data wr word            0000      0.200 uS   .
  107   043fd2    0a98  supr data wr word            0000      0.200 uS   .
  108   001226    0006  supr prog                    0000      0.200 uS   .
  109   001228    04de  supr prog                    0000      0.200 uS   .
```

To change the trace display format

- Use the **tf** command.

You can change the format of the trace information with the **tf** (trace format) command.

The **tf** command primarily allows you to arrange the columns of trace information in a different manner. However, you can include any trace label in the trace. Also, the trace label information can be displayed in various number bases, and counts can be displayed relative or absolute.

Examples

To view the trace display format:

```
U>tf
  tf addr,H mne xbits,H count,R seq
```

To change the trace format so that the address column is 12 characters wide and the external trace signals are not shown:

```
U>tf addr,H,12 mne count,R seq
U>tl -t
```

```
 Line   addr,H        68000 Mnemonic,H                           count,R  seq
-----   ------------  -----------------------------------  ---------   ---
    0   043ffa          2700  supr data rd word                  ---   +
    1   043ffc          0000  supr data rd word            0.200 uS   .
    2   043ffe          0400  supr data rd word            0.200 uS   .
    3   crt1:entry    LEA.L   sysstack:TopOfStack,A7       0.200 uS   .
    4   000402          0004  supr prog                    0.200 uS   .
    5   000404          4000  supr prog                    0.200 uS   .
    6   000406        PEA.L   0000000                      0.200 uS   .
    7   000408          0000  supr prog                    0.200 uS   .
    8   00040a          0000  supr prog                    0.200 uS   .
    9   00040c        SUBA.L  A6,A6                        0.200 uS   .
```

Qualifying Trigger and Store Conditions

This section describes tasks relating to the qualification of trigger and storage states.

You can trigger on, or store, specific states or specific values on a set of trace signals (which are identified by trace labels).

Also, you can *prestore* states. The prestore qualifier is a second storage qualifier used for storing states that occur before the normally stored states. Prestore is useful for capturing entry points to procedures or for identifying where global variables are accessed from.

This section describes how to:

- Qualify the trigger state.
- Trigger on a number of occurrences of some state.
- Change the trigger position in the trace.
- Qualify states stored in the trace.
- Activate and qualify prestore states.
- Change the count qualifier.

Expressions in Trace Commands

Expressions are used in commands which qualify the trace. Expressions may be specified in the following forms (the pound sign, #, appears before comments):

```
any/all                                  # special tokens
never/none
arm

label=<value>
label!=<value>
label=<value> and label=<value> ...      # this condition
label!=<value> or label!=<value> ...     # not this condition
label=<value>..<value>                   # this range
label!=<value>..<value>                  # not this range
```

Note that if you wish to specify an expression such as "label=<value> and label!=<value>", you must configure the analyzer so that you have access to its full capability (refer to the "Using the Emulation Analyzer - Complex Configuration" chapter).

Note also that only one range resource is available. You can, however, use this range (or "not this range") in more than one trace command.

Tokens The tokens **any** or **all** specify any or all conditions; you can use these tokens interchangeably. The tokens **never** or **none** specify false conditions; they are used to turn off qualifiers. The **never** and **none** tokens may also be used interchangeably. The **arm** token represents a condition external to the analyzer. Arm conditions are described in the "Making Coordinated Measurements" chapter.

Trace Labels Labels may be predefined trace labels or labels which you define with the **tlb** (trace label) command. Trace labels can be up to 31 characters long. When you define a trace label, you assign trace signals to the label name. The emulation analyzer trace signals are described in the table that follows.

Emulation Analyzer Trace Signals

Trace Signals	**Signal Name**	**Signal Description**
0	$\overline{\text{UDS}}$	Upper Data Strobe (A0)
1-23	A1-A23	Address Lines 1-23
31	GRDAC_L	Guarded Memory Access Cycle
30	ROMAC_L	ROM Memory Access Cycle
29 28 27	TFC2 TFC1 TFC0	Function Codes 0-2. These lines to the analyzer are derived from the 68000 processor's function code lines. During normal foreground operation (user program or a foreground monitor), the processor function code lines are passed directly to the analyzer. When a DMA tag cycle is generated, an illegal function code pattern is driven to the analyzer to indicate the tag. Two other illegal function code patterns are used during emulation monitor operations to generate additional status information. The modified function code meanings are: 000 - Monitor Program Space 001 - User Data Space 010 - User Program Space 011 - DMA tag cycle status 100 - Monitor Data Space 101 - Supervisor Data Space 110 - Supervisor Program Space 111 - Interrupt Acknowledge
26	VMA	6800 peripheral cycle
25	RD/$\overline{\text{WR}}$	High Read/Low Write
24	B/$\overline{\text{W}}$	High Byte Access/Low Word Access
32-47	D0-D15	Processor Data 0-15

Predefined Trace Labels To see the trace labels which have been predefined, enter the **tlb** (trace label) command with no options.

```
M>tlb
  #### Emulation trace labels
  tlb addr 0..23
  tlb data 32..47
  tlb stat 24..31
```

These predefined trace labels represent emulation processor signals as described below.

addr Represents the trace signals (0 through 23) which monitor the emulation processor’s address pins.

data Represents the trace signals (32 through 47) which monitor the emulation processor’s data pins.

stat Represents the trace signals (24 through 31) which monitor other emulation processor signals.

Values Values can be numeric constants (in several bases), symbols, or equates. Values can also be constants, symbols, and equates combined with operators. (Refer to the <value> description in the "Commands" chapter for information on constants and operators.)

Predefined Equates The **equ** (specify equates) command allows you to equate values with names. Equates for common trace label values are predefined. To view the equates, enter the **equ** command with no options. (These status equates are also listed in the **help proc** information.)

```
U>equ
  ### Equates ###
  equ     byte=0xxxx xxx1y     # byte cycle
  equ  cyc6800=0xxxx x0xxy     # 6800 cycle
  equ     data=0xxx0 1xxxy     # data cycle
  equ      dma=0xx01 1xxxy     # bus released to DMA device
  equ      grd=00xxx xxxxy     # guarded memory
  equ   intack=0xx11 1xxxy     # interrupt acknowledge
  equ     prog=0xxx1 0xxxy     # program cycle
  equ     read=0xxxx xx1xy     # memory read
  equ      sup=0xx1x xxxxy     # supervisor cycle
  equ  supdata=0xx10 1xxxy     # supervisor data
  equ  supprog=0xx11 0xxxy     # supervisor program
  equ     user=0xx0x xxxxy     # user cycle
  equ userdata=0xx00 1xxxy     # user data
  equ userprog=0xx01 0xxxy     # user program
  equ     word=0xxxx xxx0y     # word cycle
  equ    write=0xxxx xx0xy     # memory write
  equ    wrrom=0x0xx xx0xy     # write to rom
```

These predefined equates may be used to specify values for the **stat** trace label when qualifying trace conditions. For example:

```
stat=write
```

is the same as:

```
stat=0xxxxxx0xy
```

Equates, either predefined or user-defined, are translated to their actual values when used. Re-defining an equate will not affect commands in which the equate was previously used. For example, if you enter the commands **equ count=100; tg any count; equ count=5**, the occurrence count in the trigger specification is still 100.

To qualify the trigger state

- Use the **tg** command.

The **tg** (specify simple trigger) command allows you to specify when the analyzer should begin storing states.

Examples

Suppose you want to look at the execution of the analyzer demo program after the branch to the first instruction in the demo program's "for" loop (0A8EH), and, therefore, you would like to begin storing states after address 0A8EH occurs. To do this you could enter the commands shown below.

```
U>tinit
U>sym for_loop=0a8e
U>tg addr=for_loop
U>t
  Emulation trace started
U>ts
  --- Emulation Trace Status ---
  NEW User trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 512 (512) -1..510
  Sequence term 2
  Occurrence left 1
U>tf addr,H,12 mne count,R seq
U>tl -de
```

```
 Line   addr,H        68000 Mnemonic,H                        count,R  seq
-----   ------------  -----------------------------------  ---------  ---
   -1   000b2a        Unimplemented Instruction:0ff64            ---   .
    0   for_loop      MOVE.L  D5,D3                        0.320 uS   +
    1   000a90        MOVE.L  D2,D5                        0.200 uS   .
    2   000a92        JSR     0001224                      0.200 uS   .
    3   000a94          0000  supr prog                    0.200 uS   .
    4   000a96          1224  supr prog                    0.200 uS   .
    5   001224        MOVE.L  data_gen:__rand_seed,D0      0.200 uS   .
    6   043fd0          0000  supr data wr word            0.200 uS   .
    7   043fd2          0a98  supr data wr word            0.200 uS   .
    8   001226          0006  supr prog                    0.200 uS   .
```

To trigger on a bus arbitration sequence:

```
U>cf ba=en
U>cf bat=en
U>tg stat=dma
U>t
  Emulation trace started
```

To trigger on a number of occurrences of some state

- Use the **tg <qualifier> <occurrence count>** command.

When specifying a simple trigger, you can include an occurrence count. The occurrence count specifies that the analyzer trigger on the Nth occurrence of some state.

The default base for an occurrence count is decimal. You may specify occurrence counts from 1 to 65535.

Examples

To trigger on the 100th occurrence of the branch to the first instruction after the analyzer demo program's for loop (0A8EH):

```
U>sym for_loop=0a8e
U>tg addr=for_loop 100
U>t
  Emulation trace started
U>ts
  --- Emulation Trace Status ---
  NEW User trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 512 (512) 0..511
  Sequence term 2
  Occurrence left 1
```

To change trigger position in the trace

- Use the **tp** command.

The **tp** (trigger position) command changes the trigger position in the trace.

The trigger position default is **tp s**, which specifies that the trigger appears at the start of the trace. You can also specify that the trigger appear in the center of the trace with the **tp c** command, or that the trigger appear at the end of the trace with the **tp e** command.

Additionally, you can specify a certain number of states to appear before (**tp -b 10**) or after (**tp -a 1014**) the trigger in the trace.

When the analyzer counts time or states, the actual trigger position is within +/- 1 state of the number specified. When counts are turned OFF, the actual trigger position is within +/- 3 states of the number specified.

Examples

To place the trigger state in the center of the trace:

```
U>tp c
U>t
  Emulation trace started
U>ts
  --- Emulation Trace Status ---
  NEW User trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 512 (512) -257..254
  Sequence term 2
  Occurrence left 1
```

Notice in the trace status information that states are stored before and after the trigger.

To qualify states stored in the trace

- Use the **tsto** command.

By default, all captured states are stored; however, you can qualify which states get stored with the **tsto** (trace storage qualifier) command.

Examples

To store only the states which write random numbers to the Results area in the analyzer demo program, enter the following commands.

```
U>tsto addr=Results..Results+3ff
U>tg any
U>t
  Emulation trace started
U>tl
```

```
 Line   addr,H       68000 Mnemonic,H                     count,R  seq
-----   ------------ ----------------------------------  ---------  ---
    0   000fb8         4e75  supr prog                        ---    +
    1   06039a         0000  supr data wr word           3.280 uS    .
    2   06039c         03a7  supr data wr word           0.200 uS    .
    3   060132         0000  supr data wr word           65.32 uS    .
    4   060134         2aaf  supr data wr word           0.200 uS    .
    5   0602f2         0000  supr data wr word           65.20 uS    .
    6   0602f4         6310  supr data wr word           0.200 uS    .
    7   060312         0000  supr data wr word           64.96 uS    .
    8   060314         0f6e  supr data wr word           0.200 uS    .
    9   060096         0000  supr data wr word           64.92 uS    .
```

Notice that the trigger state (line 0) is included in the trace list; trigger states are always stored.

To activate and qualify prestore states

- Use the **tpq <qualifier>** command.

Prestore allows you to save up to two states which precede a normal store state. Prestore is turned off by default. However, you can use the **tpq** command to specify a prestore qualifier.

Prestore is useful when you want to find the cause of a particular state. For example, if a variable is accessed from many different places in the program, you can qualify the trace so that only accesses of that variable are stored. Then, you can turn on prestore to find out where accesses of that variable originate from.

States which satisfy the prestore qualifier and the storage qualifier at the same time are stored as normal states.

The analyzer uses the same resource to save prestore states as it does to save count tags. Consequently, the "prestore" string is shown in the "count" column of the trace list. Notice that the time counts are relative to the previous normal storage state. Turning off the count qualifier does not turn off prestore: however, the "prestore" string cannot be seen in the "count" column of the trace list.

Examples

To prestore LINK A6,#0 instructions (which is the first instruction in the Caller functions and whose opcode is 4E56H) on writes to the range Results through Results+3ff:

```
U>tsto addr=Results..Results+3ff
U>tpq data=4e56
U>tg any
U>t
  Emulation trace started
U>tl -de
```

```
 Line   addr,H        68000 Mnemonic,H                      count,R  seq
-----   ------------  ----------------------------------  ---------  ---
    0   043fb8          0000  supr data rd word                 ---   +
    1   Caller_0      LINK    A6,#****                     prestore   .
    2   Write_Num     LINK    A6,#****                     prestore   .
    3   060362          0000  supr data wr word            63.88 uS   .
    4   060364          7921  supr data wr word            0.200 uS   .
    5   Caller_1      LINK    A6,#****                     prestore   .
    6   Write_Num     LINK    A6,#****                     prestore   .
    7   060216          0000  supr data wr word            64.92 uS   .
    8   060218          0369  supr data wr word            0.200 uS   .
    9   Caller_1      LINK    A6,#****                     prestore   .
```

To turn off prestore states:

```
U>tpq none
```

To change the count qualifier

- To count time, use the **tcq time** command.
- To count states, use the **tcq <qualifier>** command.
- To turn OFF counting, use the **tcq none** command.

After initializing the analyzer, the default count qualifier is **time**, which means that the time between states is saved. When time is counted, up to 512 states can be stored in the trace.

When you count states, the counter is incremented each time the state is captured (not necessarily stored) by the analyzer. When a state is counted, up to 512 states can be stored in the trace.

When you turn OFF counting, up to 1024 states can be stored in the trace.

Examples

Suppose you want to know how many loops of the program occur between writes to the memory location Results+0C4H. You can use the **tcq** command to count a state that occurs once for each loop of the program.

First, set up the analyzer so that only writes to Results+0C4H are stored:

```
U>tsto addr=Results+0c4 and stat=write
```

Next, specify the count qualifier as the first instruction after the analyzer demo program's "for" loop (0A8EH):

```
U>sym for_loop=0a8e
U>tcq addr=for_loop
```

Finally, set up to trigger on ant state, start the trace, change the trace format to display relative and absolute counts, and display the trace:

```
U>tg any
U>t
  Emulation trace started
U>tf addr,H,6 mne count,R count,A
U>tl

 Line   addr,H  68000 Mnemonic,H                      count,R    count,A
-----   ------  ------------------------------------  ---------  ---------
    0   00099e    4eb9  supr prog                           ---          0
    1   06011a    0000  supr data wr word                   308        308
    2   06011a    0000  supr data wr word                   542        850
    3   06011a    0000  supr data wr word                   188       1038
    4   06011a    0000  supr data wr word                  1098   2.136e03
    5   06011a    0000  supr data wr word                    28   2.164e03
    6   06011a    0000  supr data wr word                   117   2.281e03
    7   06011a    00    supr data wr byte                    58   2.339e03
    8   06011a    00    supr data wr byte                     0   2.339e03
    9   06011a    00    supr data wr byte                     0   2.339e03
```

To return to counting time:

```
U>tcq time
```

Using the Sequencer

By using the sequencer, you can trigger after a sequence of states instead of just one state. The sequencer has several levels, called *sequence terms*.

Each sequence term can search for two states at a time: a primary state and a secondary state. The primary state may have an occurrence count specified. If the primary state occurs the number of times specified, the sequencer branches to the next term. If the secondary state is found before the primary state occurs the number of times specified, the sequencer branches back to the first term.

The same secondary branch condition is used for all sequence terms, and secondary branches are always back to the first term; therefore, the secondary branch is called the *global restart.*

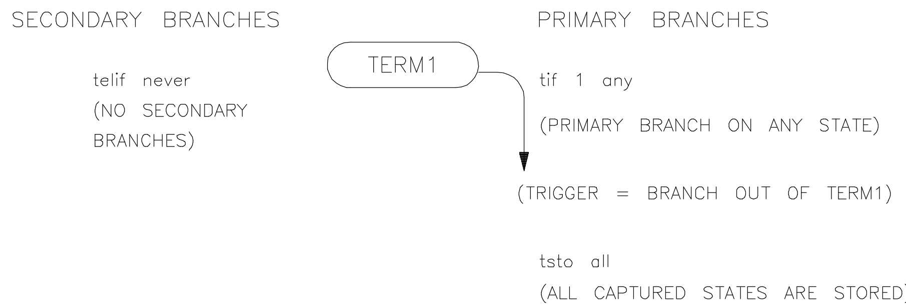
The last sequence term defines the trigger state. A branch out of this term constitutes the trigger.

This section describes how to:

- Reset the sequencer.
- Display the sequencer specification.
- Specify primary and secondary branch conditions.
- Add or insert sequence terms.
- Delete sequence terms.

The Default Sequencer Specification

After power-up, initialization, or sequencer reset, the sequencer consists of one term.



It may be helpful to think of the **tif** (primary branch expression) command as a conditional statement. For example, "If (some state occurs), then branch".

Because sequence term 1 is the last term and a branch out of the last term constitutes the trigger, the primary branch expression (**any**) of term 1 specifies the trigger condition. The expression **any** says that any captured trace state will cause a branch. Therefore, the trigger will occur immediately after the **t** (trace) command is issued (if instructions are being executed).

The **tsto** (trace storage qualifier) command specifies that **all** captured states are stored. The trace storage qualifier is a global; that is, it applies to all sequence terms. In addition to states which satisfy the trace storage qualifier, any state which causes a branch is stored in trace memory. Also, prestore states can be saved before states which satisfy the trace storage qualifier.

The **telif** command is used to specify the secondary branch expression for every sequence term; this expression is called the *global restart*. It may be helpful to think of the **telif** command as an "else if" conditional statement. For example, "Else if (some state occurs before) then branch to term 1".

The global restart in the default sequencer specification is **never**. This means no trace state can cause a secondary branch.

Simple Trigger and the Sequencer

The simple trigger command used previously in this chapter has the following effect on the sequencer:

```
U>sym for_loop=0a8e
U>tg addr=for_loop
U>tsq
  tif 1 addr=for_loop
  tsto all
  telif never
```

Notice that only the primary branch expression of the first sequence term (the trigger condition) is different than the default sequencer specification. The address 0A8EH is the first address inside the demo program's "for" loop. An address value of 0AE8H will trigger the analyzer, causing trace memory to be filled with states and stop.

When the **tg** command is entered with no options, the primary branch expression of the first sequence term is displayed. This is the trigger condition only when one term exists in the sequencer.

To reset the sequencer

- Enter the **tsq -r** command.

To reset the sequencer to its default, power-up state use the **-r** option to the **tsq** (trace sequencer) command.

Examples

To reset the sequencer:

```
U>tsq -r
```

To display the sequencer specification

- Enter the **tsq** command with no options.

To display the sequencer specification, enter the **tsq** command with no options.

Examples

```
U>tsq
  tif 1 any
  tsto all
  telif never
```

The **tif 1 any** part of the sequencer specification says that any state will cause a branch out of term 1. The **tsto all** says all states will be stored, and the **telif never** says that the global restart is turned off.

To specify primary and secondary branch expressions

- Use the **tif** and **telif** commands.

The **tif** command lets you qualify the states searched for by sequence terms.

The **telif** command lets you qualify the state that will cause a global restart (sequencer branch back to term 1).

Examples

You can use sequence terms to trace a specific combination of events. For example, Caller_3 can be used to write any random number, but suppose you want to trace only the situation where Caller_3 is used to write a random number to address Results+0C4H.

First, set up the sequencer so that it first searches for the call to Caller_3 by specifying the address of Caller_3 (0A18H) as the primary branch expression of the first sequence term.

```
U>tif 1 addr=Caller_3
```

After Caller_3 is found, the sequencer should then search for the write to address Results+0C4H. You can do this by specifying the address Results+0C4H, qualified by the "write" status, as the primary branch expression of the second sequence term.

```
U>tif 2 addr=Results+0c4 and stat=write
```

However, if the program executes the RTS instruction of the Caller_3 function (address 0A2EH) before the write to Results+0C4H, you know that Caller_3 is not used to write the random number this time, and the sequencer should start over. You can specify the global restart expression to do this.

```
U>telif addr=0a2e
```

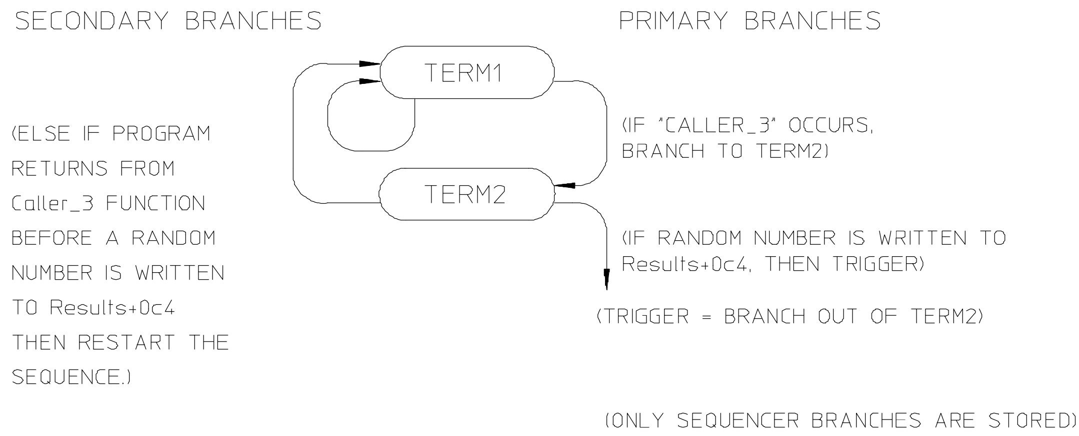
If the write to address Results+0C4H occurs before the program executes the RTS instruction at 0A2EH, the sequencer will take a primary branch out of the last term and trigger the analyzer. Set up the analyzer so that only sequencer branches are stored.

```
U>tsto never
```

The resulting sequencer specification is shown below.

```
U>tsq
  tif 1 addr=Caller_3
  tif 2 addr=Results+0c4 and stat=write
  tsto never
  telif addr=0a2e
```

The sequencer specification above is represented in the figure below. The primary branch expression of the first sequence term is the address associated with Caller_3 (0A18H). The primary branch expression for the second sequence term is the specific write condition we would like to trace; it is also the trigger condition. The primary branch out of the second term constitutes the trigger.



The sequencer works like this: After the trace is started, the first sequence term searches for the call to Caller_3. When the call to Caller_3 state is found, the sequencer branches to term 2. Now, the second sequence term searches for the address Results+0C4H. If address Results+0C4H is found before the state which satisfies the secondary branch expression (the return at address 0A2EH), the analyzer is triggered, causing the analyzer memory to be filled with states before the analyzer stops. If the RTS instruction at address 0A2EH is executed before the primary branch (in either the first or second terms), the sequencer branches back to the first sequence term.

The following commands position the trigger state in the center of the trace, start the trace, and display the trace status.

```
U>tp c
U>t
U>ts
  --- Emulation Trace Status ---
  NEW User trace complete
  Arm ignored
  Trigger in memory
  States 258 (512) -257..0
  Sequence term 3
  Occurrence left 1
```

The "seq" column in the trace list contains information about the sequencer. A "+" in the "seq" column indicates the state satisfied a branch condition. To add the "seq" column to the trace list, enter the following command.

```
U>tf addr,H,12 mne count,r seq
```

Listing the trace will result in the following display.

```
U>tl -de -7

 Line   addr,H        68000 Mnemonic,H                      count,R  seq
-----   ------------  -----------------------------------  ---------  ---
   -7   Caller_3      LINK    A6,#****                     106.8 uS   +
   -6   000a2e        RTS                                  23.60 uS   +
   -5   Caller_3      LINK    A6,#****                     954.2 uS   +
   -4   000a2e        RTS                                  23.48 uS   +
   -3   Caller_3      LINK    A6,#****                     107.3 uS   +
   -2   000a2e        RTS                                  23.48 uS   +
   -1   Caller_3      LINK    A6,#****                     107.0 uS   +
    0   06011a          0000  supr data wr word vpa        20.80 uS   +
    1
```

Remember, the primary branch out of the last term constitutes the trigger. Also, a primary branch always advances to the next higher term. A secondary branch from any term is always made back to the first sequence term (global restart).

To add or insert sequence terms

- Use the **tsq -i** command.

The sequencer may have a total of 4 terms. You can add or insert sequence terms with the **tsq** (trace sequencer) command using the **-i** (insert) option. If the term number specified already exists, the new sequence term is inserted before the existing term; otherwise, the new sequence term is added.

Examples

To insert a second sequence term:

```
U>tsq
  tif 1 addr=Caller_3
  tif 2 addr=Results+0c4 and stat=write
  tsto never
  telif addr=0a2e
U>tsq -i 2
U>tsq
  tif 1 addr=Caller_3
  tif 2 any
  tif 2 addr=Results+0c4 and stat=write
  tsto never
  telif addr=0a2e
```

To delete sequence terms

- Use the **tsq -d** command.

You delete sequence terms using the **-d** option to the **tsq** (trace sequencer specification) command.

After a term is deleted, the remaining terms are renumbered.

Examples

To delete the second sequence term:

```
U>tsq
  tif 1 addr=Caller_3
  tif 2 any
  tif 2 addr=Results+0c4 and stat=write
  tsto never
  telif addr=0a2e
U>tsq -d 2
U>tsq
  tif 1 addr=Caller_3
  tif 2 addr=Results+0c4 and stat=write
  tsto never
  telif addr=0a2e
```

6

Using the Emulation Analyzer - Complex Configuration

Using the Emulation Analyzer - Complex Configuration

This chapter describes how to use the emulation analyzer in its "complex" configuration (the "Using the Emulation Analyzer - Easy Configuration" chapter describes how to use the emulation analyzer in its easy-to-use configuration).

The basic differences between the easy configuration and the complex configuration are in the sequencer and the expressions used to qualify states. Therefore, this chapter describes the following tasks:

- Switching into the complex configuration.
- Using complex expressions.
- Using the sequencer.

Switching into the Complex Configuration

This section describes how to:

- Switch into the complex configuration
- Switch back into the easy configuration

To switch into the complex analyzer configuration

- Enter the **tcf -c** command.

To enter the "complex" analyzer configuration, use the **-c** option to the **tcf** (trace configuration) command. This will cause the analyzer to be initialized to its default "complex" configuration state.

To switch back into the easy analyzer configuration

- Enter the **tcf -e** command.

The **tcf -e** command will place the analyzer back into the "easy" configuration. Changing the analyzer configuration to "easy" will reset the trace pattern specifications, the trigger position, and the count and prestore qualifiers.

Using Complex Expressions

In the "complex" configuration, up to eight pattern resources and one range resource may be used in trace commands wherever state qualifier expressions were used in the "easy" configuration. In fact, state qualifiers are assigned to the pattern and range resources.

The additional capability allowed in the "complex" configuration is that these patterns may be used in combinations to specify more complex qualifiers. The pattern and range resources are divided into two sets, and you can combine resources with the set operators.

This section describes how to:

- Assign state qualifiers to trace patterns.
- Assign state qualifiers to the trace range.
- Combine pattern and range qualifiers.

To assign state qualifiers to trace patterns

- Use the **tpat** command.

Up to eight trace patterns can be specified with the **tpat** (trace pattern) command. The trace pattern names are **p1**, **p2**, ..., **p8**.

The expression associated with a trace pattern can be the keywords **all**, **any**, **none**, or **never**, or the expression may be trace labels equated to values (which can be ANDed together) or trace labels not equal to values (which can be ORed together).

Consider whether or not you will be using global set operators (**and** or **or**) with any of the patterns; if so, make sure those patterns are in different sets.

Examples

To assign to pattern p1 the state where the address value equals 520H, the data value equals 0XXAA1234H, and the status is a memory write:

```
U>tpat p1 addr=520 and data=0xxaa1234 and stat=write
```

To assign to pattern p1 any state except where the address value equals 5C2H, the data value equals 0XX3X5678H, and the status is a memory write:

```
U>tpat p1 addr!=5c4 or data!=0xx3x5678 or stat!=write
```

To assign state qualifiers to the trace range

- Use the **trng** command.

One trace range can be specified with the **trng** (trace range) command. The range name is **r**, and **!r** specifies "not in range".

The expression associated with a trace range can be the keywords **all**, **any**, **none**, or **never**, or the expression may be a trace label equated to a range of values.

Examples

To assign the address range 500H through 5FFH to the range resource:

```
U>trng addr=500..5ff
```

To assign the data range 80H through 8FH to the range resource:

```
U>trng data=0080..008f
```

To combine pattern and range resources

- Use the set operators.

The eight patterns (**p1**..**p8**), the range (**r** for "in range" or **!r** for "not in range"), and the **arm** qualifier (described in the "Making Coordinated Measurements" chapter) are grouped into the two sets shown below.

Set 1: **p1**, **p2**, **p3**, **p4**, **r**, and **!r**.

Set 2: **p5**, **p6**, **p7**, **p8**, and **arm**.

Resources within a set may be combined using one of the intraset operators, **|** (OR) or **~** (NOR).

The two sets can be combined with the **and** and **or** interset (between set) operators. Interset operators are also called global set operators.

The intraset (within a set) operators (~, |) are evaluated first; then, the interset operators are evaluated. You cannot use interset operators on patterns in the same set.

Though only the OR (|) and NOR (~) logical operators are available as intraset operators, you can create the AND and NAND operators by applying DeMorgan's law (the "/" character is used to represent a logical NOT):

```
AND      A and B = /(/A and /B)     NOR
NAND   /(A and B) = /A or /B        OR
```

Examples

Some valid intraset combinations follow.

```
U>tsto p1 | p2 | p3 | r
U>tsto p5 ~ p6 ~ arm
```

The following expression is invalid because you cannot use both **|** (OR) and **~** (NOR) operators within the same set.

```
U>tsto p1 | p2 ~ p3
!ERROR 1249! Invalid qualifier expression: ~ p3
```

The following expression is invalid because you cannot combine resources from different sets with the **|** (OR) or **~** (NOR) operators.

```
U>tsto p1 ~ p2 ~ p5
!ERROR 1249! Invalid qualifier expression: p5
```

Some valid combinations of the two sets follow.

```
U>tsto p1 ~ p2 and p5 | p6
U>tsto p3 | p4 | !r or p7
U>tsto p8 ~ arm and p1 ~ p2
```

The following set combination is invalid because **p1** and **p2** are in the same set.

```
U>tsto p1 and p2
!ERROR 1249! Invalid qualifier expression: p2
```

Note that "p1 ~ p1" is allowed; this type of expression may occasionally be useful if you are running out of pattern resources and wish to specify a logical NOT of some existing pattern. For example, consider the following commands:

```
tpat p1 addr=0
tif 1 p1
tif 2 p1 ~ p1
```

The primary branch of term 2 will be taken when "addr!=0".

An example of using DeMorgan’s law to create the AND operator follows.

Suppose you want to specify the following storage qualifier:

```
U>tsto p1 & p2 or p5 & p6
!ERROR 1241! Invalid qualifier resource or operator: &
```

The error occurs because the **&** operator is not a valid intraset operator. If the specifications for the trace patterns are:

```
tpat p1 addr=5f0
tpat p2 data=39xxxxxx and stat=write
tpat p5 addr=500
tpat p6 data=0xx39xxxx and stat=write
```

you can enter an equivalent expression to the one which caused the error by making the following changes to the trace patterns and using the NOR (~) operator in the **tsto** command.

```
U>tpat p1 addr!=5f0
U>tpat p2 data!=39xxxxxx or stat!=write
U>tpat p5 addr!=500
U>tpat p6 data!=0xx39xxxx or stat!=write
U>tsto p1 ~ p2 or p5 ~ p6
```

Using the Sequencer

This section describes how to use the sequencer in the "complex" configuration. The differences between using the sequencer in the "easy" configuration and in the "complex" configuration are summarized in the following table.

Differences Between the "Easy" and "Complex" Analyzer Configurations

Analyzer Feature	In the "easy" configuration . . .	In the "complex" configuration . . .
sequence terms and the trigger (**tsq**)	You can insert or delete terms from the sequencer, and the branch out of the last sequence term constitutes the trigger.	There are always eight terms in the sequencer. Any of the sequence terms except the first may be specified as the *trigger term*. Entry into the trigger term constitutes the trigger.
simple trigger (**tg**)	The simple trigger command (**tg**) sets up a one term sequencer, and the expression specified with the **tg** command becomes the primary branch expression of the first sequence term.	The simple trigger command (**tg**) sets the primary branch expression of sequence term 1, and specifies the second sequence term as the trigger term.
primary branch expressions (**tif**)	Primary branches are always made to the next higher sequence term.	Primary branches may be made to any sequence term.
secondary branch expressions (**telif**)	The secondary branch expression is a global restart. In other words, the secondary branch expression applies to all sequence terms, and the branch is always back to the first sequence term.	Secondary branch expressions may be specified for each sequence term. Also, secondary branches can be made to any sequence term.
storage qualifiers (**tsto**)	The trace storage qualifier is "global" and applies to all sequence terms.	A storage qualifier is associated with each sequence term; however, the **tsto** command still allows you to specify storage qualifiers globally.

In the complex configuration, you perform the same tasks as are performed in the easy configuration. However, in the complex configuration, you have more sequence terms, you can specify destination terms for primary and secondary branches, and you can specify storage qualifiers for each sequence term.

This section describes how to:

- Reset the sequencer.
- Specify a simple trigger condition.
- Specify primary and secondary branches.
- Specify the trigger term.
- Specify storage qualifiers.
- Trace windows of execution.

To reset the sequencer

- Enter the **tsq -r** command.

After entering the "complex" analyzer configuration, the sequencer is in its default reset state.

If the analyzer is already in the "complex" configuration, you can reset the sequencer to its default state with the **tsq -r** command.

Examples

To reset the sequencer:

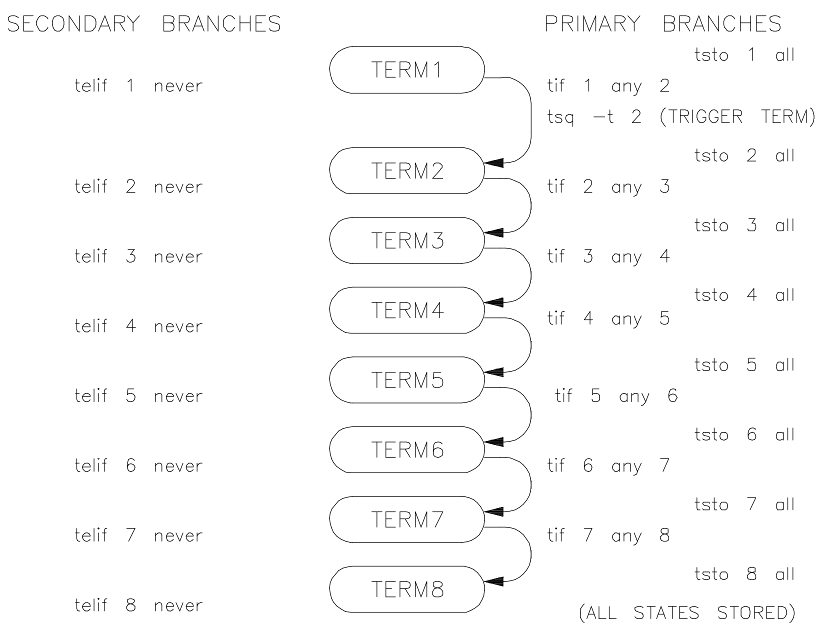
```
U>tsq -r
```

To display the default sequencer specification:

```
U>tsq
  tif 1 any 2
  tif 2 any 3
  tif 3 any 4
  tif 4 any 5
  tif 5 any 6
  tif 6 any 7
  tif 7 any 8
```

```
tif 8 never
tsq -t 2
tsto 1 all
tsto 2 all
tsto 3 all
tsto 4 all
tsto 5 all
tsto 6 all
tsto 7 all
tsto 8 all
telif 1 never
telif 2 never
telif 3 never
telif 4 never
telif 5 never
telif 6 never
telif 7 never
telif 8 never
```

There are eight terms in the "complex" configuration sequencer. By default, the primary branch expression for each term (except term 8) is **any**, the secondary branch expression for each term is **never**, and the storage qualifier for each term is **all**. The trigger term is the second sequence term. This sequencer specification will result in the same trace data as the default sequencer specification in the "easy" configuration (except that there will be more sequencer branches after the trigger). A diagram of the default sequencer specification is shown in the figure below.



If the **tsq** information scrolls off your screen, you may wish to display the sequencer specifications with a combination of other display commands; for example, you could enter the **tif**, **telif**, **tsto**, and **tsq -t** commands to display the same information.

To specify a simple trigger condition

- Use the **tg** command.

Using the **tg** (simple trigger) command in the "complex" configuration will cause the first two sequence terms to be modified. The pattern specified in the **tg** command becomes the primary branch expression of the first sequence term. The primary and secondary branch expressions of the second sequence term are set to **never**, and this term is specified as the trigger term. The secondary branch expression of the first sequencer term is also set to **never**.

The result of the **tg** command in the "complex" configuration is the same as in the "easy" configuration, and equivalent **tg** commands (where the pattern is the same as the "easy" configuration expression, and the storage qualifiers are the same) will yield identical traces in each of the trace configurations.

As in the "easy" configuration, the **tg** command with no options will display the primary branch expression of the first sequence term. This will only be the trigger condition when the second sequence term is the trigger term.

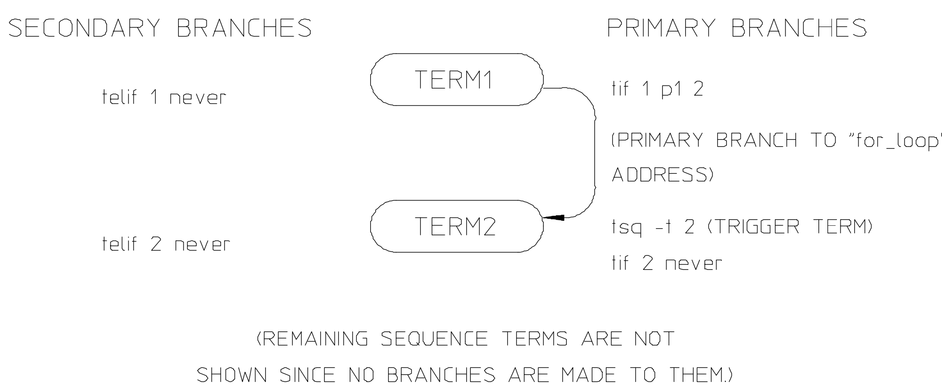
Examples

To set up a simple trigger on the branch to the first instruction in the demo program’s "for" loop (0A8EH):

```
U>sym for_loop=0a8e
U>tpat p1 addr=for_loop
U>tg p1
```

```
U>tsq
  tif 1 p1 2
  tif 2 never
  tif 3 any 4
  tif 4 any 5
  tif 5 any 6
  tif 6 any 7
  tif 7 any 8
  tif 8 never
  tsq -t 2
  tsto 1 all
  tsto 2 all
  tsto 3 all
  tsto 4 all
  tsto 5 all
  tsto 6 all
  tsto 7 all
  tsto 8 all
  telif 1 never
  telif 2 never
  telif 3 never
  telif 4 never
  telif 5 never
  telif 6 never
  telif 7 never
  telif 8 never
```

A diagram of this sequencer specification is shown in the figure below.



To specify primary and secondary branch expressions

- Use the **tif** and **telif** commands.

In the "easy" configuration, primary branches are always to the next sequence term. In the "complex" configuration, primary branches may be to any sequence term. Therefore, the number of the destination term must be specified before the occurrence count.

In the "easy" configuration, the secondary branch expression is a "global restart". It applies to all sequence terms and causes branches back to the first sequence term. In the "complex" configuration, you can specify secondary branch expressions for each sequence term and the branch may be to any sequence term. Therefore, the number of the destination term must be specified.

Examples

To specify a primary branch from sequence term 2 to sequence term 5 when the pattern p2 is found:

```
U>tif 2 p2 5
```

To specify a secondary branch from sequence term 2 to sequence term 3 when the pattern p3 is found:

```
U>telif 2 p3 3
```

To specify that the sequencer never branch out of term 5:

```
U>tif 5 never
U>telif 5 never
```

To specify the trigger term

- Use the **tsq -t** command.

In the "easy" configuration, the branch out of the last sequence term constitutes the trigger. In the "complex" configuration there are always eight terms in the sequencer, and any of the sequence terms except the first may be specified as the trigger term. Entry into the trigger term constitutes the trigger. The trigger term is specified with the **tsq -t** command.

Examples

To specify that entry into the fifth term constitutes the trigger:

```
U>tsq -t 5
```

To specify storage qualifiers

- Use the **tsto** command.

In the "easy" configuration, the trace storage qualifier is global, that is, it applies to all sequence terms. In the "complex" configuration, storage qualifiers are associated with each sequence term (though you can specify that one storage qualifier applies to all terms).

Prestore qualifiers still apply to all normal storage states; however, in the "complex" configuration, you specify pattern or range resources with the **tpq** command.

Examples

To store states matching pattern p4 while searching for the branch expressions of sequence term 7:

```
U>tsto 7 p4
```

To store states matching the range resource while searching for the branch expressions of sequence term 5:

```
U>tsto 5 r
```

To store all states when searching for branch expressions, except when searching for the branch expressions of sequence term 1:

```
U>tsto all
U>tsto 1 none
```

To trace windows of activity

1 Set up one sequence term as the window enable term.

2 Set up one sequence term as the window disable term.

3 Set up a trigger term.

4 Do not store states while searching for the window enable condition.

5 Store all states while searching for the window disable condition.

One common use for the "complex" configuration sequencer is to trace "windows" of execution or, perhaps, to eliminate "windows" of execution from traces.

For example, suppose you wish to trace only the execution within a certain range of addresses. These addresses could be a subroutine or perhaps they are just the addresses of instructions in which you are interested.

A simple windowing sequencer specification would consist of a window enable term, a window disable term, and perhaps a trigger term (if you wish to trigger on a condition other than the enable or disable terms). Only the states which occur between the window enable condition and the window disable condition are stored.

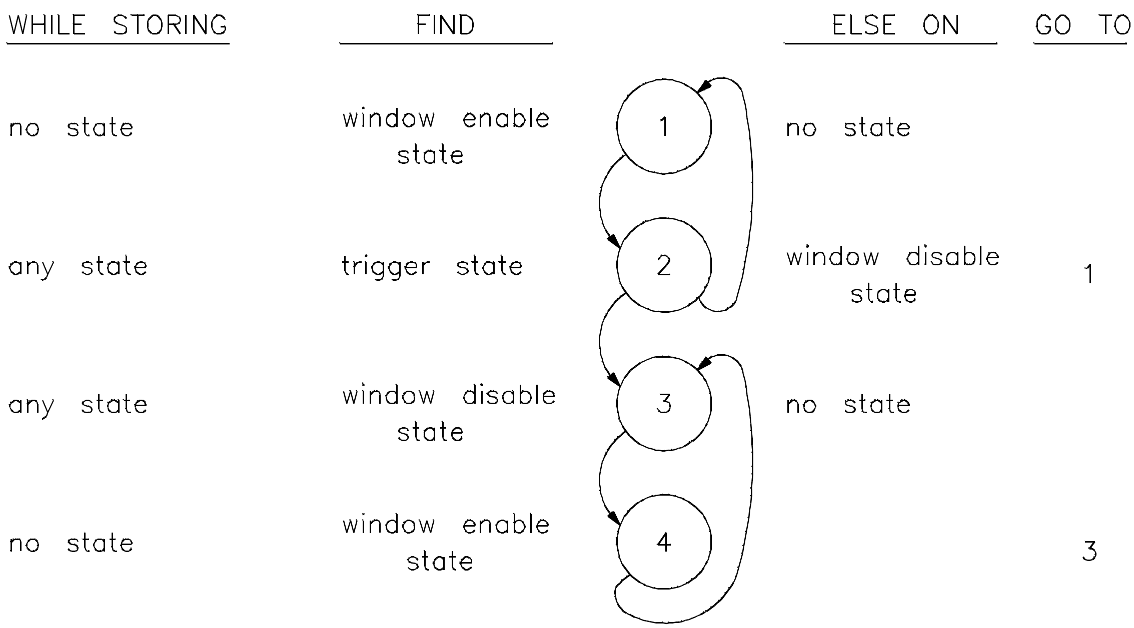
Examples

To trace only the execution of the analyzer demo program's "rand" subroutine, you would set up the sequencer specification so that the call to the "rand" subroutine is the window enable term and the return at the subroutine's "ret" instruction is the window disable term.

To display the "rand" subroutine in memory, enter the following command.

```
U>m -dm 121a..124d
  000121a  rand:_srand      MOVE.L  00004[A7],data_gen:__rand_s
  0001222  -                RTS
  0001224  -                MOVE.L  data_gen:__rand_seed,D0
  000122a  -                MOVE.L  #041c64e6d,D1
  0001230  -                JSR     lmul:intmul
  0001236  -                ADDI.L  #000003039,D0
  000123c  -                MOVE.L  D0,data_gen:__rand_seed
  0001242  -                CLR.W   D0
  0001244  -                SWAP.W  D0
  0001246  -                ANDI.L  #000007fff,D0
  000124c  -                RTS
```

Suppose that you wish to trigger on the "JSR to lmul:intmul" instruction. The diagram of the sequencer to do this is shown in the figure below.



To define symbols for the enable, disable, and trigger states:

```
U>sym enable=1224
U>sym disable=124c
U>sym trigger=1230
```

To reset the sequencer:

```
U>tsq -r
```

To specify trace patterns:

```
U>tpat p1 addr=enable
U>tpat p2 addr=disable
U>tpat p3 addr=trigger
```

To specify the primary and secondary branch expressions:

```
U>tif 1 p1 2
U>tif 2 p3 3
U>telif 2 p2 1
U>tif 3 p2 4
U>tif 4 p1 3
```

To specify the trigger term:

```
U>tsq -t 3
```

To specify the storage qualifiers so that states are stored only while searching for the window disable condition (the first command below specifies all storage qualifiers to be **none**, and the second command specifies that all states be stored while searching for the window disable condition):

```
U>tsto none
U>tsto 2 all
U>tsto 3 all
```

To place the trigger position at the center of the trace:

```
U>tp c
```

To display the sequencer specification.

```
U>tsq
  tif 1 p1 2
  tif 2 p3 3
  tif 3 p2 4
  tif 4 p1 3
  tif 5 any 6
  tif 6 any 7
  tif 7 any 8
  tif 8 never
  tsq -t 3
  tsto 1 none
  tsto 2 all
  tsto 3 all
  tsto 4 none
  tsto 5 none
  tsto 6 none
  tsto 7 none
  tsto 8 none
  telif 1 never
  telif 2 p2 1
  telif 3 never
  telif 4 never
  telif 5 never
  telif 6 never
  telif 7 never
  telif 8 never
```

Starting the trace, waiting for the measurement to complete, and displaying the trace will result in the following information.

```
U>t
  Emulation trace started
U>ts
  --- Emulation Trace Status ---
  NEW User trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 266 (266) -10..255
  Sequence term 4
  Occurrence left 1
U>tl -de -t 60
```

```
 Line   addr,H        68000 Mnemonic,H                       count,R  seq
-----   ------------  ----------------------------------  ---------  ---
  -10   enable        MOVE.L  data_gen:__rand_seed,D0           ---   +
   -9   043fd0          0000  supr data wr word           0.200 uS   .
   -8   043fd2          0a98  supr data wr word           0.200 uS   .
   -7   001226          0006  supr prog                   0.200 uS   .
   -6   001228          04de  supr prog                   0.200 uS   .
   -5   00122a        MOVE.L  #041c64e6d,D1               0.200 uS   .
   -4   :__rand_seed    3930  supr data rd word           0.200 uS   .
   -3   0604e0          e21e  supr data rd word           0.200 uS   .
   -2   00122c          41c6  supr prog                   0.200 uS   .
   -1   00122e          4e6d  supr prog                   0.200 uS   .
    0   trigger       JSR     lmul:intmul                 0.200 uS   +
    1   001232          0000  supr prog                   0.200 uS   .
```

```
  2   001234          0f2e  supr prog                       0.200 uS   .
  3   lmul:intmul   SWAP.W  D1                              0.200 uS   .
  4   043fcc          0000  supr data wr word               0.200 uS   .
  5   043fce          1236  supr data wr word               0.200 uS   .
  6   000f30        SWAP.W  D0                              0.200 uS   .
  7   000f32        TST.W   D0                              0.200 uS   .
  8   000f34        BNE.B   0000f6e                         0.200 uS   .
  9   000f36        TST.W   D1                              0.200 uS   .
 10   000f6e        TST.W   D1                              0.280 uS   .
 11   000f70        BEQ.B   0000f58                         0.200 uS   .
 12   000f72        MOVEA.L D2,A0                           0.200 uS   .
 13   000f74        MOVEA.L D3,A1                           0.400 uS   .
 14   000f76        MOVE.L  D0,D2                           0.200 uS   .
 15   000f78        SWAP.W  D0                              0.200 uS   .
 16   000f7a        MOVE.L  D1,D3                           0.200 uS   .
 17   000f7c        SWAP.W  D1                              0.200 uS   .
 18   000f7e        MULU.W  D0,D3                           0.200 uS   .
 19   000f80        MULU.W  D1,D2                           0.200 uS   .
 20   000f82        MULU.W  D1,D0                           0.200 uS   .
 21   000f84        ADD.W   D3,D2                           2.720 uS   .
 22   000f86        SWAP.W  D0                              2.800 uS   .
 23   000f88        ADD.W   D2,D0                           2.800 uS   .
 24   000f8a        SWAP.W  D0                              0.200 uS   .
 25   000f8c        MOVE.L  A0,D2                           0.200 uS   .
 26   000f8e        MOVE.L  A1,D3                           0.200 uS   .
 27   000f90        RTS                                     0.200 uS   .
 28   lrem:intremz    2f08   unused prefetch                0.200 uS   .
 29   043fcc          0000  supr data rd word               0.200 uS   .
 30   043fce          1236  supr data rd word               0.200 uS   .
 31   001236        ADDI.L  #000003039,D0                   0.200 uS   .
 32   001238          0000  supr prog                       0.200 uS   .
 33   00123a          3039  supr prog                       0.200 uS   .
 34   00123c        MOVE.L  D0,data_gen:__rand_seed         0.200 uS   .
 35   00123e          0006  supr prog                       0.200 uS   .
 36   001240          04de  supr prog                       0.400 uS   .
 37   001242        CLR.W   D0                              0.200 uS   .
 38   :__rand_seed    bfe9  supr data wr word               0.200 uS   .
 39   0604e0          9aff  supr data wr word               0.200 uS   .
 40   001244        SWAP.W  D0                              0.200 uS   .
 41   001246        ANDI.L  #000007fff,D0                   0.200 uS   .
 42   001248          0000  supr prog                       0.200 uS   .
 43   00124a          7fff  supr prog                       0.200 uS   .
 44   disable       RTS                                     0.200 uS   +
 45   enable        MOVE.L  data_gen:__rand_seed,D0         46.28 uS   +
 46   043fd0          0000  supr data wr word               0.200 uS   .
 47   043fd2          0a98  supr data wr word               0.200 uS   .
 48   001226          0006  supr prog                       0.200 uS   .
 49   001228          04de  supr prog                       0.200 uS   .
```

7

Using the External State Analyzer

Using the External State Analyzer

The HP 64703A analyzer provides 16 external trace signals (in addition to the 64 channels of emulation analysis). These trace lines allow you to analyze additional target system signals. The external analyzer may be configured as an extension to the emulation analyzer, as an independent state analyzer, or as an independent timing analyzer.

Note that the external analyzer's independent timing mode (**xtmo -t**) cannot be used from the Terminal Interface. A host computer interface is necessary to provide timing analysis. Consequently, independent timing analysis is not described in this manual. Refer to the appropriate host computer interface manual (either the *68000 Emulator User's Guide for the PC Interface* or the *68000 Emulator User's Guide for the Softkey Interface*).

The tasks you perform with the external analyzer are grouped into the following sections:

- Setting up the external analyzer.
- Using the external analyzer as part of the emulation analyzer.
- Using the external analyzer as an independent state analyzer.

Setting Up the External Analyzer

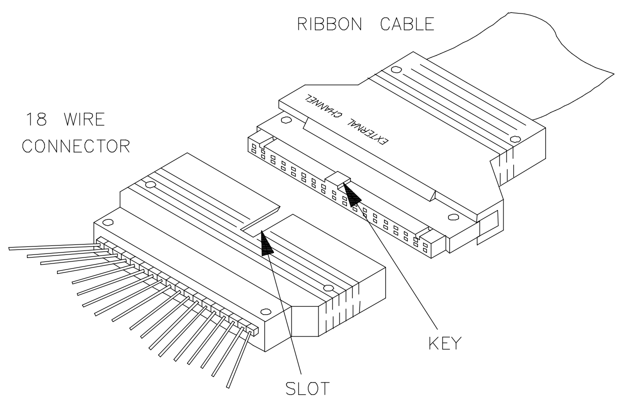
This section assumes you have already connected the external analyzer probe to the HP 64700 Card Cage as described in Step 1, panel 13 of the "Installation" chapter.

Before you can use the external analyzer, you must:

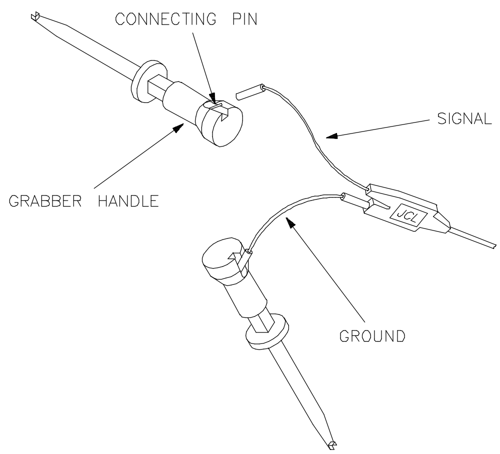
- Connect the external analyzer probe to the target system.
- Specify threshold voltages of external trace signals.
- Label the external trace signals.
- Select the external analyzer mode.

To connect the external analyzer probe to the target system

1 Assemble the Analyzer Probe. The analyzer probe is a two-piece assembly, consisting of ribbon cable and 18 probe wires (16 data channels and the J and K clock inputs) attached to a connector. Either end of the ribbon cable may be connected to the 18 wire connector, and the connectors are keyed so they may only be attached one way. Align the key of the ribbon cable connector with the slot in the 18 wire connector, and firmly press the connectors together.

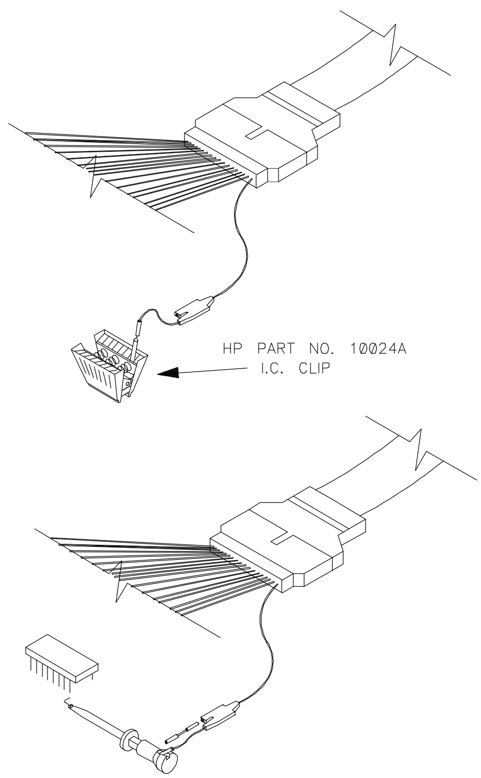


2 Attach grabbers to probe wires. Each of the 18 probe wires has a signal and a ground connection. Each probe wire is labeled for easy identification. Thirty-six grabbers are provided for the signal and ground connections of each of the 18 probe wires. The signal and ground connections are attached to the pin in the grabber handle.



CAUTION Turn OFF target system power before connecting analyzer probe wires to the target system. The probe grabbers are difficult to handle with precision, and it is extremely easy to short the pins of a chip (or other connectors which are close together) with the probe wire while trying to connect it.

3 You can connect the grabbers to pins, connectors, wires, etc., in the target system. Pull the hilt of the grabber towards the back of the grabber handle to uncover the wire hook. When the wire hook is around the desired pin or connector, release the hilt to allow the grabber spring tension to hold the connection.



To specify threshold voltages

- Use the **xtv** command.

The external analyzer probe signals are divided into two groups: the lower byte (channels 0 through 7 and the J clock), and the upper byte (channels 8 through 15 and the K clock). You can specify a threshold voltage for each of these groups.

The default threshold voltages are specified with the keyword **TTL** which translates to 1.4 volts.

Use the **xtv** (threshold voltage for external trace signals) command to specify different threshold voltages. The **-l** option to **xtv** allows you to specify threshold voltages for the lower group. The **-u** option allows you to specify threshold voltages for the upper group.

Voltages may be in the range from -6.4 volts to 6.35 volts (with a 50 mV resolution); you may also use the keywords **TTL**, **CMOS** (which translates to 2.5 volts), or **ECL** (which translates to -1.3 volts).

Examples

To specify CMOS threshold voltages for all external trace signals:

```
U>xtv -l CMOS -u CMOS
```

To define external trace labels

- Use the **xtlb** command.

You may wish to define external trace labels to make specifying qualifiers easier. External trace labels may be used in any of the external analyzer modes.

One external trace label has been predefined, **xbits**. This label is associated with all 16 external trace signals. This label appears in the default trace format and listing.

If you wish to define external trace labels to further break down the external signals, use the **xtlb** (external trace label) command.

You may change the trace listing format (**xtf** or **tf**) to include external labels in the trace after they have been defined.

Examples

To define an external analyzer label, iodata, for external analyzer signals 0 through 7:

```
U>xtlb iodata 0..7
```

Using as Part of the Emulation Analyzer

By default, on power-up or after trace initialization (**tinit**), the external analyzer is aligned with the emulator. In this mode, you have 16 external trace signals which are clocked with the same signal as the emulation analyzer. The external trace signals may be used to capture target system signals synchronized with the emulation clock.

When the external analyzer operates as an extension of the emulation analyzer, they operate as one analyzer. The only external trace commands allowed in this mode are **xtv**, **xtlb**, and **xtmo**. You can, however, display the help text for the other external trace commands. The external labels may be referenced in emulation trace commands in this mode.

External trace signal data is captured on the trace clock specified in the **tck** (trace clock source) command. You should not use the external J and K signals to clock the emulation trace; however, you may wish to use these signals to qualify the emulation trace clock (refer to the "Qualifying the Analyzer Clock" section of the "Using the Analyzer — Easy Configuration" chapter.)

To select the "emulation analyzer extension" mode

- Enter the **xtmo -e** command.

To re-select the emulation analyzer extension mode, use the **xtmo -e** command.

Using as an Independent State Analyzer

The external analyzer may also operate as an independent state analyzer. The independent state analyzer is identical to the emulation analyzer, except that only 16 bits of analysis are available. The analyzer now acts as two separate state analyzers; two sets of analyzer resources (trace memory, patterns, qualifiers, etc.) are available, one for the emulation analyzer and one for the independent state analyzer.

When the independent state analyzer mode is selected, you can use one analyzer to arm the other. You can specify the arm condition as a qualifier, perhaps as the trigger condition (cross-triggering). (Refer to the "Making Coordinated Measurements" chapter for more information on cross-triggering.)

This section describes how to:

- Select the independent state external analyzer mode.
- Specify the external analyzer clock.
- Specify the maximum qualified clock speed.
- Qualify the external analyzer clock.
- Use slave clocks for mixed clock demultiplexing.
- Use slave clocks for true demultiplexing.
- Arm the external analyzer with the emulation analyzer trigger.

To select the "independent state" mode

- Enter the **xtmo -s** command.

When you use the external analyzer as an independent state analyzer, a whole new set of external trace commands become available. Trace commands (except for the trace activity, **ta**, and trace initialization, **tinit**, commands) are duplicated for the independent state analyzer and prefixed with an **x**.

The following commands become available in the independent state mode: **xt**, **xtarm**, **xtcf**, **xtck**, **xtcq**, **xtelif**, **xtg**, **xth**, **xtif**, **xtl**, **xtlb**, **xtp**, **xtpat**, **xtpq**, **xtrng**, **xts**, **xtsck**, **xtsq**, and **xtsto**. These commands operate identically to their counterpart emulation analyzer commands.

To specify the external analyzer clock source

- Use the **xtck -r <clock>**, **xtck -f <clock>**, or **xtck -x <clock>** commands.

The independent state analyzer is typically clocked with target system clock signals connected to the J and K external clock inputs.

The independent state analyzer may also be clocked with the L, M, and N clock signals generated by the emulator. The L clock is the emulation clock derived by the emulator, the N clock is used as a qualifier to provide the user/background tracing options (**-u** and **-b**) to **tck**, and the M clock is not used.

Once a clock signal has been selected, you must specify whether the analyzer is to clock on the rising edge of the signal, the falling edge, or both the rising and falling edges. The edge is specified by the three following **xtck** options.

-r Specifies that the clock should take place on the rising edge of the signal(s) which follow.

-f Specifies that the clock should take place on the falling edge of the signal(s) which follow.

-x Specifies that the clock should take place on both edges of the signal(s) which follow.

When several clocks are specified, they are ORed; that is, each signal specified will clock the analyzer.

Examples

To specify that the external analyzer be clocked on the rising edge of the JCL signal:

```
U>xtck -r J
```

To specify that the external analyzer be clocked on the falling edge of the KCL signal:

```
U>xtck -f K
```

To specify that the external analyzer be clocked on both the rising and falling edges of the JCL signal:

```
U>xtck -x J
```

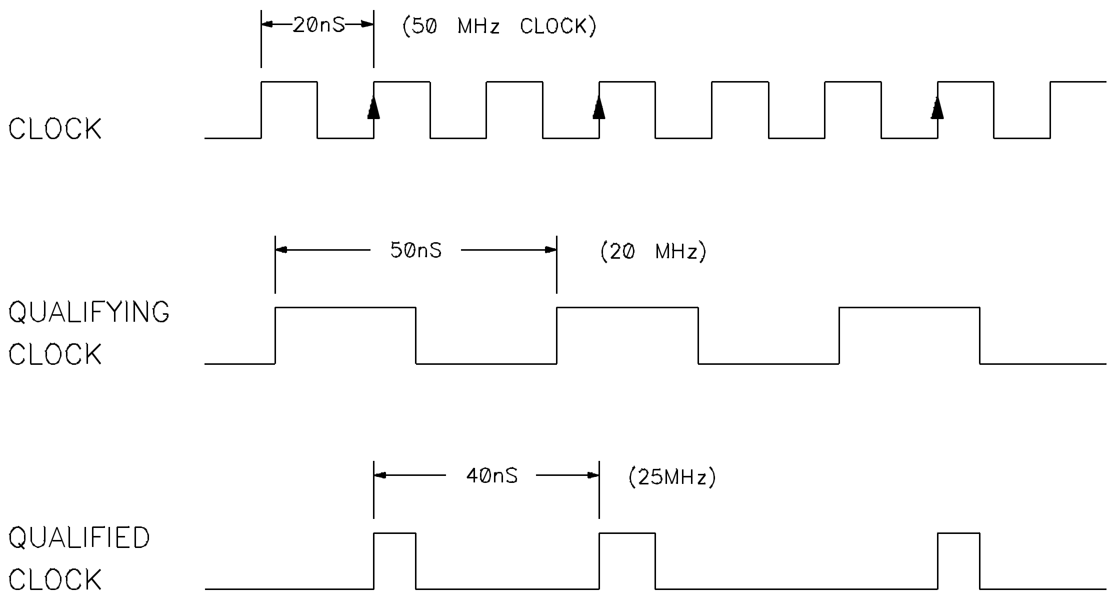
To specify the maximum qualified clock speed

- Use the **xtck -s S**, **xtck -s F**, or **xtck -s VF** commands.

The maximum qualified clock rate is the repetition rate of all specified clock signals (see the following figure). You are allowed to select the maximum qualified clock speed of the analyzer; however, there are tradeoffs involving the trace count qualifier to be considered:

- Slow (xtck -s S). Slow specifies a maximum qualified clock rate of 16 MHz. When S is selected, there are no restrictions on the trace count qualifier.
- Fast (xtck -s F). Fast specifies a maximum qualified clock rate of 20 MHz. When "F" is selected, the trace count qualifier may be used to count states but not time.
- Very Fast (xtck -s VF). Very fast specifies a maximum qualified clock rate of 25 MHz. When "VF" is selected, the trace count qualifier may not be used at all (in other words, **xtcq none**).

Examples


20nS
(50 MHz CLOCK)
CLOCK
50nS
(20 MHz)
QUALIFYING CLOCK
40nS
(25MHz)
QUALIFIED CLOCK

To qualify clocks

- Use the **xtck -h <clock>** or **xtck -l <clock>** commands.

Independent state analyzer clock signals may be qualified with other clock signals; that is, the selected signals may only clock the analyzer when the qualifying clock signal is true. Clock signals are qualified by using the **-l** and **-h** options to the **xtck** command.

The **-l** option is used to specify a qualifying signal which only allows the trace to clock when this signal is lower than the threshold voltage.

The **-h** option is used to specify a qualifying signal which only allows the trace to clock when this signal is higher than the threshold voltage.

Any signal, J, K, L, M, or N, may be used to qualify other signals.

Note that if several clock qualifiers are specified, the analyzer will be clocked if any one is true.

Qualifier setup time is approximately 25 nanoseconds when the external analyzer is aligned with emulation analyzer (**xtmo -e**). Qualifier setup time is approximately 20 nanoseconds when the external analyzer operates as an independent state analyzer (**xtmo -s**). Qualifier hold time is approximately 5 nanoseconds.

Examples

To allow the external analyzer to be clocked when the signal on KCL is high:

```
U>xtck -h K
```

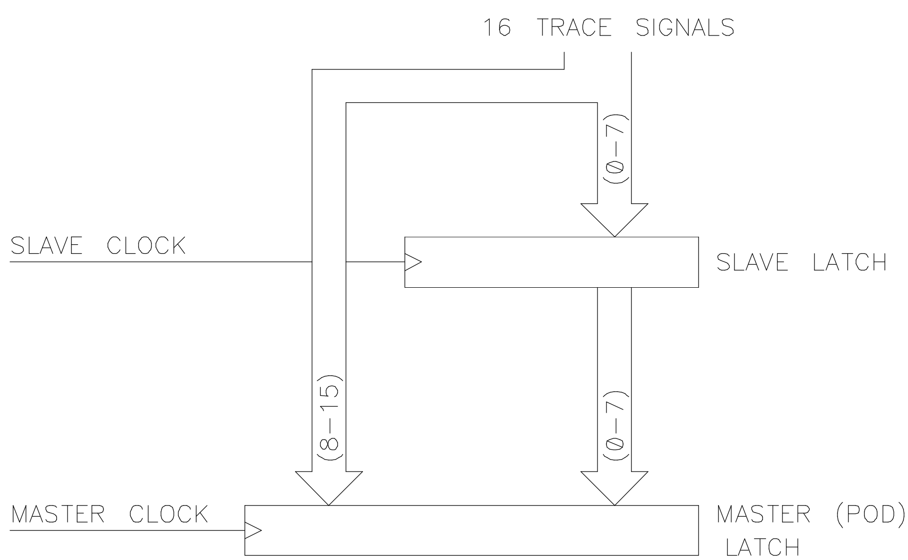
To use slave clocks for mixed clock demultiplexing

- Use the **xtsck -m** command.

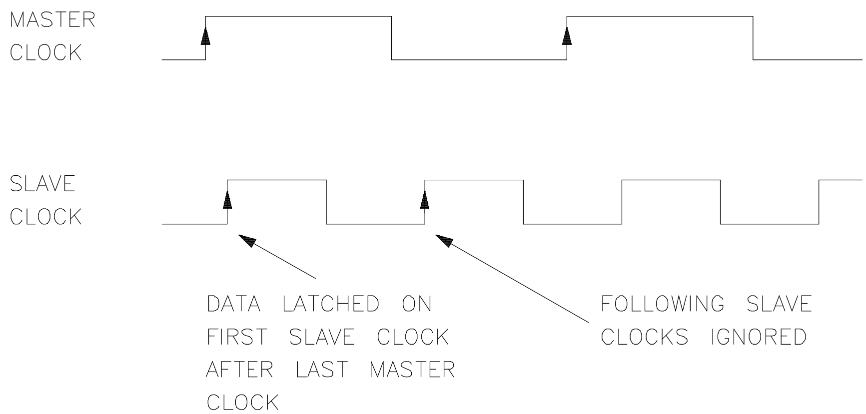
External analyzer slave clocks are specified with the **xtsck** (external trace slave clock) command. (Master clocks are specified by the **xtck** commands.) By default, the slave clocks are turned OFF, as may be specified by the **-o** option to the **xtsck** command.

Rising edges (**-r**), falling edges (**-f**), or both edges (**-x**) of clocks J, K, L, M, or N may be specified as the slave clock.

The mixed clock mode is specified with the **-m** option to the **xtsck** command. In this mode, the lower 8 channels of the pod (bits 0-7) are latched with the slave clock, and the master clock gates the entire pod (see the figure below).



If no slave clock has appeared since the last master clock, the data on the lower 8 bits of the pod will be latched at the same time as the upper 8 bits. If more than one slave clock has appeared since the last master clock, only the first slave data will be available to the analyzer (see the figure below).



Examples

To specify a slave clock on the rising edge of the KCL signal for mixed clock demultiplexing:

```
U>xtsck -m -r K
```

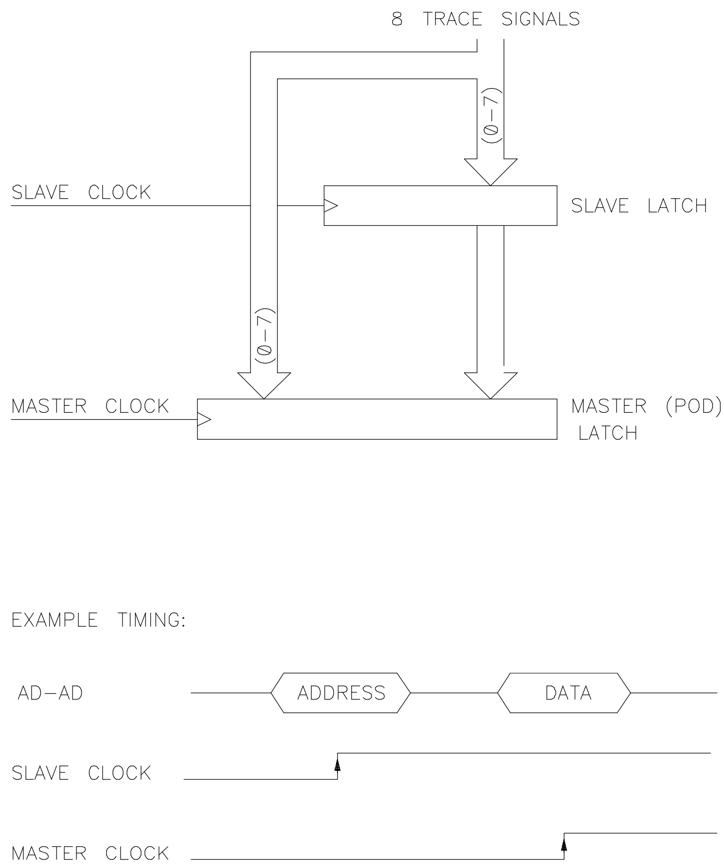
To use slave clocks for true demultiplexing

- Use the **xtsck -d** command.

External analyzer slave clocks are specified with the **xtsck** (external trace slave clock) command. (Master clocks are specified by the **xtck** commands.) By default, the slave clocks are turned OFF, as may be specified by the **-o** option to the **xtsck** command.

Rising edges (**-r**), falling edges (**-f**), or both edges (**-x**) of clocks J, K, L, M, or N may be specified as the slave clock.

The true demultiplexing mode is specified with the **-d** option to the **xtsck** command. In this mode, the lower 8 channels of the pod (bits 0-7) are latched with the slave clock; the upper 8 channels also get data from signals 0-7, but they are clocked with the master clock. Thus, the analyzer gets two copies of bits 0-7. The slave clock latches the data for bits 0-7, and the master clock then gates the entire pod into the analyzer (see the figure below).



If no slave clock has appeared since the last master clock, the data on the lower 8 bits of the pod will be the same as the upper 8 bits. If more than one slave clock has appeared since the last master clock, only the first slave data will be available to the analyzer.

Examples

To specify a slave clock on the rising edge of the KCL signal for true demultiplexing:

```
U>xtsck -d -r K
```

To arm the analyzer with the emulation analyzer trigger

- Use the **xtarm** command.

You can arm (that is, activate) the external analyzer when the emulation analyzer finds its trigger condition. The connection between the emulation analyzer and the external analyzer is made over one of the emulator's internal trigger signals (trig1 or trig2). You set up the emulation analyzer to drive the internal trigger signal when it finds its trigger condition, and you use the **xtarm** command to arm the external analyzer when the internal trigger signal appears.

Examples

To arm the external analyzer with the emulation analyzer's trigger output over the internal trig2 signal:

```
M>xtarm =trig2
M>xtg arm
M>xt
  External trace started
M>tg any
M>tgout trig2
M>r rst
U>t
  Emulation trace started
U>xts
  --- Emulation Trace Status ---
  NEW User trace complete
  Arm received
  Trigger in memory
  States 512 (512) 0..512
  Sequence term 2
  Occurrence left 1
```

8

Making Coordinated Measurements

Making Coordinated Measurements

When HP 64700 Card Cages are connected together via the Coordinated Measurement Bus (CMB), you can start and stop up to 32 emulators at the same time.

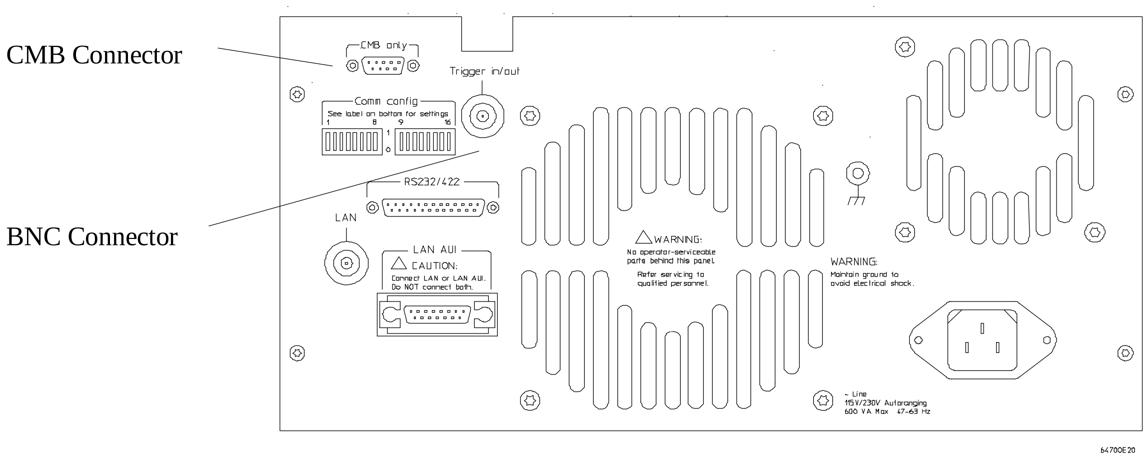
You can use the analyzer in one HP 64700 to arm (that is, activate) the analyzers in other HP 64700 Card Cages or to cause emulator execution in other HP 64700 Card Cages to break into the monitor.

You can use the HP 64700's BNC connector (labeled TRIGGER IN/OUT on the HP 64700 rear panel) to trigger an external instrument (for example, a logic analyzer or oscilloscope) when the analyzer finds its trigger condition. Also, you can allow an external instrument to arm the analyzer or break emulator execution into the monitor.

The coordinated measurement tasks you can perform are grouped into the following sections:

- Setting up for coordinated measurements.
- Starting and stopping multiple emulators.
- Using external trigger signals.

The location of the CMB and BNC connectors on the HP 64700 rear panel is shown in the following figure.



Signal Lines on the CMB

There are three bi-directional signal lines on the CMB connector on the rear panel of the emulator. These CMB signals are:

TRIGGER The CMB TRIGGER line is low true. This signal can be driven or received by any HP 64700 connected to the CMB. This signal can be used to trigger an analyzer. It can be used as a break source for the emulator.

READY The CMB READY line is high true. It is an open collector and performs an ANDing of the ready state of enabled emulators on the CMB. Each emulator on the CMB releases this line when it is ready to run. This line goes true when all enabled emulators are ready to run, providing for a synchronized start.

When CMB is enabled, each emulator is required to break to background when CMB READY goes false, and will wait for CMB READY to go true before returning to the run state. When an enabled emulator breaks, it will drive the CMB READY false and will hold it false until it is ready to resume running. When an emulator is reset, it also drives CMB READY false.

EXECUTE The CMB EXECUTE line is low true. Any HP 64700 on the CMB can drive this line. It serves as a global interrupt and is processed by both the emulator and the analyzer. This signal causes an emulator to run from a specified address when CMB READY returns true.

BNC Trigger Signal

The BNC trigger signal is a positive rising edge TTL level signal. The BNC trigger line can be used to either drive or receive an analyzer trigger, or receive a break request for the emulator.

Comparison Between CMB and BNC Triggers The CMB trigger and BNC trigger lines have the same logical purpose: to provide a means for connecting the internal trigger signals (trig1 and trig2) to external instruments. The CMB and BNC trigger lines are bi-directional. Either signal may be used directly as a break condition.

The CMB trigger is level-sensitive, while the BNC trigger is edge-sensitive. The CMB trigger line puts out a true pulse following receipt of EXECUTE, despite the commands used to configure it. This pulse is internally ignored.

Note that if you use the EXECUTE function, the CMB TRIGGER should not be used to trigger external instruments, because a false trigger will be generated when EXECUTE is activated.

Setting Up for Coordinated Measurements

This section describes how to:

- Connect the Coordinated Measurement Bus.
- Connect the rear panel BNC.

To connect the Coordinated Measurement Bus (CMB)

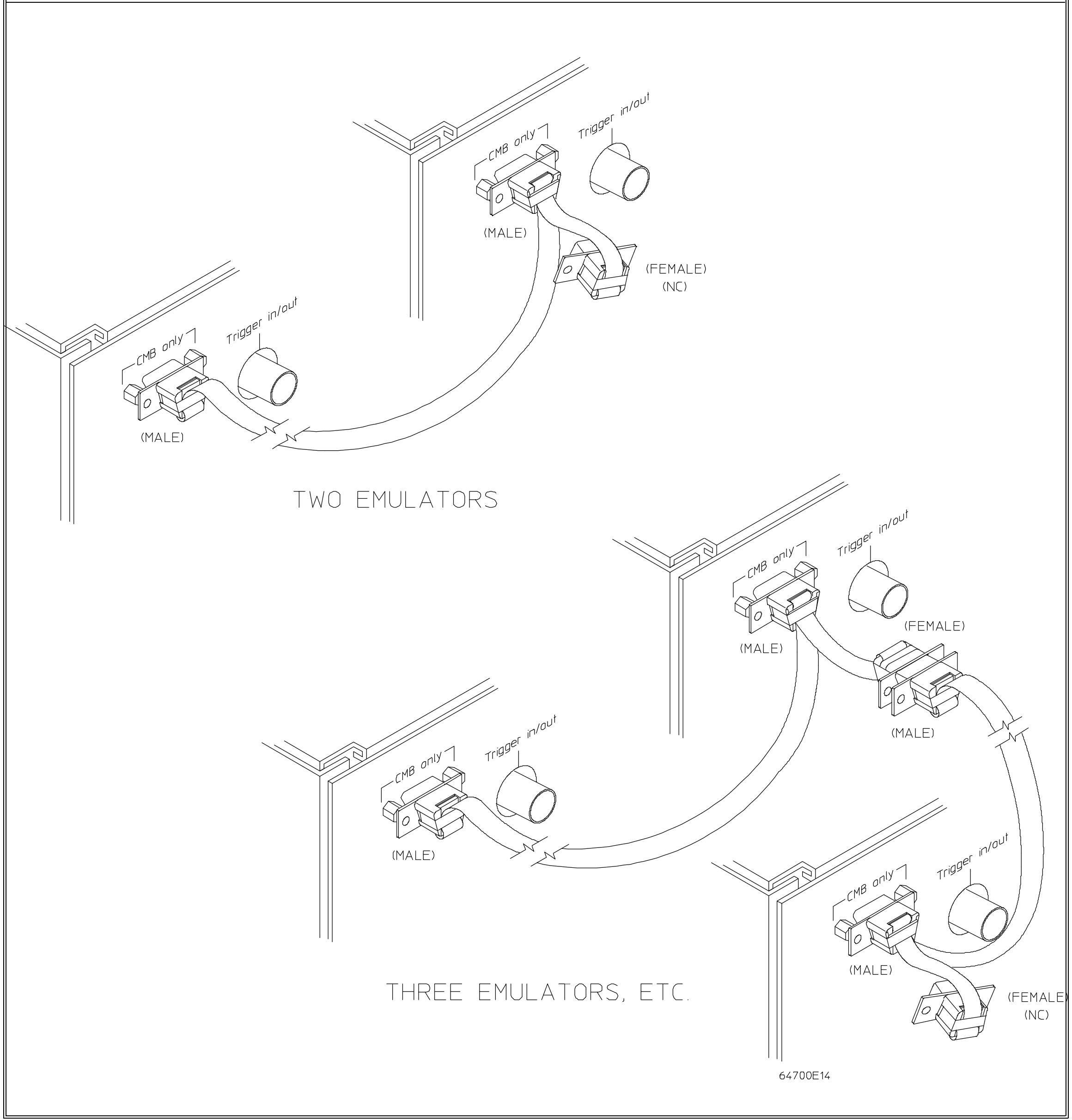
CAUTION Be careful not to confuse the 9-pin connector used for CMB with those used by some computer systems for RS-232C communications. Applying RS-232C signals to the CMB connector is likely to result in damage to the HP 64700 Card Cage.

To use the CMB, you will need one CMB cable for the first two emulators and one additional cable for every emulator after the first two. The CMB cable is orderable from HP under product number HP 64023A. The cable is four meters long.

You can build your own compatible CMB cables using standard 9-pin D type subminiature connectors and 26 AWG wire.

Note that Hewlett-Packard does not ensure proper CMB operation if you are using a self-built cable!

1 Connect the cables to the HP 64700 CMB ports.



Number of HP 64700 Series Emulators	Maximum Total Length of Cable	Restrictions on the CMB Connection
2 to 8	100 meters	None.
9 to 16	50 meters	None.
9 to 16	100 meters	Only 8 emulators may have rear panel pullups connected. *
17 to 32	50 meters	Only 16 emulators may have rear panel pullups connected. *

* A modification must be performed by your HP Customer Engineer.

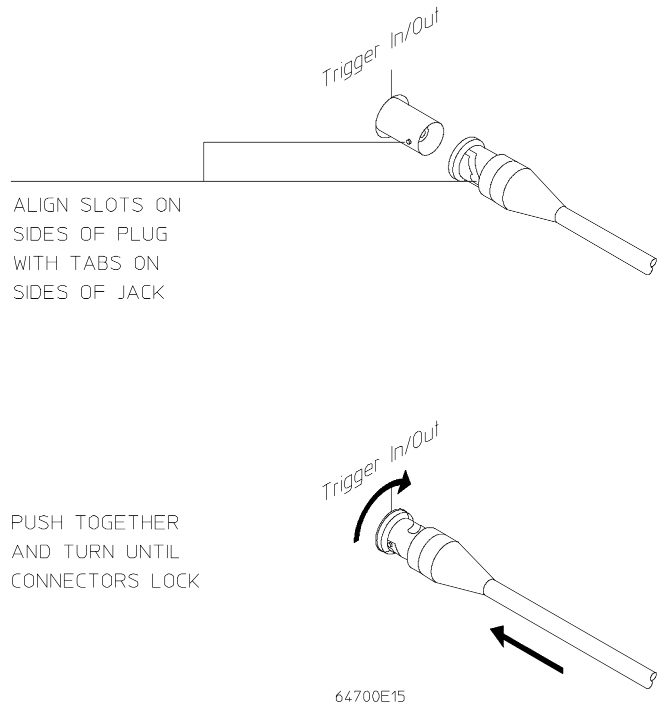
Emulators using the CMB must use background emulation monitors.

At least 3/4 of the HP 64700-Series emulators connected to the CMB must be powered up before proper operation of the entire CMB configuration can be assured.

To connect to the rear panel BNC

CAUTION The BNC line on the HP 64700 accepts input and output of TTL levels only. (TTL levels should not be less than 0 volts or greater than 5 volts.) Failure to observe these specifications may result in damage to the HP 64700 Card Cage.

1 Connect one end of a 50 ohm coaxial cable with male BNC connectors to the HP 64700 BNC receptacle and the other end to the appropriate BNC receptacle on the other measuring instrument.



The BNC connector is capable of driving TTL level signals into a 50 ohm load. (A positive rising edge is the trigger signal.) It requires a driver that can supply at least 4 mA at 2 volts when used as a receiver. The BNC connector is configured as an open-emitter structure which allows for multiple drivers to be connected. It can be used for cross-triggering between multiple HP 64700Bs when no other cross-measurements are needed. The output of the BNC connector is short-circuit protected and is protected from TTL level signals when the emulator is powered down.

Starting/Stopping Multiple Emulators

When HP 64700 Card Cages are connected together via the Coordinated Measurement Bus (CMB), you can start and stop up to 32 emulators at the same time. These are called synchronous measurements. This section describes how to:

- Enable synchronous measurements.
- Start synchronous measurements.
- Disable synchronous measurements.

To enable synchronous measurements

- Enter the **cmb -e** command.

You can enable the emulator's interaction with the CMB by using the **cmb -e** command. When the EXECUTE signal is received, the emulator will run at the address specified by the **rx** command. (Specifying an address with the **rx** command will automatically enable interaction with the CMB.)

The **tx -e** command enables the analyzer to start a measurement when the EXECUTE signal is received. If trace at execute is disabled (**tx -d**), the analyzer ignores the CMB EXECUTE signal.

Note that the **cmb** command does not affect the operation of analyzer cross-triggering.

Examples

To enable synchronous measurements with the **cmb -e** command:

```
U>cmb -e
```

To enable synchronous measurements with the **rx <address>** command:

```
U>rx 920
```

To start synchronous measurements

- Enter the **x** command.

The **x** command causes the EXECUTE line to be pulsed, thereby initiating a synchronous measurement. CMB interaction does not have to be enabled (**cmb -e**) in order to use the **x** command. (The **cmb -e** command only specifies how the emulator will react to the CMB EXECUTE signal.)

All emulators whose CMB interaction is enabled will break into the monitor when any one of those emulators breaks into its monitor.

Note that when the CMB is being actively controlled by another emulator, the step command (**s**) does not work correctly. The emulator may end up running in user code (NOT stepping). Disable CMB interaction (**cmb -d**) while stepping the processor.

```
U>x
```

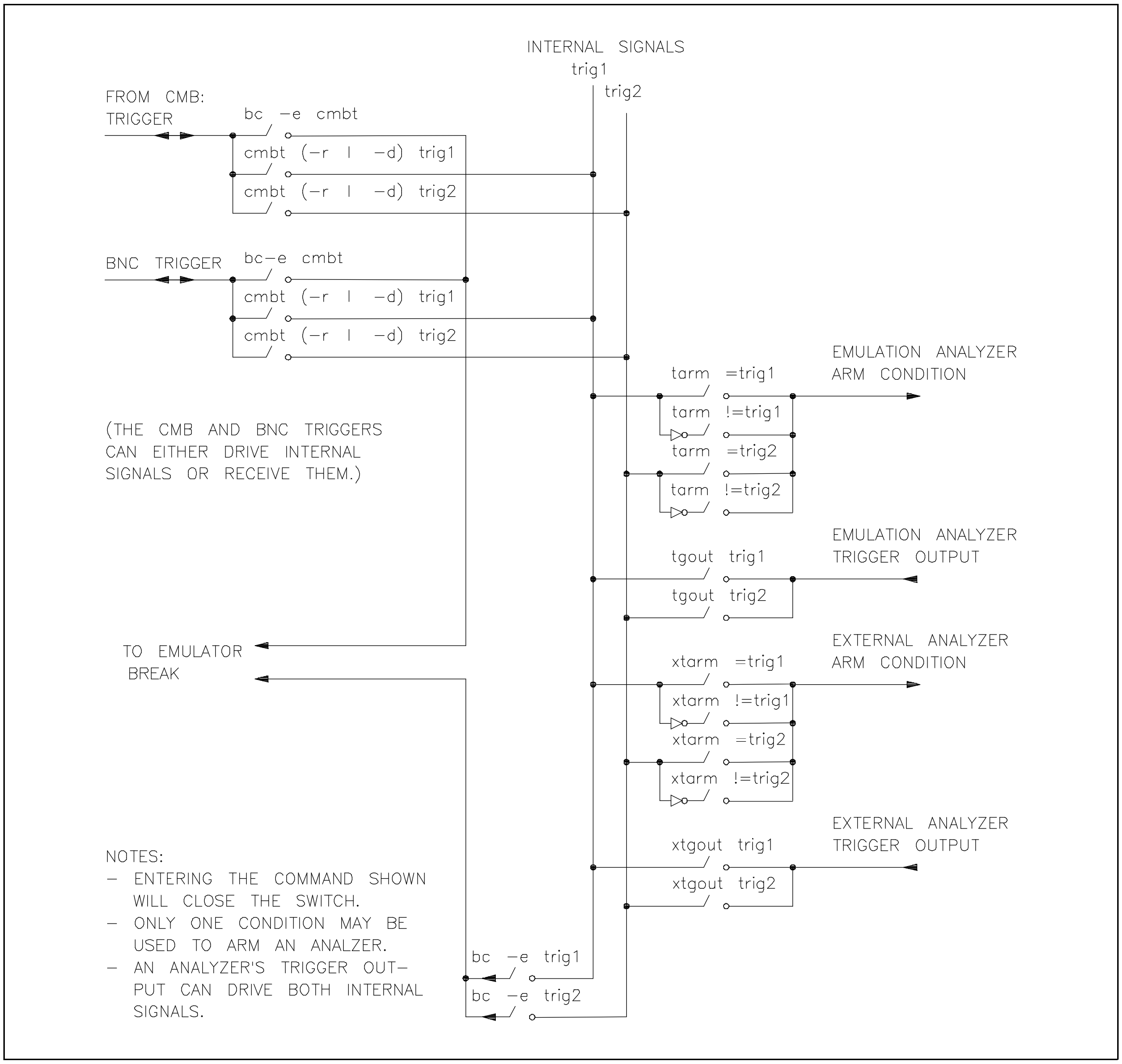
To disable synchronous measurements

- Enter the **cmb -d** command.

You can disable the emulator’s interaction with the CMB by using the **cmb -d** command. When interaction is disabled, the emulator ignores the CMB EXECUTE and READY lines.

Using External Trigger Signals

External trigger signals come from the CMB and BNC connectors. A diagram of the internal signals and the commands which may be used to drive them or to arm an analyzer with them are shown in the figure below. This diagram is only



intended to show logical connections, and does not represent actual circuitry inside the emulator.

This section describes how to:

- Arm analyzers with external trigger signals.
- Break emulator execution with external trigger signals.
- Send analyzers' trigger output signals to external lines.

To arm analyzers with external trigger signals

1 Use the **cmbt -d** or **bnct -d** commands to have the rear panel drive an internal trigger signal.

2 Use the **tarm** or **xtarm** commands to arm the analyzer on the internal signal.

By default, the analyzers are **always** armed. This means that the analyzers arm conditions are always true.

The **tarm** (trace arm condition) command is used to specify or display the emulation analyzer arm condition.

The **xtarm** (external trace arm condition) command is used to specify or display the independent state analyzer arm condition.

There are two internal signals, **trig1** and **trig2**, which may be specified as the arm condition. You can specify that the arm condition be true when one of these two signals is true (**=trig1** or **=trig2**).

By using **!=trig1** or **!=trig2**, you can specify that the analyzer be armed or never armed, depending on the state of the internal signal when the trace is started. .

The keyword **arm** may be used to specify primary and secondary branch qualifiers, as well as storage or prestore qualifiers.

It is often important to start the analyzer which receives a signal before the analyzer which drives the signal. For example, if you start the analyzer which drives a signal first, the signal may already be driven before you start the analyzer which receives the signal. The receiving analyzer will most likely capture states which execute long after the condition which caused the signal to be driven.

Examples

To arm the emulation analyzer when the external CMB trigger signal is true:

```
M>cmbt -d trig1
M>tarm =trig1
```

If you enter the following commands:

```
M>bnct -d trig2
M>tarm !=trig2
```

If the **trig2** signal is asserted when the analyzer is started, the analyzer can never be armed. If the **trig2** signal is not asserted when the analyzer is started, the analyzer is armed immediately.

To break emulator execution with external trigger signals

- Use the **bc -e cmbt** or **bc -e bnct** commands.

You can use the **bc -e cmbt** or **bc -e bnct** commands to enable emulator execution to break into the monitor when a trigger signal is received.

Examples

To enable breaks on the CMB TRIGGER signal:

```
R>bc -e cmbt
```

To enable breaks on the BNC TRIGGER signal:

```
R>bc -e bnct
```

To send analyzer trigger output signals to external lines

1 Use the **cmbt -r** or **bnct -r** commands to have the rear panel receive an internal trigger signal.

2 Use the **tgout** or **xtgout** commands to drive the trigger output to the internal signal.

The default condition of the analyzer specifies that neither the emulation analyzer nor the external analyzer will drive the internal **trig1** or **trig2** signals when the trigger is found.

The **tgout** command is used to specify that one of the internal signals be driven when the emulation analyzer trigger is found.

The **xtgout** command is used to specify that one of the internal signals be driven when the independent state analyzer trigger is found.

The **tgout** or **xtgout** commands with no options will display the signal which is currently being driven when the trigger is found (or **none** if no signal is driven when the trigger is found).

The signals which may be driven when the trigger is found are the internal signals **trig1** and **trig2**. The trig1 and trig2 signals may drive the CMB or BNC TRIGGER lines or the emulator break.

Note that you should not set up an analyzer to both drive and receive the same trigger signal. For example, if you issue the commands **tg arm; tarm =trig1; tgout trig1; bnct -d trig1 -r trig1**, the analyzer trig1 signal will become latched in a feedback loop and will remain latched until the loop is broken. To break the loop, you must first disable the signal's source, then momentarily disable either the drive or receive function. In this case, the commands **tgout none** and **bnct -d none** will break the loop.

Examples

To send the emulation analyzer trigger output to the CMB trigger line over the internal trig1 signal:

```
M>cmbt -r trig1
M>tgout trig1
```

To send the independent state analyzer trigger output to the BNC trigger line over the internal trig2 signal:

```
M>bnct -r trig2
M>xtgout trig2
```

Part 3

Reference

9

Commands

Commands

This chapter describes:

- The Terminal Interface commands.
- Analyzer state qualifier expressions.
- Values that that can be specified in commands.

<addr> - address specification in the 68000 emulators

```
XXXXXX                - 24 bit address
XXXXXX@<fc>           - 24 bit address with function code specifier
XXXXXX..XXXXXX        - address range, 24 bit address through 24 bit
                        address
XXXXXX..XXXXXX@<fc>   - address range with function code specifier
XXXXXX..              - 128 byte address range
XXXXXX@<fc>..         - 128 byte address range with function code
                        specifier
```

The 68000 emulator allows you to specify function code information in addition to the numerical address. This allows you to map separate regions of memory for user and supervisor program and data space.

The parameters are as follows:

<fc>

The function code may be any one of the following:

u — User
s — Supervisor
d — Data
p — Program
ud — User Data
up — User Program
sd — Supervisor Data
sp — Supervisor Program
x — don’t care

Examples

To map address ranges:

```
R>map 0..1fff eram
R>map 2000..3fff@d eram
R>map 2000..3fff@p eram
R>map 4000..5fff@sd eram
R>map 4000..5fff@sp eram
R>map 4000..5fff@up erom
R>map 4000..5fff@ud eram
R>map
  # remaining number of terms   : 9
  # remaining emulation memory  : 1000h bytes
  map  0000000..0001fff     eram   # term 1
  map  0002000..0003fff@d   eram   # term 2
  map  0002000..0003fff@p   eram   # term 3
  map  0004000..0005fff@sd  eram   # term 4
  map  0004000..0005fff@sp  eram   # term 5
  map  0004000..0005fff@up  erom   # term 6
  map  0004000..0005fff@ud  eram   # term 7
```

```
  map  other tram
```

Notice that you can map the same address range to different blocks of memory by using function code specifiers.

To display the contents of memory locations in these ranges:

```
R>m 4000..400f@sd
  0004000..000400f@sd  08 08 08 01 08 00 0c 01 00 00 08 01 00 08 01
01
R>m 2000..200f@d
  0002000..000200f@d   00 00 00 00 04 08 0c 01 08 00 00 00 08 00 0d
01
R>m 4000..400f@up
  0004000..000400f@up  08 00 08 00 09 00 00 01 00 00 0c 00 09 08 01
00
R>m 0..0f
  0000000..000000f     00 00 80 00 11 08 88 09 08 00 08 00 0c 00 0d
00
R>m 4000..400f
!ERROR  312! Ambiguous address: 0004000
R>m 2000..200f
!ERROR  312! Ambiguous address: 0002000
```

The "Ambiguous address" error message occurs because the address range exists in different blocks of memory (mapped with different function codes). If you don't use a function code specifier with the address range, the emulator doesn't know which block of memory you're referring to.

See Also

m (allows you to display or modify memory locations or ranges)

map (used to define the type and location of memory used by the emulator)

b - break emulation processor to monitor

b

The **b** command issues a break to the emulator, causing it to stop executing the user program and begin execution of the monitor program. If the emulator is in the reset state when a break occurs, it will be released from reset and will begin execution within the emulation monitor.

See Also

r (runs the user program from the current pc or a specified address)

s (steps the user program a number of instructions from the current pc or a specified address)

bc - set or display break conditions

```
bc                  - display current setting for all break conditions
bc -e <condition>               - enable specified break condition(s)
bc -d <condition>               - disable specified break condition(s)
bc -d <condition> <condition>   - multiple <condition>s allowed
```

The **bc** command allows you to set break conditions for the emulation system. This allows you to have the emulator break to the monitor upon error conditions (such as write to ROM), emulation processor trace events, or break to the monitor when a trigger signal is received.

The parameters are as follows:

-e — Enables the indicated break conditions (which must be specified immediately following the **-e** on the command line).

-d — Disables the indicated break conditions (which must be specified immediately following the **-d** on the command line).

<condition> — You can enable or disable the following break conditions:

bp — Software breakpoints and breakpoint registers. Software breakpoints and the processor breakpoint registers can not be configured independently. The "bp" condition enables or disables them both.

rom — Writes to ROM memory locations, as characterized when mapping memory.

bnct — Assertion of the rear panel BNC TRIGGER signal. Note that this signal may also drive either of the internal **trig1** or **trig2** signals or both.

cmbt — Assertion of the CMB (Coordinated Measurement Bus) TRIGGER signal. Note that the CMB trigger signal may also drive either of the internal **trig1** or **trig2** signals or both.

trig1 — Assertion of the internal **trig1** signal. Refer to the **tgout**, **bnct**, and **cmbt** commands for information on specifying drivers and receivers of the **trig1** signal.

trig2 — Assertion of the internal **trig2** signal. Refer to the **tgout**, **bnct**, and **cmbt** commands for information on specifying drivers and receivers of the **trig2** signal.

When you use the **bc** command, the emulator may break into the monitor while each enable/disable is being executed. If the emulator was executing your program when the **bc** command was received, it will return to your program when finished executing the command. If you request only a display of the current break conditions, the emulator does not break to the monitor.

When a hardware reset occurs during processing of the **bc** command, the result may be that a particular break condition is left in an unknown state. If this occurs, a display of the break conditions will show a question mark "?" instead of **-e** or **-d** next to the break condition.

Since the 68000 emulator prefetches instructions, it is possible that an additional instruction may execute after the one which originally caused the break condition. If this does occur, the additional instruction was already in the processor's instruction pipeline; the emulator has no way of aborting the execution of that instruction.

See Also

bnct (specify drivers and receivers of the rear panel BNC signal)

cmbt (specify drivers and receivers of the CMB trigger signal)

bp (set/delete software breakpoints)

map (specify whether memory locations are mapped as RAM or ROM)

tgout (specify whether the **trig1** and/or **trig2** signals are to be driven when the analyzer finds the trigger condition)

bnct - specify control of rear panel BNC signal

```
bnct              - display current bnct set up
bnct -d <dtype>   - rear panel BNC drives trig(1,2) signal(s)
bnct -r <rtype>   - rear panel BNC receives trig(1,2) signal(s)

--- NOTES ---
All option combinations are accepted:
'bnct -d trig1,trig2 -r trig1,trig2'   is a valid command
```

The **bnct** command allows you to specify which of the internal **trig1** or **trig2** trigger signals will drive and/or receive the rear panel BNC trigger. You can specify the signals individually, as an ORed condition for drive, or as an ANDed condition for receive; or, you can specify that the signals are not to be driven or received.

Upon powerup, **bnct** is set to **bnct -d none -r none**.

The parameters are as follows:

-d — Specifies that the rear panel BNC port drives the internal trigger signals, trig1 and trig2.

-r — Specifies that the rear panel BNC port receive the internal trigger signals, trig1 or trig2, and send them out the BNC port.

<dtype> — The valid drive options are:

trig1	When the BNC signal is received, drive trig1 signal.
trig2	When the BNC signal is received, drive trig2 signal.
none	When the BNC signal is received, drive neither signal.

<rtype> — The valid receive options are:

trig1	When trig1 signal goes true, send out the BNC signal.
trig2	When trig2 signal goes true, send out the BNC signal.
none	Neither trig1 or trig2 will send the BNC signal out.

Normally, you would use this command to cross-trigger instruments. For example, you may wish to trigger a digitizing oscilloscope hooked to various timing signals when the emulation analyzer finds a certain state, or, you may wish to do the converse and trigger the HP 64700's analyzer when an oscilloscope finds its trigger.

See Also

bc (break conditions; can be used to specify that the emulator will break into the emulation monitor upon receipt of one of the **trig1**/**trig2** signals)

cmbt (coordinated measurement bus trigger; used to specify which internal signals will be driven or received by the HP 64700 coordinated measurement bus)

tarm (analyzer trace arm; used to specify arming (begin to search for trigger) conditions for the analyzer **-- trig1/trig2** can be used to arm the analyzer)

tgout (specifies which of the **trig1/trig2** signals are to be driven when the analyzer trigger is found)

bp - set, enable, disable, remove or display software breakpoints

```
bp                - display current breakpoints
bp <addr>         - set breakpoint at <addr>
bp -e *           - enable all breakpoints
bp -e <addr>      - enable breakpoint at <addr>
bp -d *           - disable all breakpoints
bp -d <addr>      - disable breakpoint at <addr>
bp -r *           - remove all breakpoints
bp -r <addr>      - remove breakpoint at <addr>
bp <addr> <addr>  - more than one <addr> may be given
```

Upon powerup or **init** initialization, the breakpoint table is cleared and the breakpoint feature is disabled.

The parameters are as follows:

<addr> Specifies the address location where the software breakpoint is to be inserted. If you specify options **-e**, **-d**, or **-r**, the address specifies the location of the software breakpoint to be enabled, disabled, or removed.

-e Enables (activates) the breakpoint(s) at the address(es) specified. This installs the necessary breakpoint instruction in memory. If the breakpoint is already enabled, no action is taken.

-d Disables (deactivates) the software breakpoint(s) at the address(es) specified. When the software breakpoint is disabled, the original memory contents are restored if the breakpoint was enabled. The software breakpoint address(es) remain in the breakpoint definition table and can be reset by using the **bp -e <ADDRESS>** command.

-r Removes the software breakpoint(s) at the addresses specified. When the software breakpoint is removed, the original memory contents are restored if the breakpoint was enabled; then, the address is removed from the breakpoint table.

Note that when the breakpoint table is displayed with the **bp** command, the enable/disable status of each breakpoint is tested by reading the memory locations in question. If a software break instruction is found, the breakpoint is displayed as "enabled"; if not, the breakpoint is displayed as "disabled". If the software breakpoint is in target RAM and emulator is running under the real-time restriction, the breakpoint is displayed as "status unknown".

If the emulator executes an TRAP instruction that was placed by you (either through your compiler or via memory modification) and not by the **bp** command, an "undefined breakpoint" error message is generated.

If the emulator is executing in the user program when you define or modify breakpoints, it will break into the monitor for each breakpoint that is defined or modified. The emulator will return to user program execution after breakpoint definition or modification.

Remember that any operation which modifies memory or the memory map will alter the existing breakpoints. For example, if you load a new program in the same address range where breakpoints reside, the breakpoints will be destroyed. Changing the memory map will prevent the emulator from placing new breakpoints or enabling existing breakpoints.

The breakpoint break condition is enabled (**bc -e bp**) after power-up or initialization. If you disable the breakpoints break condition with the **bc -d bp** command, the software breakpoints currently defined will remain in the breakpoint table, but will be disabled and will remain in that state until the breakpoint feature is reenabled and the specified breakpoints are reenabled (**bc -e bp** and **bp -e <addr>**).

See Also

bc (enable/disable breakpoint conditions (including **bp**))

bpreg (enable/disable breakpoint registers)

mo (defines memory access and display modes; the **bp** command uses the currently defined modes when writing software breakpoints into memory)

cf - display or set emulation configuration

```
cf                 - display current settings for all config items
cf <item>          - display current setting for specified <item>
cf <item>=<value> - set new <value> for specified <item>
cf <item> <item>=<value> <item> - set and display can be combined
```

The parameters are as follows:

<item> Configuration item. The valid 68000 emulator configuration items are:

ba — Enable/disable bus arbitration with target system.

bat — Enable/disable bus arbitration tagging by analyzer.

bbk — Specify upper address bits to be driven to target system during background bus cycles.

bfc — Specify function code to be driven to target system during background bus cycles.

be — Enable/disable emulation processor response to target system /BERR signal on non-interlocked emulation memory accesses.

The emulator will always respond to the /BERR signal during all target system memory cycles and interlocked cycles regardless of the setting of the **cf be** configuration option.

dbc — Enable/disable drive of background bus cycles to target system.

lfc — Specify which function code areas in memory map will be targeted by subsequent **load** commands.

mon — Select background or foreground monitor.

Note that you can link the foreground monitor with user programs as long the user programs are not going to be loaded into target memory. The monitor program is used to load programs into target memory. Therefore, when user programs are to be located in target memory, you must assemble and link the foreground monitor separately, and load it before you load the user programs.

mondti — Enable/disable interlock of target system /DTACK with emulation /DTACK on monitor program cycles.

pdw — Select the processor data bus width. This configuration option is only valid when emulation 68HC001 or 68EC000 processors. When **cf pdw=8**, the emulator is set to 8-bit mode. When **cf pdw=16**, the emulator is set to 16-bit mode.

rrt — Restrict to real time runs (on or off). Certain commands, for example displaying target memory or processor registers, use the emulation monitor program. If the emulator is running the user program, the emulator will temporarily break into the monitor to access target system resources and return to executing the user program. If your target system requires the emulator to execute in real-time, you should restrict to real-time runs.

When **cf rrt=on** and the emulator is running user code, the system refuses all commands that cause a break except **rst** (reset), **r** (run), **b** (break to monitor), and **s** (step).

When **cf rrt=off**, the system accepts commands normally.

rssp — Specify system stack pointer value to load upon each transition from emulation reset to the monitor.

Note that a target system reset that occurs during background monitor operation will not affect the supervisor stack pointer value.

When a foreground monitor is used, the reset value of the supervisor stack pointer must be at least six bytes away from a guarded memory area. If the reset value of SSP is not six bytes away from a guarded area, a "Stack is in guarded memory" error will occur when you attempt to run the program.

swtp — Select trap vector to use for software breakpoint instruction.

ti	Enable/disable response to target system interrupts.

See Also

help (you can get an on line display of the configuration items for a particular emulator by typing **help cf**. To obtain more information regarding a particular configuration item, type **help cf <config_item>**).

cl - set or display command line editing mode

```
cl              - display command line edit mode
cl -e           - enable command line editing
cl -d           - disable command line editing
cl -l <columns> - number of columns for command line
```

The **cl** command allows you to enable or disable command line editing. Command line editing has two typing modes. The normal command entry is input mode. The input mode functions like normal (canonical) command entry. The control mode allows command modification.

The parameters are as follows:

-e
Enables command line editing.

-d
Disables command line editing.

-l <columns>
This option allows you to set the column length for the command line. The <columns> value can be from 40 to 132 columns.

The editing mode commands are as follows.

Command	Description
<ESC>	enter command editing mode
i	insert before current character
a	insert after current character
x	delete current character
r	replace current character
dd	delete command line
D	delete to end of line
A	append to end of line
$	move cursor to end of line
0	move cursor to start of line
^	move cursor to start of line
h	move left one character
l	move right one character
k	fetch previous command
j	fetch next command
/<string>	find previous command in history matching <string>
n	fetch previous command matching <string>
N	fetch next command matching <string>

cmb - enable/disable Coordinated Measurement Bus run/break

```
cmb          - display current setting
cmb -e       - enable CMB run/break interaction
cmb -d       - disable CMB run/break interaction
```

The **cmb** command allows you to enable or disable interaction on the CMB (Coordinated Measurement Bus). The CMB allows you to make measurements involving cross-triggering of multiple HP 64700 analyzers, and to synchronously run and break multiple emulators.

The **cmb** command only affects the ability for multiple emulators to run or break in a synchronized fashion. The analyzer trigger capability is unaffected by the **cmb** command.

If no options are supplied, the current state of CMB enable/disable is displayed.

The parameters are as follows:

-e Enables interaction between the emulator and the Coordinated Measurement Bus.

-d Disables interaction between the emulator and the Coordinated Measurement Bus.

When interaction is enabled via the **cmb -e** command, the emulator will run code beginning at the address specified via the **rx** command when the CMB /EXECUTE (/ means active low) pulse is received.

The CMB READY line is driven false while the emulator is running in the monitor. The line goes to the true state whenever execution switches to the user program.

Notice that if the **rx** command is given, CMB interaction is enabled just as if a **cmb -e** command was issued. Refer to the syntax pages for the **rx** command for further information.

When interaction is disabled via the **cmb -d** command, the emulator ignores the actions of the /EXECUTE and READY lines. In addition, the emulator does not drive the READY line.

See Also

rx (allows you to specify the starting address for user program execution when the CMB /EXECUTE line is asserted)

tx (controls whether or not the emulation analyzer is started when the /EXECUTE line is asserted)

x (pulses the /EXECUTE line, initiating a synchronous execution among emulators connected to the CMB and enabled)

Also, refer to the "Making Coordinated Measurements" for further information on CMB operation.

cmbt - specify control of the rear panel CMB trigger signal

```
cmbt              - display current cmbt set up
cmbt -d <dtype>   - rear panel CMB drives trig(1,2) signal(s)
cmbt -r <rtype>   - rear panel CMB receives trig(1,2) signal(s)

--- NOTES ---
All option combinations are accepted:
'cmbt -d trig1,trig2 -r trig1,trig2'   is a valid command
```

The **cmbt** command allows you to specify which of the internal **trig1/trig2** trigger signals will drive and/or receive the rear panel CMB (Coordinated Measurement Bus) trigger. You can specify the signals individually, as an ORed condition for drive, or as an ANDed condition for receive; or, you can specify that the signals are not to be driven and/or received.

If no options are specified, the current setting of **cmbt** is displayed. Upon powerup, **cmbt** is set to **cmbt -d none -r none**.

The parameters are as follows:

-d — Specifies that the rear panel CMB TRIGGER line drives the internal trigger signals, trig1 and trig2.

-r — Specifies that the rear panel CMB receive the internal trigger signals, trig1 or trig2, and send them out on the CMB TRIGGER line.

<dtype> — The valid drive options are:

trig1	When the CMB TRIGGER signal is received, drive trig1 signal.
trig2	When the CMB TRIGGER signal is received, drive trig2 signal.
none	When the CMB TRIGGER signal is received, drive neither signal.

<rtype> — The valid receive options are:

trig1	When trig1 signal goes true, send out the CMB TRIGGER signal.

trig2	When trig2 signal goes true, send out the CMB TRIGGER signal.
none	Neither trig1 or trig2 will send the CMB TRIGGER signal out.

You use this command to trigger other HP 64700 analyzers. For example, you may wish to start a trace on another HP 64700 analyzer when the analyzer in this emulator finds its trigger; or, you may wish to do the converse and trigger the analyzer in this emulator when another emulation analyzer finds its trigger.

See Also

bc (break conditions; can be used to specify that the emulator will break into the emulation monitor upon receipt of one of the **trig1**/**trig2** signals)

bnct (BNC trigger; used to specify which internal signals will be driven or received by the rear panel BNC connector)

cmb (Used to enable or disable interaction on the CMB. This does not affect whether measurement instruments can exchange triggers over the CMB; it only controls run/break interaction between multiple emulators)

tarm (analyzer trace arm; used to specify arming (begin to search for trigger) conditions for the analyzer **-- trig1/trig2** can be used to arm the analyzer)

tgout (specifies which of the **trig1/trig2** signals are to be driven when the analyzer trigger is found)

cp - copy memory block from source to destination

```
cp <dest_addr>=<addr>..<addr>  - copy range to destination address
```

The **cp** command allows you to copy a block of data from one region of memory to another.

When **cp** is executed, the data from the specified range is copied to the destination address, with the lower boundary data going to the destination address, lower boundary + 1 to destination + 1, and so on until the upper boundary of the source range is copied.

The parameters are as follows:

<dest_addr> Specifies the lower boundary of the destination range.

<addr> Specifies the lower, and possibly upper, memory address boundaries of the source range to be copied. The default is a hexadecimal number; other bases may be specified. You can use "<addr>.." to specify a range from the address through the next 127 bytes.

If the source or destination addresses reside within the target system, the emulator will break to the background monitor and will return to foreground after the copy is completed.

If memory mapped as guarded is encountered in the source or destination range during the copy, the command is aborted; however, all locations modified prior to accessing guarded memory are left in the modified state.

See Also

m (allows you to display or modify memory locations or ranges)

map (used to define the type and location of memory used by the emulator)

ser (used to search memory ranges for a specific set of data values)

dt - display or set current date and/or time

```
dt                        - display current date and time
dt <yymmdd>               - set current date
dt <hh:mm:ss>             - set current time
dt <yymmdd> <hh:mm:ss>    - set current date and time
```

The **dt** command allows you to set or display the current date and time stored by the HP 64700 series emulators.

Note that the emulator system date & time clock is reset when power is cycled.

If no parameters are specified, the current date and time settings are displayed.

The parameters are as follows:

<yymmdd> Current date in year, month, day format.

<hh:mm:ss> Current time in hour:minute:second format.

Examples

To display the current date and time settings at emulator powerup:

```
M>dt
  January 01, 1988  0:00:21
```

To set the date to August 18, 1987:

```
M>dt 870818
```

To set the date to August 18, 1987 and the time to 11:05:00 (the order of the two arguments is not significant):

```
M>dt 870818 11:05:00
```

Note that if **yy** is greater than 50, the year is assumed to be in the 20th century (in other words, **19yy**). If **yy** is less than 50, the year is assumed to be in the 21st century (in other words, **20yy**).

dump - upload processor memory in absolute file format

```
dump -i <addr>..<addr> - upload intel hex format
dump -m <addr>..<addr> - upload motorola S-record format
dump -t <addr>..<addr> - upload extended tek hex format
dump -h <addr>..<addr> - upload hp format (requires transfer protocol)
dump -b <addr>..<addr> - send data in binary (valid with -h option)
dump -x <addr>..<addr> - send data in hex ascii (valid with -h option)
dump -c <hex char> <addr>..<addr> - after uploading a hex ascii
                                    format file send this character
                                    to close the file
```

The **dump** command allows a host interface program to dump the contents of emulation and/or target system memory to a host file. The contents can be dumped in HP, Tektronix hex, Intel hex, and Motorola S-record formats by specifying various options on the command line.

When uploading the file in HP file format using the HP 64000 **transfer** software, record checking is performed automatically by the **transfer** protocol.

The parameters are as follows:

-i — Specifies in Intel hex record format. Note that the various options for HP file format transfer (such as **-x**, **-b**, and **-e**) are invalid with this format.

-m — Specifies the Motorola S-record format.

-t — Specifies the Tektronix extended hexadecimal format.

-h — Indicates that the memory contents will be dumped in HP absolute file format.

-b — Indicates that the records will be sent in binary; this is only valid with **-h** (HP file format).

-x — The records will be sent in hexadecimal; this is only valid with the **-h** option (HP file format).

-c <hex char> — Indicates that the ASCII hexadecimal character specified should be sent to the host at the end of the file upload.

<addr> — Specifies the lower, then upper, address boundaries of the memory range to be dumped. The default is a hexadecimal number; other bases may be supplied.

Note that the HP 64000 format ".X" file created with a "dump -hx" command has records that contain 136 fewer bytes of data than the file format standard allows. Because of this, HP 64000 format ".X" files which are created with the **dump** command may take longer to be processed by consumers of the ".X" file (depending on how the consumer processes sequential records).

See Also

load (used to load emulation memory from a host computer file)

echo - evaluate arguments and display results

```
echo "string"            - echo string to output
echo ‘string‘            - echo string to output
echo expression          - evaluate expression and display results in
                           hex
echo \<value>            - echo character equal to value
echo expression \<value> - multiple arguments may be evaluated
```

The **echo** command allows you to display ASCII strings or the results of evaluated expressions on the standard output device. You must enclose strings in single open quote marks (‘) (ASCII 60 hex) or double quotation marks (") (ASCII 22 hex). A string not enclosed in delimiters will be evaluated as an expression and the result will be echoed. In addition, you may supply a backslash with a two digit hex constant; the corresponding ASCII character(s) will be echoed.

Echoing strings or ASCII characters is particularly useful within macros, command files, and repeats where you wish to prompt the user to perform some action during a "wait for any keystroke" command (see description for **w**). The expression capability is useful as a quick calculator.

Note that all options may combined within the same echo command as long as they are separated by spaces.

The parameters are as follows:

string

Any set of ASCII characters enclosed between single open quote marks (‘), or double quotes ("). Since the command buffer is limited to 256 characters, the maximum number of characters in a string is 248.

Note that many keyboards (and printers) represent the single open quote mark as an accent grave mark. In any case, the correct character is ASCII 60 hexadecimal. The correct double quote character is ASCII 22 hexadecimal.

Note that a character which is used as a delimiter cannot be used within the string. For example, the string **"Type "C""** is incorrect and will return an error. The string **‘Type "C"‘** is correct.

expression

A valid expression. The expression will be evaluated and the result will be echoed.

<value>

Is the hex code for any valid ASCII character. More than one character can be echoed with a single command; each "nn" must be preceded by a backslash. A total of 62 ASCII characters can be represented within a single **echo** command.

This capability is particularly useful for sending non-displaying control characters to a terminal; refer to the examples below.

Examples

To echo the string "Set S1 to OFF" to the standard output, type the following:

```
M>echo "Set S1 to OFF"
Set S1 to OFF
```

A useful application of the backslash option is to send a terminal control characters:

```
M>echo \1b "H" \1b "J" \1b "&dBSet S1 to OFF"
```

The above command sends "<ESC>H<ESC>J<ESC>&dB Set S1 to OFF" to the terminal. On an HP 2392A terminal this homes the cursor, clears the screen, sets the video mode to inverse video, and writes the message "Set S1 to OFF". Therefore, the user would see the message "Set S1 to OFF" in inverse video at the upper left hand corner of an otherwise blank screen.

You might combine this with a macro command as part of a procedure. For example, type:

```
M>mac PROMPT={echo "Set S1 to OFF";w}
M>PROMPT
```

You will see:

```
Set S1 to OFF
Waiting for any keystroke...
```

To calculate the value of the expression (1f + 1e), type:

```
M>echo 1f+1e
03dh
```

See Also

mac (grouping a set of commands under a label for later execution)

rep (grouping a set of commands for immediate repetition)

w (wait command, allows user specified delays)

equ - define, display or delete equates

```
equ name=<value>         - equate name to number or pattern
equ name                 - display named equate
equ -d name              - delete named equate
equ -d *                 - delete all equates
equ *                    - list all equates
equ                      - list all equates
equ name1=<value> name2  - multiple operands allowed
```

The **equ** command allows you to equate arithmetic values with names that you can easily remember; these names can then be used in other commands to reference the value.

A number of equates have been predefined for common analyzer status values. The equates are present after the emulator is powered up or initialized.

The parameters are as follows:

name — A character string that names the equate to be displayed, deleted, or assigned a value. The name must be an alphanumeric designator no greater than 31 characters in length, beginning with an alpha character or underscore and including only alphanumeric characters or underscores thereafter.

<value> — An arithmetic expression to be assigned to the equate name.

-d — Deletes the named.

Note that each equate is translated to its actual value at the time of command entry. For example, if you specify an equate **count=21h**; and an expression **start=2000h**, then the command **tg addr=start count** will be entered into the system as **tg addr=start 33**. At this point, redefining the value of **addr** or **count** would not change the address expression or the occurrence counter for the trigger.

Note that the combination of a single **equ** command with all names and expressions cannot exceed 255 characters. The number of equates and symbols that may be defined is limited only by available system memory; thus, it is dependent on the number of equates, symbols, macros, etc. defined.

See Also

tg, tpat, tif, telif, and others. (**equ** provides an easy way to name expressions to use in setting up trigger or branch conditions)

r, m, bp (equates may be used to specify run addresses, memory addresses, or breakpoint addresses)

es - display current emulation system status

```
es
```

The **es** command displays the current status of emulation activity. The following types of information may be displayed:

R **-** emultor in reset state
U **-** running user program
M **-** running monitor program
W **-** waiting for CMB to become ready
c **-** no target system clock
r **-** target system reset active
h **-** processor halted
g **-** bus granted
b **-** no bus cycles
w **-** cpu in wait
? **-** unknown state

The emulator will not break to the monitor to obtain information. Therefore, any information that can only be obtained while in the monitor will not be displayed if the emulator is not in the monitor.

See Also

ta (allows you to display activity on emulation and external analyzer lines)

ts (allows you to display the current status of the emulation analyzer)

<expr> - analyzer state qualifier expressions

In the easy configuration:

```
any/all                                 - always true set
none/never                              - always false set
arm                                     - external qualifier
<label>=<value>                         - define state qualifier
<label>!=<value>                        - define state qualifier
<label>=<value> and <label>=<value> ...  - state and state
<label>!=<value> or <label>!=<value> ... - state or state
<label>=<value>..<value>                - define state range
<label>!=<value>..<value>               - define notstate range
```

In the complex configuration:

```
any/all                              - always true set
none/never                           - always false set
<set1>                               - single set
<set2>                               - single set
<set1> and <set2>                    - set global and set
<set1> or <set2>                     - set global or set
```

Analyzer state qualifier expressions are used in specifying triggers, time qualifiers, primary and secondary branch conditions, prestore qualifiers, and other analyzer setup items.

There are two types of analyzer expressions, simple and complex.

The parameters are as follows:

<label> A trace label that is currently defined via either the **tlb** or **xtlb** commands.

<value> Values are numeric constants, equates, or symbols. Also, values can be the result of constants, equates, and symbols combined with operators. Refer to the <value> description.

<set1> Consists of: p1, p2, p3, p4, r, !r. The pattern resources are assigned values with the **tpat** command and the range resource is assigned a value with the **trng** command.

<set2> Consists of: p5, p6, p7, p8, arm. The pattern resources are assigned values with the **tpat** command. The "arm" keyword specifies the arm condition as specified in the **tarm** or **xtarm** commands.

Resources within a set can be combined with intraset operators. Resources between the two sets can be combined with the interset operators.

Intraset Operators

You use intraset operators to form relational expressions between members of the same set. The operators are:

~ (intraset logical NOR)

| (intraset logical OR)

The operators must remain the same throughout a given intraset expression.

Interset Operators

You use interset operators to form relational expressions between members of set 1 and set 2. The operators are:

and (interset logical AND)

or (interset logical OR)

You can then form the following types of expressions:

(set 1 expression) and (set 2 expression)

(set 1 expression) or (set 2 expression)

The order of sets does not matter:

(set 2 expression) and (set 1 expression)

Combining Intraset and Interset Operators

You can use both the intraset and interset operators to form very powerful expressions. For example:

```
p1~p2 and p5|arm
p3 or p6~p7~p8
```

However, you cannot repeat different sets to extend the expression. The following is invalid:

```
p1~p2 and p5 and p3 and p7
```

DeMorgan's Theorem and Complex Expressions

At first glance, it seems that you only have a few operators to form logical expressions. However, using the combination of the simple and complex expression operators, along with a knowledge of DeMorgan's Theorem, you can form virtually any expression you might need in setting up an analyzer specification.

DeMorgan’s theorem in brief says that

A NOR B = (NOT A) AND (NOT B)

and

A NAND B = (NOT A) OR (NOT B)

The NOR function is provided as an intraset operator. However, the NAND function is not provided directly. Suppose you wanted to set up an analyzer trace of the condition

(addr=2000) NAND (data=23)

This can be done easily using the simple and complex expression capabilities. First, you would define the simple expressions as the inverse of the values you wanted to NAND:

```
tpat p1 addr!=2000
tpat p2 data!=23
```

Then you would OR these together using the intraset operators:

```
p1|p2
```

This is effectively the same as:

(NOT addr=2000) OR (NOT data=23) = (addr=2000) NAND (data=23)

If you need an intraset AND operator, you can use the same theory. Suppose you actually wanted:

(addr=2000) AND (data=23)

First, define the simple expressions as the inverse values:

```
tpat p1 addr!=2000
tpat p2 data!=23
```

Then you would NOR these together using the intraset operators:

```
p1~p2
```

This is effectively the same as:

(NOT addr=2000) NOR (NOT data=23) = (addr=2000) AND (data=23)

Examples

Some easy configuration examples include:

```
tg addr=2000
tif 1 data=20..30
telif addr!=3000 or data!=5
```

Some complex configuration examples include:

First, to assign values to pattern names:

```
tpat p1 addr=2000
tpat p2 addr!=3000
tpat p5 data!=5
trng data=20..30
```

Next, to create complex expressions within the analyzer commands:

```
tg p1
tif 1 r
telif 1 p2 or p5 3
```

To use intraset operators:

To store pattern 1 NOR pattern 2 NOR range:

```
tsto p1~p2~r
```

To trigger on pattern 2 OR (NOT range):

```
tg p2 | !r
```

help, ? - display help information

```
help <group>          - print help for desired group
help -s <group>       - print short help for desired group
help <command>        - print help for desired command
help                  - print this help screen
```

The **help (?)** command lets you display syntax, description and examples for any HP 64700 emulator Terminal Interface command. You may display a brief description for anything from a single command to command groups or the entire command set. Detailed information is available for single commands.

You may enter a question mark **?** instead of typing help; it performs the same function.

The parameters are as follows:

<group> The valid group names are:

gram	System grammar.
proc	Processor specific grammar.
sys	System commands.
emul	Emulation commands.
trc	Analyzer trace commands.
xtrc	External trace analysis commands.
*	All command groups.

-s Switches to the abbreviated help mode; only the expanded name of each command is displayed next to the command.

<command> Detailed help information is displayed for the named command.

Note that if you specify "*" for <command> or <group>, information for all commands will be displayed.

init - reinitialize system

```
init      - limited initialization; resets emulation and analysis
            products
            but not environment (macros, equates, date & time, etc..)
init -c   - complete initialization; does not run system memory
            integrity tests
init -p   - powerup initialization; run from reset with complete
            system verification tests
init -r   - powerup initialization; run from reset with complete
            system verification tests
            ignore all optional products
            do not use flash ROM
```

The **init** command allows you to re-initialize the emulator. Powerup, complete, and limited initializations are available through various options. In most cases you should only use this command if the emulator is not responsive to other commands.

If no options are specified, a limited initialization sequence is performed. The operating system and data communications are not affected but all of the emulation and analysis boards are reset. For example, a limited initialization would not change macro definitions, system date and time, or the data communications parameters, but the emulation memory map and breakpoint list would be reset to their default states.

The parameters are as follows:

-p — Specifies a powerup initialization sequence. This initializes the operating system, data communications, emulation and analyzer boards, and runs extensive performance verification.

-c — Specifies a complete initialization sequence. Everything is initialized as defined by the powerup sequence with the exception of the performance verification.

-r — Specifies a complete initialization with system verification tests (as with **-p**), but optional products and the flash ROM are ignored.

Note that the **init -c**, **init -p**, or **init -r** commands cause a loss of system memory. If these commands are used in macros, commands that follow them will not be executed.

See Also — **cf** (change emulation configuration)

dt (set system date and time)

map (define the emulation memory map)

stty (set data communications parameters)

tinit (reset the analyzer to powerup defaults)

lan - set configuration parameters

```
lan                 - display the current lan configuration
lan -l              - startup lan if not already started
lan -b              - enable BNC
lan -a              - enable AUI
lan -i <ip_addr>    - set Internet Protocol address
lan -g <ip_addr>    - set Internet Protocol Gateway address
lan -p <port>       - set TCP service port number
```

The parameters are as follows:

-l — Selects the LAN interface without having to change the HP 64700 configuration switch settings. Note that the serial interface is always active.

-b — Selects the LAN interface's BNC connector without having to change the HP 64700 configuration switch settings.

-a — Selects the LAN interface's AUI connector without having to change the HP 64700 configuration switch settings.

-i <ip_addr> — Internet Address in dot notation (for example, 192.6.94.2).

-g <ip_addr> — Gateway Address in dot notation (for example, 192.6.94.2).

-p <port> — Any number that is likely to be unused (for example, 6470).

lanpv - performance verification on LAN interface

```
lanpv -b   - testing performed through BNC connector
lanpv -a   - testing performed through AUI connector
lanpv -v   - print the error code value
```

To run performance verification, the connector under test must be removed from the network and capped with a terminator.

The parameters are as follows:

-b Tests the LAN interface through its BNC connector.

-a Tests the LAN interface through its 15-pin AUI connector.

-v Prints the error code value. The error codes and their meanings are:

load - download absolute file into processor memory space

```
load -i         - download intel hex format
load -m         - download motorola S-record format
load -t         - download extended tek hex format
load -S         - download symbol file
load -h         - download hp format (requires transfer protocol)
load -a         - reserved for internal hp use
load -e         - write only to emulation memory
load -u         - write only to target memory
load -b         - data sent in binary (valid with -h option)
load -x         - data sent in hex ascii (valid with -h option)
load -q         - quiet mode
load -p         - record ACK/NAK protocol (valid with -imt options)
```

The **load** command lets you load program code into emulation or target memory. Various file formats are supported via options to the load command. The destination of the program code is determined by the information contained in the program file. Additional options allow you to load only target memory or emulation memory as desired.

If a load error occurs, the current load procedure is aborted. However, records which were successfully loaded will remain in memory.

Note that at least one dash (**-**) must be included before any parameters are specified. It is optional to include or omit dashes for succeeding parameters. At least one file format option must be specified.

The parameters are as follows:

-i Specifies that the program code will be in Intel hex file format.

-m Specifies that the program code will be in Motorola S-record file format.

-t Specifies that the program code will be in extended Tektronix hexadecimal file format.

-h Specifies that the program code will be in HP file format. In this case, the file is expected to be transferred using the HP 64000 Hosted Development System **transfer** protocol.

-e Load only those portions of program code which would reside in memory mapped to emulation memory space. (Refer to the **map** command.)

-u — Load only those portions of program code which would reside in memory mapped to target memory space. (Refer to the **map** command.)

-q — The program code will be transferred in quiet mode. If **-q** is not specified, the emulator controller will write a "#" for each record successfully received and processed.

S — This allows you to download a symbol file from the host computer into the emulator.

-b — When using the HP file format, the program is expected to be in binary.

-x — When using the HP file format, the program is expected to be in hex.

-p — When using Intel, Motorola or Tektronix file formats, this option sets up a protocol checking scheme using ASCII **ACK/NAK** characters. If using this option, the host should send one record at a time and wait for the emulator to return an ASCII **ACK** character between records. If the emulator returns an ASCII **NAK** instead, there has been an error in data transmission. When the emulator receives the EOF character, it will return only the normal emulator prompt since data transmission is complete.

If, during the transfer, the host receives a **NAK** for a record, it should retransmit the record until an **ACK** is received or until a timeout value is reached, whichever occurs first.

Note that when you load an absolute file, the incoming data is examined for valid records (in the specified format). If the data being sent does not contain any valid records, the emulator will wait forever looking for valid records. The process must be terminated be entering a <CTRL>c.

See Also

dump (allows you to transfer emulation memory contents to a host)

m - display or modify processor memory space

```
m <addr>                - display memory at address
m -d<dtype> <addr>      - display memory at address with display option
m <addr>..<addr>        - display memory in specified address range
m -dm <addr>..<addr>    - display memory mnemonics in specified range
m <addr>..              - display 128 byte block starting at address A
m <addr>=<value>        - modify memory at address to <value>
m -d<dtype> <addr>=<value>       - modify memory with display option
m <addr>=<value>,<value>         - modify memory to data sequence
m <addr>..<addr>=<value>,<value> - fill range with repeating sequence
```

The **m** command allows you to display and modify emulation and target system memory. Options allow you to specify the display mode, specific address or addresses for display or modification, and the data values to be inserted.

At least one address must be specified. If no display mode is specified the display mode set by the **mo** command is used. Data items specified in memory modification are repeated as a group to fill the address range specified. The memory display defaults to the last value specified, or the default format for the emulator in use upon powerup initialization (varies dependent on the microprocessor being emulated).

If the selected address range for display or modification includes memory within the user's target system, emulator execution must break to the monitor in order to perform the access. After the command is complete, the processor will be returned to foreground execution if no errors occurred.

The parameters are as follows:

-d<dtype>

The **-d** option allows you to set the display mode for memory accesses. The valid display modes are:

b	display size is 1 byte(s)
w	display size is 2 byte(s)
l	display size is 4 byte(s)
m	display processor mnemonics

<addr>

Specifies the address to be displayed or modified. As noted in the syntax, an address followed by two periods and another address specifies a range of addresses to display or modify. The address default representation is a hexadecimal number.

<value> Data value to which a particular location is to be modified. If a range of locations is to be modified to a sequence of data values, the values must be separated by commas. Note that data may be specified in decimal, octal, or binary in addition to the hexadecimal default.

Note that the way the data item is handled depends on the display mode in effect. For example, if the display mode is byte, and the value entered is 1a3f66, the location specified will be modified to 66 hex. If the display mode is short, the location will be modified to 3f66 hex. And if the display mode is word, the location will be modified to 1a3f66.

Conversely, if you specify the value 33 hex for modification in byte mode, the value 33 is entered; in word mode, the value 0033 is entered; in long word mode, the value 000033 is entered. In other words, if the value supplied is shorter than the mode in effect, it is padded with leading zeros.

See Also

map (specify mapping of memory to emulation or user memory and to RAM or ROM)

mo (specify global access and display modes)

mac - display, define, or delete current macros

```
mac                          - display currently defined macros
mac <name>                   - display macro <name>
mac <name>={<cmd_list>}      - define macro <name> as list of commands
mac -d <name>                - delete macro <name>
mac -d *                     - delete all macros
mac -q                       - set expansion echo to quiet mode
mac -v                       - set expansion echo to verbose mode
```

The **mac** command allows you to save a group of commands under a name of your choice. This allows you to instantly recall that command group by typing in the assigned name. The emulator will then preprocess the macro to expand the commands stored in it to a normal command line. Then, the command line is then executed.

The parameters are as follows:

-d — The **-d** parameter, in conjunction with the macro <NAME>, deletes the macro defined by <NAME>. If <NAME> is given as the character "*" then all macros are deleted.

<name> — This represents the name you assign to the macro definition. Names can be any combination of alphanumeric characters; however, you cannot define a macro that has a name identical to that of another HP 64700 Terminal Interface command.

If you specify a name which is the same as a currently defined macro, that macro will be overwritten by the new macro you define.

<cmd_list> — This represents one or more emulator commands, including names which are used to define other macros. Commands in <cmd_list> must be separated from other commands by a semicolon (;).

When using command substitution, you can include pseudo-parameters in the form of "&token&" in the macro definition. Do not include any white space between the two "&" symbols. When you execute the macro, include the string to be substituted for &token& as a parameter on the command line. The macro will execute using the command expanded with the string you substituted.

-q — Sets the macro expansion echo to quiet mode. In this mode, any macro that you run will be executed without displaying the expanded command string.

-v — Sets the macro expansion echo to verbose mode. In this mode, any macro that you run will first display the expanded command string as a comment, and then will execute the macro.

Nested macro calls are permitted and limited only by constraints of system memory.

The commands within the macro definition are not checked for correct syntax until the macro is executed; therefore, it is advisable to test the command string before defining the macro.

The number of macros that can be created is limited to 100, but may be less depending on the complexity of the macros defined.

The length of the macro name combined with the macro definition is limited only by the maximum HP 64700 command length of 255 characters; thus, the macro name and definition can be a maximum of 251 characters.

A command within a macro definition cannot contain the pound sign character (#) unless the command is enclosed in a quoted string. (Otherwise, text following the # is interpreted as a comment.) This means there can be no matching brace at the end of the command. Use the **echo** command to place comments in a macro definition.

Command line substitution is possible when invoking a macro. During the macro definition, you may include pseudo-parameters which allow you to substitute parameters, such as file names, when invoking the macro.

See Also

rep (repeat; allows you to repeat any command, including macros)

map - display or modify the processor memory map

```
map                                  - display current map structure
map <addr>..<addr> <type> <attrib> - map address range as memory type
map other <type>                     - map other range as memory type
map -d <term #>                      - delete specified map term
map -d *                             - delete all map terms
```

Because the emulator can use target system memory or emulation memory (or both), it is necessary to map ranges of memory so that the emulator knows where to direct its accesses.

Up to 16 ranges of memory can be mapped, and the resolution of mapped ranges is 4 Kbytes (that is, the memory ranges must begin on 4 Kbyte boundaries and must be at least 4 Kbytes in length).

The parameters are as follows:

<addr> Specifies the address range to be mapped.

other Specifies unmapped address ranges. Only the **trom**, **tram**, or **grd** types can be specified for unmapped memory. The "other" range is unaffected when all mapper terms are deleted with the **map -d *** command.

<type> The valid types are:

eram Indicates that the given address range is to reside in emulation address space and act as RAM (read/write).

erom Indicates that the given address range resides in emulation address space; it is to act as ROM (read only).

tram Indicates that the given address range lies within target system RAM space. When the emulation processor accesses an address within this range, the target system data buffers will be enabled by a mapper signal to complete the transaction.

trom Indicates that the given address range lies within target system ROM space.

grd — The **grd** parameter indicates the given address range is to be "guarded". An emulation system break will be generated upon accesses to guarded memory.

<attrib> — The only valid emulation memory attribute is:

dti — Specifies that accesses in a range of emulation memory be synchronized with the target system. This means the termination of accesses in the range will not occur until the target system provides a /DTACK or /VPA. A /BERR signal from the target system will also terminate an emulation memory cycle and cause the emulator to begin execution of the bus error handler.

If this attribute is not specified, emulation memory accesses are terminated by a /DTACK signal generated by the emulator.

For emulation memory accesses that are not synchronized to the target system (that is, accesses to ranges that are mapped without the **dti** attribute), you can either allow the emulator to respond to target system /BERR signals by entering the **cf be=en** command, or cause the emulator to ignore target system /BERR signals by entering the **cf be=dis** command.

-d <term #> — Delete the mapped address range. The emulation system assigns a term number to each mapped address range. Term numbers are assigned in ascending order of address range.

When any map term is added or deleted the emulation processor will be reset and held in the reset state until a break or run command is issued. The processor remains reset in recognition of the fact that returning to execution directly after mapper modification is most likely invalid.

Be sure to disable all software breakpoints (**bc -d bp**) before changing the map. Software breakpoints are not cleared when the memory map is changed. After the new map and the program are set up, you can re-enable the breakpoints break condition (**bc -e bp**) and enter the **bp -e *** command to reenable the defined software breakpoints.

Note that the memory mapper re-assigns blocks of emulation memory after the insertion or deletion of mapper terms.

Note that if you map the same address range to different blocks of memory by using function code specifiers, all further references to addresses in that range must be fully specified with a function code. Otherwise, the emulator will return an "Ambiguous address" error message.

See Also

bc (break conditions; determines whether emulator breaks to monitor upon write to space mapped as ROM)

m (memory display/modify)

bp (set/delete software breakpoints)

mo - set or display current default mode settings

```
mo                  - display current mode settings
mo -d<dtype>        - set display mode to specified type
mo -a<atype>        - set access mode to specified type
```

The **mo** command allows you to modify the global access and display modes. Access mode is defined as the type of processor data cycles used by the emulation monitor to access a portion of user memory. Display mode is defined as the method used to display or modify data resident in memory.

The parameters are as follows:

-a<atype>

The **-a** option allows you to set the access mode. The valid access modes are:

d	the same as the display mode
b	byte, display size is 1 byte(s)
w	word, display size is 2 byte(s)

-d<dtype>

The **-d** option allows you to set the display mode. The valid display modes are:

b	byte, display size is 1 byte(s)
w	word, display size is 2 byte(s)
l	long, display size is 4 byte(s)
m	display processor mnemonics

At powerup or after **init**, the default access mode is set to **b** (byte) and the default display mode is set to **w** (word).

See Also

m (memory display/modify)

po - set or display prompt

```
po              - display the current port settings
po -p "string" - change the prompt string
```

The **po** command allows you to change the system prompt characters.

The parameters are as follows:

-p
Allows you to change the emulator's command prompt to one specified by **<STRING>**.

string
Any group of ASCII characters enclosed by single open quotes (‘) or double (") quote marks.

pv - execute the system performance verification diagnostics

```
pv                - display pv warning message
pv <repeat_count> - execute diagnostics <repeat_count> number of times
```

CAUTION

The **pv** command performs a system powerup initialization after all pv execution is completed. Therefore, all equates, macros, memory map, configuration settings, system clock, software breakpoints, trace specifications, and other configuration items you have altered will be cleared.

The **pv** command runs performance verification on the emulator and analyzer. The performance verification exercises all the emulator hardware and software to high confidence level.

You should only run performance verification when the emulation probe is plugged into the 64744-66509 demo board.

The parameters are as follows:

<repeat_count> Specifies the number of times to repeat the performance verification.

If **pv** reports failures, first check your hardware installation as described in the "Installation" chapter. If the failures persist, call your local HP Sales and Service office for assistance. A list of offices is provided in the *Support Services* guide.

Note that providing multiple commands such as **pv 1;r** is invalid; the second command will not execute due to the system reset.

Typing in <CTRL>c to abort the **pv** command may result in incorrect failure messages.

See Also

init (reinitializes the emulator)

r - run user code

```
r           - run from current Program Counter
r $         - run from current Program Counter
r <addr>    - run from address <addr>
r rst       - run from processor reset
```

The **r** command starts an emulation run. Execution begins at the address specified by the <addr> parameter; if no address is specified, execution begins at the address currently present in the program counter.

The parameters are as follows:

<addr> Specifies the address where execution is to begin. If you specify **$**, the processor runs from the current program counter value.

rst Specifies that the emulation processor runs from reset.

See Also

s (step; allows controlled stepping through program instructions)

rx (run only when CMB (Coordinated Measurement Bus) execute pulse is received)

x (pulse the CMB execute line if resident on the CMB)

reg - display and set registers

```
reg                          - display all basic register contents
reg *                        - display all basic register contents
reg <reg>                    - display contents of the named reg
reg <regclass>               - display contents of the named reg class
reg <reg>=<value>            - modify contents of the named reg
reg <reg> <reg>=<value> <regclass>   - display and set may be combined
```

The **reg** command allows you to display and modify emulation processor register contents. Individual registers may be displayed or modified; related groups of registers may be displayed; combinations of display and modify are permitted on the same command line.

The parameters are as follows:

<reg> The 68000 emulator supports display and modification of these individual registers:

pc	Program Counter
st	Status Register
usp	User Stack Pointer
ssp	System Stack Pointer
d0..d7	Data Registers 0 through 7
a0..a7	Address Registers 0 through 7. Notice that **a7** holds the current stack pointer value, depending on the setting of the user/supervisor bit in the **st** register.

<regclass> The only register class supported by the 68000 emulator is ***** (all registers).

<value> A numeric value.

See Also

s (step; allows you to step through program execution **--** combination with the **reg** command is useful in debugging)

rep - repeat execution of the command list multiple times

```
rep <value> {<cmd_list>}  - execute the command list <value> number
                            of times
rep 0 {<cmd_list>}        - execute the command list forever

<cmd_list> - list of valid commands separated by semicolons
```

The **rep** command allows you to repeat a group of commands and macros a specified number of times.

No other command input will be accepted until the command group has executed the indicated number of repetitions.

The parameters are as follows:

<value> An integer value specifying how many times the command list should be executed. A count of zero is a special case, meaning "repeat forever" (the repetition can be terminated by entering <CTRL>c, which issues a break signal to the emulator).

<cmd_list> Any valid HP 64700 command, including previously defined macros, may be specified with the options appropriate to the command. The list of commands must be preceded by an opening brace and followed by a closing brace. Also, the commands must be separated by semicolons. The commands will be executed in the same order as they are specified on the command line.

See Also **mac** (allows assignment of a name to a command group for easy recall of a specified command sequence)

rst - reset emulation processor

```
rst          - reset and stay in reset state
rst -m       - reset, then enter monitor
```

The **rst** command resets the emulation microprocessor. An option allows you to specify that the processor should begin executing the emulation monitor code immediately after the reset. If **-m** is not specified, the emulation processor remains in the reset state. Note that any commands which require the emulation processor to execute the monitor code for command processing will not execute while the processor is in the reset state; these include commands such as **reg**.

Commands or hardware signals which will take the emulator out of a reset state include **b**, **r**, **s**, and the CMB /EXECUTE pulse.

The parameters are as follows:

-m Causes the emulator to begin executing monitor code immediately after the reset.

rx - run at CMB-execute

```
rx          - display run-at-CMB-execute address
rx $        - when CMB-execute occurs, use the PC value at that time
rx <addr>   - set run-at-CMB-execute address
```

The **rx** command allows you to set the starting address for synchronous CMB (Coordinated Measurement Bus) execution.

The parameters are as follows:

<addr> Specifies where to start program execution when the CMB EXECUTE pulse is detected. If $ is specified for address, the current program counter value is used (default).

If the HP 64700 emulator is connected to the CMB, and the CMB-EXECUTE pulse is detected, followed by the CMB-READY line in the true state, the emulator will begin execution at the address specified by the **rx** command. If no **rx** command has been issued, execution begins at the current program counter value (same as **rx $**).

Execution will begin at the address specified by **rx** every time the conditions listed above are met. For example, if you type the command **rx 100**, the emulator will start executing at address 100 hex every time the CMB-EXECUTE line is pulsed.

The **rx** command automatically turns on CMB interaction by effectively performing the equivalent of a **cmb -e** command whether or not you have done so.

See Also

cmb (enables or disables CMB interaction)

x (initiates a synchronous CMB interaction by pulsing the CMB-EXECUTE line)

s - step emulation processor

```
s                      - step one from current PC
s <count>              - step <count> from current PC
s <count> $            - step <count> from current PC
s <count> <addr>       - step <count> from <addr>
s -q <count> <addr>    - step <count> from <addr>, quiet mode
s -w <count> <addr>    - step <count> from <addr>, whisper mode
```

The **s** command allows you to single-step the emulation processor through a program. You can specify the number of steps to execute at a single time; or, you can direct the emulator to step continuously. In addition, you may specify the starting address for stepping.

The parameters are as follows:

<count>

Specifies the number of steps to execute in sequence before returning command control.

The default base for <decimal> is decimal; however, other number bases may be specified.

If you do not specify a value for <count>, then a value of one (1) is assumed. If you specify a step count of zero (0), the emulator interprets this as "step continuously". Continuous stepping can be aborted with the <CTRL>c command; or, it will be terminated upon receipt of an emulation break condition such as a write-to ROM.

<addr>

Specifies the starting address for stepping. If you substitute **$** for the <addr> parameter, the current program counter value will be used as the <addr> value. The same will occur if no address parameter is specified.

Note that if you specify a value for <addr>, then you must specify a value for <count>. Otherwise, the address value will be interpreted as a step count; the emulator will step the number of locations specified.

-q

Stepping will occur in quiet mode; that is, the instructions and program counter are not displayed upon execution of each step.

-w

Stepping will be done in whisper mode; only the final program counter value is displayed after the step is executed.

If the emulator was in the run state (U> prompt) executing a user program when you request the step, it will break to the monitor program before executing the step.

Note that when the Coordinated Measurement Bus (CMB) is being actively controlled by another emulator, the step command (**s**) does not work correctly. The emulator may end up running in user code (NOT stepping). Disable CMB interaction (**cmb -d**) while stepping the processor.

See Also

r (run emulation processor from a specified address)

reg (view or modify processor register contents)

ser - search through processor memory for specified data

```
ser <addr>..<addr>=<value>           - search for data value in range
ser -d<dtype> <addr>..<addr>=<value> - search with display option
ser <addr>..<addr>=<value>,<value>   - search for data sequence in
                                       range
ser <addr>..<addr>="CDE"             - search for string "CDE" in
                                       range
```

The **ser** command allows you to search memory for a data value, a character string, or a combination of both. For every pattern match, the starting address of the match is displayed.

Using the **-d** (display mode) option, the method of interpreting the pattern supplied by the user can be altered. If no option is given, the display mode used is taken from global default set by the **mo** command.

The parameters are as follows:

<addr> — Specifies first the lower, and possibly the upper, address boundaries of the memory range to search for the given data pattern. You can use "<addr>.." to specify the range from the address through the next 127 bytes.

<value> — Either a numeric expression or a string to be used as a reference pattern in the search.

Strings must be bounded by single open quote marks (‘) or double quotes (").

Note that many keyboards (and printers) represent the single open quote mark as an accent grave mark. In any case, the correct character is ASCII 60 hexadecimal. The correct double quote character is ASCII 22 hexadecimal.

Note that if the character string you are searching for contains double quotes, you must delimit the string with single open quotes and vice versa. For example, the string **"Type "C""** will return an error; the string **‘Type "C"‘** is correct.

-d<dtype> — Allows you to specify the display mode used for the search. The valid display modes are:

b	display size is 1 byte(s)
w	display size is 2 byte(s)

l	display size is 4 byte(s)

If addresses specified in the search reside in target system memory, the emulator is broken to the monitor and returned to the user program when the command is completed.

Note that you can concatenate various combinations of values to form more complex search patterns by separating the parameters with commas (**,**).

See Also

cp (used to copy the contents of one memory range to another)

m (used to display/modify memory locations)

stty - set or display current communications settings

```
stty <port> <options>
```

Parameter	**<options>**
Parity	noparity, evenp, onep, zerop
Character Size	cs7, cs8
Baud	300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800
Protocol	rs232, rs422
Stop Bits	1stopb, 2stopb
Request/Clear to Send	crts, -crts
Data Set/Terminal Rdy	cdsr, -cdsr
Start/Stop	xon, -xon
Line Terminator	onlcr, ocrnl, ocr, onl
Echo	echo, -echo
Data Term/Comm Equip	dte, dce

The **stty** command allows you to modify the parameters of the serial data communications port without changing the configuration switch settings.

The serial port, port A, may be modified by **stty**.

The powerup default configuration for the serial port is determined by the rear panel configuration switches; refer to the "Installation" chapter for more information.

The parameters are as follows:

rs232
rs422

RS-422 utilizes balanced transmission lines and therefore can achieve much higher data rates with reliability over long distances than RS-232. Otherwise, the interfaces are similar.

dte
dce

The serial port may be set to operate either as Data Communications Equipment (DCE) or as Data Terminal Equipment (DTE). This configures the handshake lines and transmit/receive lines for the proper signal to pin relationships on the interface.

onlcr

Generate new-line and carriage-return on output.

ocrnl

Generate carriage-return and new-line on output.

ocr

Generate carriage-return on output.

onl

Generate new-line on output.

crts
-crts

The option **crts** enables the Request To Send/Clear To Send handshake. Specifying **-crts** disables this handshake.

cdsr
-cdsr

The option **cdsr** enables exchange and recognition of the Data Set Ready/Data Terminal Ready status lines. Specifying **-cdsr** disables the exchange.

xon
-xon

If you specify **xon**, the system generates XON/XOFF (DC1/DC3 characters) software handshaking to control the amount of data received at a given time. Specifying **-xon** disables this handshake sequence.

(When the emulator’s receive buffer is full, it will send a DC3 (XOFF) character to the host to stop transmission; when it is ready for more data, it will send a DC1 (XON) character to restart transmission.)

Note that if you toggle the **xon** parameter when running at 1200 baud and below, the **stty** command will return invalid characters. The PC Interface attempts to do this when starting up and fails with a datacomm error. To get around this problem, set switch 13 on the emulator’s back panel (enable xon) to allow the PC Interface to start up successfully. In the Terminal Interface, just enter another carriage return to regain proper communications.

echo
-echo

If you specify **echo**, all characters received by the emulator datacomm are echoed back to the sending system. Specifying **-echo** means the system will not echo back characters received.

You will normally use this in conjunction with the echo settings required by your host computer and your terminal. Most Hewlett-Packard systems will require that you enable the echo feature, as HP host computers automatically echo characters back to data terminal devices.

For further information on the meanings of various data communications parameters, you may refer to the book entitled *Touring Datacomm: A Data Communications Primer*. This book is orderable from HP's Direct Marketing Division under the part number 5957-4622. Another book which may be helpful is *The RS-232 Solution*, orderable from HP under the product number 92234X. You also may need to refer to the hardware and software reference manuals that are supplied with your terminal and/or host computer for further information on required data communications parameters for links to those devices.

Examples

To display the current data communications settings:

```
M>stty
```

```
stty A 9600 cs8 1stopb noparity dce rs232 -crts cdsr -xon onlcr echo
```

To set the baud rate to 1200 baud:

```
M>stty 1200
M>stty
```

```
stty A 1200 cs8 1stopb noparity dce rs232 -crts cdsr -xon onlcr echo
```

sym - define, display or delete symbols

```
sym <name>          - display all or named symbols
sym -g <name>       - display all or named global symbol
sym -u <name>       - display all or named user symbol
sym -l              - display all local modules
sym -l <name>       - display symbols in local module
sym <name>=<addr>   - define user symbol
sym -d              - delete all symbols
sym -du             - delete all user symbols
sym -du <name>      - delete named user symbol
sym -dg             - delete all global symbols
sym -dl             - delete all local symbols in all modules
sym -dl <name>      - delete all local symbols in module
```

The **sym** command defines, displays, or deletes symbols in the emulator. The **sym** command without any parameters displays all of the symbols currently defined.

Three types of symbols are supported: global, local, and user. Global symbols reference addresses anywhere in memory using an absolute reference. Local symbols also use absolute addressing but are grouped within a "module." User symbols are defined at the command line. Global and local symbols cannot be defined at the command line.

The definition of a module for grouping local symbols depends on the environment being used. For local symbols created by a high-level language, a module might be a function, a procedure, or a separately compilable source file. When you define local symbols through the use of a symbol file, a module, in effect, becomes a technique to manage the symbols. It can be a mnemonic device to refer to modules, or it can be a simple way to group local symbols into a set for display and deletion purposes since the **sym** command facilitates manipulation of local symbols by their module name.

Symbols are used like equated variables. When using symbols in expressions, only the **+** and **-** operators can be used immediately before and after the symbol name. The expression can contain literals and equated (**equ**) labels, but not other symbols.

When using symbols, if a symbol and an equated value have the same name, the equated value will be used.

The symbol table can be updated in three ways:

- You can enter user symbols at the command line.
- You can update it from an external "symbol file" using the **load -So** command.

- You can load an absolute file (such as an IEEE-695 file) which can contain symbols as well as program code.

A "symbol file" is a text file containing user-specified symbols.

The parameters are as follows:

<name> — This represents the symbol label to be defined or referenced. The format of the symbol name reference is determined by the type of symbol, where:

name	Is a user symbol or module name.
:name	Is a global symbol name.
name:	Is a local module name.
module:name	Is a symbol name in a local module.

In addition, symbols can be referenced using a "wild card" expression when displaying and deleting names. Only one wildcard character can appear in a symbol name. An asterisk ("*") character is used to represent zero or more characters at the end of a symbol name. A wildcard can be used in any of the following symbol types:

name*	Represents a user symbol name followed by zero or more of any character or characters.
:name*	Represents a global symbol name followed by zero or more of any character or characters.
module:name*	Represents a local module:symbol followed by zero or more of any character or characters.

<addr> — Specifies the value to assign to a user symbol.

-d — Deletes all symbols.

-du — Deletes user symbols. If a <name> parameter is not included, all user symbols are deleted. If a <name> parameter is included, only user symbols matching the entered name are deleted.

-dg — Deletes all global symbols. No option exists to delete one global symbol.

-dl Deletes local symbols in a module. If a <name> parameter is not included, all local symbols are deleted for all modules. If a <name> parameter is included to specify a module name, only local symbols in the module matching the entered name are deleted.

-g Specifies the display of global symbols. If a <name> parameter is not included, all global symbols are displayed. If a <name> parameter is included, only global symbols matching the entered name are displayed.

-l This option allows you to display local modules and symbols. If a <name> parameter is not included, all local modules are displayed. If a <name> parameter is included, only local symbols matching the symbol name or module are displayed.

-u This option allows you to display user symbols. If a <name> parameter is not included, all user symbols are displayed. If a <name> parameter is included, only user symbols matching the entered name are displayed.

See Also

equ (used to equate names to expressions)

load (used to load a program file with symbols, or a symbol text file)

t, xt - start a trace

```
t - start an emulation trace

xt - start an external trace
```

The **t** and **xt** commands start emulation and external traces, respectively. These commands (or **tx** if making a synchronous CMB execution) must be entered to actually begin a measurement; most other trace commands are used only for specification of triggering, sequencer, and storage parameters; or to display trace results or status.

If the external analyzer has been linked to the emulation analyzer via the **xtmo** command, the **xt** command is invalid and both analyzers begin a trace when the **t** command is entered.

See Also

r (starts a user program run; normally will be specified after entering the **t** command)

th (halts a trace in process)

ts (allows you to determine the current status of the emulation analyzer)

tx (specifies whether a trace is to begin upon start of CMB execution)

x (begins synchronous CMB execution)

xtmo (specifies whether or not the external analyzer bits are to be treated as a separate analyzer or integrated with the emulation analyzer. If associated with the emulation analyzer, the **xt** command is invalid; the **t** command starts the trace on both analyzers.)

ta - current status of analyzer signals is displayed

```
ta
```

The **ta** command allows you to display the activity on each of the analyzer input lines. Each signal may be low (0), high (1), or moving (?).

Each pod (group of 16 lines) is displayed on a single line with bit 0 (LSB) at the far right and bit 15 (MSB) on the far left.

See Also

xtv (used to set the threshold voltages for the optional external analyzer inputs; incorrect specification may show up as lack of activity in a **ta** display)

tarm, xtarm - specify the arm condition

```
tarm  <signal>    - arm the emulation analyzer

xtarm  <signal>    - arm the external analyzer
```

The **tarm** (**xtarm**) command allows you to specify an arming condition for the emulation and external analyzers. You can specify the arm condition as the assertion of the trig1 or trig2 signals or as **tarm always**.

The trig1 or trig2 signals can be asserted from many sources including the analyzer itself or the rear panel BNC connector or the CMB. See **bnct**, **cmbt**, and **tgout** for examples.

The arm condition may be used in specifying the analyzer trigger or in specifying branch conditions for the sequencer, as well as count or prestore qualifiers.

If the analyzers are connected through use of the **xtmo** command, then the **xtarm** command is invalid. In this case, the **tarm** command will set the arming condition for the analyzer combination.

If no parameters are supplied, the current **tarm** condition is displayed. The default setting after powerup or **tinit** is **tarm always**.

The parameters are as follows:

=trig1 — The assertion of the trig1 signal will arm the analyzer.

=trig2 — The assertion of the trig2 signal will arm the analyzer.

!=trig1 — If the trig1 signal is asserted when the analyzer is started, the analyzer can never be armed.

If the trig1 signal is not asserted when the analyzer is started, the analyzer is armed immediately.

!=trig2 — If the trig2 signal is asserted when the analyzer is started, the analyzer can never be armed.

If the trig2 signal is not asserted when the analyzer is started, the analyzer is armed immediately.

always — The analyzer is always armed.

Note that if the external analyzer is configured to operate as a timing analyzer (**xtmo -t**) then the **!=** operator is invalid when used in the **xtarm** command as given to the external analyzer. Only the **=** operator will be recognized.

See Also

bc (can be used to cause the emulator to break to monitor execution upon receipt of the trig1 and/or trig2 signals)

bnct (used to define connections between the internal trig1 and trig2 signals and the rear panel BNC connector)

cmbt (used to define connections between the internal trig1 and trig2 signals and the CMB trigger signal)

tgout (defines whether or not the trig1 or trig2 signals are driven when the analyzer finds the trigger state)

tcf, xtcf - set or display trace configuration

```
tcf             - display trace configuration
tcf -c          - set complex trace configuration
tcf -e          - set easy trace configuration

xtcf             - display trace configuration
xtcf -c          - set complex trace configuration
xtcf -e          - set easy trace configuration
```

The **tcf** (**xtcf**) commands are used to set the configuration for the emulation (external) analyzer. There are two possible configurations for the analyzer, an easy configuration (**tcf -e**) and a complex configuration (**tcf -c**).

The easy configuration hides some of the complexity of the analyzer sequencer and makes it easy to use. The complex configuration gives you greater capability when using the sequencer and gives you greater flexibility when using expressions to qualify states.

In the easy configuration, you can insert up to five sequence terms in the sequencer. The branch out of the last sequence term constitutes the trigger.

In the complex configuration, there are always eight terms in the sequencer. Any of the sequence terms except the first may be specified as the *trigger term*. Entry into the trigger term constitutes the trigger.

In the complex configuration, up to eight pattern resources and one range resource may be used in trace commands wherever state qualifier expressions are used in the easy configuration. These patterns are grouped in to sets and may be combined with set operators to specify more complex qualifiers.

If no parameters are supplied, the current analyzer configuration is displayed. After powerup or **tinit**, the default analyzer configuration is **tcf -e**.

The parameters are as follows:

-e — Sets the analyzer to the easy configuration.

-c — Sets the analyzer to the complex configuration.

See Also

tarm (used to set the analyzer arm specification; this specification can only be used in analyzer expressions in complex configuration)

telif (sets the global restart in easy configuration, secondary branch condition in complex configuration)

tg (used to set a trigger expression in either analyzer configuration)

tif (sets primary branch specification in either analyzer configuration)

tpat (used to label complex analyzer expressions with a pattern name; the pattern name is then used by the analyzer setup commands. Only valid in complex configuration)

tpq (specifies trace prestore qualifier in either analyzer configuration)

trng (defines a range of values to be used in complex analyzer expressions)

tsto (specifies a qualifier to be used when storing analyzer states)

tsq (used to modify the trace sequencer's number of terms and trigger term)

xtmo (used to append or disconnect the external analyzer to/from the emulation analyzer)

tck, xtck - set or display clock specification for the analyzer

```
tck -r <clock>     - clock analyzer on rising edge of clock
tck -f <clock>     - clock analyzer on falling edge of clock
tck -x <clock>     - clock analyzer on either edge of clock
tck -l <clock>     - qualify on low level of clock
tck -h <clock>     - qualify on high level of clock
tck -b             - qualify when emulation in background
tck -u             - qualify when emulation in user
tck -s <speed>     - define the clock speed

xtck -r <clock>     - clock analyzer on rising edge of clock
xtck -f <clock>     - clock analyzer on falling edge of clock
xtck -x <clock>     - clock analyzer on either edge of clock
xtck -l <clock>     - qualify on low level of clock
xtck -h <clock>     - qualify on high level of clock
xtck -b             - qualify when emulation in background
xtck -u             - qualify when emulation in user
xtck -s <speed>     - define the clock speed
```

The **tck** (**xtck**) command allows specification of clock qualifiers and master edges of the master clocks used for the emulation and external analyzers.

The **tck** command is included with the system for the purpose of internal system initialization and system control through high-level software interfaces.

You should ONLY use the **tck** command when you wish to trace background execution or perhaps to qualify the emulation analyzer clock on some external signal. In other words, **do not change the the "tck -r L" setting**.

If you are using the external analyzer as an independent state analyzer, you will use the **xtck** command to specify and qualify the clock signal for the external analyzer.

The parameters are as follows:

<clock>

Five clock signals are defined: J, K, L, M, and N.

The L, M, and N clocks are generated by the emulator. The L clock is the emulation clock derived by the emulator, the N clock is used as a qualifier to provide the user/background tracing options (**-u** and **-b**) to **tck**, and the M clock is not used. The L and N clocks may also be used to clock and qualify the external analyzer as well as the emulation analyzer.

The J and K clocks are the clock inputs on the external trace probe (if one is present). These clock signals should only be used to clock the external trace; they should not be used to clock the emulation trace although it may occasionally be useful to use the external clock signals as qualifiers for the emulation trace.

r	The analyzer is clocked on the rising edge of the indicated clock signal.
f	The analyzer is clocked on the falling edge of the indicated clock signal.
x	The analyzer is clocked on both the rising and falling edges of the indicated clock signal.
l	Qualifies the analyzer clock so that the analyzer is only clocked when this clock signal is low (less positive/more negative voltage).
h	Qualifies the analyzer clock so that the analyzer is only clocked when this clock signal is high (more positive/less negative voltage).
-b	The analyzer is only clocked when the emulator is executing in background (in other words, the background monitor).
-u	The analyzer is only clocked when the emulator is executing in foreground (in other words, the user program). This is the default.
-s <speed>	Specifies the maximum qualified clock speed. The <speed> parameter can be:

S	SLOW, less than or equal to 16 MHz.
F	FAST, between 16 MHz and 20 MHz.
VF	VERY FAST, between 20 MHz and 25 MHz.

Changing the clock speed affects the **tcq** command parameters. When speed is set to **S** (slow), the **tcq** command may either count states or time. When speed is set to **F** (fast), the **tcq** command may be used to count states but not time. If clock speed is set to **VF** (very fast), **tcq** cannot count either state or time and should be set to **tcq none.**

If no parameters are specified, the current clock definitions are displayed. After powerup or **tinit**, the **-u** option is always set.

When several clock edges are specified with the **-r**, **-f**, or **-x** options, any one of the edges clocks the given trace. If several qualifiers are specified with the **-l** or **-h** options, they are ORed so that the trace is clocked when any of the qualifiers are met.

Note that the **-u** and **-b** qualifiers are ORed with all of the other qualifiers specified.

See Also

ta (display current trace signal activity. This can be useful after you have modified the clocks for the external analyzer; you can issue a **ta** command and verify that you are seeing activity on the signals of interest.)

tcq (specifies the trace count qualifier as states, time, or none. The maximum qualified clock speed set by **tck -s** affect which **tcq** parameters are valid.)

tsck (used to define slave clock signals used by the analyzer; **tck** defines the master clock signals. Default mode for **tsck** is off on all pods.)

xtv (specifies threshold voltages for external analyzer input lines; must be set correctly to ensure that the **J** and **K** clock signals are recognized)

xtmo (specifies mode of operation for the external analyzer; that is, whether it acts as an independent analyzer or is appended to the emulation analyzer)

tcq, xtcq - set or display the count qualifier specification

```
tcq             - display count qualifier specification
tcq time        - define count on time
tcq <expr>      - define count on state

xtcq             - display count qualifier specification
xtcq time        - define count on time
xtcq <expr>      - define count on state
```

The **tcq** (**xtcq**) command allows you to specify a qualifier for the emulation (external) trace tag counter.

When the tag counter is active, the analyzer counts occurrences of the expression you specify (which may include simple or complex expressions (depending on analyzer configuration), **time**, or **none**). Each time a trace state is stored, the value of the counter is also stored and the counter is reset. The tag counter shares trace memory with stored states, so only half as many states can be captured by the analyzer when the tag counter is active. (The analyzer can store 1024 states with **tcq none**, 512 states otherwise.)

If no parameters are given, the current count qualifier is displayed. Upon powerup or after **tinit** initialization, the clock qualifier defaults to the state **tcq time**.

The parameters are as follows:

time

If you specify **time** rather than an analyzer expression, the trace tag counter measures the amount of time between stored states.

Note that the **tcq time** qualifier is only available when the analyzer clock speed is set to the slow (**S**) speed setting (default). If the clock speed is set to very fast (**VF**), then trace tag counting must be turned off by specifying **tcq none**. Refer to the **tck** command (analyzer clock specification) for further information.

<expr>

State qualifier expression. Refer to the <expr> description in this chapter.

Note that the count qualifier **tcq arm** is not permitted in any configuration.

See Also

tck (used to specify the clock source and clock parameters for the analyzer)

tp (specifies position of the trigger within the trace; note that **tcq** affects the number of states the analyzer can store and therefore may affect trigger positioning)

tpat (assigns analyzer expressions to pattern names in complex configuration; the pattern names are then used to specify qualifiers in other analyzer commands such as **tcq**)

trng (specifies a range of values to be used as a complex mode qualifier; this range definition can be used as a count qualifier by **tcq**)

tsq (used to manipulate the trace sequencer)

xtmo (used to choose the external analyzer mode; the external analyzer can operate as an independent state or timing analyzer, or it may be appended to the emulation analyzer. If appended, the **xtcq** command has no effect and the **tcq** command specifies the count qualifier for both analyzers.)

telif, xtelif - set or display secondary branch specification

In the easy configuration:

```
telif              - display global restart specification
telif <expr>       - define global restart specification

xtelif              - display global restart specification
xtelif <expr>       - define global restart specification
```

In the complex configuration:

```
telif            - display all secondary branch specifications
telif X          - display secondary branch specification X
telif X <expr>   - define secondary branch specification X
telif X <expr> Y - secondary branch X will jump to Y (default next
                   term)

xtelif            - display all secondary branch specifications
xtelif X          - display secondary branch specification X
xtelif X <expr>   - define secondary branch specification X
xtelif X <expr> Y - secondary branch X will jump to Y (default next
                    term)
```

The **telif** (**xtelif**) command allows you to set the global restart qualifier (in easy configuration) for the emulation (external) analyzer sequencer. In complex configuration, **telif** (**xtelif**) lets you set the secondary branch qualifier for each term of the emulation (external) analyzer sequencer.

Note that the **telif** command is used as a global restart qualifier in easy configuration and a secondary branch qualifier in complex configuration. The hierarchy of the **tif** and **telif** commands is such that either branch will be taken if found before the other; however, if both branches are found simultaneously, the **tif** branch is always taken over the **telif** branch.

When in easy configuration, the sequencer will restart by jumping to sequencer term number one (1) when the expression specified by **telif** occurs.

When in complex configuration, the sequencer will branch to the sequencer level specified by the Y parameter when the expression specified is found. There are always eight sequencer terms available. Position of the trigger term is defined with the **tsq** command. If both the **tif** and **telif** expressions are satisfied simultaneously, the **tif** branch is taken; otherwise, branching occurs according to which expression is first satisfied.

The parameters are as follows:

<expr> State qualifier expression. Refer to the <expr> description in this chapter.

X Specifies a sequence term number to associate with the given <expr>. When you associate a term number with a complex expression, that expression is only used as a secondary branch qualifier at the sequencer level specified by the term number. If you specify X without an expression, the secondary branch qualifier currently associated with that term number is displayed.

Y Specifies the branch destination when <expr> is found. For example, if you wish to have the sequencer branch from term 1 to term 3 after the expression is found you would be specified as **telif 1 <expr> 3**. If you do not specify a term number, the default is to increment the sequencer level (**telif X <expr> (X+1)**).

If **telif** is entered with no parameters, the global restart qualifier or secondary branch qualifiers (depending on analyzer configuration) for all sequencer levels are displayed. If **telif** is entered with only an X parameter in complex configuration, the secondary branch qualifier for only that term number is displayed.

Upon initialization via a powerup sequence or the **tinit** command, the secondary branch specifiers are set to **telif never**.

Note that the default branch to condition for sequence term 8 is 8; that is, branch to the same term.

Note that if the **tif** expression for the given sequence term X has a <count> parameter other than one (1), the counter is reset to zero (0) if the **telif** branch is taken before the occurrence counter parameter is satisfied. For example, if the **tif** counter parameter is 7, and the **tif** expression has been found 5 times, then the **telif** expression is satisfied, the **telif** branch will be taken and the **tif** counter will be reset from 5 to 0. This might cause you difficulty if you happen to have **telif** branching back to the same term; your occurrence condition may or may not be satisfied.

See Also

tarm (allows you to specify that the **trig1** or **trig2** signal will arm the analyzer. This arm condition can then be used as part of the secondary branch qualifier)

tcf (used to select whether the analyzer is operated in easy configuration or complex configuration)

tif (used to specify a primary branch specification for the analyzer)

tg (used to set up a simple trigger qualifier in either analyzer configuration. Specifying the **tg** command overrides the current sequencer specification and will modify the existing **telif** qualifier stored in sequence term number 1)

tpat (used to assign pattern names to simple expressions for use in specifying complex expressions. These complex expressions are used to specify **telif** qualifiers in analyzer complex configuration)

trng (used to set up an expression which assigns a range of values to a range variable. This range information may be used in specifying complex **telif** qualifiers)

tsto (specifies a global trace storage qualifier in both easy & complex configurations; also specifies a trace storage qualifier for each sequencer term in complex configuration. Used to control the types of information stored by the analyzer)

tsq (used to manipulate the trace sequencer)

xtmo (specifies whether the external analyzer operates as an independent state or timing analyzer or is appended to the emulation analyzer. If appended to the emulation analyzer, the **xtelif** command is invalid; all secondary branch qualifiers are specified with the **telif** command)

tf, xtf - specify trace display format

```
tf                          - display current format
tf <label>,<base>           - display the label in the specified base
tf mne                      - disassembled mnemonic
tf count                    - count, absolute (relative to trigger)
tf count,a                  - count, absolute (relative to trigger)
tf count,r                  - count, relative to preceding state
tf seq                      - sequencer state change
tf mne <label>,<base> count count,r seq
                            - multiple fields may be specified
tf addr,H mne count,r seq   - default format
tf addr,<base>,<width>      - column width (addr column only)

xtf                         - display current format
xtf <label>,<base>          - display the label in the specified base
xtf count                   - count, absolute (relative to trigger)
xtf count,a                 - count, absolute (relative to trigger)
xtf count,r                 - count, relative to preceding state
xtf seq                     - sequencer state change
xtf <label>,<base> <label>,<base> count seq
                            - multiple fields may be specified
xtf xbits count,r seq       - default format
```

The **tf** (**xtf**) command allows you to specify which pieces of information from the emulation (external) analyzer trace will be displayed by **tl** (**xtl**) (trace list) commands. Each label represents a column in the trace list display.

The parameters are as follows:

<label> — A label defined via the **tlb** or **xtlb** commands. The analyzer bits associated with that label will be displayed in a column of the trace listing.

<base> — Specifies the numeric base in which <label> is to be displayed. The choices are **Y** (binary), **Q** or **O** (octal), **T** (decimal), **H** (hexadecimal), or **A** (ASCII). The specifiers are not case sensitive. In ASCII mode, non printing characters are displayed as periods (.). If <base> is not specified, the default base is hexadecimal.

mne — Displays mnemonic information about a state. Depending on the trace list disassembler options (see **tl -o**), disassembled instructions may be displayed in this column. To ensure correct operation of **mne**, the predefined labels **addr**, **data**, and **stat** must not be redefined. The **mne** column is only allowed with the **tf** command, and not with **xtf**.

count
count,a — Absolute time counts are displayed. That is, the the time shown for each state is relative to the trigger state.

count,r — Relative time counts are displayed. That is, the time count shown for each state is relative to the previous state.

seq — If you specify **seq**, a "+" is displayed in the seq column for each state that causes the sequencer to branch from one term to another.

<width> — This option allows you to set the width of the column for the "addr" predefined trace label to values from 4 to 50. A wider address field is useful when displaying symbols in the trace list.

The minimum width is really defined by the base of the "addr" column. For example, if the 24-bit address is displayed in binary, the minimum width is 24.

Note that changing the trace format DOES NOT change the type of information captured by the analyzer; it only specifies how the captured data should be displayed.

See Also

tl,xtl (displays the current data in emulation (external) trace memory according to the specifications set up by **tf**)

tlb,xtlb (define labels which represent groups of emulation (external) analyzer input lines; these labels may be used to create special trace list displays by including the labels in the **tf** definition)

xtmo (defines whether the external analyzer acts as an independent state/timing analyzer or is appended to the emulation analyzer)

tg, xtg - set and display trigger condition

```
tg                   - display sequence term 1 primary branch
tg <expr>            - define trigger
tg <expr> <count>    - define trigger and occurrence count

xtg                   - display sequence term 1 primary branch
xtg <expr>            - define trigger
xtg <expr> <count>    - define trigger and occurrence count
```

The **tg** (**xtg**) command sets a trigger condition for the emulation (external) analyzer. When the expression specified occurs the number of times specified, the analyzer triggers.

The parameters are as follows:

<expr> State qualifier expression. Refer to the <expr> description in this chapter.

<count> Specifies the number of times the expression must occur before the trigger condition is satisfied. The <count> value specified must be from 1 to 65535. The default number base for <count> is decimal. If <count> is not specified, the occurrence count is 1.

If no parameters are specified, the current primary branch condition for sequencer term 1 is displayed. Note that this is not necessarily the trigger condition if other sequence terms are used. After powerup or **tinit** initialization, **tg** is set to **tg any**.

The **tg** command modifies the current analyzer sequence specification. The manner in which the sequencer is modified is dependent upon the analyzer configuration.

If the analyzer is in easy configuration (**tcf -e**), the sequencer is reduced to a one term sequence triggering upon exit from term 1. The global restart qualifier is set to never (**telif never**); the primary branch condition is set to the specified trigger expression (**tif 1 <expr> <count>**).

If the analyzer is in complex configuration (**tcf -c**), the sequencer is modified to trigger upon entrance to the second sequence term (**tsq -t 2**), the secondary branch qualifier is set to never (**telif 2 never**), and the primary branch qualifier for term number 1 is set to the specified expression (**tif 1 <expr> 2 <count>**).

The analyzer storage qualifier (**tsto**) is not affected in either configuration; therefore, the analyzer uses the storage qualifier from the most recent **tsto** command.

See Also

bc (allows you to break the emulator to the monitor when various conditions occur; you can have the emulator break upon analyzer trigger by specifying **tgout trig1** and **bc -e trig1** (or you could use the trig2 signal to perform the same function))

t (starts an emulation trace)

tarm (used to specify an analyzer arm condition; the analyzer will not trigger until the arm condition is received if you specify **tg arm**)

tcf (used to specify whether the analyzer is operated in easy or complex configuration)

tpat (used to assign pattern names to simple analyzer expressions; the pattern names are then used in creating complex analyzer expressions which could be used with the **tg** command to trigger the analyzer)

trng (used to specify a range of values for a particular group of analyzer lines; this range may be used in specifying complex analyzer expressions for triggering the analyzer)

tsto (specifies which states encountered by the analyzer should be stored in trace memory)

tsq (used to manipulate the trace sequencer. Note that the sequencer's current status is affected by the **tg** command.)

xtmo (specifies whether the external analyzer is treated as a separate state or timing analyzer or is appended to the emulation analyzer. If appended to the emulation analyzer, the **xtg** command is no longer valid; **tg** sets the trigger condition for both analyzers.)

tgout, xtgout - specify signals to be driven by the analyzer

```
tgout  <signal>    - find trigger then drive signal

xtgout  <signal>    - find trigger then drive signal
```

The **tgout** (**xtgout**) command allows you to specify which of the internal trig1 and/or trig2 signals will be driven when the emulation analyzer (external analyzer) finds its trigger condition.

The parameters are as follows:

<signal> Specifies the internal signal to drive when the trigger is found. This signal can be:

trig1	The trig1 signal is driven by the analyzer when the trigger state is found.
trig2	The trig2 signal is driven by the analyzer when the trigger state is found.
trig1,trig2	Both trig1 and trig2 should are driven when the analyzer trigger is found.
none	Neither the trig1 nor trig2 signals are driven when the analyzer finds its trigger state.

If no parameters are specified, the current state of **tgout** is displayed. Upon powerup or **tinit**, the default state is **tgout none**.

Note that if the analyzer is receiving trig1 or trig 2 via the **tarm** command, then that signal cannot be driven, although no error message will be issued to that effect.

If the external analyzer has been appended to the emulation analyzer via the **xtmo** command, then the **xtgout** command is invalid and the **tgout** command specifies the trigger signals to be driven when either analyzer finds its trigger.

See Also **bc** (allows you to specify a break to emulation monitor when the tgout condition is satisfied)

bnct (specifies whether or not trig1 and trig2 are used to drive and/or receive the rear panel BNC connector signal line)

cmbt (specifies whether or not trig1 and trig2 are used to drive and/or receive the CMB trigger signal)

tarm (used to specify that the analyzer will be armed upon assertion or negation of trig1 or trig2)

th, xth - halt the trace

```
th          - halt the emulation trace
th -w       - suppress output and errors

xth         - halt the external trace
xth -w      - suppress output and errors
```

The **th** (**xth**) command stops an emulation (external) trace.

The parameters are as follows:

-w

Suppresses the output and errors. In other words, "Emulation trace halted" is not shown.

If the external analyzer has been appended to the emulation analyzer with the **xtmo** command, the **xth** command is invalid and **th** halts both the emulation and external trace in process.

The analyzer will stop driving the **trig1** and **trig2** signals when the trace is halted. This may cause you difficulty in making measurements with instruments connected to the BNC. For example, if you set the analyzer to drive **trig1** (**tgout trig1**) when the trigger condition is found, then drive this to the BNC connector with **bnct -d trig1**, the BNC signal will be driven high when the analyzer finds its trigger while a trace is in progress; it will fall low when the trace finishes.

You should start the trace after you have begun the external instrument’s measurement. Otherwise, the following measurement errors may occur, depending on the type of external instrument you are using:

- With an edge sensitive instrument, starting the instrument after the analyzer trigger is found will mean that the instrument never sees the transition of the **trig1** line and therefore never triggers.
- With a level sensitive instrument, starting the instrument after the analyzer trigger is found will mean that the instrument triggers immediately; although many states of interest have probably already passed.

Note that if the analyzer trigger specification has not been found, you will need to use the **th** command to halt the analyzer before you can display the trace list.

See Also

t (used to start an analyzer trace)

ts (allows you to determine the current status of the emulation analyzer)

tx (starts an analyzer trace upon receipt of the CMB execute signal)

x (starts a synchronous CMB execution)

tif, xtif - set or display primary sequence branch specifications

In the easy configuration:

```
tif                    - display all primary branch specifications
tif X                  - display primary branch X specification
tif X <expr>           - define primary sequence branch X
tif X <expr> <count>   - branch X jump to next term after count times

xtif                    - display all primary branch specifications
xtif X                  - display primary branch X specification
xtif X <expr>           - define primary sequence branch X
xtif X <expr> <count>   - branch X jump to next term after count times
```

In the complex configuration:

```
tif                    - display all primary branch specifications
tif X                  - display primary branch X specification
tif X <expr>           - define primary sequence branch X
tif X <expr> Y         - define primary sequence branch X jump to Y
tif X <expr> Y <count> - define branch X jump to Y after count times

xtif                    - display all primary branch specifications
xtif X                  - display primary branch X specification
xtif X <expr>           - define primary sequence branch X
xtif X <expr> Y         - define primary sequence branch X jump to Y
xtif X <expr> Y <count> - define branch X jump to Y after count times
```

The **tif** (**xtif**) command allows you to set the primary branch qualifier for each term of the emulation (external) analyzer sequencer.

The parameters are as follows:

X — Specifies the sequence term whose primary branch qualifier is to be modified with the <expr> state qualifier. If you specify X without an expression, the **tif** qualifier for that term number is displayed.

<expr> — State qualifier expression. Refer to the <expr> description in this chapter.

<count> — Specifies the number of times the expression must occur before the trigger condition is satisfied. The <count> value specified must be from 1 to 65535. The default number base for <count> is decimal. If <count> is not specified, the occurrence count is 1.

Note that, in the complex configuration, if you specify the <count> parameter, you must also specify a Y parameter. If you omit the Y parameter when specifying <count>, the system will interpret the count as "branch to term" information; if greater than eight (8), an error will be returned; otherwise, you will have just specified an incorrect branch.

Y — Specifies the branch destination when the state qualifier is found. For example, if you wish to have the sequencer branch from term 1 to term 3 after the expression is found, this would be specified as **tif 1 <expr> 3**. If you do not specify a term number, the default is to increment the sequencer level (**tif X <expr> (X+1)**).

If **tif** is entered with no parameters, the primary branch qualifiers for all sequencer levels are displayed. If **tif** is entered with only an X parameter, the primary branch qualifier for only that term number is displayed.

Upon initialization via a powerup sequence or the **tinit** command, the primary branch specifiers are set to **tif X any (X+1)**.

Note that the **telif** command is used as a global restart qualifier in easy configuration and a secondary branch qualifier in complex configuration. The hierarchy of the **tif** and **telif** commands is such that either branch will be taken if found before the other; however, if both branches are found simultaneously, the **tif** branch is always taken over the **telif** branch.

When in easy configuration, the sequencer will increment to the next sequencer level when the expression specified by **tif** occurs the number of times specified by the <count> parameter. There is a maximum of five sequence levels; only one is available at initialization. If you require more sequencer levels, you must insert them with the **tsq** command. (The term you are specifying a primary branch for with the **tif** command must be present in the sequence.) The branch out of the last sequencer term constitutes the trigger.

When in complex configuration, the sequencer will branch to the sequencer level specified by the Y parameter when the expression specified occurs the number of times indicated in the <count> parameter. There are always eight sequencer terms available. Position of the trigger term is defined with the **tsq** command.

Note that, in the complex configuration, at sequencer term number 8, the default branch to condition is also term 8; that is, branch to the same term.

See Also

tarm (allows you to specify that the **trig1** or **trig2** signal will arm the analyzer. This arm condition can then be used as part of the primary branch qualifier)

tcf (used to select whether the analyzer is operated in easy configuration or complex configuration)

telif (used to specify a secondary branch specification for the analyzer)

tg (used to set up a simple trigger qualifier in either analyzer mode. Specifying the **tg** command overrides the current sequencer specification and will modify the existing **tif** qualifier stored in sequence term number 1)

tpat (used to assign pattern names to simple expressions for use in specifying complex expressions. These complex expressions are used to specify **tif** qualifiers in analyzer complex configuration)

trng (used to set up an expression which assigns a range of values to a range variable. This range information may be used in specifying complex **tif** qualifiers)

tsto (specifies a global trace storage qualifier in both easy and complex configurations; also specifies a trace storage qualifier for each sequencer term in complex configuration. Used to control the types of information stored by the analyzer)

tsq (used to manipulate the trace sequencer)

xtmo (specifies whether the external analyzer operates as an independent state or timing analyzer or is appended to the emulation analyzer. If appended to the emulation analyzer, the **xtif** command is invalid; all primary branch qualifiers are specified with the **tif** command)

tinit - initialize emulation and external analyzers to powerup defaults

```
tinit
```

The **tinit** command restores all trace specification items to their powerup default values which are as follows:

Trace Specification	**Default Value**
Analyzer arm	tarm always
Trace Configuration	tcf -e
Analyzer master clocks	tck -r L -u -s VF
Trace format	tf addr,H mne xbits,H count,R seq
Trace trigger	tg any tgout none
Analyzer signal line labels	#### Emulation trace labels tlb addr 0..23 tlb data 32..47 tlb stat 24..31
Trigger Position	tp s
Trace Prestore Qualifier	tpq none
Trace sequencer (includes branch and store conditions)	tif 1 any tsto all telif never
Trace slave clocks	tsck -o 1 tsck -o 2 tsck -o 3 tsck -o 4 tsck -o 5 tsck -o 6 tsck -o 7
Trace Upon Execute?	tx -d # ignore the execute signal

See Also

init (used to initialize selected portions of the emulator or the entire emulator, dependent on the options given)

tl, xtl - display trace list

```
tl                 - display next states
tl *               - display all states
tl -d -15..3       - display states -15 through 3, disassembled
tl -s 20..30       - display symbols only in addr column
tl -a 50..60       - display absolute addresses only in addr column
tl -e 12..25       - display symbols and absolutes in addr column
tl -h              - display next states, no header
tl -n 3            - display next 3 states
tl -t 123          - display top 123 states
tl 10..20          - display states 10 through 20
tl -b              - upload binary trace list
tl -h -n -d 15     - some options may be combined

xtl                 - display next states
xtl *               - display all states
xtl -h              - display next states, no header
xtl -n 3            - display next 3 states
xtl -t 123          - display top 123 states
xtl 10..20          - display states 10 through 20
xtl -b              - upload binary trace list
xtl -h -n -d 15     - some options may be combined
```

The **tl** (**xtl**) command allows you to display the current emulation (external) analyzer trace list information.

If the trigger specification has not yet been satisfied, the trace list cannot be displayed until the trace in progress is halted with the **th** command. Entering the **tl** command before the trace is halted results in the message "**** Trigger not in memory ****."

If the analyzer was halted before any states were captured, the message "**** No trace data ****" is displayed upon entry of the **tl** command.

The parameters are as follows:

-d Disassemble instructions in the trace.

-s Display symbols in the address column.

-a Display absolute addresses in the address column. This is the default.

-e Display symbols and absolute addresses in the address column.

-h Suppresses the display of column headers.

-n Display the next number of states of the trace. If you do not specify a number, the same number of states will be displayed as the last time you used **tl** to display part (but not all) of the trace.

-t Displays the top number of states of the trace. If you do not specify a number, the number of states displayed is the same number as the last time **tl** was invoked to display part (but not all) of the trace.

-b Dumps the trace list in binary format using the HP 64000 **transfer** protocol.

Note that the **-h** and **-d** options cannot be used with the **-b** option.

If no parameters are given, the trace list is displayed starting with the first state that has not yet been displayed. The number of states displayed is identical to the number of states displayed by the last **tl** command. For example, if the last trace list display was **tl -t 5**, then the next **tl** command will start the display at state 6 and display a total of five states.

Note that the HP 64700 remembers the last option specified for the address field (**-s**, **-a**, or **-e**), and uses it for the next **tl** command if no other option is specified.

See Also

t (starts an analyzer trace)

tf (specifies the display format for the trace)

th (halts a trace in process)

tlb (defines analyzer signal line labels; these may be used by **tf** in specifying the trace list display format)

ts (allows you to determine the current status of the emulation analyzer)

tlb, xtlb - define and display trace labels

```
tlb addr 0..15         - define addr, positive polarity, bits
                         0 through 15
tlb data 16..23        - define data, positive polarity, bits
                         16 through 23
tlb stat 24..31        - define stat, bits 24 through 31
tlb stat 31..24        - define stat, bits 24 through 31;
                         31..24 is the same bit range as 24..31
tlb -n LRESET 25       - define LRESET, negative polarity, bit 25
tlb -d LRESET          - delete named label
tlb -d *               - delete all labels
tlb addr               - display named label
tlb                    - display all labels

xtlb MYLABEL 8..12      - define MYLABEL, positive polarity,
                          external analyzer bits 8 through 12
xtlb myStatus 3         - define myStatus, positive polarity,
                          external analyzer bit 3
xtlb -n AuxBits 2..11   - define AuxBits, negative polarity,
                          external analyzer bits 2 through 11
xtlb -n AuxBits 11..2   - define AuxBits, negative polarity,
                          external analyzer bits 2 through 11;
                          11..2 is the same bit range as 2..11
xtlb -n LRESET16 7      - define LRESET16, negative polarity,
                          external analyzer bit 7
xtlb -d myStatus        - delete named external analyzer label
xtlb -d *               - delete all external analyzer labels
xtlb LRESET16           - display named external analyzer label
xtlb                    - display all external analyzer labels
```

The **tlb** (**xtlb**) command allows you to define new labels for emulation (external) analyzer lines, as well as display or delete previously defined analyzer labels. Since labels are pre-defined for the address, data, and status lines of the emulation analyzer, **xtlb** will be the more frequently used command.

The parameters are as follows:

-d

Delete the named label. If the label is currently used in a trace specification or in the trace display format (tf command), it will not be deleted until removed from all of the specifications. If * is used, all labels are deleted.

-n

Defines the named label with negative polarity. That is, after label definition, bits that are a one (1) refer to a signal lower than the threshold voltage and bits that are a zero (0) refer to a signal higher than the threshold voltage. If **-n** is not specified, the named label defaults to positive polarity.

If no parameters are specified, the current label definitions are displayed. Upon emulator powerup, or after a **tinit** command, the following labels are defined:

```
M>tlb
  #### Emulation trace labels
  tlb addr 0..23
  tlb data 32..47
  tlb stat 24..31

M>xtlb
  #### External trace labels
  xtlb xbits 0..15
```

Note that the predefined emulation trace labels are special labels, used for trace list disassembly. They should not be changed or deleted.

The external analyzer has 16 lines that may be assigned to labels, numbered 0 through 15, where 0 is the least significant bit. The emulation analyzer has 108 lines, where 0 is the least significant bit.

In emulation analyzer labels, no more than 32 signal lines may be assigned to a given label. Also, an emulation analyzer label may not cross more than a multiple of 16 boundary. For example, a label cannot be defined for emulation analyzer lines 15..32 since one multiple of 16 boundary is crossed from 15 to 16 and another boundary is crossed from 31 to 32.

Labels are made up of alphanumeric characters; upper and lower case are distinguished. Labels can be up to 31 characters in length.

Labels can be made to overlap; for example, you may wish to define a label for a particular status line or data bit so that you can easily track its state in the trace list.

The number of labels that can be defined is limited only by system memory.

See Also

tf (used to specify the trace list format; **tlb** <LABEL> definitions can be specified as output columns in the trace listing through the **tf** command)

tpat (trace pattern definition; labels defined in **tlb** can be used in pattern definitions)

trng (trace range, used to specify a range of valid values to be used in a trace specification; labels defined by **tlb** may be used in defining the trace range)

xtv (threshold voltage setting for analyzer lines; **tlb** can be used to define positive and negative logic for labels encompassing those lines)

tp, xtp - set and display trigger position within the trace

```
tp                - display trigger position
tp s              - trigger position start of the trace
tp c              - trigger position center of the trace
tp e              - trigger position end of the trace
tp -b <offset>    - trigger is offset states from beginning of trace
tp -a <offset>    - trigger has offset states after the trigger

xtp                - display trigger position
xtp s              - trigger position start of the trace
xtp c              - trigger position center of the trace
xtp e              - trigger position end of the trace
xtp -b <offset>    - trigger is offset states from beginning of trace
xtp -a <offset>    - trigger has offset states after the trigger
```

The **tp** (**xtp**) command allows you to specify where the trigger state will be positioned within the emulation (external) trace list.

The position number specified has an accuracy of +/**-** 1 state when counting states or time; when counts are turned off, the accuracy is +/**-** 3 states.

The parameters are as follows:

s — The trigger is positioned at the start of the trace list.

c — The trigger is positioned at the center of the trace list.

e — The trigger is positioned at the end of the trace list.

-b — Indicates that the trigger is to be placed in the trace list with <offset> number of states before the trigger position to the beginning of the trace.

-a — Indicates that the trigger is to be placed in the trace list with <offset> number of states after the trigger position to the end of the trace.

<offset> — A decimal value from 0 to 1023.

If no parameters are supplied, the current trigger position setting is displayed. Upon powerup or after **tinit**, the trigger position is **tp s**.

Note that the **s**, **c**, and **e** options are the only position parameters that are valid for the optional external analyzer set to timing mode (**xtmo -t**).

See Also

tg (defines the trigger expression)

tl (used to display the trace list)

tsq (used to specify the trigger position within the trace sequencer; reference the sequencer operation when deciding where to position the trigger in the trace list, if you want to capture all of the sequence conditions)

xtmo (specifies whether the external analyzer acts independently or is appended to the emulation analyzer)

tpat, xtpat - set and display pattern resources

```
tpat                                  - display all patterns
tpat <pattern>                        - display named patterns
tpat <pattern> <label>=<value>        - equals pattern
tpat <pattern> <label>!=<value>       - not equals pattern
tpat <pattern> <label>=<value> and <label>=<value>
tpat <pattern> <label>!=<value> or <label>!=<value>

xtpat                                 - display all patterns
xtpat <pattern>                       - display named patterns
xtpat <pattern> <label>=<value>       - equals pattern
xtpat <pattern> <label>!=<value>      - not equals pattern
xtpat <pattern> <label>=<value> and <label>=<value>
xtpat <pattern> <label>!=<value> or <label>!=<value>
```

The **tpat** (**xtpat**) command allows you to assign pattern names to simple emulation (external) analyzer expressions. These pattern names are then used in building complex expressions for other analyzer commands.

The **tpat** command is only valid in the complex analyzer configuration (**tcf -c**).

The parameters are as follows:

<pattern> One of the pattern names **p1** through **p8**.

<label> A trace label that is currently defined via either the **tlb** or **xtlb** commands.

<value> Values are numeric constants, equates, or symbols. Also, values can be the result of constants, equates, and symbols combined with operators. Refer to the <value> description.

If no parameters are given, or if the pattern name is given as *****, all eight of the current pattern assignments are displayed. If one of the pattern names is given, the expression assigned to that pattern is displayed.

Upon entering complex configuration after powerup or a **tinit** initialization, all eight patterns are defined as **tpat <pattern> any**.

See Also

tcf (defines whether the analyzer is in easy configuration or complex configuration; the **tpat** command is only valid in complex configuration)

telif (specifies a secondary branch qualifier in analyzer complex configuration; **tpat** patterns may be used in qualifier specification)

tg (used to specify a simple trigger in either easy configuration or complex configuration; **tpat** patterns may be used in complex configuration trigger specification)

tif (used to specify a primary branch qualifier in either analyzer configuration; **tpat** patterns may be used in complex configuration branch specifications)

tpq (specifies a trace prestore qualifier; **tpat** patterns may be used in qualifier specification)

trng (defines a range of values on a set of analyzer input lines; this range may be used in conjunction with the patterns defined by **tpat** in setting up complex analysis qualifiers)

tsq (used to manipulate the trace sequencer)

tsto (used to define global storage qualifiers in both analyzer configurations; may also be used to define storage qualifiers for each sequencer level in complex configuration. The patterns defined by **tpat** may be used in complex configuration storage qualifier definition.)

xtmo (determines whether the external analyzer acts as an independent state or timing analyzer or is appended to the emulation analyzer. If appended, the **xtpat** command is no longer valid; **tpat** defines patterns to be used across both analyzers.)

tpq, xtpq - set or display prestore specification

```
tpq               - display prestore specification
tpq <expr>        - set prestore specification

xtpq               - display prestore specification
xtpq <expr>        - set prestore specification
```

The **tpq** (**xtpq**) command allows you to specify a prestore qualifier for the emulation (external) trace.

During the trace, the analyzer fills a two stage pipe with states that satisfy the prestore qualifier. Each time a trace state is stored into the trace buffer, the prestore qualifier is also stored and then cleared. Therefore, up to two prestore events may be stored for each normal store event; the prestore events in the trace buffer will correspond to the most recent states that satisfied the prestore qualifier immediately prior to a store event but following the previous store event.

The parameters are as follows:

<expr> State qualifier expression. Refer to the <expr> description in this chapter.

If no parameters are given, the current prestore qualifier setting is displayed. Upon powerup or after **tinit** initialization, the prestore qualifier defaults to **tpq none**.

See Also

tcf (specifies whether the analyzer is to operate in easy configuration or complex configuration)

tsq (used to manipulate the trace sequencer)

tsto (used to specify a global storage qualifier for both easy configuration and complex configuration; also used to specify individual sequence term storage qualifiers in complex configuration)

xtmo (specifies whether the external analyzer will act as an independent state or timing analyzer or whether it will be appended to the emulation analyzer. If appended to the emulation analyzer, the **xtpq** command has no effect; the **tpq** command sets the prestore qualifier for both analyzers.)

trng, xtrng - set or display range pattern

```
trng                           - display range
trng <label>=<value>..<value>  - define range

xtrng                           - display range
xtrng <label>=<value>..<value>  - define range
```

The **trng** (**xtrng**) command lets you specify a range of acceptable values for an emulation (external) trace label. This range may then be used in complex qualifiers for the trace specification. The **trng** (**xtrng**) command is only available in the analyzer's complex configuration (see **tcf** syntax pages).

There is no need for a not equals operator in specifying ranges, as the trace specification commands which allow "range" as a parameter also accept "not range" in the form **!r**.

If the optional external analyzer has been appended to the emulation analyzer via the **xtmo** command, the **xtrng** command is invalid; **trng** sets a range pattern to be used by both analyzers.

The parameters are as follows:

<label> A trace label that is currently defined via either the **tlb** or **xtlb** commands.

<value> Values are numeric constants, equates, or symbols. Also, values can be the result of constants, equates, and symbols combined with operators. Refer to the <value> description.

If no parameters are supplied, the current range definition is displayed. After powerup or **tinit** initialization, the **trng** command is set to **trng any**. (Note that **trng** is not directly available after analyzer initialization; the analyzer is set to easy configuration when initialized. You must then switch to complex configuration to access **trng**.)

Ranges can be specified that encompass more bits than the number of bits defined for the specified label.

Note that the **tcf -e** (set trace configuration to easy) command also will reset **trng**. In other words, any **trng** defined when the analyzer was in complex configuration is destroyed when the analyzer is set to easy configuration; you cannot return to complex configuration and use the old **trng**.

See Also

tcf (sets analyzer to complex or easy configuration; analyzer must be in complex configuration to utilize the **trng** command)

telif (specifies the sequencer secondary branch expression; in complex configuration, this expression can include references to the range)

tg (specifies analyzer trigger; may trigger on references to range)

tif (specifies the sequencer primary branch expression; in complex configuration, branch expression may include range qualifier)

tpat (trace pattern definition; assigns pattern names to simple expressions for later use in analyzer specifications. **tpat** essentially commits only one pattern to a label; whereas **trng** allows a range of values to be assigned to the range pattern)

tpq (defines trace prestore qualifier; the range specification may be used in complex configuration prestore qualifier expressions)

tsq (trace sequencer definition)

tsto (defines trace storage qualifier; that is, specifies exactly what states are actually to be stored by the analyzer. In complex configuration, this can include states that fall within the specification defined by **trng**)

xtmo (specifies the mode of the external analyzer; either an independent state or timing analyzer or an analyzer appended to the emulation analyzer)

ts, xts - display status of emulation trace

```
ts          - display complete emulation trace status
ts -w       - display short status

xts          - display complete emulation trace status
xts -w       - display short status
```

The **ts** (**xts**) command allows you to determine the current status of the emulation (external) analyzer.

The parameters are as follows:

-w The **-w** option indicates that the trace status should be printed in whisper mode; this gives an abbreviated version of the status. See "Whisper Mode Trace Display" below for interpretation of the whisper status information.

Trace Status Displays

The emulation and external state trace status is displayed in the following form:

```
---[Emulation | External] Trace Status---
(NEW) [User | CMB ] trace [complete | halted | running ]
Arm [ ignored | (not) received ]
Trigger (not) found
Arm to trigger armcount
States visible (history) first..last
Sequence term term
Count remaining count
```

The external timing trace status is displayed in the following form:

```
--- External Timing Trace Status---
(NEW) [User | CMB ] trace [complete | halted | running ]
Arm [ ignored | (not) received ]
trace status
Arm to trigger armcount
Samples visible (history) first..last
```

The trace status header indicates whether this status is for the emulation or external state trace.

Whether the trace status is displayed as Emulation or External depends on:

- Presence of the optional external analyzer.

- Whether you entered the **ts** (emulation trace status) or **xts** (external trace status) command.
- The current mode setting of the optional external analyzer. If set as a state analyzer (**xtmo -s**), you can have an external state trace status. If set as a timing analyzer (**xtmo -t**), there is a different display for timing status (described below). If appended to the emulation analyzer, the **xts** command is invalid; the external analyzer acts as an extension to the emulation analyzer and their status is reported under the Emulation Trace Status.

Status Display Interpretation

The first line of the trace status indicates the initiator of the trace, whether the trace is completed, running, or halted, and whether or not this trace has been displayed.

NEW — This trace has not been displayed. The **tl** (**xtl**) command will clear this flag until the next trace is started. Halting a trace that is running (as opposed to complete), marks the trace as being NEW even though the trace may have been displayed while running. The next **tl** command with no options will list the trace from the top.

User — The operator initiated this trace with the **t** (**xt**) command.

CMB — This trace was initiated by a /EXECUTE pulse on the CMB after a **tx** command was entered.

complete — The trace has found its trigger and completed.

halted — The trace was halted in response to a **th** (**xth**) command.

running — The trace is still running; either the complete sequencer specifications have not yet been satisfied; or not enough qualified store states have been found to fill trace memory.

The second line of the trace display indicates the analyzer arm status.

ignored — The arm condition specified for this trace was **tarm always.**

received — The arm condition has been satisfied.

not received — The arm condition was not satisfied. (If you specified an arm condition but didn't use it in trigger qualification, this will be displayed if the arm condition is not satisfied. However, the analyzer may still find the correct trigger and complete the trace.)

The third line of the state trace display indicates the trigger status. Because of the pipelined analyzer architecture, it is possible that the trace status may display "not found" when in fact the trigger has been found. This will occur when not enough states satisfying the storage specification are found to push the trigger out of the pipeline and into trace memory. In any case, the trace will not be displayable until the trigger is in trace memory (unless you halt the analyzer).

found — The trigger condition has been found.

not found — The trigger condition has not yet been satisfied.

For the external timing status, the third line indicates the timing trace status. This will be one of the following strings:

Tracepoint found

Trigger found - delaying

Pattern found - waiting for edge

Prestore complete - waiting for trigger

Waiting for prestore

Waiting for arm

The fourth line of the trace display indicates the amount of time that passed between the arm signal and the trigger condition.

armcount — This will be from -0.04 microseconds to 41.943 milliseconds. The arm to trigger counter may underflow or overflow, in which case "<-0.04 microseconds" or ">41.943 milliseconds" are reported, respectively. If the arm signal was ignored or if the trigger was not found, the character "?" (unknown) is displayed.

The fifth line of the trace display indicates the number of states displayable by **tl**. (Number of samples in the case of the external timing trace.)

visible — Number of states which can be displayed by tl (xtl); this will be a number from 0 to 1024 (or 0 to 512 if tcq is active).

history — Number of states which can be displayed if the current trace is halted; this may include history states which may be overwritten and thus unavailable if the current trace runs to completion.

first Number of the first state stored in trace memory, relative to the trigger state. This will be a number from -1024 to 0. The character "?" is displayed if the trigger state is not yet in memory.

last Number of the last state stored in trace memory, relative to the trigger state. This will be a number from -1 to 1023. The character **?** is displayed if the trigger state is not yet in memory.

The sixth line of the trace display indicates the current sequencer term position. (Not used in the external timing trace status.)

term Current sequence term position (1 through 5 in easy configuration; 1 through 8 in complex configuration). If the trace is completed or halted, the last sequence term number is displayed. A "?" is displayed if the trace is running and the sequencer is running too quickly for the current term number to be read.

The seventh line of the trace display indicates the count qualifier status for the primary branch condition of the current sequence term, see **tif** for further details. (Not used in the external timing trace status.)

count Remaining number of occurrences of the primary branch qualifier needed to satisfy the qualifier so that the primary branch will be taken. A "?" is displayed if the trace is running and the counter is updating too quickly to be read.

Whisper Mode Trace Display

If the **-w** option is given, an abbreviated version of the trace status is given as follows:

Trace run status:

- **R -** trace running
- **C -** trace completed
- **H -** trace halted

Trace arm status:

- **A -** Arm has been received
- **a -** arm has not yet been received
- **x -** arm signal is being ignored

Trace trigger status:

- **T -** trace trigger has been found
- **t -** trace trigger has not yet been found

Trace list status:

*** -** indicates that this trace has not been displayed

See Also

es (allows you to determine general emulator status)

t (starts an emulation trace)

tarm (arm the analyzer based on state of the trig1 and trig2 signals)

tg (specify the analyzer trigger state)

th (halt the current trace in process)

tif (specify sequencer primary branch condition and number of occurrences)

tx (specify that trace is to begin upon receiving the CMB /EXECUTE pulse)

x (begin a synchronous CMB execution)

tsck, xtsck - set or display slave clock specification for the analyzer

```
tsck -o <pod number>            - turn slave clock off in pod
tsck -d <pod number> -r <clock> - demux pod, rising edge of clock(s)
tsck -d <pod number> -f <clock> - demux pod, falling edge of clock(s)
tsck -d <pod number> -x <clock> - demux pod, both edges of clock(s)
tsck -m <pod number> -r <clock> - mix pod clocks, rising edge of
                                  clock(s)
tsck -m <pod number> -f <clock> - mix pod clocks, falling edge of
                                  clock(s)
tsck -m <pod number> -x <clock> - mix pod clocks, both edges of
                                  clock(s)

xtsck -o              - turn slave clock off in external pod
xtsck -d -r <clock>   - demux pod, rising edge of clock(s)
xtsck -d -f <clock>   - demux pod, falling edge of clock(s)
xtsck -d -x <clock>   - demux pod, both edges of clock(s)
xtsck -m -r <clock>   - mix pod clocks, rising edge of clock(s)
xtsck -m -f <clock>   - mix pod clocks, falling edge of clock(s)
xtsck -m -x <clock>   - mix pod clocks, both edges of clock(s)
```

The **tsck** (**xtsck**) command allows you to specify the slave clock edges used for the emulation (external) analyzer trace.

Each analyzer pod has the capability of latching certain signals with a slave clock instead of the master clock. (You set up the master clock with the **tck** command.)

The **xtsck** command controls the slave clock for the optional external analyzer. No pod number is necessary since the external analyzer has only one pod.

The parameters are as follows:

<pod number> Specifies one of 4 groups of analyzer input lines. These are as follows:

Pod #	Bits
1	0-15
2	16-31
3	32-47
4	0-15 of the external analyzer

Note that you only need to specify pod 4 if you are using the **tsck** command to operate on the optional external analyzer. You would typically do this only if you had logically joined the analyzers using the **xtmo** command.

-d

Specifies that the slave clock operates in demultiplexed mode. In this mode, the lower 8 channels of the analyzer pod (bits 0-7) are latched with the slave clock and the upper 8 channels (bits 8 through 15) are replaced with the lower 8 channels. In other words, the upper 8 bits are identical to the lower 8 at the pod.

However, the data is not clocked into the analyzer itself until the next master clock occurs. Therefore, if no slave clocks have occurred since the last master clock, the data on the lower 8 analyzer lines is identical to the upper 8. If one or more slave clocks have occurred since the last master clock, the data on the lower 8 bits is the only data available to the analyzer.

When using the **-d** option, you must specify one of the **-r**, **-f**, or **-x** options to indicate the active edge(s) of the slave clock.

-m

Specifies that the slave clock operates in mixed mode. In the mixed mode, the lower 8 channels of the analyzer pod (bits 0-7) are latched with the slave clock, and the master clock latches in the entire pod. Therefore, if no slave clock has occurred since the last master clock, the data on the lower 8 bits of the pod will be clocked into the analyzer at the same time as the upper 8 bits. If more than one slave clocks has occurred since the last master clock, only the first slave clock data will be available to the analyzer.

When using the **-m** option, you must specify one of the **-r**, **-f**, or **-x** options to indicate the active edge(s) of the slave clock.

-r

Indicates that the pod should latch data on the **rising** edge of the slave clock.

-f

Indicates that the pod should latch data on the **falling** edge of the slave clock.

-x

Indicates that the pod should latch data on **both** edges of the slave clock.

<clock>

Five clock signals are defined: J, K, L, M, and N.

The L, M, and N clocks are generated by the emulator. The L clock is the emulation clock derived by the emulator, the N clock is used as a qualifier to provide the user/background tracing options (**-u** and **-b**) to **tck**, and the M clock is not used. The L and N clocks may also be used to clock and qualify the external analyzer as well as the emulation analyzer.

The J and K clocks are the clock inputs on the external trace probe (if one is present). These clock signals should only be used to clock the external trace; they should not be used to clock the emulation trace although it may occasionally be useful to use the external clock signals as qualifiers for the emulation trace.

-o

If you specify **-o** with a <pod number>, the slave clock is ignored on that pod. Remember that you don't need to specify <pod number> with the **xtsck** command; this command operates only on the single external analyzer pod.

If no parameters are specified, the current slave clock definitions are displayed. The default for all slave clocks is **off** after powerup or **tinit** initialization.

See Also

ta (allows you to display active signals on the analyzer input lines; useful in verifying that you have selected the correct clock conditions)

tck (used to define master clock signals used by the analyzer; **tsck** defines the slave clock signals. Default mode for **tsck** is off on all pods.)

xtv (specifies threshold voltages for external analyzer input lines; must be set correctly to ensure that the **J** and **K** clock signals are recognized)

xtmo (specifies mode of operation for the external analyzer; that is, whether it acts as an independent analyzer or is appended to the emulation analyzer)

tsq, xtsq - modify or display sequence specification

In the easy configuration:

```
tsq              - display entire sequence specification
tsq -r           - reset the sequence specification
tsq -i X         - insert sequence term X into sequence
tsq -d X         - delete sequence term X from sequence

xtsq              - display entire sequence specification
xtsq -r           - reset the sequence specification
xtsq -i X         - insert sequence term X into sequence
xtsq -d X         - delete sequence term X from sequence
```

In the complex configuration:

```
tsq              - display entire sequence specification
tsq -t           - display sequence trigger specification
tsq -t X         - set the sequence to trigger on entrance to term X
tsq -r           - reset the sequence specification

xtsq              - display entire sequence specification
xtsq -t           - display sequence trigger specification
xtsq -t X         - set the sequence to trigger on entrance to term X
xtsq -r           - reset the sequence specification
```

The **tsq** (**xtsq**) command allows you to manipulate or display the emulation (external) trace sequencer.

When the analyzer is in easy configuration (**tcf -e**), the sequencer has a maximum of four sequence terms with a minimum of one term.

If the analyzer is in complex configuration (**tcf -c**), the sequencer always has eight terms (although the particular sequencer setup may mean that only two are ever accessed).

The parameters are as follows:

-r

Resets the sequencer.

In the easy configuration, the result is a simple one term sequence which stores all states and triggers on the first occurrence of any state. This is equivalent to issuing the commands:

In the complex configuration, the result is an eight term sequence with the trigger term at term number 2. The sequencer will be set to **tsto any** (store any state). All secondary branch qualifiers are turned off (**telif X never**), and all primary branch qualifiers will jump to the next higher numbered term on any state (**tif X any (X+1)**).

-i Inserts a new sequence term at X. The new sequence term will use the default storage qualifier (which can be modified with the **tsto** command). It will also use the secondary branch qualifier (global restart in easy configuration) specified by the **telif** command.

If there is already a sequence term with number X, terms with number X and above will be renumbered (X becomes X+1) to make room for the new term.

You must insert terms in a contiguous manner; for example, you cannot insert a term number 4 if the sequencer only has two terms defined. Instead, you must next insert a term numbered 1, 2 or 3.

The primary branch qualifier for the new term will be defined as **tif X any** unless it is the last term in the sequence (by definition, the trigger term), in which case the primary branch qualifier is set to **tif X never**.

-d Deletes the term specified and renumbers higher numbered terms downward to fill the gap.

X Specifies a sequence term number.

In the easy configuration, X is in the range from 1 through 4 when inserting or deleting terms.

In the complex configuration, X is in the range 2 through 8 to use as the trigger term.

-t Specifies the trigger term when a sequence term number is included. When no sequence term number is included, the trigger term is displayed. The analyzer triggers on a sequencer branch to the trigger term.

If no options are given, all of the sequencer storage and branch qualifiers are displayed along with the trigger term position. Upon powerup or after **tinit** initialization, the sequencer defaults to the following state:

```
tif 1 any
tsto all
telif never
```

In other words, the sequencer powers up with two sequence terms; the second sequence term is the trigger term. Any state will cause a branch from the first term to the second term; global restart is set to never and all states are stored by the analyzer.

Switching analyzer configurations from easy to complex or vice versa also resets the sequencer (that is, **tcf -c** or **tcf -e**).

See Also

tcf (defines whether analyzer is operated in complex configuration or easy configuration)

telif (sets global restart qualifier in easy configuration; secondary branch qualifier in complex configuration)

tg (defines the trigger qualifier)

tif (sets the primary branch qualifier in both easy and complex configuration)

tsto (defines the analyzer global storage qualifier)

tsto, xtsto - set or display trace storage specification

In the easy configuration:

```
tsto              - display storage specification
tsto <expr>       - define storage specification

xtsto              - display storage specification
xtsto <expr>       - define storage specification
```

In the complex configuration:

```
tsto              - display all storage specifications
tsto X            - display storage qualifier X specification
tsto <expr>       - define global storage specification
tsto X <expr>     - define storage qualifier X specification

xtsto              - display all storage specifications
xtsto X            - display storage qualifier X specification
xtsto <expr>       - define global storage specification
xtsto X <expr>     - define storage qualifier X specification
```

The **tsto** (**xtsto**) command allows you to specify a trace storage qualifier for the emulation (external) analyzers. The expression parameter qualifies the states to be stored by the analyzer.

The parameters are as follows:

<expr> — State qualifier expression. Refer to the <expr> description in this chapter.

X — Specifies the sequence term number whose storage qualifier is either displayed or assigned as <expr>.

If no parameters are given, the current trace storage qualifier settings are displayed. Upon powerup or after **tinit** initialization, the trace storage qualifier defaults to **tsto all**. Using the **tcf** command to switch from complex configuration to easy configuration or vice versa will also reset the storage qualifier to **tsto all**.

If the analyzer is in easy configuration (**tcf -e**), the expression is specified by <expr> and this serves as a global storage qualifier. In other words, the same expression is used as a storage qualifier regardless of the current sequencer state.

If the analyzer is in complex configuration (**tcf -c**), the expression is specified by <expr> and may be assigned to a sequencer state with the X parameter. When an expression is assigned to a specific term number, the analyzer will only store states corresponding to the given expression when at the given sequencer level. If no sequence term number is used, the associated expression is defined as global; the analyzer stores states satisfying the expression regardless of the sequencer level.

See Also

tcf (used to specify whether the analyzer is in easy configuration or complex configuration)

telif (used to specify a global restart qualifier in easy configuration; specifies a secondary branch qualifier for each sequencer level in complex configuration)

tg (used to specify a trigger condition in either easy configuration or complex configuration; overrides the current sequencer specification. Note that **tg** does not affect **tsto**; therefore, the current **tsto** specifications remain in effect whenever a **tg** command is entered)

tif (used to specify a primary branch qualifier in either analyzer configuration)

tpat (used to assign pattern names to simple analyzer expressions for use in constructing complex analyzer expressions; these expressions can be used in specifying storage qualifiers for the **tsto** command)

trng (used to specify a range of values of a set of analyzer inputs; this range information can be used in constructing complex configuration qualifiers for the **tsto** command)

tsq (used to manipulate the trace sequencer)

tx, xtx - enable/disable execute condition

```
tx -e    - start a measurement when the execute signal is received
tx -d    - ignore the execute signal

xtx -e    - start a measurement when the execute signal is received
xtx -d    - ignore the execute signal
```

The **tx** command allows you to specify that the analyzer will begin a measurement when the CMB /EXECUTE line is asserted.

The parameters are as follows:

-e Specifies that the analyzer will start a measurement upon receiving the CMB /EXECUTE signal.

-d The analyzer will NOT start a measurement upon receiving the CMB /EXECUTE signal.

If no options are specified, the current state of **tx** enable/disable is displayed. Upon powerup or after a **tinit**, the system defaults to **tx -d**.

If **tx -e** is given, enabling measurement on execute, the CMB trigger is immediately driven true upon receiving the /EXECUTE signal. If the analyzer is not driving either trig1 or trig2, it is then started. The CMB trigger is then disabled and the HP 64700 waits for all other participants in the measurement to release the CMB trigger. When the last instrument releases the CMB trigger, the trigger will go false; at this point any analyzers driving trig1 or trig2 will be started.

See Also

cmbt (specifies whether the CMB trigger signal is driven or received by the internal trig1 and trig2 signals)

tarm (specifies the arm condition for the analyzer)

tg (specifies a trigger condition for the analyzer)

<value> - values in Terminal Interface commands

Values are numeric constants, equates, or symbols. Also, values can be the result of constants, equates, and symbols combined with operators. Equates are defined with the **equ** command. Symbols can be loaded with the **load** command or defined with the **sym** command.

Constants

A value may be specified as a constant in any of the following number bases. (Constants with no base specified are assumed to be hexadecimal numbers.)

- Hexadecimal (base **H** or **h**). For example: 6eh, 9xH, 0f3, or 0cfh. (The leading digit of a hexadecimal constant must be 0-9.)
- Decimal (base **T** or **t**, for base "ten"). For example: 27t or 99T. (Don’t cares are not allowed in decimal numbers.)
- Binary (base **Y** or **y**). For example: 1101y, 01011Y, or 0xx10xx11y. (The leading digit of a binary constant must be 0 or 1. Do not use the characters "B" or "b" to specify the base of binary numbers because they will be interpreted as hexadecimal numbers; for example, 1B equals 27 decimal.)
- Octal (base **Q**, **q**, **O**, or **o**). For example: 777o, 6432q, or 7xx3Q. (The leading digit of an octal constant must be 0-7.)

Don’t cares are not allowed in ranges or decimal numbers. A value of all don’t cares may be represented by a question mark (**?**).

Operators. When specifying values, constants can be combined with the following operators (in descending order of precedence):

-, **~**	Unary two’s complement, unary one’s complement. The unary two’s complement operator is not allowed on constants containing don’t care bits.
*****, **/**, **%**	Integer multiply, divide, and modulo. These operators are not allowed on constants containing don’t care bits.
+, **-**	Addition, subtraction. These are not allowed on constants containing don’t care bits.

<<, <<<, >>, >>>	Shift left, rotate left, shift right, rotate right.
&	Bitwise AND.
^	Bitwise exclusive or, XOR.
|	Bitwise inclusive OR.
&&	Logical AND/bit-wise merge. When bits are different, the first value overrides the second; e.g., 10xxy && 11x1y == 10x1y.

Note that all operations are carried out on 32-bit numbers.

ver - display system software and hardware version numbers

```
ver
```

The **ver** command instructs the emulator to return the current emulator Terminal Interface software version numbers.

w - wait for specified condition before continuing

```
w          - wait for any keystroke
w <value>  - wait for <value> number of seconds or any keystroke
w -m       - wait for measurement complete or any keystroke
```

The **w** command is used to program automatic waits into macros, repeats, and command files. Normal operation is to wait for any keystroke before executing the next operation; optionally, the wait can be programmed for a specific time period or for completion of a measurement in process (such as a trace).

The parameters are as follows:

<value> The number of seconds before proceeding.

-m Wait for completion of the current measurement before proceeding.

Examples

To cause the emulator to wait for any keystroke before proceeding to the next command, type:

```
U>w
```

You might use this in a situation where you wish the operator to make a judgment regarding some other condition before proceeding with the next measurement.

To cause the emulator to wait for 32 seconds or for any keystroke, type:

```
U> w 32
```

This might be used where you know the desired system state will be reached in a definite amount of time (or should be reached within that time).

To have the emulator wait until another measurement is completed or for any keystroke entry, type:

```
U> w -m
```

Note that the above examples, taken exactly as shown, don't provide you with a useful function -- they are provided only to show correct examples of command line syntax. To use the wait command effectively, it should be applied within macros, repeat commands, or command files. Refer to the **rep** and **mac** commands for further examples.

x - emit a Coordinated Measurement Bus execute signal

x

The **x** command allows you to initiate a synchronous CMB (Coordinated Measurement Bus) measurement execution.

When **x** is performed, the CMB /EXECUTE line is pulsed. If **tx** (trace at execute) is enabled, an analyzer measurement will begin. If the CMB is enabled via the **cmb -e** command, a break will occur, followed by a run at execute as specified by the **rx** command.

The **x** command is available whether CMB and trace at execute are enabled or not. Specifically, the **cmb** and **tx** commands control how this HP 64700 emulator will respond when an /EXECUTE or READY is detected. The **x** command only controls when this emulator will issue an /EXECUTE signal.

See Also

cmb (used to enable or disable interaction with the CMB)

rx (used to specify an address to start a program run when the /EXECUTE pulse is received from the CMB)

tx (used to specify that an analyzer measurement should begin when the /EXECUTE pulse is received from the CMB)

xteq - set/display external timing edge qualifier

```
xteq                - display current channels which will cause
                      a trigger on a rising or falling edge
xteq -f <rangelist> - set falling edge channels to <rangelist>
xteq -r <rangelist> - set rising edge channels to <rangelist>
xteq -f <unarypat> - set falling edge channels to <unarypat>
xteq -r <unarypat> - set rising edge channels to <unarypat>
xteq -r <rangelist> -f <unarypat> - multiple arguments accepted
```

The **xteq** command allows you to specify the channels which will cause an edge trigger.

The trigger will occur following a valid duration of a pattern specified by **xtt** when a transition occurs on any of the lines specified in **xteq**. Note that **xteq** allows you to qualify the transitions to trigger only on the rising edge or the falling edge of the given input lines.

Note that the timing trace information is only accessible through the binary trace list option (**tl -b**).

The parameters are as follows:

-r — The trigger will occur on the rising edge of any signal on the input lines specified.

-f — The trigger will occur on the falling edge of any signal on the input lines specified.

<unarypat> — Valid unary pattern values are:

all any	Set qualifier to all 16 channels.
none never	Set qualifier to 0 channels.

<rangelist> — This parameter can be one or more of the following range arguments:

<num>	Channel identifier (0 to 16). This specifies the bit which will cause an edge trigger.
<num>..<num>	Channel range (0 to 16). This specifies the range of bits which will cause an edge trigger.

<label> Use full range of <label>. All of the bits assigned to the label will cause an edge trigger. Refer to the **tlb, xtlb** description for information on defining labels.

<label>:<num> Use <num> as offset into <label>.

<label>:<num>..<num>

Specifies a subrange of <label>. This specifies the bits to be used within that label which will cause an edge trigger.

Note that when specifying a range of bits to use within a label, the bit range specified is relative to the label, not to the input bit. For example, if you define a label named STATUS with input bits 8..11, then want to specify the least significant two bits of STATUS in a trigger specification, you can use either **STATUS:0..1** or simply the range **8..9**.

If no parameters are specified, the current edge qualifier is displayed. Upon powerup or **tinit** initialization, the default setting is **xteq -r any -f any**.

When multiple arguments are used, the combinations are ORed together to form a single pattern.

See Also

tlb,xtlb (specifies labels assigned to input lines for the emulation (external) analyzer)

xtgq (specifies an glitch qualifier used in conjunction with **xtt** to determine a valid trigger state)

xtm (specifies timing analyzer mode)

xtt (specifies timing analyzer trigger pattern and duration)

xtgq - set/display external timing glitch qualifier

```
xtgq             - display current channels which will
                   cause a glitch trigger
xtgq <rangelist> - set glitch channels to <rangelist>
xtgq <unarypat>  - set glitch channels to <unarypat>
```

The **xtgq** command allows you to specify the channels which will cause a glitch trigger.

A glitch trigger will occur following a valid duration of a pattern as specified in the **xtt** command while the pattern is still present. A less than duration specified in **xtt**, or a timing mode other than **xtm -g** will cause the **xtgq** command to be ignored.

You might use this command to look for glitch occurrences related to a specific bit pattern.

Note that the timing information is only accessible through the binary trace list option (**tl -b**).

The parameters are as follows:

<unarypat> Valid unary pattern values are:

all any	Set qualifier to all 16 channels.
none never	Set qualifier to 0 channels.

<rangelist> This parameter can be one or more of the following range arguments:

<num>	Channel identifier (0 to 16). This specifies the bit which will cause a glitch trigger.
<num>..<num>	Channel range (0 to 16). This specifies the range of bits which will cause a glitch trigger.
<label>	Use full range of <label>. All of the bits assigned to the label will cause a glitch trigger. Refer to the **tlb, xtlb** description for information on defining labels.
<label>:<num>	Use <num> as offset into <label>.

<label>:<num>..<num>

Specifies a subrange of <label>. This specifies the bits to be used within that label which will cause a glitch trigger.

Note that when specifying a range of bits to use within a label, the bit range specified is relative to the label, not to the input bit. For example, if you define a label named STATUS with input bits 8..11, then want to specify the least significant two bits of STATUS in a trigger specification, you can use either **STATUS:0..1** or simply the range **8..9**.

If no parameters are specified, the current glitch qualifier is displayed. Upon powerup or **tinit** initialization, the default setting is **xtgq none**.

When multiple arguments are used, the combinations are ORed together to form a single pattern.

See Also

tlb,xtlb (specifies labels assigned to input lines for the emulation (external) analyzer)

xteq (specifies an edge qualifier used in conjunction with **xtt** to determine a valid trigger state)

xtm (specifies timing analyzer mode; must be in mode **xtm -g** for **xtgq** use)

xtt (specifies timing analyzer trigger pattern and duration)

xtm - set/display external timing mode

```
xtm     - display current timing mode
xtm -s  - turn on standard timing mode
xtm -g  - turn on glitch timing mode
xtm -t  - turn on transitional timing mode
```

The **xtm** command allows you to specify the mode of operation for the timing analyzer.

This command is only available if the HP 64700 emulator is equipped with the external state/timing analyzer option.

The parameters are as follows:

-s Selects the standard timing analyzer mode and samples data at the period selected by **xtsp**; up to 1024 samples can be stored during a single trace.

-g The timing analyzer operates in standard mode with glitch detection added. Again, the sample rate is selected by **xtsp**. When glitch mode is selected, the maximum number of samples per trace is reduced to 512.

-t Selects the transitional timing analyzer mode. Data is only stored when an input transition is detected. For the analyzer to record these transitions accurately, some trace memory must be dedicated to storing the delta time between transitions, so the number of state transitions that can be stored is reduced to a maximum of 512.

If no parameters are supplied, the current mode setting for the timing analyzer is displayed. Upon powerup or **tinit**, the timing analyzer mode is set to **xtm -t**.

See Also

xtmo (specifies whether to use the external analyzer as a separate state analyzer, separate timing analyzer, or append the lines to the emulation analyzer)

xtsp (defines the timing sample period)

xtmo - external analyzer trace mode

```
xtmo -e    - emulation analyzer has external bits
xtmo -s    - external state analyzer
xtmo -t    - external timing analyzer
```

The **xtmo** command allows you to specify the mode of operation for the external analyzer. The analyzer can be configured to run as an independent state or timing analyzer; or, the external analyzer can be associated with the emulation analyzer to synchronize measurements made by the two analyzers.

The parameters are as follows:

-s The external analyzer acts as an independent state analyzer.

-t The external analyzer acts as an independent timing analyzer.

-e The external analyzer is appended to the emulation analyzer.

If no parameters are specified, the current operation mode of the external analyzer is displayed. Upon powerup, the default operation mode is **xtmo -e**.

Note that if the emulation and external analyzers are clocking data off of the same clock, the setup/hold times of the data on the external analyzer probe inputs may not be met properly. The timing relationship between a target system processor signal and the setup/hold time of the external probe signals must be specified for each emulator. This is because each emulator has unique circuitry that generates the emulation analyzer clock and each processor has different timing requirements. Therefore, each emulator must specify the setup/hold time requirements of the external probe inputs with respect to a target processor signal.

If the external analyzer has been associated with the internal analyzer with the **xtmo -e** command, and trace specifications have been defined referencing lines present on the external analyzer, the analyzer cannot be reconfigured as an independent state or timing analyzer with the **xtmo -s** or **xtmo -t** commands until the trace specifications referencing the external analyzer lines are removed.

If the external analyzer is in the independent state or timing mode, and an **xtmo -e** command is issued to append it to the emulation analyzer, the trace specifications for the external analyzer lines are reinitialized.

See Also

bnct (specifies whether trig1 and/or trig2 are to be driven or received by the rear panel BNC connector)

cmbt (specifies whether the trig1 and/or trig2 signals are to be driven or received by the CMB trigger line)

tarm (specifies the arm condition for the analyzer)

tgout (specifies whether or not the trig1 and/or trig2 signals are to be driven when the analyzer finds its trigger)

tx (specifies that the analyzer is to commence a trace upon receiving the CMB execute pulse)

xtsp - set/display external timing sample period

```
xtsp            - display current timing sample period
xtsp <period> n - set timing sample period to <period> nanoseconds
xtsp <period> u - set timing sample period to <period> microseconds
xtsp <period> m - set timing sample period to <period> milliseconds
```

The **xtsp** command allows you to define the sample period for timing analyzer measurements.

Larger sample periods enable coverage of more events; however, there is the danger that some transitions may be missed if they change during the sample period. Conversely, small sample periods virtually guarantee recording of all transitions but allow the measurement of only a small total number of events in time.

The parameters are as follows:

<period> Defines the sample period for the analyzer. This is an a integer value.

The valid range for <period> is between 10 ns and 50 ms in a 1,2,5 sequence (that is, 10 ns, 20 ns, 50 ns,..., 50 ms) for standard timing modes.

For glitch mode valid periods are between 20 ns and 50 ms in the same step sequence.

For transitional timing mode, the only valid sample period is 10 ns.

n Indicates that the given sample period is in nanoseconds.

u Indicates that the given sample period is in microseconds.

m Indicates that the given sample period is in milliseconds.

If no parameters are given, the current setting of the sample period is displayed. Upon powerup or **tinit** initialization, the sample period setting is **xtsp 10 n**.

See Also **xtm** (defines the timing analyzer run mode; if mode is **xtm -s** or **xtm -g**, then **xtsp** defines the amount of time between samples; if mode is **xtm -t**, the timing analyzer runs in transitional mode; the sample period (10 nanoseconds only) is used as a clock to measure the delta time between transitions)

xtt - set/display external timing trigger condition

```
xtt                     - display current timing trigger setting
xtt <expr> < <period> n - trigger when <expr> is true
                        - for less than <period> ns
xtt <expr> > <period> u - trigger when <expr> is true
                        - for greater than <period> us
xtt <expr> > <period> m - trigger when <expr> is true
                        - for greater than <period> ms
```

The **xtt** command lets you specify the timing analyzer trigger. The trigger specification includes the trigger pattern and the duration of that pattern.

If <expr> is found but <period> is not satisfied, there is a 20 ns reset time before the analyzer will search for another pattern.

The parameters are as follows:

<expr> Defines a simple expression of the general form **label=pattern** or **label=pattern and label=pattern ...**. Also, **any**, **all**, **none**, and **never** may be used as the expression.

Refer to the **tlb, xtlb** description for information on defining labels.

<period> Specifies, in conjunction with the greater than (>) and less than (<) operators, and the **n**, **u** and **m** designators, define a duration for which the trigger must be present to satisfy the trigger condition. The <period> is always expressed as an integer value.

If **> <period>** is specified, <period> must fall within the range of 30 ns to 10 ms in 10 ns increments. The trigger will occur at the end of the specified duration.

If **< <period>** is specified, <DURATION> must fall within the range of 40 ns to 10 ms in 10 ns increments. The pattern must remain stable for at least 20 ns; the trigger will occur after the pattern changes states from the designated pattern.

n Indicates that the duration is specified in nanoseconds.

u Indicates that the duration is specified in microseconds.

m Indicates that the duration is specified in milliseconds.

If no parameters are specified, the current timing analyzer trigger expression and duration are displayed. Upon powerup or **tinit** initialization, the timing trigger is set to **xtt any**.

See Also

xteq (specifies that certain timing channels will qualify the trace trigger specified by **xtt**; the pattern and duration are specified by **xtt**, the trigger occurs when the signal transition specified by **xteq** occurs)

xtgq (specifies a glitch qualifier for **xtt**; the trigger occurs after the pattern and duration specified by **xtt** is satisfied when the glitch specified by **xtgq** occurs)

xtlb (defines labels for external analyzer input lines)

xtm (sets the timing mode for the analyzer to standard, glitch, or transitional)

xttd - set/display external timing trigger delay

```
xttd            - display current timing trigger delay
xttd <period> n - set timing trigger delay to <period> nanoseconds
xttd <period> u - set timing trigger delay to <period> microseconds
xttd <period> m - set timing trigger delay to <period> milliseconds
```

The **xttd** command allows you to specify the amount of time to delay the timing analyzer trigger after a valid trigger condition has occurred.

The parameters are as follows:

<period> — Specifies, along with the **n**, **u**, or **m** parameters, the trigger delay period for the analyzer. This is an a integer value; the valid range for <period> is between 0 and 10 ms in 10 ns increments.

n — Indicates that the given delay is in nanoseconds.

u — Indicates that the given delay is in microseconds.

m — Indicates that the given delay is in milliseconds.

If no parameters are given, the current setting of the delay is displayed. Upon powerup or **tinit** initialization, the delay setting is **xttd 0**.

See Also — **xtt** (specifies the timing analyzer trigger pattern and duration)

xttq - set/display external timing transition qualifier

```
xttq             - display current channels which will
                   cause a transition in transitional mode
xttq <rangelist> - set transition channels to <rangelist>
xttq <unarypat>  - set transition channels to <unarypat>
```

The **xttq** command allows you to specify the channels which will cause a transition record when the timing analyzer mode is set to transitional (**xtm -t**).

The parameters are as follows:

<unarypat>

Valid unary pattern values are:

all
any

Set qualifier to all 16 channels.

none
never

Set qualifier to 0 channels.

<rangelist>

This parameter can be one or more of the following range arguments:

<num> Channel identifier (0 to 16). This specifies the bit which will cause a timing transition record.

<num>..<num> Channel range (0 to 16). This specifies the range of bits which will cause a timing transition record.

<label> Use full range of <label>. All of the bits assigned to the label will cause a timing transition record. Refer to the **tlb, xtlb** description for information on defining labels.

<label>:<num> Use <num> as offset into <label>.

<label>:<num>..<num>

Specifies a subrange of <label>. This specifies the bits to be used within that label which will cause a timing transition record.

Note that when specifying a range of bits to use within a label, the bit range specified is relative to the label, not to the input bit. For example, if you define a label named STATUS with input bits 8..11, then want to specify the least significant two bits of STATUS in a trigger specification, you can use either **STATUS:0..1** or simply the range **8..9**.

If no parameters are specified, the current transition qualifier is displayed. Upon powerup or **tinit** initialization, the default setting is **xttq any**.

See Also

tlb,xtlb (specifies labels assigned to input lines for the emulation (external) analyzer)

xteq (specifies an edge qualifier used in conjunction with **xtt** to determine a valid trigger state)

xtgq (specifies a glitch qualifier used in conjunction with **xtt** to determine a valid trigger state)

xtm (specifies timing analyzer mode; must be in mode **xtt -t** (transitional mode) for **xttq** to be useful)

xtt (specifies timing analyzer trigger pattern and duration)

xtv - threshold voltage for the external analyzer

```
xtv                       - display current threshold voltage
xtv -l<level>             - set lower byte and J clock
xtv -u<level>             - set upper byte and K clock
xtv -u<level> -l<level>   - multiple arguments accepted
```

The **xtv** command allows you to set the logic threshold voltages for the external trace probes.

The parameters are as follows:

-l Specifies the threshold voltage that is to be used for the lower 8 bits of the analyzer probe. These are bits 0 through 7 and the **J** clock.

-u Specifies the threshold voltage specified that is to be used for the upper 8 bits of the analyzer probe. These are bits 8 through 15 and the **K** clock.

<level> Specifies the voltage level. Valid options for <level> are:

ECL	Voltage levels for ECL logic, **-**1.3 volts.
TTL	Voltage levels for TTL logic, +1.4 volts.
CMOS	Voltage levels for CMOS logic, +2.5 volts.
+/-x.xx	User definable levels -6.40 to +6.35 volts.

If no parameters are specified, the current threshold voltage settings are printed. Upon powerup or **tinit** initialization, the threshold voltage settings are set to **xtv -u TTL -l TTL**.

See Also

ta (allows you to view trace input signal activity; useful in verifying the correct threshold levels)

10

Error Messages

Error Messages

This chapter contains descriptions of error messages that can occur while using the Terminal Interface. The error messages are listed in numerical order, and each description includes the cause of the error and the action you should take to remedy the situation.

The emulator can return messages to the display only when it is prompted to do so. Situations may occur where an error is generated as the result of some command, but the error message is not displayed until the next command (or a carriage return) is entered.

A maximum number of 8 error messages can be displayed at one time. If more than 8 errors are generated, only the last 8 are displayed.

Emulator Error Messages

1 **I/O port access not supported**

Cause: You attempted to use the **io** command for an emulator whose processor does not support separate I/O (such as the 68000).

Action: Use the **m** command to modify I/O ports on these emulators.

20 **Attempt to change foreground monitor map term**

Cause: The **cf mon=fg** command that sets up use of a foreground monitor also maps a memory range for the monitor's use. You attempted to alter that term using the **map** command.

Action: Try using another memory range for the new map term. If you need to have the range used by the foreground monitor, then switch to a background monitor, delete the old foreground monitor map term, and add the new term. Now you can return to using a foreground monitor; remember you will need to reload the monitor code.

21 **Insufficient emulation memory**

Cause: You have attempted to map more emulation memory than is available.

Action: Reduce the amount of emulation memory that you are trying to map.

40 **Restricted to real time runs**

Cause: While the emulator is restricted to real-time execution, you have attempted to use a command that requires a temporary break in execution to the monitor. The emulator does not permit the command and issues this error message.

Action: You must break the emulator's execution into the monitor before you can enter the command.

61 **Emulator is in the reset state**

Cause: You have entered a command that requires the emulator to be running in the monitor (for example, displaying registers).

Action: Enter the **b** (break) command to cause the emulator to run in the monitor, and enter the command that caused the error again.

80 **Stack pointer is odd**

Cause: You have attempted to modify the stack pointer to an odd value for a processor that expects the stack to be aligned on a word boundary (such as the 68000).

Action: Modify the stack pointer to an even value.

81 **Stack is in guarded memory**

Cause: Your stack pointer pointed to a location in memory mapped as guarded; you then attempted to run or step the emulation processor. The emulator was unable to access the stack to complete the transition from the monitor to the user program or vice versa.

Action: Either remap memory so the stack pointer points to a location in RAM, or change the stack pointer value (either with your program or with the **cf** command options, if available) to point to a location in RAM.

82 **Stack is in target ROM**

Cause: Your stack pointer pointed to a location in memory mapped as target ROM; you then attempted to run or step the emulation processor. The emulator was unable to access the stack to complete the transition from the monitor to the user program or vice versa.

Action: Either remap memory so the stack pointer points to a location in RAM, or change the stack pointer value (either with your program or with the **cf** command options, if available) to point to a location in RAM.

83 **Stack is in emulation ROM**

Cause: Your stack pointer pointed to a location in memory mapped as emulation ROM; you then attempted to run or step the emulation processor. The emulator was unable to access the stack to complete the transition from the monitor to the user program or vice versa.

Action: Either remap memory so the stack pointer points to a location in RAM, or change the stack pointer value (either with your program or with the **cf** command options, if available) to point to a location in RAM.

84 **Program counter is odd**

Cause: You attempted to modify the program counter to an odd value using the **reg** command on a processor which expects even alignment of opcodes.

Action: Modify the program counter only to even numbered values.

102 **Monitor failure; no clock input**

Cause: The monitor is unable to run because no emulation processor clock is available.

Action: Make sure a clock meeting the microprocessor's specifications is input to the clock pin of the target system probe.

103 **Monitor failure; no processor cycles**

Cause: The monitor is unable to run since the processor is not running. The monitor is unable to determine the cause of the failure.

Action: If running in-circuit, troubleshoot the target system. If running out of circuit, reinitialize the emulator and try the procedure again.

104 **Monitor failure; bus grant**

Cause: The monitor is unable to run. The emulation processor is not running because it has granted the bus to another device.

Action: Wait until the processor has regained bus control, then retry the operation.

105 **Monitor failure; halted**

Cause: The monitor is unable to run because the processor is halted (due to an external halt line or a halt instruction).

Action: Release the external halt and retry the operation. If the processor halted due to a halt instruction, try the **rst** command, then retry the operation.

106 **Monitor failure; wait state**

Cause: The monitor is unable to run because the processor is in a continuous wait state.

Action: A continuous wait state may indicate target system problems. Troubleshoot the wait line. If you were running out of circuit, try initializing the emulator with **init**, then retry the procedure.

107 **Monitor failure; bus error**

Cause: The monitor is unable to run because the processor has encountered a bus fault (such as the 68000 /BERR line).

Action: Release the /BERR line and determine why it was activated.

68000 Emulator Messages

The following error messages are unique to the 68000 emulator.

140 **Supervisor stack pointer not initialized**

Cause: The supervisor stack pointer was not initialized to a value on the transition from emulation reset to the monitor.

Action: Use the **cf rssp** command to define a supervisor stack pointer lying within an area mapped as RAM and reserved for stack space, and execute the transition from emulator reset to the monitor (**rst -m** will perform the transition). Or, you can modify the supervisor stack pointer directly using the command **reg ssp=<value>**.

141 **Foreground monitor operating in USER mode**

Cause: The foreground monitor was found operating in the 68000 user program state.

Action: Reset the emulator. Check your foreground monitor source code to verify that it keeps the processor in the supervisor state and does not make transitions into the user program state.

142 **Supervisor stack in guarded memory at <address>**

Cause: The supervisor stack either was defined in or grew into a memory range mapped as guarded.

Action: Reset the emulator. Then, define the supervisor stack pointer within a memory range mapped as emulation or target RAM and allow sufficient room for the stack to grow as procedures are activated and deactivated.

143 **Supervisor stack is in ROM at <address>**

Cause: The supervisor stack either was defined in or grew into a memory range mapped as ROM.

Action: Reset the emulator. Then, define the supervisor stack pointer within a memory range mapped as emulation or target RAM and allow sufficient room for the stack to grow as procedures are activated and deactivated.

145 **BERR occurred during background operation**

Cause: A bus error was encountered while the emulator was executing the background monitor.

Action: Reset the emulator.

146 **BERR during background access to supervisor stack**

Cause: A bus error occurred while the emulation monitor was attempting to push or pop data on the supervisor stack.

Action: Define the supervisor stack pointer within a memory range mapped as emulation or target RAM and allow sufficient room for the stack to grow as procedures are activated and deactivated.

147 **RESET during background operation**

Cause: A target system RESET occurred while the emulator was executing in the background monitor.

Action: Verify the register state is as expected, if so, you may continue with no further action. If not, reset the emulator from the emulation system (**rst**).

149 **Unexpected stack format on background entry**

Cause: The stack format word was different from that expected upon normal background entry.

Action: Set up the desired register states; or, reset the emulator.

156 **Coverage not supported**

Cause: You attempted to use the **cov** command for an emulator that does not provide coverage memory.

161 **Copy target image not supported**

Cause: You attempted to use the **cim** command for an emulator that does not support the command.

General Emulator and System Messages

201 **Out of system memory**

Cause: Macros and equates that you have defined have used all of the available system memory.

Action: Delete some of the existing macros (**mac -d <NAME>**) and equates (**equ -d <NAME>**), which will free additional memory.

204 **FATAL SYSTEM SOFTWARE ERROR**

205 **FATAL SYSTEM SOFTWARE ERROR**

208 **FATAL SYSTEM SOFTWARE ERROR**

Cause: The system has encountered an error from which it cannot recover.

Action: Write down the sequence of commands which caused the error. Cycle power on the emulator and reenter the commands. If the error repeats, call your local HP Sales and Service office for assistance.

206 **Incompatible compatibility table entry**

Cause: The emulation firmware (ROM) is not compatible with the analysis or system firmware in your HP 64700 system.

Action: The ROMs in your emulator must be compatible with each other for your emulation system to work correctly. Contact your Hewlett-Packard Representative.

300 **Invalid option or operand**

305 **Invalid option or operand: %s**

Cause: You have specified incorrect option(s) to a command. %s, if printed, indicates the incorrect option(s).

Action: Reenter the command with the correct syntax. Refer to the on-line help information.

307 **Invalid expression: %s**

Cause: You have entered an expression with incorrect syntax; therefore, it cannot be evaluated. %s is the bad expression.

Action: Reenter the expression, following the syntax rules for that type of expression. Refer to the command description to determine the expression type; then refer to the expression syntax pages to determine the correct syntax for that type.

308 **Invalid number of arguments**

Cause: You have either entered too many options to a command or an insufficient number of options.

Action: Re-enter the command with correct syntax. Refer to the command syntax pages in this manual for information.

310 **Invalid address: %s**

Cause: You specified an invalid address value as an argument to a command (other than an analyzer command). For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number (even zero (0)).

Action: Re-enter the command and the address specification. See the <ADDRESS> and <EXPRESSION> syntax pages in the "Commands" chapter for information on address specifications.

311 **Invalid address range: %s**

Cause: You specified an invalid address range as an argument to a command (other than an analyzer command). For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number, or the upper boundary of the range you specified is less than the lower boundary.

Action: Re-enter the command and the address specification. See the <ADDRESS> and <EXPRESSION> syntax pages in the "Commands" chapter for information on address specifications. Also, make sure that the upper boundary specification is greater than the lower boundary specification (the lower boundary must always precede the upper boundary on the command line).

312 **Ambiguous address: %s**

Cause: Certain emulators support segmentation or function code information in addressing. The emulator is unable to determine which of two or more address ranges you are referring to, based upon the information you entered.

Action: Re-enter the command and fully specify the address, including segmentation or function code information.

313 **Missing option or operand**

Cause: You have omitted a required option to the command.

Action: Re-enter the command with the correct syntax. Refer to the "Commands" chapter for further information on required syntax.

314 **Option conflict: %s**

Cause: You have entered a command with two options which cannot be used together. For example, you might have entered **tl -bx**; you cannot ask for both a binary and hexadecimal trace list dump.

Action: Reenter the command, specifying only non-conflicting options. Refer to the command description to determine which options may be used together.

315 **Invalid count: %s**

Cause: This error occurs when the emulation system expects a certain number (of arguments, for example), but you specify a different number.

Action: Enter the number the system expects to receive.

316 **Invalid range expression: %s**

Cause: In the **tl** command, you specified an illegal range. For example, you might have specified **tl -10..a**.

Action: Use only legitimate range numbers in the **tl** command (-1024..1023); the second range value must be greater than the first.

317 **Range out of bounds: %s**

Cause: In the **tl** command, you specified a range number which was greater than the number of states available in the analyzer. For example, you might have specified **tl -2048..2048**; the analyzer only has 1024 states.

Action: Specify range numbers between -1024 and 1023.

318 **Count out of bounds: %s**

Cause: You specified an occurrence count less than 1 or greater than 65535 for **tg** or **tif**. For example, you might have entered **tif 1 any 2 69234**.

Action: Re-enter the command, specifying a count value from 1 to 65535. For example: **tif 1 any 2 65535**.

319 **Invalid base: %s**

Cause: This error occurs if you have specified an invalid base in the **tf** or **xtf** commands.

Action: Enter the **help tf** or **help xtf** command to view the valid base options.

320 **Invalid label: %s**

Cause: You tried to define a label with characters other than letters, digits, or underscores.

Action: Re-enter the **tlb** command with a label consisting only of letters, digits, or underscores.

321 **Label not defined: %s**

Cause: You entered an analyzer expression in which the label was not present in the analyzer label list. For example, if the label list includes **addr**, **data**, and **stat**, you might have entered something such as **tg lowerdata=24t**. This error also occurs if you try to delete a label that does not exist.

Action: You can re-enter the command, using one of the previously defined labels and adjust the expression as necessary to accommodate the fit of that label to the analyzer input lines. Or, you can define a new label using the **tlb** command, then re-enter the analyzer command using the newly defined label.

400 **Record checksum failure**

Cause: During a **transfer** operation, the checksum specified in a file did not agree with that calculated by the HP 64700.

Action: Retry the **transfer** operation. If the failure is repeated, make sure that both your host and the HP 64700 data communications parameters are configured correctly.

401 **Records expected: %s; records received: %s**

Cause: The HP 64700 received a different number of records than it expected to receive during a **transfer** operation.

Action: Retry the **transfer**. If the failure is repeated, make sure that the data communications parameters are set correctly on the host and on the HP 64700. Refer to the "Installation" chapter for details.

410 **File transfer aborted**

Cause: A **transfer** operation was aborted due to a break received, most likely a <CTRL>c from the keyboard.

Action: If you typed <CTRL>c, you probably did so because you thought the transfer was about to fail. Retry the transfer, making sure to use the correct command options. If you are unsuccessful, make sure that the data communications parameters are set correctly on the host and on the HP 64700, then retry the operation.

411 **Severe error detected, file transfer failed**

Cause: An unrecoverable error occurred during a **transfer** operation.

Action: Retry the transfer. If it fails again, make sure that the data communications parameters are set correctly on the host and on the HP 64700. Also make sure that you are using the correct command options, both on the HP 64700 and on the host.

412 **Retry limit exceeded, transfer failed**

Cause: The limit for repeated attempts to send a record during a **transfer** operation was exceeded, therefore the transfer was aborted.

Action: Retry the transfer. Make sure you are using the correct command options for both the host and the HP 64700. The data communications parameters need to be set correctly for both devices. Also, if you are in a remote location from the host, it is possible that line noise may cause the failure.

413 **Transfer failed to start**

Cause: Communication link or transfer protocal incorrect.

Action: Check link and transfer options.

415 **Timeout, receiver failed to respond**

Cause: Communication link or transfer protocol incorrect.

Action: Check link and transfer options.

420 **Unknown mode: %s**

Cause: This error occurs when you have specified an unknown option in the **stty** command.

Action: Enter the **help stty** command to view the valid options.

425 **Load option conflict: %s and option: %s**

Cause: Two or more options in the load command cannot be used together.

Action: Enter the **help load** command to view the options that cannot be used together.

520 **Equate not defined: %s**

Cause: You tried to delete an equate that did not exist in the equate table. For example suppose the equates **a=1** and **b=2** were in the equate table. If you typed **equ -d c**, you would receive the above error message.

Action: Use **equ** to display the list of named equates before deleting equates.

600 **Adjust PC failed during break**

Cause: System failure or target condition.

Action: Run performance verification (**pv** command), and check target system.

602 **Break failed**

Cause: The **b** command was unable to break the emulator to the monitor.

Action: Determine why the break failed, then correct the condition and retry the command. See message 608.

603 **Read PC failed during break**

Cause: System failure or target condition.

Action: Try again.

604 **Disable breakpoint failed: %s**

Cause: System failure or target condition.

Action: Run performance verification (**pv** command), and check target system.

605 **Undefined software breakpoint: %s**

Cause: The emulator has encountered a software breakpoint in your program that was not inserted with the **bp** command.

Action: If your processor allows different software breakpoint instructions, either modify the ones you inserted in your code, or modify the ones inserted by **bp** using your emulator's configuration options (**cf** command). If only one instruction is available, remove those inserted in your code before assembly and link, then reinsert them using the **bp** command.

606 **Unable to run after CMB break**

Cause: System failure or target condition.

Action: Run performance verification (**pv** command), and check target system.

608 **Unable to break**

Cause: This message is displayed if the emulator is unable to break to the monitor because the emulation processor is reset, halted, or is otherwise disabled.

Action: First, look at the emulation prompt and other status messages displayed to determine why the processor is stopped. If reset by the emulation controller, use the **b** command to break to the monitor. If reset by the emulation system, release that reset. If halted, try **rst -m** to get to the monitor. If there is a bus grant, wait for the requesting device to release the bus before retrying the command. If there is no clock input, perhaps your target system is faulty. It's also possible that you have configured the emulator to restrict to real time runs, which will prohibit temporary breaks to the monitor.

610 **Unable to run**

Cause: System failure or target condition.

Action: Run performance verification (**pv** command), and check target system.

611 **Break caused by CMB not ready**

Cause: This status message is printed during coordinated measurements if the CMB READY line goes false. The emulator breaks to the monitor. When CMB READY is false, it indicates that one or more of the instruments participating in the measurement is running in the monitor.

Action: None, information only.

612 **Write to ROM break**

Cause: This status message will be printed if you have set **bc -e rom** and the emulation processor attempted a write to a memory location mapped as ROM.

Action: None (except troubleshooting your program).

613 **Analyzer Break**

Cause: Status message.

Action: None.

614 **Guarded memory access break**

Cause: This message is displayed if the emulation processor attempts to read or write memory mapped as guarded.

Action: Troubleshoot your program; or, you may have mapped memory incorrectly.

615 **Software breakpoint: %s**

Cause: This status message will be displayed if a software breakpoint entered with **bp** and enabled with **bc -e bp** is encountered during a program run. The emulator is broken to the monitor. The string %s indicates the address where the breakpoint was encountered.

Action: None.

616 **BNC trigger break**

Cause: This status message will be displayed if you have set **bc -e bnct** and the BNC trigger line is activated during a program run. The emulator is broken to the monitor.

Action: None.

617 **CMB trigger break**

Cause: This status message will be displayed if you have set **bc -e cmbt** and the CMB trigger line is activated during a program run. The emulator is broken to the monitor.

Action: None.

618 **trig1 break**

Cause: This status message will be displayed if you have set the analyzer to drive **trig1** upon finding the trigger, **bc -e trig1** is set, and the analyzer has found the trigger condition while tracing a program run. The emulator is broken to the monitor.

Action: None.

619 **trig2 break**

Cause: This status message will be displayed if you have set the analyzer to drive **trig2** upon finding the trigger, **bc -e trig2** is set, and the analyzer has found the trigger condition while tracing a program run. The emulator is broken to the monitor.

Action: None.

620 **Unexpected software breakpoint**

Cause: If you have enabled software breakpoints with **bc -e bp**, this message is displayed if a software breakpoint instruction is encountered in your program that was not inserted by **bp** and is therefore not in the breakpoint table.

Action: If your processor allows different software breakpoint instructions, either modify the ones you inserted in your code, or modify the ones inserted by **bp** using your emulator's configuration options (**cf** command). If only one instruction is available, remove those inserted in your code before assembly and link, then reinsert them using the **bp** command.

621 **Unexpected step break**

Cause: System failure.

Action: Run performance verification (**pv** command).

622 **%s**

Cause: Monitor specific message.

Action: None.

623 **CMB execute break**

Cause: This message occurs when coordinated measurements are enabled and an EXECUTE pulse causes the emulator to run; the emulator must break before running.

Action: This is a status message; no action is required.

624 **Configuration aborted**

Cause: Occurs when a <CTRL>c is entered during **cf** display command.

Action: None.

625 **Invalid configuration value: %s**

Cause: You have entered a configuration option incorrectly, such as typing **cf rrt=onn** instead of **cf rrt=on**.

Action: Re-enter the configuration command, specifying only the correct options. Enter the **help cf** command for a description of the configuration options for your emulator.

626 **Configuration failed; setting unknown: %s=%s**

Cause: Target condition or system failure.

Action: Check target system, and run performance verification (**pv** command).

627 **Invalid configuration item: %s**

Cause: You specified a non-existent configuration item in the **cf** command. For example, you would see this message if you tried to enter **cf clk=int** since there is no **clk** configuration item for the emulator.

Action: Re-enter the command, specifying only configuration items that are supported by your emulator. Enter the **help cf** command for a description of the configuration options for your emulator.

628 **Guarded memory break: %s"**

Cause: A memory access to a location mapped as guarded memory has occurred during execution of the user program.

Action: Investigate the cause of the guarded memory access by the user program.

628 **Write to ROM break: %s"**

Cause: When the **rom** break condition is enabled, a memory write access to a location mapped as ROM has occurred during execution of the user program.

Action: Investigate the cause of the write to ROM by the user program. You can disable the break condition with the **bc -d rom** command.

630 **Register access aborted**

Cause: Occurs when a <CTRL>c is entered during register display.

Action: None.

631 **Unable to read registers in class: %s**

Cause: The emulator was unable to read the registers you requested.

Action: To resolve this, you must look at the other status messages displayed. Most likely, the emulator was unable to break to the monitor to perform the register read. See message 608.

632 **Unable to modify register: %s=%s**

Cause: The emulator was unable to modify the register you requested.

Action: To resolve this, you must look at the other status messages displayed. It's likely that emulator was unable to break to the monitor to perform the register modification. See message 608.

634 **Display register failed: %s**

Cause: The emulator was unable to display the register you requested.

Action: To resolve this, you must look at the other status messages displayed. It's likely that emulator was unable to break to the monitor to perform the register display. See message 608.

636 **Register not writable: %s**

Cause: This error occurs when you attempt to modify a read only register.

Action: If this error occurs, you cannot modify the contents of the register with the **reg** command.

637 **Register class cannot be modified: %s**

Cause: You tried to modify a register class instead of an individual register.

Action: You can only modify individual registers. Refer to the **reg** command description for a list of register names.

640 **Unable to reset**

Cause: Target condition or system failure.

Action: Check target system, and run performance verification (**pv** command).

641 **Unable to reset into monitor**

Cause: You have entered a **rst -m** command and the emulator is unable to break into the monitor.

Action: Reload monitor (**rst** for background).

650 **Unable to configure break on write to ROM**

Cause: The emulator controller is unable to execute the **bc -e rom** command correctly, possibly because the emulator was left in an unknown state or because of a hardware failure.

Action: Initialize the emulator or cycle power. Then reenter the command. If the same failure occurs, call your HP sales and service office.

651 **Unable to configure break on software breakpoints**

Cause: The emulator controller is unable to execute the **bc -e bp** command, possibly because the emulator is in an unknown state or because of a hardware failure.

Action: Initialize the emulator or cycle power, then re-enter the command. If the same failure occurs, call your HP sales and service office.

652 **Break condition must be specified**

Cause: You entered **bc -e** or **bc -d** without specifying a break condition to enable or disable.

Action: Re-enter the **bc** command along with the enable/disable flag and the break condition you wish to modify.

653 **Break condition configuration aborted**

Cause: Occurs when <CTRL>c is entered during **bc** display.

Action: None.

661 **Software breakpoint break condition is disabled**

Cause: You entered the **bp** command and options; however, the software breakpoint break condition is disabled.

Action: Enable the software breakpoint feature with **bc -e bp**, then enter the desired breakpoints with **bp**.

663 **Specified breakpoint not in list: %s**

Cause: You tried to enable a software breakpoint (**bp -e <ADDRESS>**) that was not previously defined. The string %s prints the address of the breakpoint you attempted to enable.

Action: Insert the breakpoint into the table and memory by typing **bp <ADDRESS>**.

664 **Breakpoint list full; not added: %s**

Cause: The software breakpoint table is full. The breakpoint you just requested, with address %s, was not inserted.

Action: Remove breakpoints that are no longer in use with **bp -r <ADDRESS>**. Then insert the new breakpoint.

665 **Enable breakpoint failed: %s**

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

666 **Disable breakpoint failed: %s**

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

667 **Breakpoint code already exists: %s**

Cause: You attempted to insert a breakpoint with **bp <ADDRESS>**; however, there was already a software breakpoint instruction at that location which was not already in the breakpoint table.

Action: Your program code is apparently using the same breakpoint instruction as **bp**. If multiple breakpoint instructions are available on your processor, either change those in your program code or modify the one **bp** uses with your emulator's configuration options (**cf** command). If only one instruction is available, remove the breakpoints from your program code and use **bp** to insert breakpoints.

668 **Breakpoint not added: %s**

Cause: You tried to insert a breakpoint in a memory location which was not mapped or was mapped as guarded memory.

Action: Insert breakpoints only within memory ranges mapped to emulation or target RAM or ROM.

669 **Breakpoint remove aborted**

Cause: Occurs when <CTRL>c is entered during a **bp -r** command.

Action: None.

670 **Breakpoint enable aborted**

Cause: Occurs when <CTRL>c is entered during a **bp -e** command.

Action: None.

671 **Breakpoint disable aborted**

Cause: Occurs when <CTRL>c is entered during a **bp -d** command.

Action: None.

680 **Stepping failed**

Cause: Stepping has failed for some reason.

Action: Usually, this error message will occur with other error messages. Refer to the descriptions of the accompanying error messages to find out more about why stepping failed.

682 **Invalid step count: %s**

Cause: You specified an non-cardinal value for a step count in the **s** command (such as entering **s 22.1**).

Action: Reenter the step command, using only cardinal values (positive integers) for the step count.

684 **Failed to disable step mode**

Cause: System failure.

Action: Run performance verification (**pv** command).

685 **Stepping aborted**

Cause: This message is displayed if a break was received during a **s** (step) command with a stepcount of zero (0). The break could have been due to any of the break conditions in **bc** or a <CTRL>c break.

Action: None.

686 **Stepping aborted; number steps completed: %d**

Cause: This message is displayed if a break was received during a **s** (step) command with a stepcount greater than zero. The break could have been due to any of the break conditions in **bc** or a <CTRL>c break. The number of steps completed is displayed.

Action: None.

688 **Step display failed**

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

689 **Break due to cause other than step**

Cause: An activity other than a **step** command caused the emulator to break. This could include any of the break conditions in a **bc** command or a <CTRL>c break.

Action: None.

692 **Trace error during CMB execute**

Cause: System failure.

Action: Run performance verification (**pv** command).

693 **CMB execute; run started**

Cause: This status message is displayed when you are making coordinated measurements. The CMB /EXECUTE pulse has been received; the emulation processor started running at the address specified by the **rx** command.

Action: None; information only.

694 **Run failed during CMB execute**

Cause: System failure or target condition.

Action: Run performance verification (**pv** command), and check target system.

700 **Target memory access failed**

Cause: This message is displayed if the emulator was unable to perform the requested operation on memory mapped to the target system.

Action: In most cases, the problem results from the emulator's inability to break to the monitor to perform the operation. See message 608.

702 **Emulation memory access failed**

Cause: System failure.

Action: Run performance verification (**pv** command).

707 **Request access to guarded memory: %s**

Cause: The address or address range specified in the command included addresses within a range mapped as guarded memory. When the emulator attempts to access these during command processing, the above message is printed, along with the specific address or addresses accessed.

Action: Re-enter the command and specify only addresses or address ranges within emulation or target RAM or ROM. Or, you can remap memory so that the desired addresses are no longer mapped as guarded.

710 **Memory range overflow**

Cause: Accessing a word or short word, for example "**m -dw 0ffffffff**" will cause a rounding error that overflows physical memory.

Action: Reduce memory display request.

720 **Invalid map term number: %s**

Cause: You attempted to delete a mapper term that does not exist. For example, you may have tried **map -d 17** (there are a maximum of 16 mapper terms). Or you may have tried **map -d 2**, when only one mapper term has been defined.

Action: Use the **map** command to determine the numbers of the terms currently mapped. Then delete the appropriate mapper term.

721 **No map terms available; maximum number already defined**

Cause: You tried to add more than 16 mapper terms.

Action: Either combine map ranges to conserve on the number of terms or delete mapper terms that aren't needed to free another mapper term.

723 **Invalid map address range: %s**

Cause: You specified an invalid address range as an argument to the **map** command. For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number, or the upper boundary of the range you specified is less than the lower boundary.

Action: Re-enter the **map** command and the address specification. See the <ADDRESS> and <EXPRESSION> syntax pages in the "Commands" chapter for information on address specifications. Also, make sure that the upper boundary specification is greater than the lower boundary specification (the lower boundary must always precede the upper boundary on the command line).

725 **Unable to load new memory map; old map reloaded**

Cause: There is not enough emulation memory left for this request.

Action: Reduce the amount of emulation memory requested.

726 **Unable to reload old memory map; hardware state unknown**

Cause: System failure.

Action: Run performance verification (**pv** command).

730 **Invalid memory map type: %s**

Cause: You specified a memory type while mapping that is not one of the supported types: **eram**, **erom**, **tram**, **trom**, or **grd**.

Action: Re-enter the **map** command, specifying only one of the five types listed above.

731 **Invalid memory map attribute: %s**

Cause: You have entered an unknown attribute when mapping a range of memory.

Action: Only the **dti** attribute is available, and they are only valid for emulation memory ranges.

732 **Invalid memory type for ’other’ range: %s**

Cause: The unmapped memory type must be **tram**, **trom**, or **grd**. If you see the above message, you have tried to map the "other" range to **eram** or **erom**.

Action: Map the "other" range to **tram**, **trom**, or **grd**.

734 **Map range overlaps with term: %d**

Cause: You entered a map term whose address range overlaps with one already mapped. For example, you may have entered a term **map 1000..2fff eram**, then tried to enter a term **map 2000..3fff erom**.

Action: Re-enter the map term so that ranges do not overlap, or combine terms and change the memory type.

736 **Memory not mapped as emulation: %s**

Cause: This error occurs when a feature available only for emulation memory is attempted with target memory. For example, this error occurs when you attempt to perform coverage measurements (see the **cov** command) on target memory.

Action: You must remap the address range as emulation memory.

738 **Unable to reset coverage bit data**

Cause: System failure.

Action: Run performance verification (**pv** command).

740 **I/O port access failed**

Cause: The emulator was unable to read or write the port specified in the **io** command. This message is also printed if your processor does not support separate I/O.

Action: If your processor does not support separate I/O, use the **m** command to modify I/O ports. Otherwise, retry the operation, and make sure that you are specifying a valid I/O address.

752 **Copy memory aborted; next destination: %s**

754 **Memory modify aborted; next address: %s**

756 **Memory search aborted; next address: %s**

Cause: One of these message is displayed if a break occurs during processing of the **cp**, **m**, or **ser** commands, respectively. The break could result from any of the break conditions (except **bp**) or could have resulted from a <CTRL>c break.

Action: Retry the operation. If breaks are occurring continuously, you may wish to disable some of the break conditions with the **bc** command.

800 **Invalid command: %s**

Cause: You have entered a command which is not part of the standard Terminal Interface command set (documented in this manual) and was not found in the currently defined macros.

Action: Enter only commands defined in this manual or in the macro set. You can display the macro set using **mac**. You can rename commands or name command groups using the **mac** command.

801 **Invalid command group: %s**

Cause: This error occurs when you specify an invalid group name in the **help -s <group>** command.

Action: Enter the **help** command with no options for a listing of the valid group names.

802 **Invalid command format**

Cause: This error occurs when an invalid macro is entered, for example, **mac {help;{}**.

Action: Refer to the **mac** command description.

807 **Macro list full; macro not added**

Cause: The maximum number of macros have been defined.

Action: You must delete macros before adding any new macros.

809 **Macro buffer full; macro not added**

Cause: This error occurs when the memory reserved for macros is all used up.

Action: You must delete macros to reclaim memory in the macro buffer.

812 **Invalid macro name: %s**

Cause: You tried to delete a macro that did not exist; or you tried to define a new macro with a name containing characters other than letters, digits, or underscores.

Action: Use the **mac** command to display the names of macros in the macro table before deleting them with **mac -d <NAME>**. Define new macro names using only letters, digits, and underscore characters.

813 **Command line too long; maximum line length: %d**

Cause: This error occurs when the command line exceeds the maximum number of characters.

Action: Split the command line into two command lines.

814 **Command line too complex**

Cause: There was not enough memory for the expressions in the command line.

Action: Split up the command line, or use fewer expressions.

815 **Missing macro parameter: %s**

Cause: This error occurred because you did not include a parameter with the specified **mac** command for macro expansion.

Action: Enter the command again, and include the appropriate parameter for the macro expansion.

816 **Command line too complex**

Cause: Too many expression operators are used.

Action: Split up the command line, or use fewer expressions.

818 **Command line too complex**

Cause: A maximum nesting level has been exceeded for nested command execution.

Action: Reduce the number of nesting levels.

820 **Unmatched quote encountered**

Cause: In entering a string, such as with the **echo** command, you didn’t properly match the string delimiters (either **‘‘** or **""**). For example, you might have entered

echo "set S1 to off

Action: Re-enter the command and string, making sure to properly match opening and closing delimiters. Note that both delimiters must be the same character. For example: **echo "set S1 to off"**.

822 **Unmatched command group encountered**

Cause: You entered the **mac** or **rep** command group without matching braces **{}**. For example: **mac test={rst -m;cf** or **rep 2 {rst -m;map**.

Action: Re-enter the command, making sure to match braces around commands you want grouped into the macro or repeat. For example: **mac test={rst -m;cf}**.

824 **Maximum number of arguments exceeded**

Cause: Exceeding the limit of 100 arguments per command.

Action: Reduce the number of arguments in the command.

826 **Maximum argument buffer space exceeded**

Cause: Exceeding space limits for argument lists.

Action: Reduce request.

840 **Invalid date: %s**

Cause: You have specified the date format incorrectly in the **dt** command.

Action: Re-enter the command with the correct date format. Refer to the **dt** command description for the correct format.

842 **Invalid time: %s**

Cause: You have incorrectly specified the time format in the **dt** command.

Action: Re-enter the command with the correct time format. Refer to the **dt** command description for the correct format.

844 **Invalid repeat count: %s**

Cause: You entered a non cardinal value for the repeat count in the **rep** command, such as **rep 22.1 <command_group>**.

Action: Re-enter the **rep** command, specifying only a cardinal number (positive integer) for the repeat count.

850 **Attempt to load code outside of allocated bounds**

Cause: This error occurs when the **lcd** command attempts to load an absolute file that contains code or data outside the range allocated for system code.

Action: Generally, you will not use the **lcd** command. The **lcd** command is intended to be used by high-level interfaces to the HP 64700.

875 **Invalid syntax for global or user symbol name: %s**

Cause: This error occurs when you enter a global or user symbol name with incorrect syntax.

Action: Make sure that you enter the global or user symbol name using the correct syntax. When specifying a global symbol, make sure that you precede the global symbol with a colon (for example, :glb_sym). When specifying a user symbol (created with the **sym** command), make sure that you enter the name correctly without a colon.

876 **Invalid syntax for local symbol or module: %s**

Cause: This error occurs when you enter a local symbol or module name with incorrect syntax.

Action: When entering a local symbol name using the **sym** command, make sure that you specify the module name, followed by a colon, then the symbol name (for example module:loc_sym). Make sure that you specify the module name correctly.

877 **Symbol not found: %s**

Cause: This occurs when you try to enter a symbol name that doesn't exist.

Action: Enter a valid symbol name.

878 **Symbol cannot contain wildcard in this context**

Cause: You tried to enter a global, local, or user symbol name using the wildcard (*) incorrectly.

Action: When you enter the symbol name again, include the wildcard (*) at the end of the symbol.

879 **Symbol cannot contain text after the wildcard**

Cause: You tried to include text after the wildcard specified in the symbol name (for example, sym*text).

Action: Enter the symbol again, but don't include text after the wildcard (*).

880 **Conflict between expected and received symbol information**

Cause: The information you supplied in a symbol definition is not what the HP 64700 expected to receive.

Action: Make sure that all symbols in the symbol file are defined correctly. Verify that there are no spaces in the address definitions for the symbols in the symbol file being downloaded.

881 **Ascii symbol download failed**

Cause: This error occurs because the system is out of memory.

Action: You must either reduce the number of symbols to be loaded, or free up additional system space and try the download again.

882 **No module specified for local symbol**

Cause: This error occurs because you tried to specify a local symbol name without specifying the module name where the symbol is located.

Action: Enter the module name where the local symbol is located, followed by a colon, then the local symbol name.

901 **Invalid firmware for emulation subsystem**

Cause: This error occurs when the HP 64700 system controller determines that the emulation firmware (ROM) is invalid.

Action: This message is not likely to occur unless you have upgraded the ROMs in your emulator. Be sure that the correct ROM is installed in the emulation controller.

902 **Invalid analysis subsystem; product address: %s**

Cause: This error occurs when the HP 64700 system controller determines that the analysis firmware (ROM) is invalid.

Action: This message is not likely to occur unless you have upgraded the ROMs in your emulator. Be sure that the correct ROMs are installed in the analyzer board.

903 **Invalid ET subsystem; product address: %s**

Cause: Detects an invalid ET. Used only internally.

Action: None.

904 **Invalid auxiliary subsystem; product address: %s**

Cause: For future products.

Action: None.

911 **Lab firmware for emulation subsystem**

Cause: This message should never occur. It shows that you have an unreleased version of emulation firmware.

Action: None.

912 **Lab firmware analysis subsystem; product address: %s**

Cause: This message should never occur. It shows that you have an unreleased version of analysis firmware.

Action: None.

913 **Lab firmware subsystem; product address: %s**

Cause: This message should never occur. It shows that you have an unreleased version of system controller firmware.

Action: None.

914 **Lab firmware auxiliary subsystem; product address: %s**

Cause: This message should never occur. It shows that you have an unreleased firmware version of the auxiliary subsystem.

Action: None.

Analyzer Messages

1102 **Invalid bit range; crosses two multiples of 16: <sig#>..<sig#>**

Cause: This error occurs when defining trace labels. A trace label may not contain trace signals crossing two 16-bit boundaries. For example, the command "**tlb name 1..32**" will cause this error because "name" contains signals which cross the 15-16 and 31-32 16-bit boundaries.

Action: Redefine your trace label so that no more than one 16-bit boundary is crossed.

1103 **Invalid bit range; out of bounds: <sig#>..<sig#>**

Cause: This error occurs when defining trace labels, and you have attempted to assign non-existent trace signals to a label.

Action: Enter the trace activity command to view the trace signals present, and use only these signals when defining trace labels.

1104 **Invalid bit range; too wide: <sig#>..<sig#>**

Cause: This error occurs when defining trace labels, and you have attempted to assign more than 32 trace signals to a label.

Action: Use more than one trace label to define over 32 trace signals.

1105 **Unable to delete label; used by emulation analyzer: <label>**

Cause: This error occurs when you attempt to delete an emulation trace label which is currently being used as a qualifier in the emulation trace specification or is currently specified in the emulation trace format.

Action: Display the emulation trace sequencer specification in the easy configuration, display the emulation trace patterns in the complex configuration, or display the trace format to see where the label is used. Also, you should check **tpq** for uses of that label. You must change the pattern or format specification to remove the label before you can delete it.

1106 **Unable to delete label; used by external state analyzer: <label>**

Cause: This error occurs when you attempt to delete an external trace label which is currently being used as a qualifier in the external state trace specification or is currently specified in the external trace format.

Action: Display the external trace sequencer specification in the easy configuration, display the external trace patterns in the complex configuration, or display the external trace format to see where the label is used. Also, check **tpq** for uses of that label. You must change the pattern or format specification to remove the label before you can delete it.

1107 **Unable to delete label; used by external timing analyzer: <label>**

Cause: This error occurs when you attempt to delete an external trace label which is currently being used as a qualifier in the external timing trace specification.

Action: Remove the label from the external timing analyzer specifications, and then delete the label.

1108 **Unable to redefine label; used by emulation analyzer: <label>**

Cause: This error occurs when you attempt to redefine an emulation trace label which is currently used as a qualifier in the emulation trace specification.

Action: Display the emulation trace sequencer specification in the easy configuration, display the emulation trace patterns in the complex configuration, or display the emulation trace format to see where the label is used. You must change the pattern or format specification to remove the label before you can redefine it.

1109 **Unable to redefine label; used by external state analyzer: <label>**

Cause: This error occurs when you attempt to redefine an external trace label which is currently used as a qualifier in the external state trace specification.

Action: Display the external trace sequencer specification in the easy configuration, or display the external trace patterns in the complex configuration to see where the label is used. You must change the pattern or format specification to remove the label before you can redefine it.

1110 **Unable to redefine label; used by external timing analyzer: <label>**

Cause: This error occurs when you attempt to redefine an emulation or external trace label which is currently being used as a qualifier in the external timing trace specification.

Action: Remove the label from the external timing analyzer specifications, and then redefine the label.

1111 **Unable to redefine label; belongs to external analyzer: <label>**

Cause: This error occurs when you attempt to redefine an external analyzer label with the emulation trace label command (for example, tlb xbits 0..16).

Action: Either use a different label name, or delete the external analyzer label before defining a label of the same name for the emulation analyzer.

1112 **Unable to redefine label; belongs to emulation analyzer: <label>**

Cause: This error occurs when you attempt to redefine an emulation analyzer label with the external trace label command (for example, xtlb addr 0..19).

Action: Either use a different label name, or delete the emulation analyzer label before defining a label of the same name for the external analyzer.

1114 **Label belongs to external analyzer: <label>**

Cause: When the external analyzer is in an independent mode, this error occurs when you attempt to use an external analyzer label in an emulation trace command (for example, tg xlabel=0).

Action: Only use external trace labels in external trace commands (when the external analyzer is in an independent mode).

1115 **Label belongs to emulation analyzer: <label>**

Cause: When the external analyzer is in an independent mode, this error occurs when you attempt to use an emulation analyzer label in an external trace command (for example, xtg addr=5).

Action: Only use emulation trace labels in emulation trace commands (when the external analyzer is in an independent mode).

1130 **Illegal base for count display**

Cause: When specifying the trace format, counts may only be displayed relative or absolute. When counting states, the count is always displayed as a decimal number.

Action: Respecify the trace format without using a base for the count column. Also, you can use "**,A**" to specify that counts be displayed absolute, or you can use "**,R**" to specify that counts be displayed relative.

1131 **Illegal base for mnemonic disassembly display**

Cause: When specifying the trace format, you cannot specify a number base for the column containing mnemonic information.

Action: Respecify the trace format without using a base for the mnemonic column.

1132 **Illegal base for sequencer display**

Cause: When specifying the trace format, you cannot specify a number base for the column containing sequencer information.

Action: Respecify the trace format without using a base for the sequencer column.

1133 **Trace format command failed; using old format**

Cause: This error occurs when the trace format command fails for some reason. This error message always occurs with another error message.

Action: Refer to the "Action" description for the other error message displayed.

1137 **Mnemonic disassembly not supported for external trace**

Cause: This error occurs when you attempt to specify a mnemonic information column in the external trace format. There is no mnemonic disassembly for the external trace.

Action: Respecify the trace format without the mnemonic column.

1138 **Illegal width for symbol display: %s**

Cause: This error occurs when the value specified for the trace format address field width is not valid.

Action: Enter the **tf** command again, and specify the width of the address field for symbol display within the range of 4 to 55.

1139 **Illegal width for addr display, mne not specified**

Cause: This error occurs when you specify a width for the address field in the **tf** command, but do not include the **mne** option.

Action: Enter the command again, and include the **mne** option.

1140 **Symbol display unsupported**

Cause: This error occurs when you try to display symbols in the trace list, but the emulator you are using doesn't support symbols.

Action: Enter the **tl** command again, but don't try to display symbols.

1141 **Symbol display unavailable without mne field**

Cause: This error occurs when you try to display symbols, but have not included the **mne** option to the **tf** command.

Action: Don't try to display symbols unless the **mne** field has already been specified.

1202 **Trigger position out of bounds: <bounds>**

Cause: This error occurs when you attempt to specify a number of lines to appear either before or after the trigger which is greater than the number of lines allowed. The <bounds> string indicates the incorrect range that you typed (not the correct limits on the range).

Action: Be sure that the trigger position specified is within the range -1024 to 1023.

1207 **Invalid clock channel: <name>**

Cause: Valid clock channels are L, M, and N. If you have an external analyzer, the J and K channels are also valid.

Action: Respecify the command using valid clock channels.

1209 **Operator must be "and" or "or": <expression>**

Cause: When combining trace labels to specify trace patterns (in simple expressions or with the **tpat** command), an operator of either "and" or "or" must appear between the label qualifiers.

Action: Refer to the "Qualifying Trigger and Store Conditions" section of the "Using the Emulation Analyzer - Easy Configuration" chapter.

1210 **Illegal mix of = and !=**

Cause: When combining trace labels to specify patterns (in simple expressions or with the tpat command), all labels must either be equal to values or not equal to values.

Action: Refer to the "Qualifying Trigger and Store Conditions" section of the "Using the Emulation Analyzer - Easy Configuration" chapter.

1211 **Illegal mix of and/or**

Cause: When combining trace labels to specify patterns (in simple expressions or with the tpat command), all label qualifiers must either be ANDed together or ORed together. You cannot mix these operators.

Action: Refer to the "Qualifying Trigger and Store Conditions" section of the "Using the Emulation Analyzer - Easy Configuration" chapter.

1212 **Conflict with overlapping label: <label>**

Cause: When combining trace labels to specify patterns (in simple expressions or with the tpat command), you cannot combine labels which are defined for common trace signals. For example, the following easy configuration commands will result in this error: tlb low8 0..7; tlb low16 0..15; tg low8=0 and low16=1.

Action: Either omit one of the overlapping labels, or redefine your labels so that they do not contain common trace signals. You could also circumvent this error by using don't cares in the appropriate places; for the example shown in cause, you could specify patterns **tg low8=0xx0xY and low16=1**.

1213 **Illegal mix of !=/and**

Cause: When combining trace labels to specify patterns (in simple expressions or with the tpat command), labels which are not equal to values must be ORed together so that the entire pattern specifies a "not equals" condition.

Action: Refer to the "Qualifying Trigger and Store Conditions" section of the "Using the Emulation Analyzer - Easy Configuration" chapter.

1214 **Illegal mix of =/or**

Cause: When combining trace labels to specify patterns (in simple expressions or with the tpat command), labels which are equal to values must be ANDed together so that the entire pattern specifies an "equals" condition.

Action: Refer to the "Qualifying Trigger and Store Conditions" section of the "Using the Emulation Analyzer - Easy Configuration" chapter.

1215 **Comparator must be = or !=: <label>**

Cause: When combining trace labels to specify patterns (in simple expressions or with the tpat command), the value of the label can only be specified with the "=" or "!=" operators.

Action: Refer to the "Qualifying Trigger and Store Conditions" section of the "Using the Emulation Analyzer - Easy Configuration" chapter.

1217 **Illegal pattern name: <name>**

Cause: Valid pattern names are p1 through p8.

Action: Use only valid pattern names.

1218 **Illegal comparator for range qualifier: !=**

Cause: When specifying a range with the trng command, you cannot use the"!=" operator.

Action: Use the "!r" range name.

1219 **Range cannot be combined with any other qualifier**

Cause: For example, the following easy configuration command will result in this error: tsto addr=400..4ff and data=40.

Action: Do not attempt to combine labels when using range qualifiers.

1221 **Range resource in use**

Cause: This error occurs when you attempt to use two different range expressions in the "easy" configuration trace specification or when you attempt to redefine the "complex" configuration range resource while it is currently being used as a qualifier in the trace specification.

Action: Only one range expression may be used in the "easy" configuration trace specification. In the "complex" configuration, display the sequencer specification to see where the range resource is being used and remove it; then, you can redefine the range resource.

1224 **Sequence term number out of range: <term>**

Cause: This error occurs when a sequencer qualification command (**tif**, **telif**, **tsq**, or **tsto**) specifies a non-existent sequence term. The easy configuration sequencer may have a maximum of 4 sequence terms. Eight sequence terms exist in the complex configuration sequencer.

Action: Re-enter the command using an existing sequence term.

1225 **Sequence term not contiguous: <term>**

Cause: This error occurs when you attempt to insert a sequence term which is not between existing terms or after the last term. For example, the following easy configuration commands will result in this error: tg any; tsq -i 4.

Action: Be sure that the sequence term you enter is either between existing sequence terms or after the last sequence term.

1226 **Too many sequence terms**

Cause: This error occurs when you attempt to insert more than 4 sequence terms.

Action: Do not attempt to insert more than 4 sequence terms.

1227 **Sequence term not defined: <term>**

Cause: This error occurs when you attempt to delete, or specify a primary branch expression for, a sequence term number which is possible, but which is not currently defined.

Action: Insert the sequence term, and respecify the primary branch expression for that term.

1228 **One sequence term required**

Cause: This error occurs when you attempt to delete terms from the sequencer when only one term exists.

Action: At least one term must exist in the sequencer. Do not attempt to delete sequence terms when only one exists.

1234 **Invalid occurrence count: <number>**

Cause: Occurrence counts may be from 1 to 65535.

Action: Re-enter the command with a valid occurrence count.

1235 **Illegal threshold value: <value>**

Cause: Threshold voltage specifications may be from -6.4 V to +6.35 V in increments of 0.05 V.

Action: Re-enter the command with a valid threshold voltage.

1237 **Option specified more than once: <option>**

Cause: When specifying external threshold voltages, this error occurs when you attempt to specify the threshold voltage for either the upper or lower byte twice.

Action: You must re-enter the command so that the threshold voltage is only specified once for each option (upper or lower byte).

1239 **Clock speed not available with current count qualifier.**

Cause: This error occurs when you attempt to specify a fast (F) or very fast (VF) maximum qualified clock speed when counting time (tcq time). This error also occurs when you attempt to specify a very fast (VF) maximum qualified clock speed when counting states (for example, tcq addr=400).

Action: Change the count qualifier; then, re-enter the command.

1240 **Count qualifier not available with current clock speed.**

Cause: This error occurs when you attempt to specify the "time" count qualifier when the current maximum qualified clock speed is fast (F) or very fast (VF). This error also occurs when you attempt to specify a "state" count qualifier when the maximum qualified clock speed is fast (F).

Action: Change the clock speed; then, change the count qualifier.

1241 **Invalid qualifier resource or operator: <expression>**

Cause: When specifying complex expressions, you have either specified an illegal pattern or used an illegal operator.

Action: Refer to the "Using Complex Expressions" section of the "Using the Emulation Analyzer - Complex Configuration" chapter for information on valid patterns and operators.

1245 **Range qualifier not accessible in easy configuration**

Cause: This error occurs when you attempt to use the **trng** command in the easy configuration.

Action: Changing into the complex configuration will allow you to use the **trng** command; otherwise, specify the range in easy configuration command expressions.

1246 **Pattern qualifiers not accessible in easy configuration**

Cause: This error occurs when you attempt to use the **tpat** command in the easy configuration.

Action: Changing into the complex configuration will allow you to use the **tpat** command; otherwise, specify the patterns in easy configuration command expressions.

1248 **Range term used more than once**

Cause: This error occurs when you attempt to use the range resource more than once in a sequencer branch expression.

Action: You cannot use the range resource more than once in a sequencer branch expression.

1249 **Invalid qualifier expression: <expression>**

Cause: This error message is shown with the errors that occur when patterns, the range, or the arm condition is used more than once within a set. This error message also occurs when intraset operators are not the same. For example, the following complex expression will result in this error: p1 ~ p2 | p3.

Action: Refer to the "Using Complex Expressions" section of the "Using the Emulation Analyzer - Complex Configuration" chapter for information on valid patterns and operators.

1250 **Arm term used more than once**

Cause: This error occurs when you attempt to use the "arm" qualifier more than once in a sequencer branch expression.

Action: You cannot use the "arm" qualifier more than once in a sequencer branch expression.

1251 **Trigger term cannot be term 1**

Cause: This error occurs when to attempt to specify the first sequence term as the trigger term. The trigger term may be any term but the first.

Action: Respecify the trigger term as any other sequence term.

1253 **Invalid pod number: <pod#>**

Cause: This error message occurs when you attempt to specify a slave clock for a non-existent analyzer pod.

Action: Use the trace activity command to display the valid pod numbers, and use only these numbers when entering commands.

1257 **Pod belongs to external analyzer: <pod#>**

Cause: This error occurs when you attempt to specify a slave clock for the external analyzer pod with the emulation analyzer's trace slave clock command. This error only occurs when the external analyzer is in its independent state mode.

Action: Use the external trace slave clock command to specify a slave clock for the external analyzer pod.

1300 **Incompatible external trace mode**

Cause: This error message occurs when you attempt to use an external trace command (other than **xtv**, **xtlb**, or **xtmo**) while the external analyzer is aligned with the emulation analyzer. The message is also display if you attempt to use external state trace commands when the external analyzer is in timing mode; or if you attempt to use external timing trace commands when the external analyzer is in state mode.

Action: Change the external trace mode, and re-enter the command.

1301 **External label in use: <label>**

Cause: This error occurs when you attempt to select the external analyzer's independent state mode while an external trace label is currently used as a qualifier in the emulation analyzer trace specification.

Action: Remove any external trace label qualifiers from emulation trace specifications before selecting the external analyzer's independent state mode.

1302 **Trig1 signal cannot be driven and received**

Cause: This error occurs when you attempt to specify the internal trig1 signal as the trace arm condition while the same analyzer's trigger output is currently driving the trig1 signal. This error also occurs if you attempt to specify that the trigger output drive the internal trig1 signal while that signal is currently specified as the arm condition for the same analyzer.

Action: You can either change the arm or the trigger output specification; in either case, make sure that they do not use the same internal signal.

1303 **Trig2 signal cannot be driven and received**

Cause: This error occurs when you attempt to specify the internal trig2 signal as the trace arm condition while the same analyzer's trigger output is currently driving the trig2 signal. This error also occurs if you attempt to specify that the trigger output drive the internal trig2 signal while that signal is currently specified as the arm condition for the same analyzer.

Action: You can either change the arm or the trigger output specification; in either case, make sure that they do not use the same internal signal.

1304 **Analyzer trace running**

Cause: This error occurs when you attempt to change the external analyzer mode while a trace is in progress.

Action: Halt the trace before changing the external analyzer mode.

1305 **CMB execute; emulation trace started**

Cause: This status message informs you that an emulation trace measurement has started as a result of a CMB execute signal (as specified by the "**tx -e**" command).

1306 **CMB execute; external trace started**

Cause: This status message informs you that an emulation trace measurement has started as a result of a CMB execute signal (as specified by the "**xtx -e**" command).

2021 **Period not in 1/2/5 sequence: <period>**

Cause: This error message occurs when the external timing sample period is not in a 1/2/5 sequence; for example, 10ns, 20ns, 50ns, 100ns, 200ns, 500ns, 1us, 2us, 5us, etc. Some examples of invalid sample period specifications are: 12ns, 18ns, 25ns, 60ns, 80ns, etc.

Action: Use a number in the 1/2/5 sequence when specifying the external timing sample period.

2022 **Sample period out of bounds: <bounds>**

Cause: The external timing sample period must be between 10 ns and 50 ms (in a 1/2/5 sequence).

Action: Re-enter the command with the sample period between the bounds shown.

2030 **Negated patterns not allowed in timing**

Cause: This error occurs when you attempt to specify a "not equals" expression when defining the external timing trigger. You can only specify labels which equal patterns (of 1's, 0's, or X's).

Action: Do not attempt to specify negated timing patterns.

2031 **Invalid trigger duration: <duration>**

Cause: This error occurs when you attempt to specify an external timing trigger duration which is in the valid range but is not a multiple of 10 ns.

Action: Re-enter the command with the trigger duration as a multiple of 10 ns.

2032 **Trigger duration out of bounds: <bounds>**

Cause: This error occurs when you attempt to specify an external timing trigger duration outside the valid range. A "greater than" duration must fall within the range of 30 ns to 10 ms (and must be a multiple of 10 ns). A "less than" duration must fall within the range 40 ns to 10ms (and must be a multiple of 10 ns).

Action: Re-enter the command with the trigger duration within the bounds shown.

2042 **Trigger delay out of bounds: <bounds>**

Cause: This error occurs when you attempt to specify an external timing trigger delay outside the valid range. The external timing trigger delay must be between 0 and 10 ms (in 10 ns increments).

Action: Re-enter the command with the trigger delay within the bounds shown.

11

Specifications and Characteristics

Emulator Specifications and Characteristics

This section contains the following types of emulator specifications and characteristics:

- Electrical characteristics (including emulator timing).
- Physical characteristics.
- Environmental characteristics.

Electrical

This section describes the electrical characteristics of the HP 64744 68000/HC001/EC000 Emulator and the HP 64700 Card Cage.

Electrical Characteristics of the HP 64744 Emulator

The AC characteristics of the HP 64744 emulator are listed in the following tables.

AC Electrical Specifications — Clock Timing

Num.	Characteristic	Symbol	MC68HC001 16.67 MHz Min	MC68HC001 16.67 MHz Max	HP 64744 Min	HP 64744 Max	Unit
	Frequency of Operation	f	4.0	16.7	4.0	16.7	MHz
1	Cycle Time	t_{cyc}	60	250	60	250	ns
2 3	Clock Pulse Width	t_{CL} t_{CH}	27 27	62.5 62.5	27 27	62.5 62.5	ns
4 5	Clock Rise and Fall Times	t_{Cr} t_{Cf}	— —	5 5	— —	5 5	ns

AC Electrical Specifications — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

			MC68HC001		HP 64744				
			16.67 MHz		Worst Case		Typical[a]		
Num.	Characteristic	Symbol	Min	Max	Min	Max	Min	Max	Unit
6	Clock Low to Address Valid	t_{CLAV}	—	30	—	68.5	—	32	ns
6A	Clock High to FC Valid	t_{CHFCV}	—	30	—	58.5	—	35	ns
7	Clock High to Address, Data Bus High Impedance (Maximum)	t_{CHADZ}	—	50	—	—	—	—	ns
8	Clock High to Address, FC Invalid, (Minimum)	t_{CHAFI}	0	—	0	—	0	—	ns
9[1]	Clock High to $\overline{AS}$, $\overline{DS}$ Asserted	t_{CHSL}	3	30	3	68.5	3	32	ns
11[2]	Address Valid to $\overline{AS}$, $\overline{DS}$ Asserted (Read)/$\overline{AS}$ Asserted (Write)	t_{AVSL}	15	—	10	—	15	—	ns
11A[2]	FC Valid to $\overline{AS}$, $\overline{DS}$ Asserted (Read)/$\overline{AS}$ Asserted (Write)	t_{FCVSL}	45	—	40	—	45	—	ns
12[1]	Clock Low to $\overline{AS}$, $\overline{DS}$ Negated	t_{CLSH}	—	30	—	68.5	—	31	ns
13[2]	$\overline{AS}$, $\overline{DS}$ Negated to Address, FC Invalid	t_{SHAFI}	15	—	10	—	15	—	ns
14[2]	$\overline{AS}$, (and $\overline{DS}$ Read) Width Asserted	t_{SL}	120	—	109	—	120	—	ns
14A[2]	$\overline{DS}$ Width Asserted (Write)	t_{DSL}	60	—	49	—	60	—	ns
15[2]	$\overline{AS}$, $\overline{DS}$ Width Negated	t_{SH}	60	—	60	—	66	—	ns
16	Clock High to Control Bus High Impedance	t_{CHCZ}	—	50	—	50	—	50	ns
17[2]	$\overline{AS}$, $\overline{DS}$ Negated to R/$\overline{W}$ Invalid	t_{SHRH}	15	—	9	—	15	—	ns
18[1]	Clock High to R/$\overline{W}$ High (Read)	t_{CHRH}	0	30	0	64.3	0	33	ns

AC Electrical Specifications — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

20[1]	Clock High to R/$\overline{W}$ Low (Write)	t_{CHRL}	0	40	0	78.3	0	44	ns
20A[2,6]	$\overline{AS}$ Asserted to R/$\overline{W}$ Low (Write)	t_{ASRV}	—	10	—	15.1	—	10	ns
21[2]	Address Valid to R/$\overline{W}$ Low (Write)	t_{AVRL}	0	—	0	—	0	—	ns
21A[2]	FC Valid to R/$\overline{W}$ Low (Write)	t_{FCVRL}	30	—	24.9	—	31	—	ns
22[2]	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	t_{RLSL}	30	—	24.9	—	29	—	ns
23	Clock Low to Data-Out Valid (Write)	t_{CLDO}	—	30	—	68.6	—	32	ns
25[2]	$\overline{AS}$, $\overline{DS}$ Negated to Data-Out Invalid (Write)	t_{SHDOI}	15	—	9.9	—	15	—	ns
26[2]	Data-Out Valid to $\overline{DS}$ Asserted (Write)	t_{DOSL}	15	—	9.9	—	15	—	ns
27[5]	Data-In Valid to Clock Low (Setup Time on Read)	t_{DICL}	5	—	18.5	—	15	—	ns
28[2]	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DTACK}$ Negated (Asynchronous Hold)	t_{SHDAH}	0	110	0	85.4	0	95	ns
28A	Clock High to $\overline{DTACK}$ Negated	t_{CHDH}	0	110	0	85.4	0	95	ns
29	$\overline{AS}$, $\overline{DS}$ Negated to Data-In Invalid (Hold Time on Read)	t_{SHDII}	0	—	0	—	0	—	ns
29A	$\overline{AS}$, $\overline{DS}$ Negated to Data-In High Impedance	t_{SHDZ}	—	90	—	75	—	81	ns
30	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{BERR}$ Negated	t_{SHBEH}	0	—	0	—	0	—	ns
31[2,5]	$\overline{DTACK}$ Asserted to Data-In Valid (Setup Time)	t_{DALDI}	—	50	—	43.4	—	46	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	t_{RHr} t_{RHf}	0	150	**[b]	**[b]	**[b]	**[b]	ns
33	Clock High to $\overline{BG}$ Asserted	t_{CHGL}	—	30	—	58.2	—	42	ns

AC Electrical Specifications — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

34	Clock High to $\overline{BG}$ Negated	t_{CHGH}	—	30	—	58.2	—	42	ns
35	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted	t_{BRLGL}	1.5	3.5	1.5	3.5	1.5	3.5	Clks
36[7]	$\overline{BR}$ Negated to $\overline{BG}$ Negated	t_{BRHGH}	1.5	3.5	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	t_{GALGH}	1.5	3.5	1.5	3.5	1.5	3.5	Clks
37A[8]	$\overline{BGACK}$ Asserted to $\overline{BR}$ Negated	t_{GALBRH}	10	1.5	10	1.5	10	1.5	Clks/ns
38	$\overline{BG}$ Asserted to Control, Address, Data Bus High Impedance ($\overline{AS}$ Negated)	t_{GLZ}	—	50	—	50	—	50	ns
39	$\overline{BG}$ Width Negated	t_{GH}	1.5	—	1.5	—	1.5	—	Clks
40	Clock Low to $\overline{VMA}$ Asserted	t_{CLVML}	—	50	—	66.5	—	53	ns
41	Clock Low to E Transition	t_{CLET}	—	35	—	51.5	—	38	ns
42	E Output Rise and Fall Time	$t_{Er,f}$	—	15	—	15	—	15	ns
43	$\overline{VMA}$ Asserted to E High	t_{VMLEH}	80	—	74.9	—	80	—	ns
44	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{VPA}$ Negated	t_{SHVPH}	0	50	—	37.5	—	41	ns
45	E Low to Control, Address Bus Invalid (Address Hold Time)	t_{ELCAI}	10	—	4.5	—	10	—	ns
46	$\overline{BGACK}$ Width Low	t_{GALI}	1.5	—	1.5	—	1.5	—	Clks
47[5]	Asynchronous Input Setup Time	t_{ASI}	5	—	16.6	—	10	—	ns
48[2,3]	$\overline{BERR}$ Asserted to $\overline{DTACK}$ Asserted	t_{BELDAL}	10	—	10	—	10	—	ns
49[9]	$\overline{AS}$, $\overline{DS}$, Negated to E Low	t_{SHEL}	-35	35	-41.6	41.6	-41.6	41.6	ns
50	E Width High	t_{EH}	220	—	220	—	220	—	ns
51	E Width Low	t_{EL}	340	—	340	—	340	—	ns
52	Data-In Hold from Clock High	t_{CHDII}	0	—	0	—	0	—	ns

AC Electrical Specifications — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

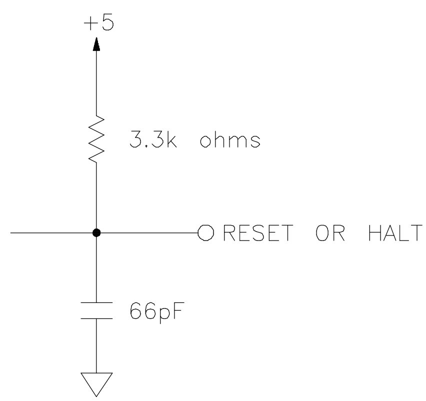
53	Data-Out Hold from Clock High	t_{CHDOI}	0	—	0	—	0	—	ns
54	E Low to Data-Out Invalid	t_{ELDOI}	10	—	10	—	10	—	ns
55	R/$\overline{W}$ Asserted to Data Bus Impedance Change	t_{RLDBD}	0	—	0	—	0	—	ns
56[4]	$\overline{HALT}$/$\overline{RESET}$ Pulse Width	t_{HRPW}	10	—	10	—	10	—	Clks
57	$\overline{BGACK}$ Negated to $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ Driven	t_{GASD}	1.5	—	1.5	—	1.5	—	Clks
57A	$\overline{BGACK}$ Negated to FC, $\overline{VMA}$ Driven	t_{GAFD}	1	—	1	—	1	—	Clks
58[7]	$\overline{BR}$ Negated to $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ Driven	t_{RHSD}	1.5	—	1.5	—	1.5	—	Clks
58A[7]	$\overline{BR}$ Negated to FC Driven	t_{RHFD}	1	—	1	—	1	—	Clks

AC Electrical Specifications — Read and Write Cycles (Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)
MC68HC001 NOTES: 1. For a loading capacitance of less than or equal to 50 pF, subtract 5 ns from the value given in the maximum columns. 2. Actual value depends on clock period. 3. If #47 is satisfied for both $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$, #48 may be ignored. In the absence of $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$ is an asychronous input using the asychronous input setup time (#47). 4. For power-up, the MC68HC001 must be held in the reset state for 100 ms to allow stabilization of on-chip circuitry. After the system is powered up, #56 refers to the minimum pulse width required to reset the controller. 5. If the asychronous input setup time (#47) requirement is satisfied for $\overline{\text{DTACK}}$, the $\overline{\text{DTACK}}$ asserted to data setup time (#31) requirement can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle. 6. When $\overline{\text{AS}}$ and R/$\overline{\text{W}}$ are equally loaded (+/-20%), subtract 5 ns from the values given in these columns. 7. The processor will negate $\overline{\text{BG}}$ and begin driving the bus again if external arbitration logic negates $\overline{\text{BR}}$ before asserting $\overline{\text{BGACK}}$. 8. The minimum value must be met to guarantee proper operation. If the maximum value is exceeded, $\overline{\text{BG}}$ may be re-asserted. 9. The falling edge of S6 triggers both the negation of the strobes ($\overline{\text{AS}}$ and x$\overline{\text{DS}}$) and the falling edge of E. Either of these events can occur first, depending on the loading on each signal. Specification #49 indicates the absolute maximum skew that will occur between the rising edge of the strobes and the falling edge of the E clock.

AC Electrical Specifications — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

HP 64744 NOTES:
a. Typical outputs measured with 130 pF load input driven by 74F244.

b. $\overline{\text{RESET}}$, $\overline{\text{HALT}}$ input circuit should meet MC68HC001 microprocessor specification #32 and drive the equivalent vcircuit shown in the figure below.



Electrical Characteristics of the HP 64700

The electrical characteristics of the HP 64700 communication ports are as follows.

Communications

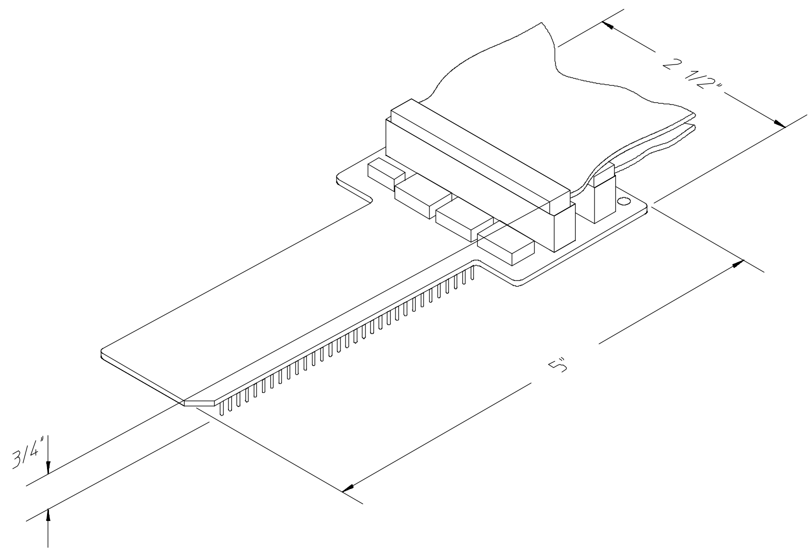
Serial Port	RS-232-C DCE or DTE to 38.4 Kbaud. RS-422 DCE to 460.8 Kbaud.
BNC (labeled TRIGGER IN/OUT)	Input. The signal must drive approximately 4 mA at 2 V. Edge Sensitive. Minimum pulse width is approximately 25 ns. Output. Driven active high only; equals +2.4V into a 50 ohm load.

Physical

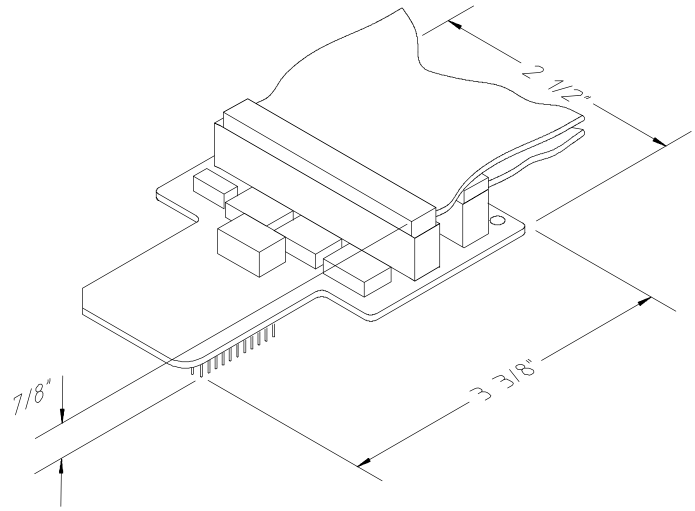
Dimensions of Emulator Probe Package Adapters

There must be enough clearance in the target system to allow the emulation probe adapter to be plugged in and the cable routed from the target system to the emulator control box. The following figures show probe dimensions and pin orientation.

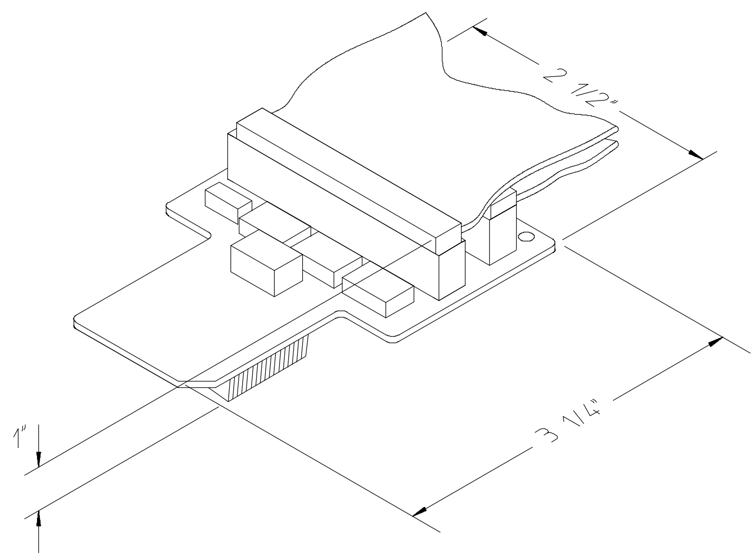
HP 64744B 68000 DIP Package Adapter



HP 64744C 68000/HC001 PGA Package Adapter



HP 64744D 68HC000/001 PLCC Package Adapter
HP 64744E 68EC000 PLCC Package Adapter



Emulator Dimensions

Width	325 mm (12.8 in.)
Height	173 mm (6.8 in.)
Length	389 mm (15.3 in.)

Emulator Weight

HP 64760	8.2 kg (18 lb)

Cable Length

Emulator to target system	approximately 610 mm (2 ft).

Probe Dimensions

92 mm (3.625 in.) width x 16 mm (0.626 in.) height x 159 mm (6.25 in.) length

Communications

Serial Port	25-pin female type "D" subminiature connector.
CMB Port	9-pin female type "D" subminiature connector.

CAUTION **Possible damage to emulator**. Any component used in suspending the emulator **must** be rated for 30 kg (65 lb) capacity.

Environmental

Temperature

Operating	0°C to +55°C (+32°F to 131°F)
Non-operating	-40°C to +70°C (-40°F to 158°F)

Altitude

Operating	4 600m (15 000 ft)
Non-operating	15 300m (50 000 ft).

Relative Humidity

15% to 95%.

External Analyzer Specifications

- Threshold Accuracy = +/- 50 mV.
- Dynamic Range = +/- 10 V about threshold setting.
- Minimum Input Swing = 600 mV pp.
- Minimum Input Overdrive = 250 mV or 30% of threshold setting, whichever is greater.
- Absolute Maximum Input Voltage = +/- 40 V.
- Probe Input Resistance = 100K ohms +/- 2%.
- Probe Input Capacitance = approximately 8 pF.
- Maximum +5 Probe Current = 0.650 A.
- +5 Probe Voltage Accuracy = +5.0 +/- 5%.

External State Analyzer Specifications

- Data Setup Time = 10 ns min.
- Data Hold Time = 0 ns, typical.
- Qualifier Setup Time = 20 ns min.
- Qualifier Hold Time = 10 ns, typical.
- Minimum Clock Width = 10 ns
- Minimum Clock Period:
  - No Tagging Mode = 40 ns (25 Mhz clock).
  - Event Tagging Mode = 50 ns (20 MHz clock).
  - Time Tagging Mode = 60 ns (16 MHz clock).
- Minimum Time from Slave Clock to Master Clock = 10 ns.
- Minimum Time from Master Clock to Slave Clock = 50 ns.

Part 4

Concept Guide

12

Concepts

Concepts

This chapter provides conceptual information on the following topic:

- Demo program descriptions.

Demo Program Description

Two demo programs have been used to generate examples throughout this manual. The emulator demo program is used in the "Quick Start" chapter's tutorial and to generate examples in the "Using the Emulator" chapter. The analyzer demo program is used to generate examples in the "Using the Analyzer" chapters. Both these demo programs are simple C language programs.

Emulator Demo Program

The emulator demo program used in this chapter is a simple command interpreter. The "cmd_rdr.c" source file is shown below.

```
volatile char Cmd_Input;
char Msg_Dest[0x20];

void Write_Msg (char *s)
{
   char *Dest_Ptr;

   Dest_Ptr = Msg_Dest;
   while (*s != '\0')
   {
      *Dest_Ptr = *s;
      Dest_Ptr++;
      s++;
   }
}

main ()
{
   static char Msg_A[] = "Command A Entered               ";
   static char Msg_B[] = "Entered B Command               ";
   static char Msg_I[] = "Invalid Command                 ";
   char c;

   for (;;)
   {
      Cmd_Input = '\0';
      while ((c = Cmd_Input) == '\0');
      switch (c) {
         case 'A' :
            Write_Msg (Msg_A);
            break;
         case 'B' :
            Write_Msg (Msg_B);
            break;
         default :
            Write_Msg (Msg_I);
            break;
      }
   }
}
```

The "cmd_rdr" program continuously reads values from **Cmd_Input**; when a value other than NULL is found, the program calls the **Write_Msg** function to copy a string to the **Msg_Dest** array.

Building the Emulator Demo Program

The emulator demo program was built using the Hewlett-Packard 68000 Advanced C Cross Compiler and the 68000/10/20 Assmbler/Linker/Librarian software development tools on the HP 9000 Series 300 host computer with the following command:

`$` **`cc68000 -hOGNr hp64742 -Wl,-Lfx -o`** `cmd_rdr cmd_rdr.c > cmd_rdr.map`

Analyzer Demo Program

The "anly.c" source file is shown below.

```
#include <stdlib.h>

int Results[0x100];

void Write_Num (int Number, int Offset)
{
   Offset = Offset % 256;
   Results[Offset] = Number;
}

void Caller_0 (int Num, int Ofs)
{
   Write_Num (Num, Ofs);
}

void Caller_1 (int Num, int Ofs)
{
   Write_Num (Num, Ofs);
}

void Caller_2 (int Num, int Ofs)
{
   Write_Num (Num, Ofs);
}

void Caller_3 (int Num, int Ofs)
{
   Write_Num (Num, Ofs);
}

static int cmp_function (const int *item1, const int *item2)
{
   if (*item1 < *item2)
   {
      return (-1);
   }
   else if (*item1 > *item2)
   {
```

```
      return (1);
   }
   else
   {
      return (0);
   }
}

void main ()
{
   int Rand_Num=0, Rand_Num_Old=0, Rand_Num_Older, Counter;

   Counter = 4096;
   for (;;)
   {
      Rand_Num_Older = Rand_Num_Old;
      Rand_Num_Old = Rand_Num;
      Rand_Num = rand();

      switch (Rand_Num_Old % 4) {
         case 0 :
            Caller_0 (Rand_Num, Rand_Num_Older);
            break;
         case 1 :
            Caller_1 (Rand_Num, Rand_Num_Older);
            break;
         case 2 :
            Caller_2 (Rand_Num, Rand_Num_Older);
            break;
         case 3 :
            Caller_3 (Rand_Num, Rand_Num_Older);
            break;
         default :
            break;
      }
      Counter--;
      if (Counter == 0)
      {
         qsort (Results, 0x100, sizeof(*Results),            (int
(*)(const void *, const void *))cmp_function);
         Counter = 4096;
      }
   }
}
```

The "anly" program uses four different functions to call the Write_Num function to simulate situations in real programs where routines are called from many different places.

Building the Analyzer Demo Program

The analyzer demo program was built using the Hewlett-Packard 68000 Advanced C Cross Compiler and the 68000/10/20 Assmbler/Linker/Librarian software development tools on the HP 9000 Series 300 host computer with the following command:

```
$ cc68000 -hOGNr hp64742 -Wl,-Lfx -o anly anly.c > anly.map
```

Part 5

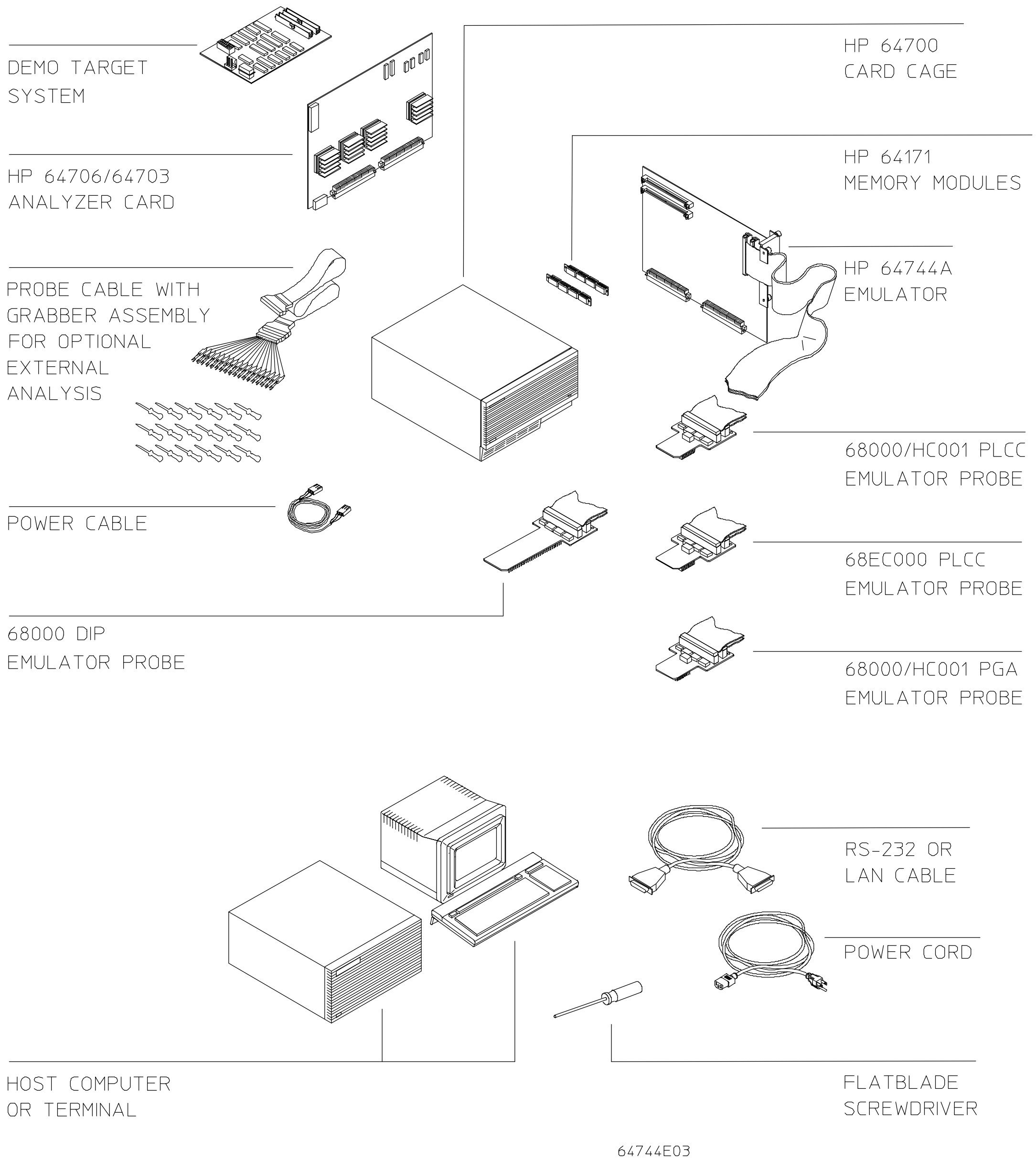
Installation Guide

Part 5

13

Installation

Installation at a Glance



The HP 64744 68000/HC001/EC000 Emulator card and the HP 64706/3 Emulation Bus Analyzer card, when installed into the HP 64700 Card Cage, are used for emulation and analysis of 68000/HC000/HC001/EC000 microprocessor systems.

Equipment supplied

The minimum system contains:

- HP 64700 Card Cage.
- HP 64744A 68000/HC001/68EC000 emulator control card with 60 Kbytes of emulation memory (which includes the 64744-66509 demo board) and one of the following probe adapters:
  - HP 64744B 68000 DIP Probe Adapter.
  - HP 64744C 68HC000/001 PGA Probe Adapter.
  - HP 64744D 68HC000/001 PLCC Probe Adapter.
  - HP 64744E 68EC000 PLCC Probe Adapter.
- HP 64706A 48-Channel Emulation Bus Analyzer card.

Optional parts are:

- HP 64171A 256 Kbyte Memory Modules or HP 64171B 1 Mbyte Memory Modules (which provide additional emulation memory).
- HP 64703A 64-Channel Emulation Bus Analyzer and 16-Channel External State/Timing Analyzer (instead of HP 64706A).
- HP 64704A 80-Channel Emulation Bus Analyzer card (instead of HP 64706A).
- HP 64794A 80-Channel Deep Memory Emulation Bus Analyzer card (instead of HP 64706A).

Equipment and tools needed

In order to install and use the 68EC000 emulation system, you need:

- Host computer or terminal with RS-232/RS-422 port.
- RS-232/RS-422 cable.
- Flat-blade screwdriver.

Installation overview

The steps in the installation process are:

1 Install memory modules on emulator board, and install emulator and analyzer boards into the HP 64700 Card Cage.
2 Connect the HP 64700 Card Cage to a host computer or terminal.
3 Connect the emulator probe cables to the 64744-66509 demo board.
4 Apply power to the HP 64700.
5 Verify emulator and analyzer performance.

Your emulation and analysis system may already be assembled (depending on how parts of the system were ordered), and you may only need to connect the HP 64700 to a host computer or terminal and the target microprocessor system.

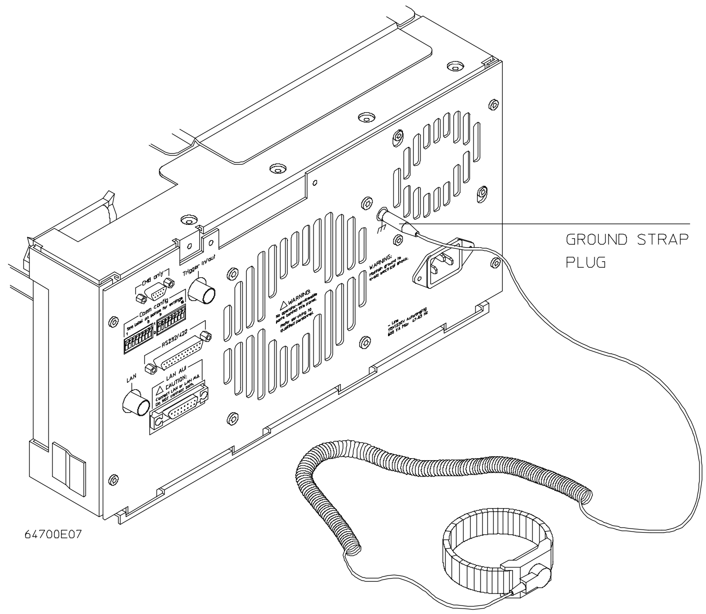
Step 1. Install Boards into the HP 64700 Card Cage

WARNING **Before removing or installing parts in the HP 64700 Card Cage, make sure that the card cage power is off and that the power cord is disconnected.**

CAUTION Do NOT stand the HP 64700 on the rear panel. You could damage the rear panel ports and connectors.

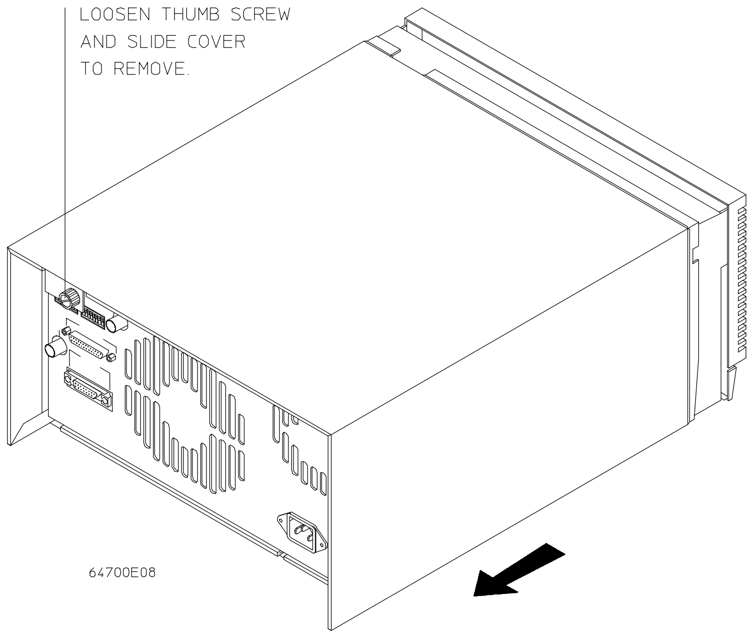
If your emulator and analyzer boards are already installed in the HP 64700 Card Cage, go to "Step 2. Connect the HP 64700 to a Host Computer or Terminal".

1 Use a ground strap when removing or installing boards into the HP 64700 Card Cage. A jack on the rear panel of the HP 64700 Card Cage is provided for this purpose.

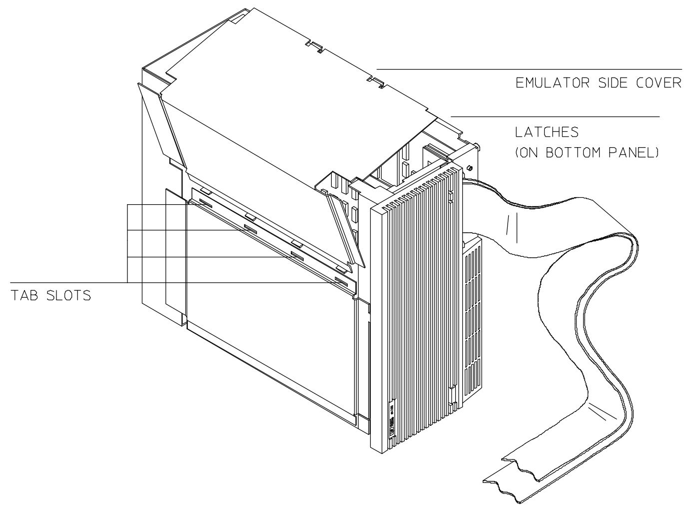


When removing or installing cards, you should use a ground strap to reduce the chances of damaging circuit cards from static discharge.

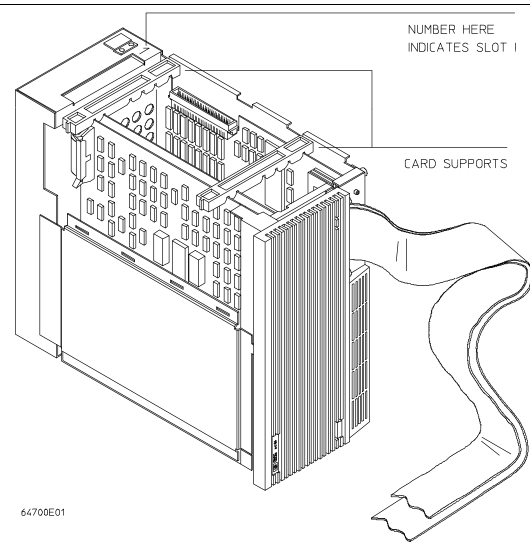
2 Turn the thumb screw and remove the top cover by sliding the cover toward the rear and up.



3 Remove the side cover by unsnapping the two latches and lifting off.

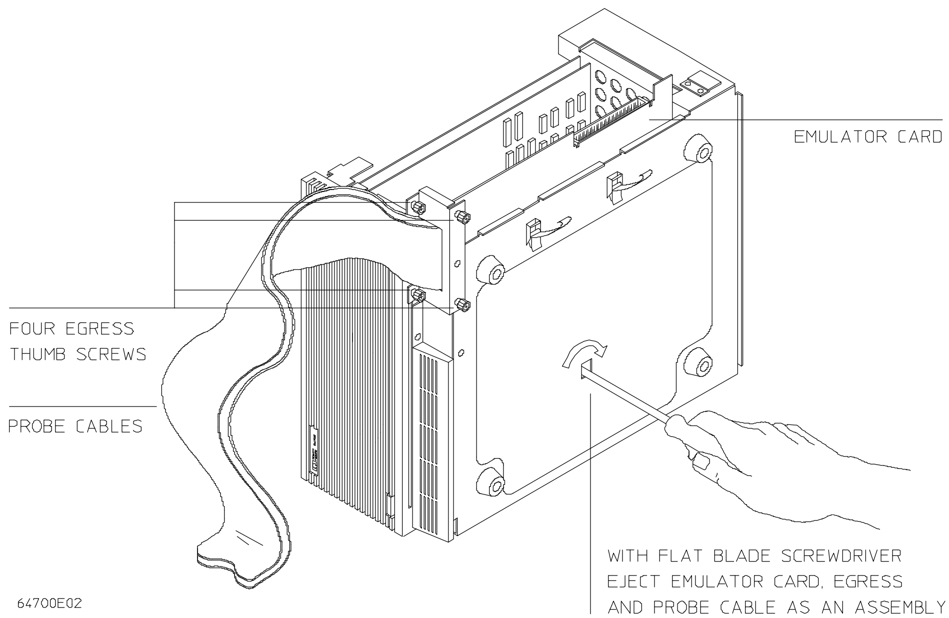


4 Remove the card supports.

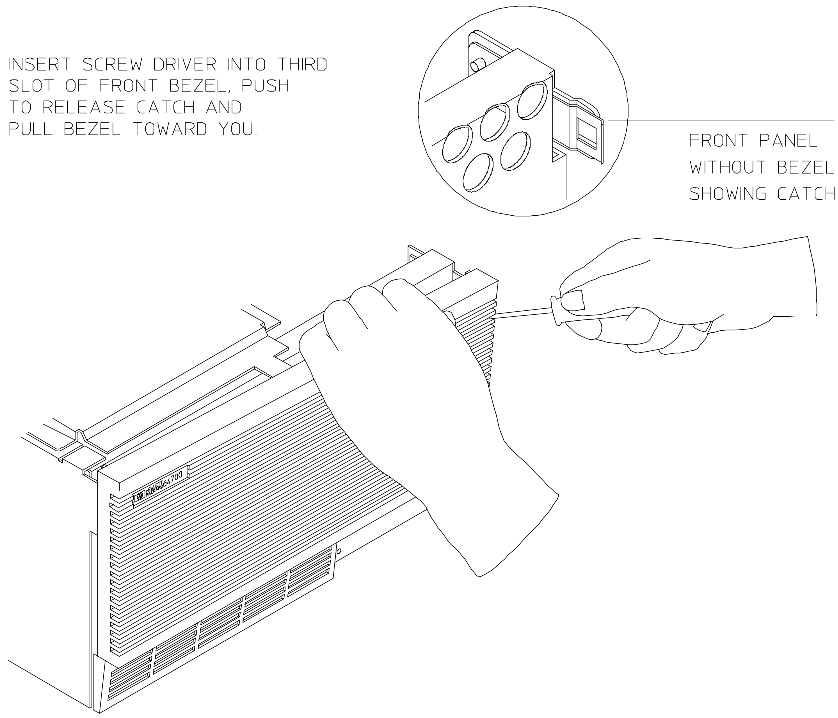


5 First, completely loosen the four egress thumb screws.

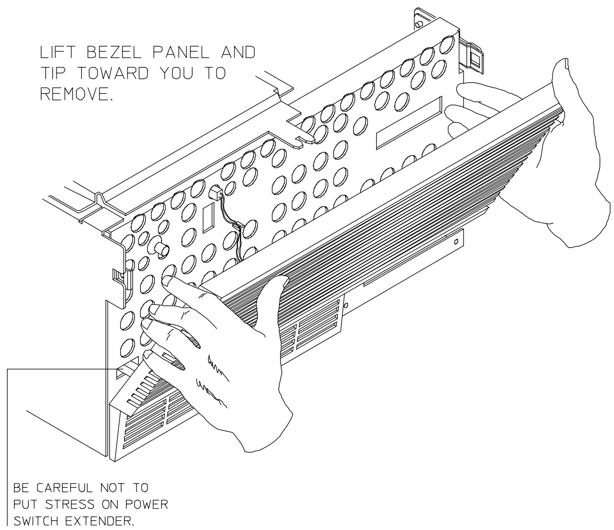
To remove emulator cards, insert a flat blade screwdriver in the access hole and eject the emulator cards by rotating the screwdriver.



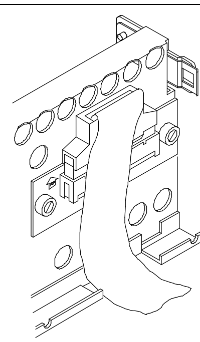
6 Insert a screw driver into the third slot of the right side of the front bezel, push to release catch, and pull the right side of the bezel about one half inch away from the front of the HP 64700. Then, do the same thing on the left side of the bezel. When both sides are released, pull the bezel toward you approximately 2 inches.



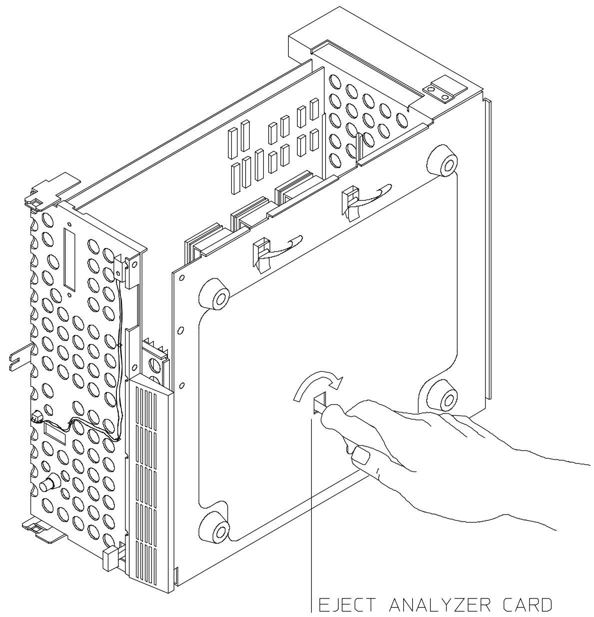
7 Lift the bezel panel to remove. Be careful not to put stress on the power switch extender.



8 If you're removing an existing analyzer card that provides external analysis, remove the right angle adapter board by turning the thumb screws counter-clockwise.



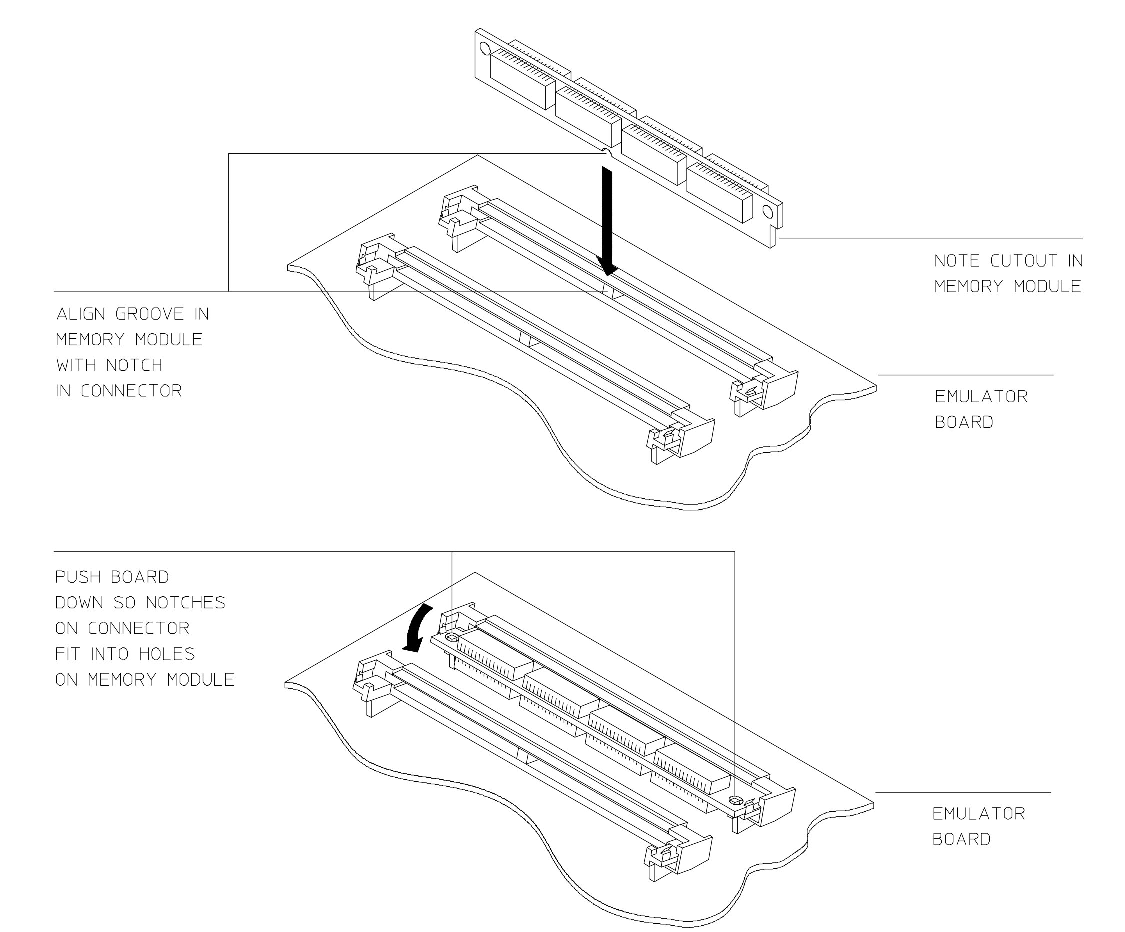
9 To remove the analyzer card, insert a flat blade screwdriver in the access hole and eject the analyzer card by rotating the screwdriver.



Do not remove the system control board. This board is used in all HP 64700 emulation and analysis systems.

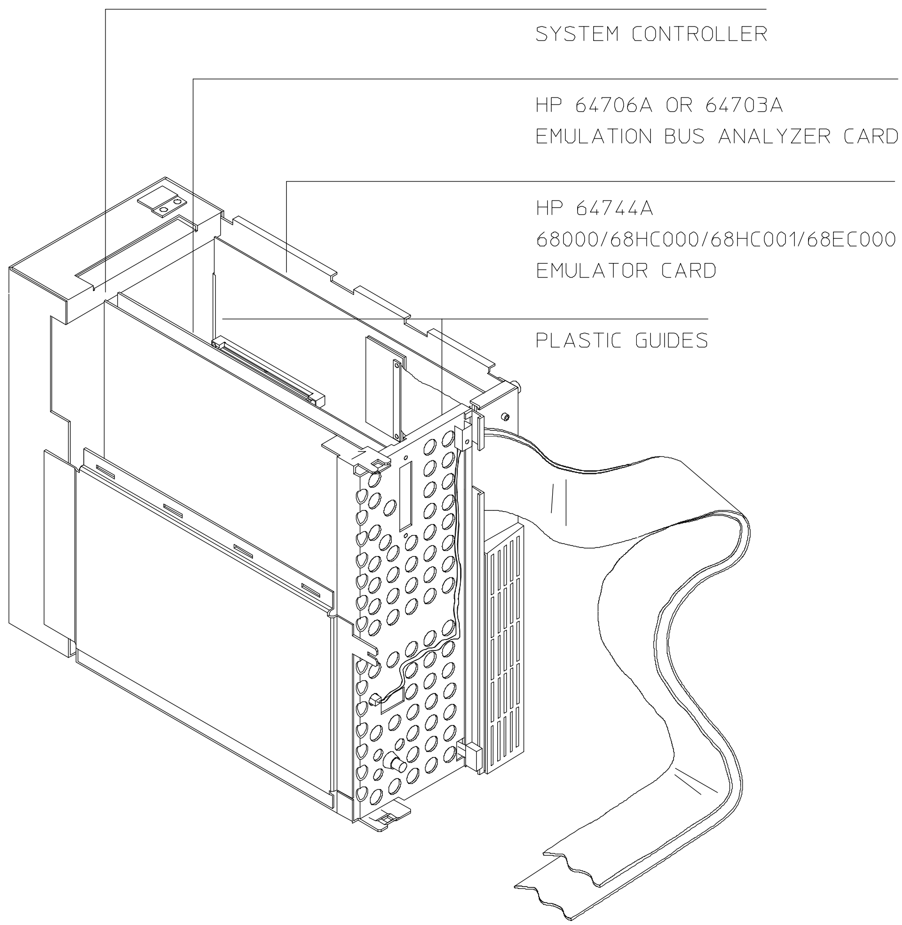
10 Install emulation memory modules on emulator board. Two types of modules may be installed: 256 Kbyte (HP 64171A), and 1 Mbyte (HP 64171B). Either type of module may be installed in any slot. There is a cutout on one side of the memory modules so that they can only be installed one way.

To install memory modules, place the memory module into the socket groove. Firmly press the memory module into the socket to make sure that it is completely seated. Once the memory module is seated in the connector groove, push the memory module down so that the notches on the socket fit into the holes on the memory module. There are two latches on the sides of the socket that hold the memory module in place.

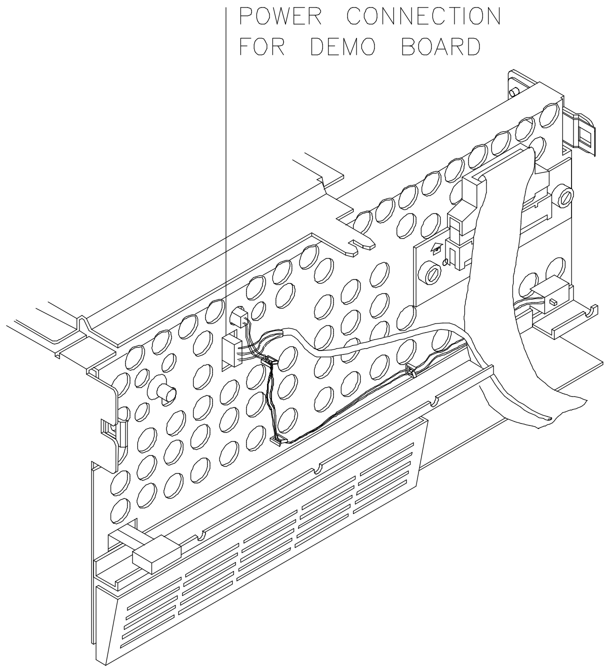


11 Install the emulation bus analyzer board (HP 64706A, HP 64703A, HP 64704A, or HP 64794A) and the HP 64744A emulation control board. The emulation bus analyzer board is installed in the slot next to the system controller board. The HP 64744A is installed in the second slot from the bottom of the HP 64700. These boards are identified with labels that show the model number and the serial number.

To install a card, insert it into the plastic guides. Make sure the connectors are properly aligned; then, press the card into mother board sockets. Check to ensure that the cards are seated all the way into the sockets. If the cards can be removed with your fingers, the cards are NOT seated all the way into the mother board socket.

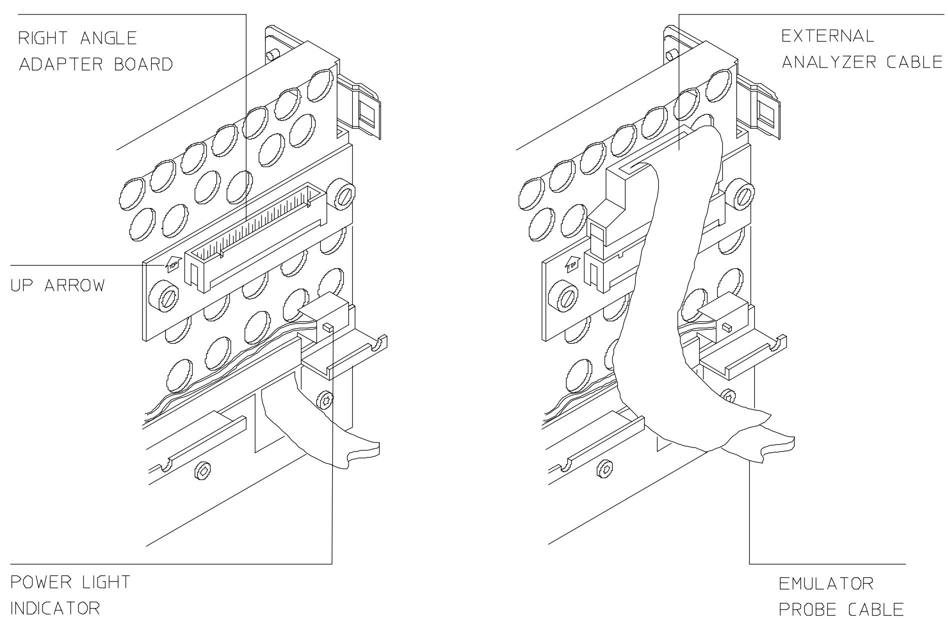


12 Connect the +5 V power cable to the connector in the HP 64700 front panel.

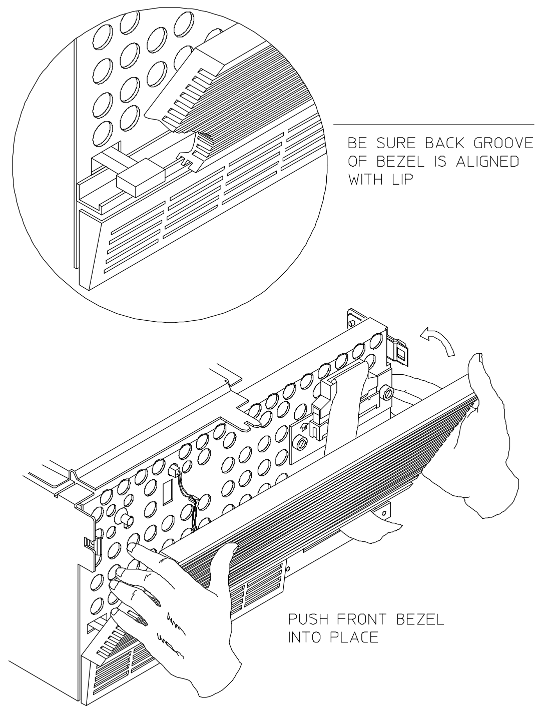


13 If you installed the HP 64703A analyzer card, install the right angle adapter board by turning the thumb screws clockwise.

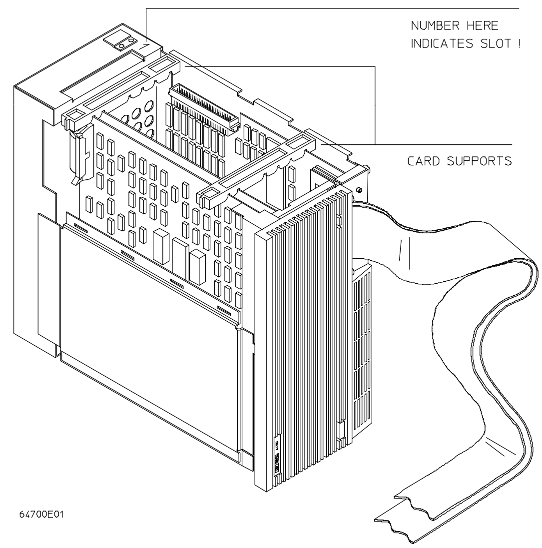
Connect the external analyzer probe cable to the right angle adapter board. Each connector of the external analyzer cable is keyed so that it can be connected to the right angle adapter board in only one way. Check for bent connector pins before connecting the analyzer probe cable. Align the key of the external analyzer cable connector with the slot in the right angle adapter board, and gently press the external analyzer cable connector into the connector on the right angle adapter board. Position the cable as shown below.



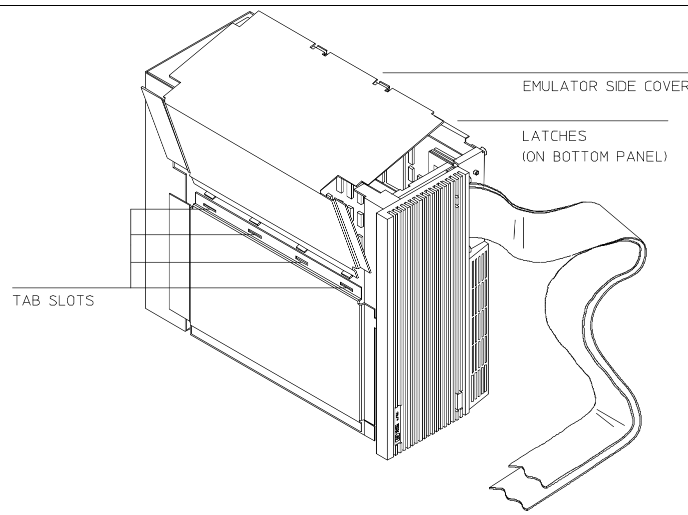
14 To reinstall the front bezel, be sure that the bottom rear groove of the front bezel is aligned with the lip as shown below.



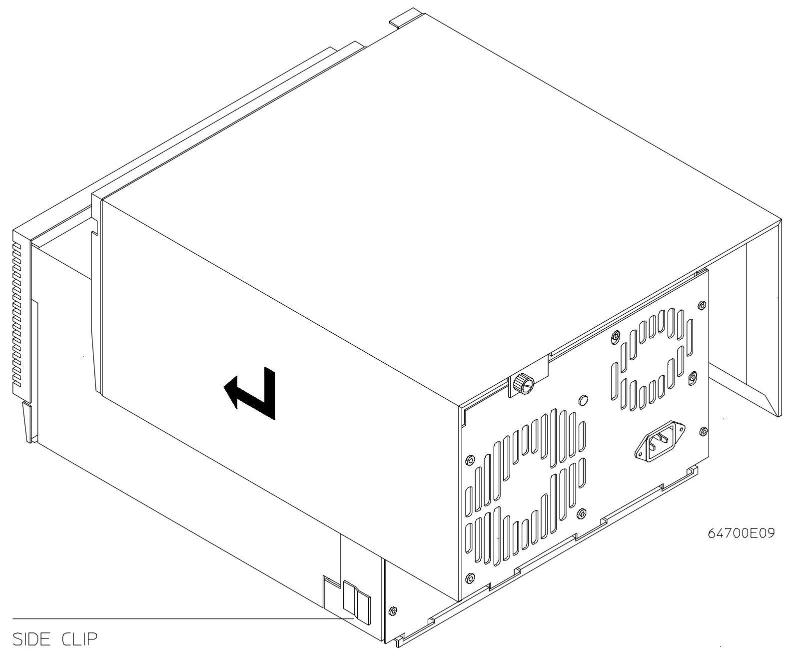
15 Install the card supports.



16 To install the side cover, insert the side cover into the tab slots and fasten the two latches.



17 Install the top cover in reverse order of its removal, but make sure that the side panels of the top cover are attached to the side clips on the frame.

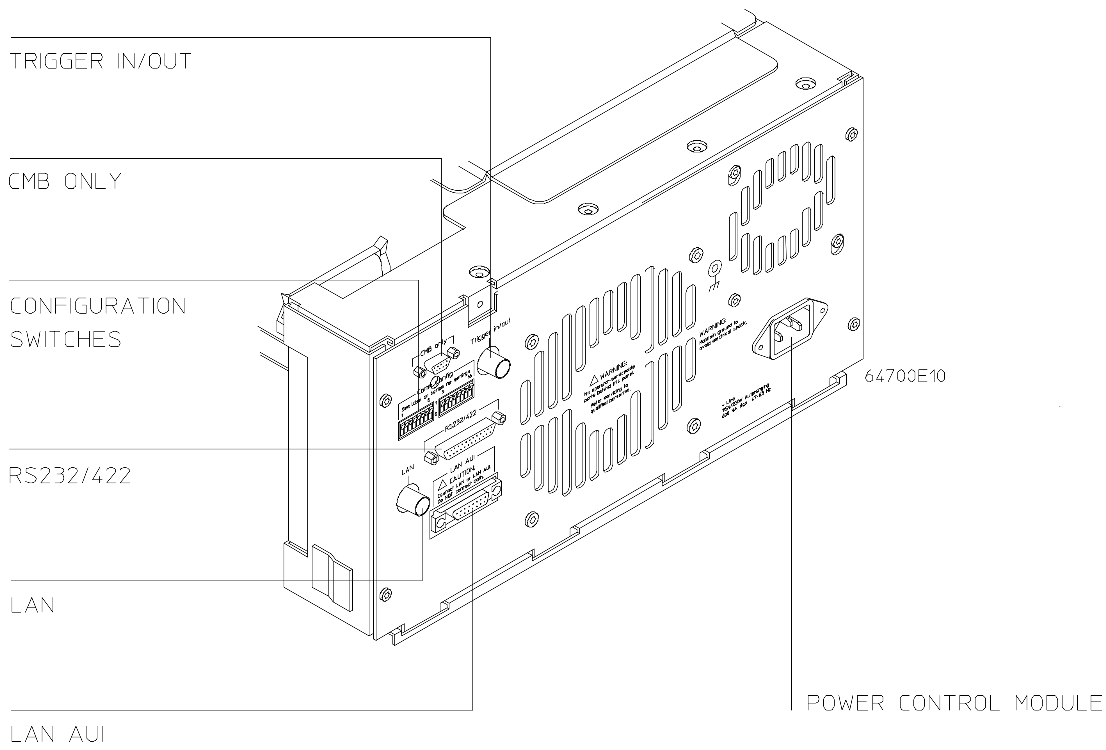


Step 2a. Connect the HP 64700 via RS-232/RS-422

If you wish to connect the HP 64700 to a host computer via the LAN interface, go to "Step 2b. Connect the HP 64700 via LAN".

1 Set the data communications configuration switches so that the HP 64700 port will have characteristics compatible with the terminal or host computer interface to which it will be connected (see the following switch summary tables). Note that the configuration switch settings are only read when the HP 64700 is powered ON or when the **init -p** command is entered.

The locations of the data communications ports and configuration switches are shown below.



HP 64700B Configuration Switch Summary

The information in the following table is also on an adhesive label attached to each HP 64700B.

<table>
<tr><th colspan="8">Configuration Switches S1-S8</th></tr>
<tr><th>S1</th><th>S2</th><th>S3</th><th>S4</th><th>S5</th><th>S6</th><th>S7</th><th>S8</th></tr>
<tr><td colspan="3" rowspan="2">RS-232/RS-422 Baud Rate
1 1 1 = 230400
1 1 0 = 115200
1 0 1 = 38400
1 0 0 = 57600
0 1 1 = 1200
0 1 0 = 2400
0 0 1 = 19200
0 0 0 = 9600</td><td>1 =
DTE</td><td>1 =
RS-422</td><td>1 =
Service</td><td>1 =
Service</td><td>1 =
Reserved for future use</td></tr>
<tr><td>0 =
DCE</td><td>0 =
RS-232</td><td>0 =
Normal</td><td>0 =
Normal</td><td>0 =
Normal</td></tr>
</table>

NOTES:

S1 - S3: Asynchronous baud rates include 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200. The rear panel switches can be used to initialize at 1200, 2400, 9600, 19200, 38400, or 115200 baud. Rates of 300 baud and 4800 baud are only selectable through the Terminal Interface **stty** command. This entire range of rates are supported at RS-422 signal levels. The EIA-RS232-D standard only covers data rates up to 20,000 bits per second (actual 19200). Asynchronous connections using RS-232 signal levels above this rate can be used but cannot be guaranteed.

Isosynchronous rates of 230400 baud and 460800 baud are supported at RS-422 signal levels using a 1X clock. The rate of 230400 can be selected through the rear panel switches but 460800 is only selectable through the **stty** command.

S4: DCE = Data Communications Equipment, DTE = Data Terminal Equipment. This switch is ignored if S5 sets the serial port to be an RS-422 device (which is always DCE).

S6: When this switch is set to "1", self diagnostic information is displayed by a flashing LED on the control board during the powerup cycle. This information is intended to be used by a qualified service technician only.

S7: When this switch is set to "1", the HP 64700B firmware is forced to execute from ROM instead of Flash EPROM. This mode is intended to be used by a qualified service technician only.

Configuration Switches S9-S16

S9	S10	S11	S12	S13	S14	S15	S16
1 = 7 Bit character size	1 = Parity enabled	1 = Parity even	1 = RTS/CTS DSR/DTR	1 = XON/ XOFF	1 = LAN BOOTP enabled	1 = 15 pin AUI	1 = LAN
0 = 8 Bit character size	0 = Parity disabled	0 = Parity odd	0 = No HW handshake	0 = No SW handshake	0 = LAN BOOTP disabled	0 = BNC ThinLAN	0 = Serial

NOTES:

S12: Hardware pacing uses a modified handshake. When hardware handshake is enabled, the DTE uses Clear to Send (CTS) to control its output. When CTS is true, data may be output, when CTS is false, data output will stop at the end of the current character. The DCE is expected to negate CTS during receipt of a character if the internal hardware buffer is full. Once a position is available in the internal hardware buffer, CTS is to be set true.

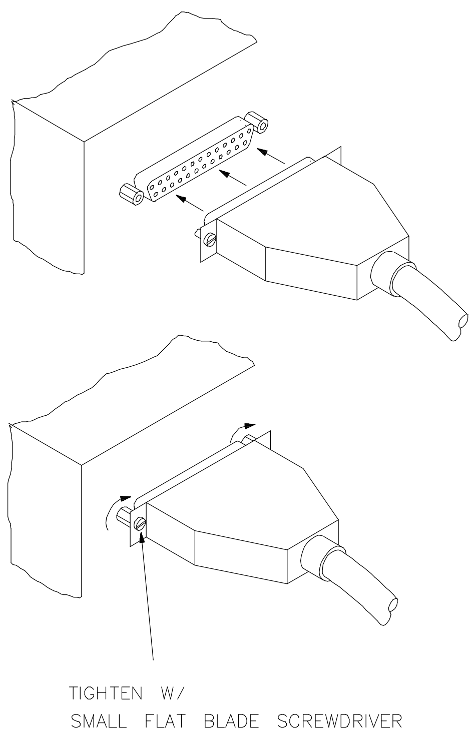
A modification is made in the use of Request to Send (RTS) as a reverse channel Clear to Send to control the output of the DCE. The DTE sets RTS false during the receipt of a character if there is no room in its hardware buffer. The DCE must stop transmission of data at the conclusion of the current character and wait until the DTE sets RTS true before resuming transmission.

This modified RTS/CTS handshake protocol provides full bi-directional hardware handshaking of the data streams. The HP 64700B can support baud rates up to 460800 using this protocol.

S13: Software pacing uses XON/XOFF protocols (DC1/DC3). Upon receipt of an XOFF, the HP 64700B can continue to transmit up to 3 additional characters. The HP 64700B sends an XOFF when its internal buffer can accept only 64 additional bytes before overflow. Software pacing is only valid on the transmission of ASCII data streams. It is not supported for binary transfers. It will support a maximum baud rate of 57600. Above this rate hardware handshaking must be used to prevent data loss.

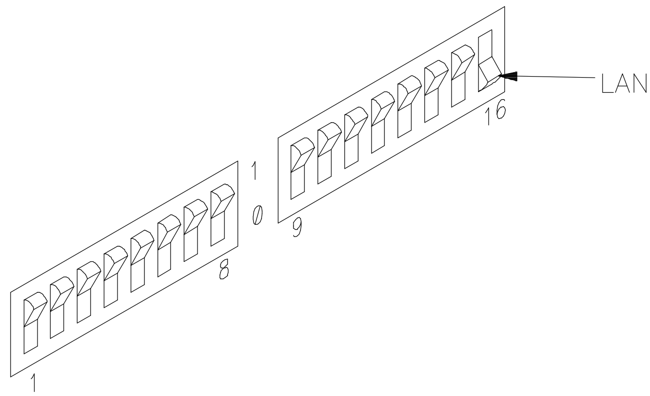
2 Select and connect the RS-232/RS-422 cable.

To connect cables to the HP 64700, simply align the cable with the serial port and insert the 25-pin male connector of the cable until it is firmly seated. You should then tighten the holding screws on each side of the cable with a small flat blade screwdriver. This will ensure that the cable pins and shield hood make good contact with the HP 64700 connector and will also guard against accidental disconnection of the cable.

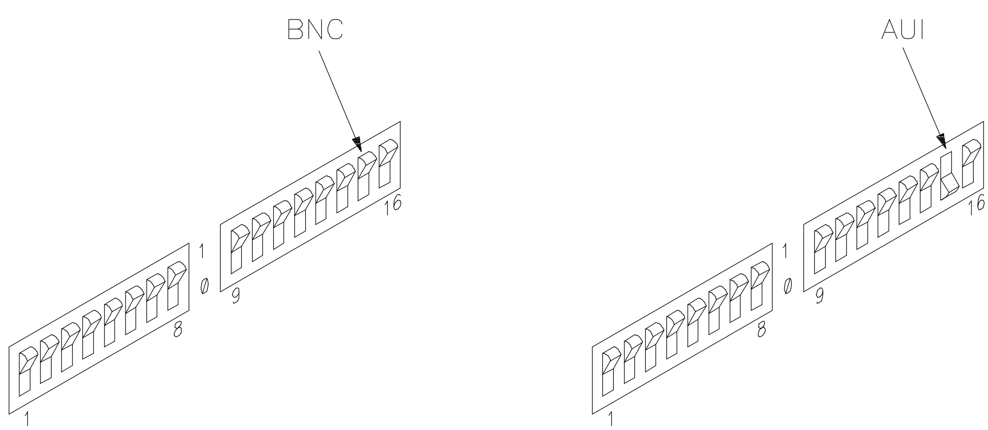


Step 2b. Connect the HP 64700 via LAN

1 Enable the LAN interface. If you are using the HP 64700's LAN interface, you must enable it by setting switch S16 is set to one (1). Set all other switches (S1 through S13) to zero.



2 Select the BNC or 15-pin AUI port. S15 is used to select which of the HP 64700's LAN connectors will be used: either the BNC connector (S15 = 0) or the 15-pin AUI connector (S15 = 1).

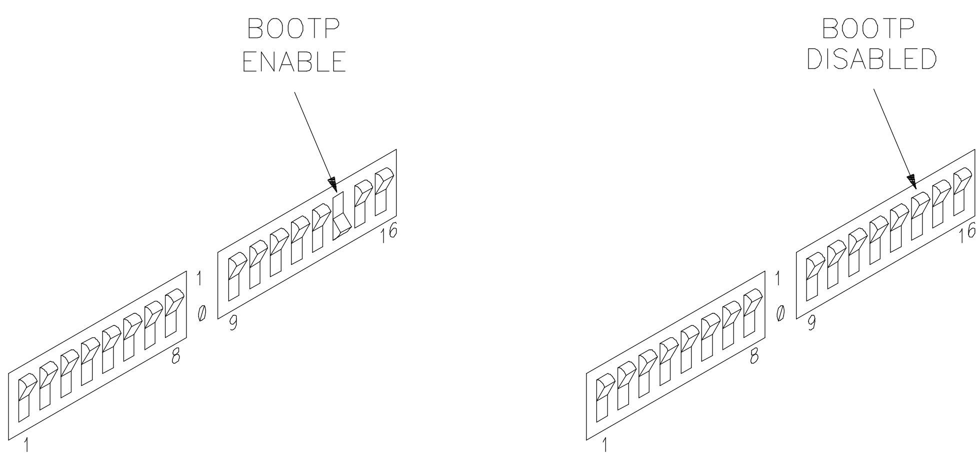


3 Enable or disable BOOTP.

BOOTP is a network service running on a host computer that allows the HP 64700's LAN parameters to be set automatically when the emulator is powered up.

When S14 is set to (1) and the host computer's "bootptab" table file has been modified to include information for the HP 64700, BOOTP will be used to set the HP 64700's LAN parameters when the emulator is powered up.

When S14 is set to zero (0), BOOTP is disabled and LAN parameters must be set by connecting the HP 64700 to a terminal or host computer via the serial port (as described in the previous Step 2a) and use the Terminal Interface **lan** command to set the HP 64700's LAN parameters. Once the LAN parameters are set (they are saved in EEPROM), you can change the configuration switch settings and connect the HP 64700 to the LAN.

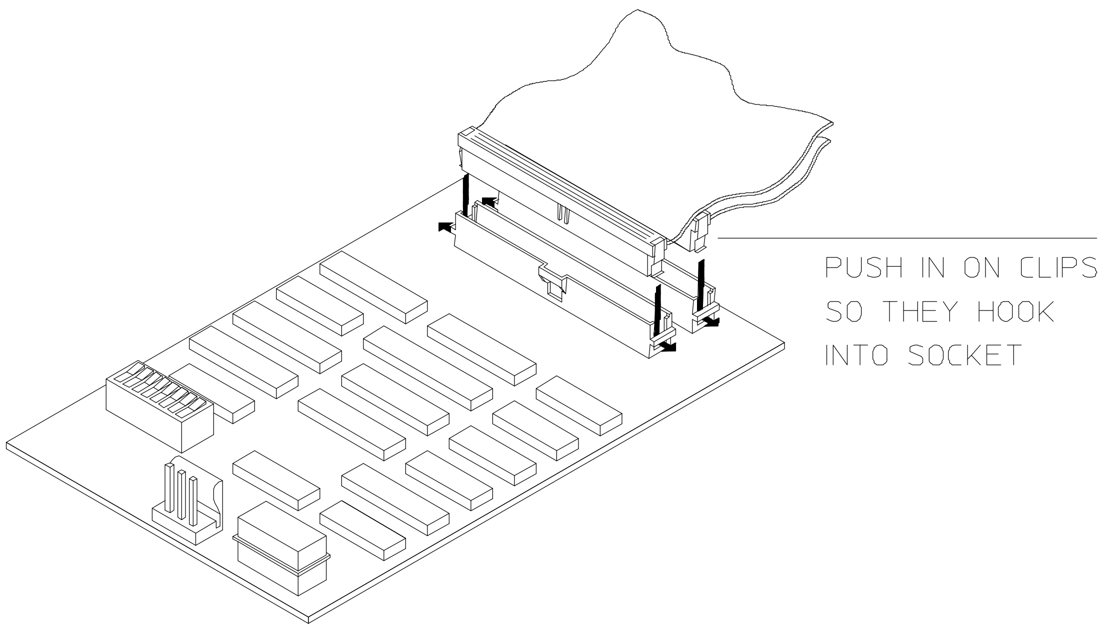


Step 3. Connect the demo board probe adapter

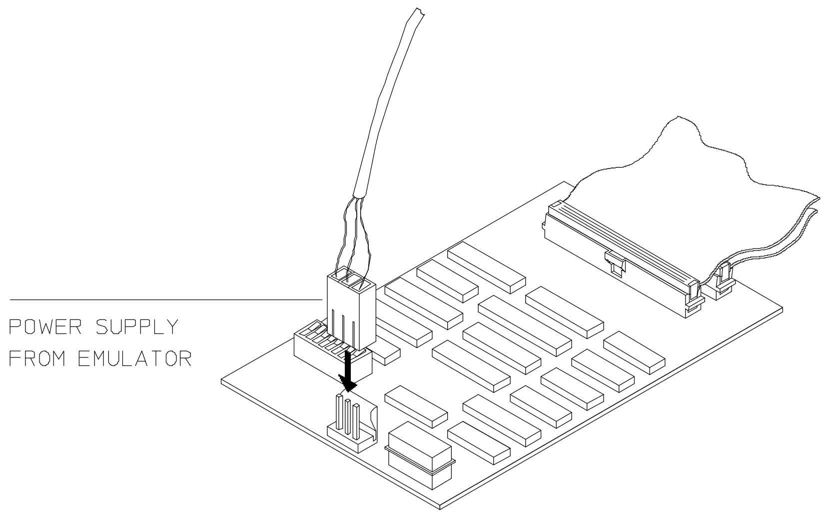
1 With HP 64700 power OFF, connect the emulator probe cables to the 64744-66509 demo board.

The emulator requires an external clock signal in order to run. Therefore, a target system is required in order to use the emulator, and the minimum target system must provide a clock. The demo board is one such minimum target system.

Also, the demo board must be used when verifying the emulator's performance.



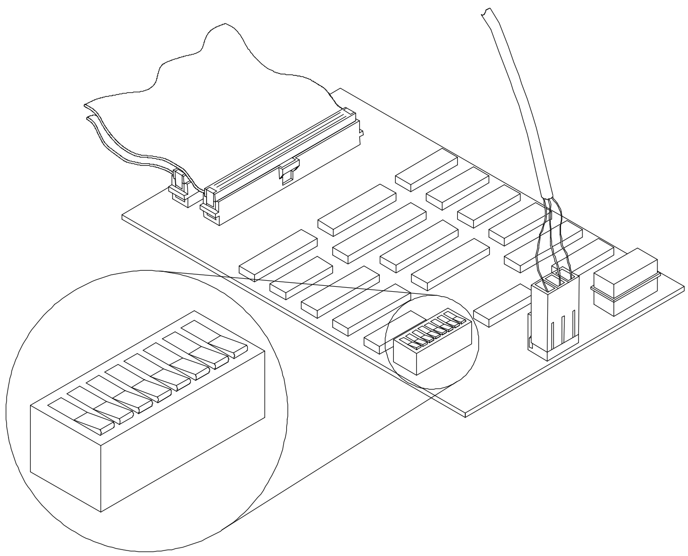
2 Connect the power supply wires from the emulator to the demo board. The 3-wire cable has 1 power wire and 2 ground wires. **When attaching the 3-wire cable to the demo board, make sure the connector is aligned properly so that all three pins are connected**.



3 Set the demo board SW1 switches.

If you're using the demo board to run performance verification tests on the emulator, all switches should be closed.

If you're using the demo board for out-of-circuit emulation, switches 1 through 8 should be open.



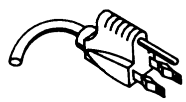
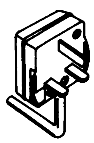
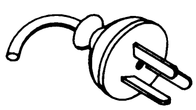
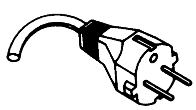
Step 4. Apply power to the HP 64700

The HP 64700B automatically selects the 115 Vac or 220 Vac range. In the 115 Vac range, the HP 64700B will draw a maximum of 345 W and 520 VA. In the 220 Vac range, the HP 64700B will draw a maximum of 335 W and 600 VA.

The HP 64700 is shipped from the factory with a power cord appropriate for your country. You should verify that you have the correct power cable for installation by comparing the power cord you received with the HP 64700 with the drawings under the "Plug Type" column of the following table.

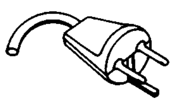
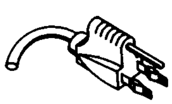
If the cable you received is not appropriate for your electrical power outlet type, contact your Hewlett-Packard sales and service office.

Power Cord Configurations

Plug Type	Cable Part No.	Plug Description	Length in/cm	Color
Opt 903 124V ** 	8120-1378 8120-1521	Straight * NEMA5-15P 90^{o}	90/228 90/228	Jade Gray Jade Gray
Opt 900 250V 	8120-1351 8120-1703	Straight * BS136A 90^{o}	90/228 90/228	Gray Mint Gray
Opt 901 250V 	8120-1369 8120-0696	Straight * NZSS198/ASC 90^{o}	79/200 87/221	Gray Mint Gray
Opt 902 250V 	812001689 8120-1692 8120-2857	Straight * CEE7-Y11 90^{o} Straight (Shielded)	79/200 79/200 79/200	Mint Gray Mint Gray Coco Brown

* Part number shown for plug is industry identifier for plug only.
Number shown for cable is HP part number for complete cable including plug.
** These cords are included in the CSA certification approval for the equipment.

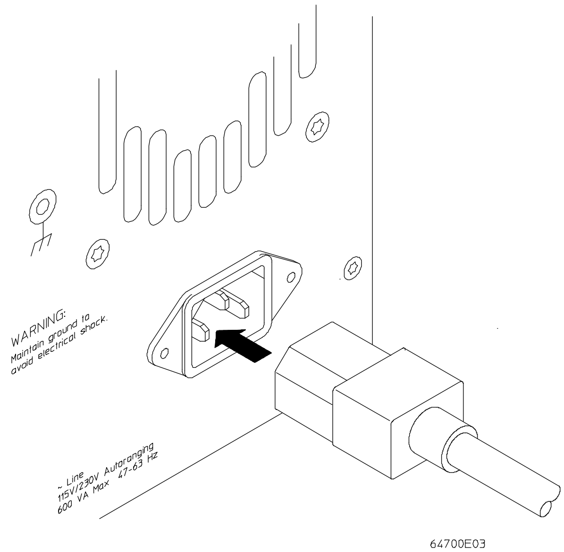
Power Cord Configurations (Cont'd)

Plug Type	Cable Part No.	Plug Description	Length in/cm	Color
Opt 906 250V 	8120-2104 8120-2296	Straight * SEV1011 1959-24507 Type 12 90^o	79/20 79/200	Mint Gray Mint Gray
Opt 912 220V 	 8120-2957	Straight *DHCK107 90^o	79/200 79/200	Mint Gray Mint Gray
Opt 917 250V 	8120-4600 8120-4211	Straight SABS164 90^o	79/200 79/200	Jade Gray
Opt 918 100V 	8120-4753 8120-4754	Straight Miti 90^o	90/230 90/230	Dark Gray

* Part number shown for plug is industry identifier for plug only.
Number shown for cable is HP part number for complete cable including plug.
** These cords are included in the CSA certification approval for the equipment.

1 Connect the power cord and turn on the HP 64700.

The line switch is a push button located at the lower left hand corner of the front panel. To turn ON power to the HP 64700, push the line switch button in to the ON (1) position. The power light at the lower right hand corner of the front panel will be illuminated.



When the emulator powers up, it sends a message (similar to the one that follows) to the selected command port and then displays the Terminal Interface prompt. You can verify that your data communications configuration is at least partially functional by looking for the message and prompt on the controlling device (terminal or terminal emulation program running on a host computer).

```
         Copyright (c) Hewlett-Packard Co. 1987
All Rights Reserved.  Reproduction, adaptation, or translation without prior
written permission is prohibited, except as allowed under copyright laws.

 HP64700B Series Emulation System
  Version:   B.01.00 20Dec93
  Location:  Flash
  System RAM:1 Mbyte

 HP64744A (PPN: 64742A) Motorola 68000/68HC001/68EC000 Emulator
 HP64740 Emulation Analyzer with External State/Timing Analyzer
R>
```

If the HP 64700 does not provide the Terminal Interface prompt

When using the RS-232/RS-422 interface:

If the HP 64700 does not provide the Terminal Interface prompt to the controlling device when power is applied:

- [ ] Make sure that you have connected the emulator to the proper power source and that the power light is lit.

- [ ] Make sure that you have properly configured the data communications switches on the emulator and the data communications parameters on your controlling device. You should also verify that you are using the correct cable.

The most common type of data communications configuration problem involves the configuration of the HP 64700 as a DCE or DTE device and the selection of the RS-232 cable. If you are using the wrong type of cable for the device selected, no prompt will be displayed.

When the serial port is configured as a DCE device, a modem cable should be used to connect the HP 64700 to the host computer of terminal. Pins 2 and 3 at one end of a modem cable are tied to pins 2 and 3 at the other end of the cable.

When the serial port is configured as a DTE device, a printer cable should be used to connect the HP 64700 to the host computer of terminal. Pins 2 and 3 at one end of a printer cable are swapped and tied to pins 3 and 2, respectively, at the other end of the cable.

If you suspect that you may have the wrong type of cable, try changing the S4 setting and cycling power.

When using the LAN interface:

You must use the **telnet** command on the host computer to access the HP 64700. After powering up the HP 64700, it takes a minute before the HP 64700 can be recognized on the network. After a minute, try the **telnet <internet address>** command.

If **telnet** does not make the connection:

- ☐ Make sure that you have connected the emulator to the proper power source and that the power light is lit.
- ☐ Make sure that the LAN cable is connected. Refer to your LAN documentation for testing connectivity.
- ☐ Make sure that the HP 64700's Internet Address is set up correctly. You must use the RS-232/RS-422 port to verify this that the Internet Address is set up correctly. While you accessing the emulator via the RS-232/RS-422 port, run performance verification on the LAN interface with the **lanpv** command. See "To run PV on the LAN interface".

If **telnet** makes the connection, but no Terminal Interface prompt is supplied:

- ☐ It's possible that the HP 64000 software is in the process of running a command (for example, if a repetitive command was initiated from telnet in another window). You can use <CTRL>c to interrupt the repetitive command and get the Terminal Interface prompt.
- ☐ It's also possible for there to be a problem with the HP 64700 firmware while the LAN interface is still up and running. In this case, you must cycle power on the HP 64700.

To run PV on the LAN interface

1 Connect a host computer or terminal to the HP 64700 using the RS-232 interface.

The HP 64700 LAN interface can be tested through a Terminal Interface command called **lanpv**. You can only use this command when communicating with the HP 64700 over an RS-232 connection. Do not use this command when communicating with the HP 64700 over the LAN.

2 Disconnect the HP 64700 from the LAN and terminate the HP 64700's LAN port you want to test.

Before you run the test, the HP 64700 must be disconnected from the network.

The connector you wish to test must be completely terminated, and the other connector must not be terminated. Only one connector can be tested at a time.

To properly terminate the BNC port, place a BNC "T" connector on the port and place 50 ohm terminators on each end of the T-connector.

To properly terminate the 15-pin AUI port, leave the MAU attached to the port and, using the appropriate loopback hood or loopback connector, terminate the end of the MAU that is normally connected to the LAN.

3 Access the Terminal Interface and enter the **lan -va** command to test the 15-pin AUI connector or the **lan -vb** command to test the BNC connector.

This command will return "PASSED" or "FAILED" before issuing a prompt. For example, to test the BNC connector:

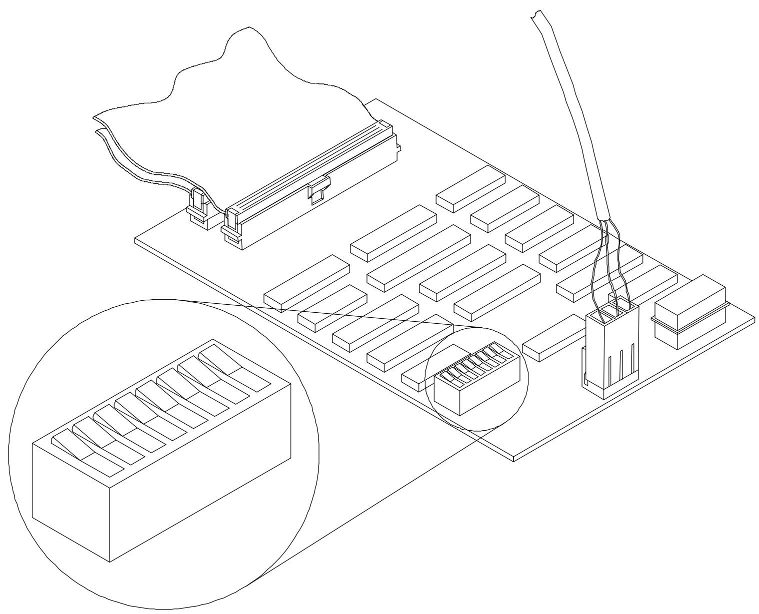
```
R>lanpv -vb
  Testing: HP 64700B LAN interface (BNC connector)
      PASSED
```

Step 5. Verify emulator and analyzer performance

The emulator probe cables must be connected to the 64744-66509 demo board when you run the performance verification tests.

After the emulator probe cables are connected to the 64744-66509 demo board (make sure the power lines from the emulator are connected to the demo board), power has been applied to the HP 64700, and the HP 64700 has supplied the Terminal Interface prompt to the controlling device, you can run performance verification tests on the emulator and analyzer.

1 Make sure that all the switches of SW1 on the demo board are closed.



2 Type the "pv" command, along with the number of times you want to execute the command.

For example:

```
R>pv 1
```

```
  Testing: HP64744A Motorola 68000/68HC001/68EC000 Emulator
  Emulation trace started
  --- Emulation Trace Status ---
  NEW User trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 1024 (1024) -3..1020
  Sequence term 2
  Occurrence left 1
  Emulation trace started
  Emulation trace halted
    PASSED
    Number of tests: 1           Number of failures: 0
  Testing: HP64740 Emulation Analyzer with External State/Timing Analyzer
    PASSED
    Number of tests: 1           Number of failures: 0

                  Copyright (c) Hewlett-Packard Co. 1987
All Rights Reserved.  Reproduction, adaptation, or translation without prior
written permission is prohibited, except as allowed under copyright laws.

  HP64700B Series Emulation System
    Version:   B.01.00 20Dec93
    Location:  Flash
    System RAM:1 Mbyte

  HP64744A Motorola 68000/68HC001/68EC000 Emulator
  HP64740 Emulation Analyzer with External State/Timing Analyzer
R>
```

If performance verification fails

☐ Make sure the emulator probe cables are connected to the 64744-66509 demo board correctly (see Step 3) and that the power lines from the emulator are connected to the demo board.

☐ Make sure the emulator and analyzer boards have been installed into the HP 64700 Card Cage correctly (see Step 1) and that there are no bent or broken pins on any of the connectors.

If this does not seem to solve the problem, call the nearest Hewlett-Packard Sales and Service office listed in the *Support Services* manual.

Glossary

access mode Specifies the types of cycles used to access target system memory locations. For example a "byte" access mode tells the monitor program to use load/store byte instructions to access target memory.

active emulator probe An emulator probe that contains circuitry that allows the emulator to more closely imitate the electrical characteristics of the microprocessor thereby avoiding the timing problems that can occur with passive probes.

analyzer An instrument that captures data on signals of interest at discreet periods.

background The emulator mode in which foreground operation is suspended so the emulation processor can be used for communication with the emulation controller. The background monitor does not occupy any processor address space.

background emulation monitor An emulation monitor program that does not execute as part of the user program, and therefore, operates in the emulator's background mode.

background memory Memory space reserved for the emulation processor when it is operating in the background mode. Background memory does not take up any of the processor's address space.

coverage measurements Provide information about memory accesses or code execution within a range of memory.

display mode When displaying memory, this mode tells the emulator the size of the memory locations to display. When modifying memory, the display mode tells the emulator the size of the values to be written to memory.

embedded microprocessor system The microprocessor system that the emulator plugs into.

emulation bus analyzer The internal analyzer that captures emulator bus cycle information synchronously with the processor's clock signal.

emulation monitor program A program that is executed by the emulation processor which allows the emulation controller to access target system resources. For example, when you display target system memory locations, the monitor program executes microprocessor instructions that read the target memory locations and send their contents to the emulation controller.

emulator An instrument that performs just like the microprocessor it replaces, but at the same time, it gives you information about the operation of the processor. An emulator gives you control over target system execution and allows you to view or modify the contents of processor registers, target system memory, and I/O resources.

foreground The mode in which the emulator is executing the user program. In other words, the mode in which the emulator operates as the target microprocessor would.

global restart When the same secondary branch condition is used for all terms in the analyzer's sequencer, and secondary branches are always back to the first term.

prestore The analyzer feature that allows up to two states to be stored before normally stored states. This feature is useful when you want to find the cause of a particular state. For example, if a variable is accessed from many different places in the program, you can qualify the trace so that only accesses of that variable are stored and turn on prestore to find out where accesses of that variable originate from.

primary sequencer branch Occurs when the analyzer finds the primary branch state specified at a certain level and begins searching for the states specified at the primary branch's destination level.

real-time Refers to continuous execution of the user program without interference from the emulator. (Such interference occurs when the emulator temporarily breaks into the monitor so that it can access register contents or target system memory or I/O.)

secondary sequencer branch Occurs when the analyzer finds the secondary branch state specified at a certain level before it found the primary branch state and begins searching for the states specified at the secondary branch's destination level.

sequence terms Individual levels of the sequencer. The analyzer provides 8 sequence terms.

sequencer The part of the analyzer that allows it to search for a certain sequence of states before triggering.

sequencer branch Occurs when the analyzer finds the primary or secondary branch state specified at a certain level and begins searching for the states specified at another level.

target system The microprocessor system that the emulator plugs into.

trace A collection of states captured on the emulation bus (in terms of the emulation bus analyzer) or on the analyzer trace signals (in terms of the external analyzer) and stored in trace memory.

trigger The captured analyzer state about which other captured states are stored. The trigger state specifies when the trace measurement is taken.

Index

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Safety

Summary of Safe Procedures

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground The Instrument

To minimize shock hazard, the instrument chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate In An Explosive Atmosphere

Do not operate the instrument in the presence of flammable gases or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with the power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service Or Adjust Alone

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

Do Not Substitute Parts Or Modify Instrument

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

WARNING Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

The following is a list of general definitions of safety symbols used on equipment or in manuals:

Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.

Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).

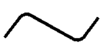
 OR 

Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.

Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual before operating the equipment.

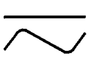
 OR 

Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.



Alternating current (power line).

Direct current (power line).



Alternating or direct current (power line).

Caution

The Caution sign denotes a hazard. It calls your attention to an operating procedure, practice, condition, or similar situation, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product.

Warning

The Warning sign denotes a hazard. It calls your attention to a procedure, practice, condition or the like, which, if not correctly performed, could result in injury or death to personnel.