User's Guide

Emulation-Bus Analyzer with deep trace memory (HP 64794)

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
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.

A software code may be printed before the date below; 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 **64794-97000** **June 1993**

Safety information and Certification and Warranty

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

In This Book

This manual documents the HP 64794 Emulation-Bus Analyzer with deep trace memory. It supplements the manuals you received with your emulator/analyzer product. Use those manuals for operating and service of your emulator/analyzer product, except as noted in this manual. This manual contains:

Chapter 1 presents an overview of the Emulation-Bus Analyzer with deep trace memory and shows you differences between it and the Emulation-Bus Analyzer with 1K trace memory it is designed to replace.

Chapter 2 discusses tasks that are available to you when using the Emulation-Bus Analyzer with deep trace memory. These tasks are not available when using the Emulation-Bus Analyzer with 1K trace memory it replaces. Tasks that can be performed the same in both analyzers are not described in this manual.

Chapter 3 discusses the differences between the interface of the Emulation-Bus Analyzer with deep trace memory and the interface of the Emulation-Bus Analyzer with 1K trace memory that is described in the emulator/analyzer manual(s) you received from Hewlett-Packard.

Chapter 4 discusses two concepts that will help you understand functions of the Emulation-Bus Analyzer with deep trace memory: trace memory depth, and operation of the trace tag counter.

Chapter 5 discusses error and status messages that you may see when using the Emulation-Bus Analyzer with deep trace memory.

Chapter 6 shows how to install and service the Emulation-Bus Analyzer with deep trace memory.

In this manual, the HP 64794 Emulation-Bus Analyzer with deep trace memory is called the deep analyzer. The HP 64703, HP 64704, and HP 64706 Emulation-Bus Analyzers with 1K trace memories are called the 1K analyzer.

Contents

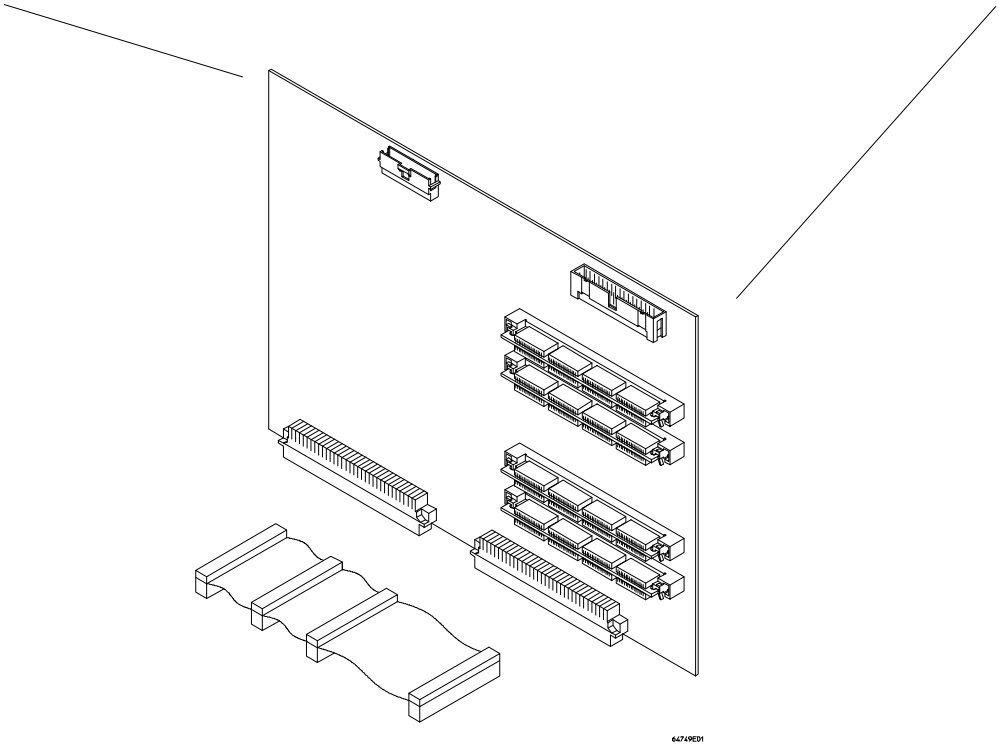
1

Deep Analyzer, at a glance

Deep Analyzer - at a glance

Sample terminal interface trace list

```
U>tcf -deep
U>tl -d -e 5320..5328
     Line  addr,H    68040 Mnemonic                                    count,R
  -------  --------  ----------------------------------------- -------------
     5320  000008f8  BLE.B     $000008E4                        -------------
          =000008fa  MOVEQ     #$00000050,D0
     5321  000008fc  CMP.L     D2,D0                                  0.08uS
          =000008fe  BLS.B     $000008E4
     5322  000008e8  TST.B     ($00,A2,D2.L)                          0.08uS
     5323  000008ec  BEQ.B     $000008F6                              0.08uS
          =000008ee  ADDQ.L    #1,D2
     5324  000008f0  MOVEQ     #$00000050,D0                          0.14uS
          =000008f2  CMP.L     D0,D2
     5325  _sysbuf    $00------ log sdata byte read                   0.08uS
     5326  000008f4  BCS.B     $000008E8                              0.08uS
          =000008f6  TST.L     D2
     5327  000008f0  MOVEQ     #$00000050,D0                          0.12uS
          =000008f2  CMP.L     D0,D2
     5328  000008f4  BCS.B     $000008E8                              0.08uS
          =000008f6  TST.L     D2
```



This chapter describes the HP 64794 Emulation-Bus Analyzer with deep trace memory, and lists differences between it and the HP 64703, HP 64704, and HP 64706 Emulation-Bus Analyzers with 1K trace memories it is designed to replace.

Note

In this manual, the HP 64794 Emulation-Bus Analyzer with deep trace memory will be called the deep analyzer. The HP 64703, HP 64704, and HP 64706 Emulation-Bus Analyzers with 1K trace memories will be referred to as the 1K analyzer.

The following information is covered in this chapter:

- Software versions required for compatibility with the deep analyzer.
- General discussion of differences between the deep analyzer and the 1K analyzer it replaces.
- Discussion of the purpose and use of the compatible mode and the deep mode of the deep analyzer.

Software compatibility

To use the deep analyzer with an HP 64700 Series Emulation System, the firmware of the emulation system and the software of the hosted user interface (if used) must be compatible with the deep analyzer. If you are using a hosted interface that is not compatible with the deep analyzer, the deep analyzer will automatically confine its operation to the specifications of the 1K analyzer. Refer to "Compatible mode vs deep mode" later in this chapter. The following paragraphs show you how to find your software versions and determine whether or not they are compatible with the deep analyzer.

Compatibility with the HP64700 Series Emulation System firmware

The HP 64700 emulation system is composed of several subsystems, each of which contains its own firmware and unique software version. The way to see the firmware versions of these subsystems depends on the interface you are using. The method to use in each interface is described below:

- Terminal Interface: Type **ver** beside your system prompt.
- Softkey User Interface: Enter the following commands:

  display pod_command
  pod_command "ver"

- Graphical User Interface: Choose **Settings→Pod Command Keyboard**. Move the cursor to the command line and type **ver**. Then press the **suspend** softkey.
- PC Interface: Select **S**ystem **T**erminal. Type **ve**r. Then press the **Ctr**l-\ keyboard keys.
- Real-Time C Debugger Interface (for PC): Choose the Help→About Debugger/Emulator (ALT,H,A) command.

To use the deep analyzer in an HP 64700 series emulation system, you must have system firmware version A.03.00, or higher. Any version of the emulator or LAN subsystem firmware may be used with the deep analyzer. (Future versions of the emulation subsystem firmware may automatically switch on features of the deep analyzer that are not available in the 1K analyzer.) The deep analyzer contains its own firmware; therefore, it is always compatible.

```
->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.

HP64700 Series Emulation System
  Version:   A.03.00 13Dec90

HP64742C/D Motorola 68000 emulator
  Version:   A.00.09 16Mar92
  Speed:     12.5 MHz
  Memory:    510 KBytes

HP64740 Compatible (PPN: 64794) Deep Emulation Analyzer
  Version:   A.03.00 27May93
  PC Board:  794-01-A
  Depth:     80ch X 1K states selected, 80ch X 8K states available
    Bank 1:    not loaded
    Bank 2:    not loaded
    Bank 3:    not loaded
    Bank 4:    not loaded

HP64701A LAN Interface
  Version:   A.00.05 18Mar93

->
```

← This firmware version number must be A.03.00 or later version number. Ignore the other product firmware versions in this list.

Compatibility with the hosted user interface software

Hosted user interfaces for emulation subsystems are available for execution on workstations and PCs. Each hosted user interface has its own software version, which is displayed when the interface first appears on screen.

The following table lists the hosted user interfaces of the emulator/analyzer and shows versions of interface software you must use in order to obtain the full features of the deep analyzer. (All software versions of the terminal interface allow use of the full features of the deep analyzer.)

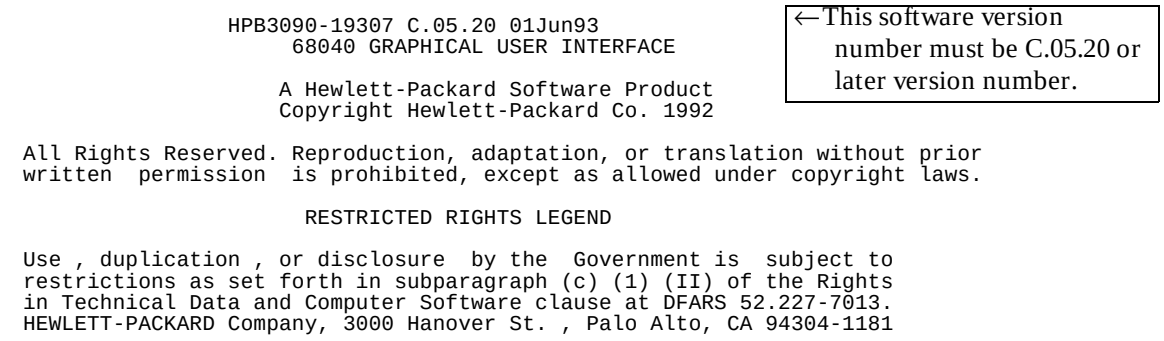
Emulator/analyzer interface	**Minimum software version required**
Workstation interfaces:	
Softkey User Interface	A.05.20 or higher.
Graphical User Interface	C.05.20 or higher.
PC interfaces:	
PC interface (older interface to PC)	not supported in any software version. (However, the deep analyzer can replace the 1K analyzer to provide improved counting speed and resolution with no tradeoffs).
Real-Time C Debugger interface	all software versions.

Workstation interfaces

Workstation interfaces for the most popular microprocessor emulators have been updated to support deep analyzer capabilities. These workstation interfaces have software version C.05.20 or higher. Older software versions of workstation interfaces will work with the deep analyzer but will limit the depth to 512/1K states (refer to compatible mode, discussed later in this chapter).

You can check the software version number of your Graphical User Interface or Softkey User Interface in two ways:

- The software version number of your interface is shown in the first display after you turn power on.



- If you are working in the Graphical User Interface, choose **Help→Version...** The software version number will be shown when the new display window opens.

PC interfaces

The PC Interface will not be updated for the deep analyzer; therefore, it only provides a 1K trace depth. However, the deep analyzer can replace the 1K analyzer when you use the PC Interface, and it will provide improved counting and resolution.

A new version of the PC interface (soon to be released) is called the Real-Time C Debugger Interface. It will replace the PC Interface. The first release of the Real-Time C Debugger Interface will support deep analyzer capabilities.

Overview of differences between the deep analyzer and the 1K analyzer

The deep analyzer has a width of 80 data channels. It can be used to replace 1K analyzers having 48 channels, 64 channels, and 80 channels. Some versions of the 1K analyzer also offered external analysis channels. The deep analyzer does not offer external analysis channels.

Differences between the deep analyzer and the 1K analyzer consist of the depth of the trace memory, and counting capability at different analysis clock rates. When using the 1K analyzer, the types of trace counting that can be performed are limited by your analyzer clock rate. Counts of time and counts of occurrences of selected states can only be made with clock rates below specified speeds. Also, you must sacrifice trace memory depth when making these counts. The deep analyzer can perform counts of time between trace states, and counts of occurrences of selected states regardless of the emulator clock speed, and with no sacrifice of trace memory depth. Specific differences between the deep analyzer and the 1K analyzer are summarized in the table below. Specifications not shown in the table are the same for the deep analyzer and the 1K analyzer.

Specification	Deep Analyzer	1K Analyzer
Trace Depth	8K base Expandable to 64K or 256K using optional plug-in memory modules Depth does not change when counting is used	1K without time or state count 512 with time or state count
Counting vs Analyzer Clock Speed	Works at full speed with all supported emulators There are no feature tradeoffs with analyzer clock speed	25 Mhz without time or state count 20 MHz with state count but without time count 16.66 MHz with time count or state count

Overview of differences between the deep analyzer and the 1K analyzer

Specification	Deep Analyzer	1K Analyzer
Time Count Resolution	20 nsec Full resolution retained for up to 22.9 minutes Resolution retained after 22.9 minutes, but no indication is given for the number of times counter overflowed	40 nsec If greater than 82 usec between stored states, resolution is reduced
Maximum State Count at full Resolution	68,719,476,735 (2^{36} -1)	If greater than 2047 counts between stored states, resolution is reduced
Maximum Prestore Memory	1 prestore state per store state	2 prestore states per store state
Cross Trigger/Break Selections Available using TRIG1/TRIG2	Drive Sources for TRIG1/TRIG2: - Trace Point - Trace Complete - N states before trace complete - Any resource or combination of resources (pattern, range or arm) or combination of resources and sequencer states. Receive TRIG1/TRIG2	Drive sources for TRIG1/TRIG2: - Trace Point Receive TRIG1/TRIG2
Analyzer Status	Trace Activity command not implemented	Channel activity shows h, l, or t in response to terminal interface command "ta"
Firmware Upgrade Requirements	Flash ROMs built in - no additional requirements	Requires additional Flash Board to upgrade firmware

Compatible mode vs deep mode

The deep analyzer has a compatible mode and a deep mode. By default, the deep analyzer is initialized in the compatible mode. When in the compatible mode, the deep analyzer appears to be the 1K analyzer; it has the same trace memory limitations as the 1K analyzer: 1024 states of maximum depth, and 512 states when making time or state counts. Some emulators have high level interfaces that cannot work with an analyzer that has a trace memory depth greater than 1K. Therefore, in order to be compatible with all of the emulator interfaces that might use the deep analyzer, the compatible mode is the default mode at power up.

If your Graphical User Interface or Softkey Interface can work with the full memory depth of the deep analyzer, your system will automatically switch to the deep mode during startup. All Real-Time C Debugger Interfaces switch to the deep mode. If your emulator interface cannot work with the full memory depth of the deep analyzer (refer to the paragraph titled "Software Compatibility"), it will remain in the compatible mode; you will still be able to use the counting abilities of the deep analyzer to make state and time counts during measurements at full analysis speed.

Note that all emulator terminal interfaces can work with the full memory depth of the deep analyzer. If your high-level interface is one that cannot work with the full memory depth of the deep analyzer, and if you must make a measurement that requires the depth of the deep analyzer, you can use the terminal interface of your emulator/analyzer to make the measurement; then switch back to your high-level interface after you have completed your measurement. Make sure you return the analyzer to the compatible mode before switching back to your high-level interface. The terminal interface command that selects memory depth is **tcf** (trace configuration). Refer to Chapter 2 of this manual for details of how to set memory depth in the terminal interface.

Note that certain terminal interface commands can cause problems within the high-level interface. The high-level interface may not know about configuration changes made by using terminal interface commands. A high-level interface command may default a setup you made through a terminal interface command. For additional information, read the "---WARNING---" statement that appears on screen when you first access the the pod_command display for terminal interface commands.

The following table lists emulators whose workstation interfaces have been enhanced to work with the greater trace memory depths of the deep analyzer (refer to the paragraph titled "Software Compatibility"). If your emulator is not listed in the table below, it may still work with the deep analyzer, but you may have to accept the limitation of 1024 states of trace memory depth, and the memory depth tradeoffs when using its high-level interface.

Emulators That Can Use The Full Power Of The Deep Analyzer			
HP Model No.	**For Motorola Processors**	**HP Model No.**	**For Intel Processors**
64742	68000	64762/64763	8086/88
	68EC000		80C86/C88
	68HC001	64764/64765	80186/188
64748	68020/EC020		80C186/C188
64747	68030/EC030		80C186EA/C188EA
64783	68040/EC040/LC040		80C186EB/C188EB
64749	68331/332	64767	80C186XL/C188XL
64746	68302		
64751	68340		

At the time this manual was printed, the following emulators were not able to work with the deep analyzer:

- HP 64774, 29000 Emulator - because of power supply limitations in the card cage containing the emulator.
- HP 64760, 80960K Emulator - because it uses a special analyzer, not the 1K analyzer that is replaced by the deep analyzer.
- HP 64759, 78K2 Emulator - because this emulator uses special clocking hardware, which is available in the 1K analyzer, but not in the deep analyzer.

2

Unique Deep Analyzer Tasks

This chapter discusses tasks that are available to you when using the deep analyzer. These tasks are not available when using the 1K analyzer. Types of tasks in this chapter include:

- Tasks you need to do when using the deep analyzer that you do not need to do when using the 1K analyzer.
- Example measurements you may wish to make with the deep analyzer that are not possible with the 1K analyzer.

The example tasks described in this chapter are shown being performed in one interface and not in the others. The reasons one interface was chosen and not another to perform a task are:

- A workstation interface provides access to the host's file system for storing trace data.
- Certain cross-triggering and cross-arming features for coordinating measurements between emulators and analyzers are only available in commands of the terminal interface.

Note

Tasks that can be performed in the 1K analyzer the same as in the deep analyzer are not described in this chapter. Refer to the emulator/analyzer manuals you received with your emulator.

Tasks that must be performed in the workstation interface

If you are using the Graphical User Interface or the Softkey User Interface, you will need to perform the following tasks when using the deep analyzer:

- Setting the depth of the trace memory that will be unloaded after a measurement.
- Setting the format for display of the trace list.

To set trace memory depth to be unloaded in the deep analyzer

- Choose **Trace→Display Options ...**, and in the dialog box, enter the desired **< DEPTH>** in the entry field beside Unload Depth. Then click OK or Apply.

- Using the command line, enter **display trace depth < DEPTH>** .

  < DEPTH> is the portion of the trace memory that the deep analyzer will unload for display or file transfer after capturing new trace data. You can specify unload of any depth from 9 states to the maximum depth of the trace memory of your deep analyzer. If you enter zero (0) as the unload **< DEPTH>** , the analyzer will be set to unload its full depth.

  The unload depth you specify will affect how quickly the analyzer responds to a **wait measurement_complete** command. The analyzer detects measurement complete after the trigger has been captured, the trace memory has been filled to the unload depth, and the unload depth has been unloaded. Measurement complete is also detected if the trace has been halted and all available data has been unloaded.

  You can respecify the unload depth after the trace is complete. For example, you may have captured 256K states, but specified unload of only 8K states. After viewing them, you may specify unload of 16K states. By specifying an unload depth of 16K, the trace unload will be completed more quickly than if you specified unload of the full 256K states. When you increase the unload depth, the additional depth will be unloaded as long as a new trace has not started.

  Trace data is always unloaded around the trigger position. For example, if you specified unload of 8K memory depth:

  - If trigger is the first state in memory (**trace after**), the first 8K states in memory will be unloaded.
  - If trigger is the center state in memory (**trace about**), the 4K states before trigger and the 4K states after trigger will be unloaded.
  - If trigger is the last state in memory (**trace before**), the 8K states before trigger will be unloaded.

Examples

To unload the first 4K memory depth for display in your deep analyzer:

Choose **Trace→Display Options ...**, and in the dialog box, type 4096 in the entry field beside Unload Depth. Then click OK or Apply.

On the command line, enter:

```
display trace depth 4096
```

To unload the full memory depth in your deep analyzer:

Choose **Trace→Display Options ...**, and in the dialog box, type either 0 or the full memory depth in the entry field beside Unload Depth. Then click OK or Apply.

On the command line, enter:

```
display trace depth 0 or <full memory depth>
```

To format the trace list display

- Choose **Trace→Display Options ...**, and in the dialog box, select the desired format for the display of trace data beside Data Format. Mnemonic provides a trace list that combines the information in the data and status fields into an assembly language program listing. Absolute provides a trace list that shows numerical values to represent the data and status fields in trace memory. Real Time OS shows a trace list obtained from the HP Real Time Operating System product, if it is installed. Note that some emulated processors do not supply sufficient status to identify the start of an instruction. If your emulator is for one of these processors, you must enter a separate command to tell the analyzer where to begin trace disassembly: either use the **disassemble from** option in the display trace popup menu, or enter a **display trace disassemble_from_line_number < NO.>** command on the command line. Also, you can move the cursor over the desired starting line in the trace list and single-click the right mouse button.

- If you selected Data Format Absolute, select the desired type of status display beside Status Format. Mnemonic obtains a mnemonic representation of the value of the status bits. Hex shows the value of the status bits as a hexadecimal number. Binary shows the values of the individual status bits as a binary number.

- Select the Count Mode desired. Relative shows the change in the count of time or states since the last displayed state in the trace list. Absolute shows the total count of time or states since the trigger state. Refer to the concepts chapter in this manual for details of counter overflow and how it affects the information shown in relative and absolute counts.

- Select Dequeue Enable, if desired and available for your emulator. Dequeuing a trace removes unexecuted prefetches from your trace list, and aligns instructions with their related operand cycles on the display. An undequeued trace list shows instructions and operand fetches in the order they occurred on the processor's buses.

- Select the desired Unload Depth. This is the depth of the trace memory that will be unloaded for display and/or file transfer after trace data is captured. Note that the entire trace memory will be used to store captured states

regardless of the Unload Depth you specify. Unload Depth is discussed in detail in the concepts chapter in this manual.

- In the entry field beside Address Offset, enter the value to be subtracted from the address and symbol/source line references of each instruction to yield the address that is displayed. You can specify an offset value to cause the listed addresses to match the addresses in compiler or assembler listings.

- Shift the display window to any desired location in the trace list by typing a trace memory line number in the entry field beside Move to Line. The display window will place the specified trace memory line number in the center of the screen and show trace information before and after it.

Example tasks you may wish to perform in the workstation interface

The example shown in this section uses commands in the Graphical User Interface and commands in the terminal interface. The terminal interface commands are accessed through the "pod_command" method available in the Graphical User Interface and Softkey User Interface. The reason that the workstation interface was chosen to perform this example task is that it provides access to the file system for storing trace data.

The following task is described in this section:

- How to capture all of the execution of your target program in a file or series of files.

To capture a continuous stream of program execution no matter how large your program

The following example shows you how to capture all of the execution of your target program. You may wish to capture target program execution for storage, for future reference, and/or for comparison with execution after making program modifications. The execution of a typical target program will require more memory space than is available in the trace memory of an analyzer. This example shows you how to capture all of your target program execution while excluding unwanted execution of the emulation monitor.

1 Choose **Trace→Display Options ...**, and in the dialog box, enter 0 or the total depth of your deep analyzer trace memory in the entry field beside Unload Depth. Then click OK or Apply. This sets unload depth to maximum.

2 For this measurement, the analyzer will drive trig1 and the emulator will receive trig1 from the trigger bus inside the 64700 card cage. The trig1 signal is used to cause the emulator to break to its monitor program shortly before the trace memory is filled. This use of trig1 is not supported in workstation interface commands. Therefore, terminal interface commands (accessible through the pod command feature) must be used. Enter the following commands:

Settings→Pod Command Keyboard
tgout trig1 -c < states before end of memory> (trigger output trig1 before trace complete)
bc -e trig1 (break conditions enabled on trig1)
Click the **suspend** softkey

Note that "tgout trig1 -c < states...> " means generate trig1 as an output when the state that is < states...> before the end of the trace memory is captured in the trace memory; "bc -e trig1" means enable the emulator to break to its monitor program when it receives trig1.

Select a value for **< states before end of memory>** that allows enough time and/or memory space for the emulator to break to its monitor program before the trace memory is filled. Otherwise, some of your program execution will not be captured in the trace. Many states may be executed before the emulation break occurs, depending on the state of the processor when the trig1 signal arrives. Also, if your program executes critical routines in which

interrupts are masked, the occurrence of trig1 may be ignored until the critical routine is completed (when using a foreground monitor).

3 If you are using a foreground monitor, enter the following additional pod commands to prevent the trace memory from capturing monitor execution. The following example commands will obtain this result in some emulators:

Settings→Pod Command Keyboard
trng addr= < address range occupied by your monitor> (trigger on range address = < address range>)
where < address range> is expressed as < first addr> ..< last addr>
tsto !r (trace store not range)
Click the **suspend** softkey

Note that "trng addr= < addr> ..< addr> " means define an address range for the analyzer; "tsto !r" means store all trace activity except activity occurring in the defined address range.

4 Start the analyzer trace with the command, **Trace→Again**

5 Start your program running using **Execution→Run→from()**, **from Transfer Address**, or **from Reset**, as appropriate.

The **Trace→Again** (or **trace again**) command starts the analyzer trace with the most recent trace specifications (including the pod_command specifications you entered). The **trace** command cannot be used by itself because it defaults the "bc -e trig1", "trng addr= ...", and "tsto !r" specifications, returning them to their default values before the trace begins.

You can see the progress of your trace with the command, **Display→Status**. A line in the Trace Status listing will show how many states have been captured.

6 The notation "trig1 break" usually followed by "Emulation trace complete" will appear on the status line. If "trig1 break" remains on the status line without "Emulation trace complete", manually stop the trace with the command:

Trace→Stop

You must wait for the notation "trig1 break" and/or "Emulation trace complete" to appear on the status line; this ensures the trace memory is filled during the trace (except for the unfilled space you specified in Step 2 above).

Note that when you set a delay specification using **tgout -c** or **tgout -t** (trigger output delay before trace complete/after trigger), the trace will indicate complete as soon as the analyzer has captured the state specified, even though the entire trace memory has not been filled.

If the notation "trig1 break" remains on the status line without being replaced by "Emulation trace complete", it indicates the trace memory is not completely filled, and no more states are being captured.

7 Store the entire trace memory content in a file with a command like:

wait measurement_complete ; copy trace to < directory/filename>

The "wait" command is inserted ahead of the "copy" command to ensure that the unload of trace data is complete before you try to store it. Without "wait", you will get an ERROR message warning that the unload is still in process. The **< filename>** is an ASCII filename for a binary file that can be viewed using the **load trace** command.

8 Start a new trace with the command: **trace again**

9 Resume the program run from the point where it was interrupted when the emulator broke to the monitor with the command: **run**

10 Wait until the notation "trig1 break" and/or "Emulation trace complete" appears on the status line. Then store the new trace memory content in a new file with commands like:

stop_trace
wait measurement_complete ; copy trace to < directory/filename+ 1>

Note that "filename+ 1" in the above command suggests use of consecutive filenames to store your execution files, such as FILENAME1, FILENAME2, etc.

Repeat steps 8 through 10 above until all program execution has been captured. Your destination directory will have a set of files that, taken together, contain all of your program execution. Note that if you did not prevent capture of foreground monitor cycles in step 3 above, the last few trace lines in each file may contain monitor cycles.

Tasks that must be performed in the terminal interface

If you are using the terminal interface, you will need to perform the following tasks when using the deep analyzer:

- Setting the depth of the trace memory.
- Displaying the trace list.

To set trace memory depth in the deep analyzer

- Set the depth of the analyzer trace memory by typing **tcf -deep** < **DEPTH**> (trace configure deep < DEPTH>).

  < **DEPTH**> is the portion of the trace memory that the deep analyzer will use to store captured data during a trace.
  < **DEPTH**> can be any depth from 1 state to the maximum number of states that can be stored in the trace memory of your deep analyzer.
  Maximum trace memory depth can be obtained by omitting the < **DEPTH**> parameter from your command, or by specifying < **DEPTH**> as 0.

  The memory depth you specify will affect the analyzer response to a **w -m** (wait for measurement complete) command. The analyzer detects measurement complete after the trigger has been captured and the trace memory depth you specify has been filled.

Examples

To obtain a 4K memory depth in your deep analyzer, enter:

```
tcf -deep 4096
```

To obtain the full memory depth in your deep analyzer, enter:

```
tcf -deep <full memory depth available in your analyzer>
```

or

```
tcf -deep 0
```

or

```
tcf -deep
```

Entering **tcf -deep** 0 or **tcf -deep** allows you to achieve maximum trace memory depth even if you are unsure of exactly how much depth is available on the analyzer card.

To display the trace list

- Display the trace list using the default parameters by typing: **tl**

If the trigger has been captured in memory, the trace list can be displayed while the trace is in progress. Otherwise, "** Trigger not in memory **" will appear under the column headings in the trace list. The only way to see the trace list before the trigger is captured is to halt the trace. Then the trace list will show a history of states captured (if any), leading up to the point where you halted the trace.

The trace list buffer is as deep as you set it. Refer to the paragraph titled, "To set trace memory depth in the deep analyzer". You can selectively display portions of the trace memory using the **tl** (trace list) command.

Examples

To return to the top of the trace list and disassemble instructions, type:

```
U> tl -td
```

To vary the number of states displayed, type:

```
U> tl -td 5
```

To display the first 20 states, type

```
U> tl -td 20
```

To suppress display of the column headers, use the **-h** option:

```
U> tl -h
```

To align the instruction on trace list line number 38 with the operand cycles on line number 47, enter the command:

```
U> tl -d -od 38 47
```

Note that the **-d** and **-od** options shown above are not used in all emulators. Check the **tl** command in your emulator.

Example tasks you may wish to perform in the terminal interface

Tasks you may wish to perform using terminal interface commands include:

- Setting up the emulation-bus analyzer to exclude sequencer-advance states from the trace memory.
- Setting up the emulation-bus analyzer to begin its sequence on a term other than term 1.
- Setting up the emulation-bus analyzer so that its counts are enabled by an external instrument.
- Setting up your emulator to break to its monitor routine after the emulation-bus analyzer completes a trace.
- Setting up the emulation-bus analyzer to identify the state it is presently capturing as its trigger state when it receives a trigger-recognition signal from another emulation-bus analyzer.

The last three examples in this section show measurement coordination between the deep analyzer and other emulators and analyzers. The first coordination example shows measurement functions of the deep analyzer being enabled by an externally supplied "arm" signal. The remaining two examples show coordination of measurements using trig1/trig2 on the trigger bus inside the 64700 card cage. Execution in associated emulators and/or measurements in other analyzers can be started or stopped by the deep analyzer when it recognizes events during a measurement. The events include:

- The analyzer recognizes a state that meets its trigger specification.
- The analyzer captures a specified number of states after capturing its trigger state.
- The analyzer completes its trace (capturing trigger plus filling trace memory).
- The analyzer captures a state that is a specified number of states before its trace memory is filled.

- The analyzer recognizes an arbitrary event or set of events specified by the user.

The cross-trigger and arming features listed above are not implemented directly with commands available in workstation or PC interfaces (except for generation of trig1/trig2 when the analyzer trigger is recognized). These features can only be obtained by using commands in the terminal interface. You can use the following methods to gain access to terminal interface commands from within high-level interfaces:

- If using the Graphical User Interface or Softkey User Interface, use the "pod command" method of accessing terminal interface commands.
- In a PC Interface, access the terminal interface through the **S**ystem **T**erminal command.
- In the Real-Time C Debugger Interface, use HP ARPA Services to select TELNET.

After typing the terminal interface commands desired, you can return to your high-level interface to continue your test procedure. Refer to the paragraph titled "To capture a continuous stream of program execution no matter how large your program" in the "Tasks that are performed more easily in the Graphical User Interface and Softkey Interface" section of this chapter for an example using terminal interface commands.

Note that the current capability of the softkey user interface is "break on trigger". This sets up the analyzer to generate trig1 only when its trigger specification is satisfied.

To prevent storage of sequencer-advance states in the trace memory

- Prevent storage of the states that cause the sequencer to advance from one sequence state to another by typing: **tsq -inc dis** (trace sequencer "include" disabled).

- Restore the default specification that causes the analyzer to store all sequencer-advance states by typing: **tsq -inc en** (trace sequencer "include" enabled)

  The default setup causes the deep analyzer to store all states that advance the sequencer from one sequence state to another. These sequence-advance states are stored regardless of whether they are qualified by your store-qualifier specification (**tsto**) or not. There may be times when you are using an elaborate sequence to identify one or two trace lines that you want to capture in memory. This selection allows you to prevent storage of the series of sequence-advance states.

To specify trace start with a sequencer term other then term one active

- Specify any sequence term to be the first active term at trace start by typing:
  tsq -init < TERM# > (trace seqencer initial term = < TERM# >)

  Where < TERM# > is the sequence term you want to be the first active term at trace start.

  There may be times when you want an elaborate sequencer setup to begin with a term other than term 1 as the first active term.

To set up the emulation-bus analyzer so its counts are enabled by an external instrument

1 Connect the rear panel BNC to deliver trig1 to the analyzer by typing: **bnct -d trig1** (BNC input drives trig1).

2 Set the emulation-bus analyzer to be armed when it receives trig1 by typing: **tarm = trig1** (trace analyzer arm signal supplied on trig1). The first time that the trig1 signal goes from false to true after the trace is started, the arm signal will switch to true and remain true for the rest of the measurement.

3 Set the emulation-bus analyzer to perform its count only when it is armed by typing: **tcq arm** (trace tag counter armed (enabled) by the arm signal).

You can connect a logic analyzer to the rear panel BNC and give it the power to control when the emulation-bus analyzer counts states. Perhaps you would like to have the emulation-bus analyzer count the number of calls to a particular routine, but only after execution of an external event. You could set up the logic analyzer to monitor that event, and to supply a TTL level to the rear panel BNC when the event occurs. Then the emulation-bus analyzer could be set up to count calls to the routine of interest, but only after the TTL-true is supplied from the logic analyzer.

To break the emulator to its monitor after the emulation-bus analyzer completes a trace

1 Set the analyzer to drive trig1 when the analyzer completes its trace (captures trigger plus enough states to fill its trace memory) by typing: **tgout trig1 complete** (trigger output on trig1 when measurement complete).

2 Set the emulator to break to its monitor program on receipt of the trig1 signal by typing: **bc -e trig1** (break condition enabled on trig1)

You may use this setup to stop your program at some point in its execution so you can single step through a portion of your target program after a complex set of conditions have been established in your target system.

Trig1 and trig2 are used to coordinate measurements between instruments in the instrumentation card cage. The analyzer can drive or receive either or both of these lines. Also, the rear-panel BNC and the CMB trigger signal can drive or receive either of these signals.

The above steps cause the emulator to break to its monitor program when the emulation-bus analyzer completes its trace. When you use trig1 and/or trig2 to coordinate actions in associated equipment there is delay in the coordination. In the example above, the emulator may execute several states between the time the analyzer completes its trace and the emulator breaks to its monitor program.

The analyzer can be set up to generate a trigger output on trig1 and/or trig2 on the following events:

- Recognition of the analyzer trigger (**tgout trig1 trigger**).
- Completion of a trace, the trigger captured and trace memory filled (**tgout trig1,trig2 complete**).
- Capture of a state that is < DELAY> number of states after the trigger was captured (**tgout trig2 -t 20**).
- Capture of a state that occurs after the trigger state and is < DELAY> number of states before the end of trace memory (**tgout trig1,trig2 -c 10**).
- Recognition of a specific state (**tgout trig1 addr= 100**).
- Recognition of a pattern when a particular sequence term is active in the complex mode (**tgout 1 p1**)

To trigger one emulation-bus analyzer from another emulation-bus analyzer

1 Connect a CMB cable (coordinated measurement bus) between the two 64700 card cages.

2 In the first analyzer, specify the analyzer trigger by typing: **tg < state>** (trigger on **< state>**)

where **< state>** is a unique state to be recognized by the analyzer (such as, **addr= 1000 and data= 44 and stat= write**)

3 Set the first analyzer to drive trig1 when it captures a state that meets the trigger specification by typing: **tgout trig1 trigger** (trigger output on trig1 when trigger specification satisfied)

4 Set the second analyzer to trigger its trace when it receives trig1 from the first analyzer by typing: **tarm = trig1; tg arm** (trace arm signal is on trig1, trigger when arm switches to true)

5 Establish interaction through the rear panel CMB connection on each card cage, as follows: In the interface of the first analyzer, set the rear panel CMB connection to receive trigger by typing: **cmbt -r trig1** (CMB trigger line receives trig1). In the interface of the second analyzer, set the rear panel CMB connection to drive trigger to the second analyzer by typing: c**mbt -d trig1** (CMB trigger line drives trig1).

6 Make sure the first (driving) analyzer is not driving trig1 by issuing the **th** (trace halt) command.

7 Start the second (receiving) analyzer using the **t** (trace) command.

8 Start the first (driving) analyzer using the **t** (trace) command.

With this coordination, you can effectively widen your analysis bus to any number of channels, limited only by the number of analyzers available in your system.

Trig1 and trig2 are used to coordinate measurements between instruments in the instrumentation card cage. An analyzer can drive or receive either or both of these lines. Also, the rear-panel BNC and the CMB trigger signal can drive or receive either of these signals.

When you use trig1 and/or trig2 to coordinate actions in associated equipment, there is delay in the coordination. In the example above, several states may be executed between the time the first analyzer recognizes its trigger and the time the second analyzer recognizes its trigger.

An analyzer can be set up to generate a trigger output pulse on trig1 and/or trig2 on any of the following events:

- Recognition of the analyzer trigger (**tgout trig1 trigger**).
- Completion of a trace, the trigger captured and trace memory filled (**tgout trig1,trig2 complete**).
- Capture of a state that is < DELAY> number of states after the trigger was captured (**tgout trig2 -t 20**).
- Capture of a state that occurs after the trigger state and is < DELAY> number of states before the end of trace memory (**tgout trig1,trig2 -c 10**).
- Recognition of a specific state (**tgout trig1 addr= 100**).
- Recognition of a pattern when a particular sequence term is active in the complex mode (**tgout 1 p1**)

3

Interfaces of the Deep Analyzer

This chapter discusses the differences between the interface of the deep analyzer and the interface of the 1K analyzer that is described in the emulator/analyzer manual(s) you received with your equipment.

The following information is covered in this chapter:

- Differences you will see when controlling the deep analyzer through the workstation interfaces (Graphical User Interface and Softkey User Interface).
- Differences you will see when controlling the deep analyzer through the PC interfaces (PC Interface and Real-Time C Debugger Interface).
- Differences you will see when controlling the deep analyzer through the terminal interface. Each of the terminal interface commands that have different syntactical structures and capabilities between the deep analyzer and the 1K analyzer are described in detail.

Graphical User Interface and Softkey User Interface Differences

The following pages show differences you will see when using the Graphical User Interface or the Softkey User Interface (for workstations) with the deep analyzer instead of with the 1K analyzer described in your emulator/analyzer manual(s).

New commands

No new commands were added in the Softkey User Interface.

One new dialog box (shown later in this chapter) was developed for use in the Graphical User Interface. It is called the Trace Options dialog box. It helps you set up trace display specifications for the deep analyzer. The dialog box maps directly to the command-line options.

One new popup menu (shown later in this chapter) was developed for use in the trace list display. It is called the trace list popup menu. It allows you to begin trace disassembly from any selected line in the trace list, open a window to edit the source file where a trace list line resides, and display the program memory content associated with a line in the trace list.

The Trace Options dialog box and the trace list popup menu are also available for the 1K analyzer in systems using the latest software versions, listed in the "Software compatibility" paragraph in Chapter 1.

Trace list differences

The trace list display of the deep analyzer has an additional item of information in its heading that may not appear in the heading of your 1K analyzer: it shows trace unload depth. It will appear as:

```
Trace List    Depth=256k    Offset=0
```

If you specified a trace memory unload depth as described in the section titled "Tasks that must be performed in the workstation interface" of Chapter 2 , the depth you specified will be shown. For example, if you installed the optional memory modules on your deep analyzer board to obtain a depth of 256K, but you entered the command **display trace depth** 4096, then the heading line

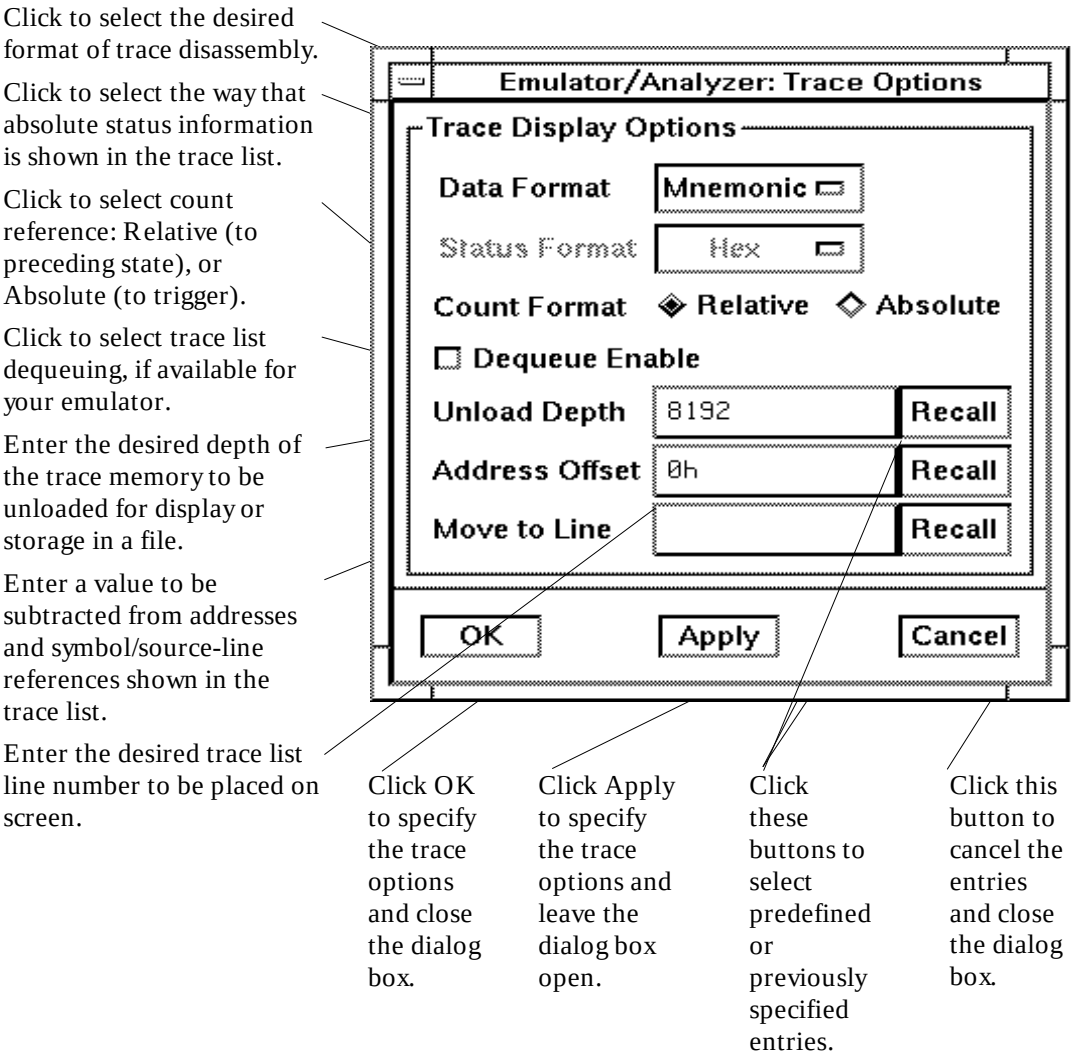
would show "Depth= 4096". If you do not specify a trace memory unload depth, the maximum trace memory depth (default depth) will be shown, and the full depth will be unloaded. The unload depth of the trace memory is discussed in detail in the concepts chapter of this manual.

Negative time or state counts in trace lists

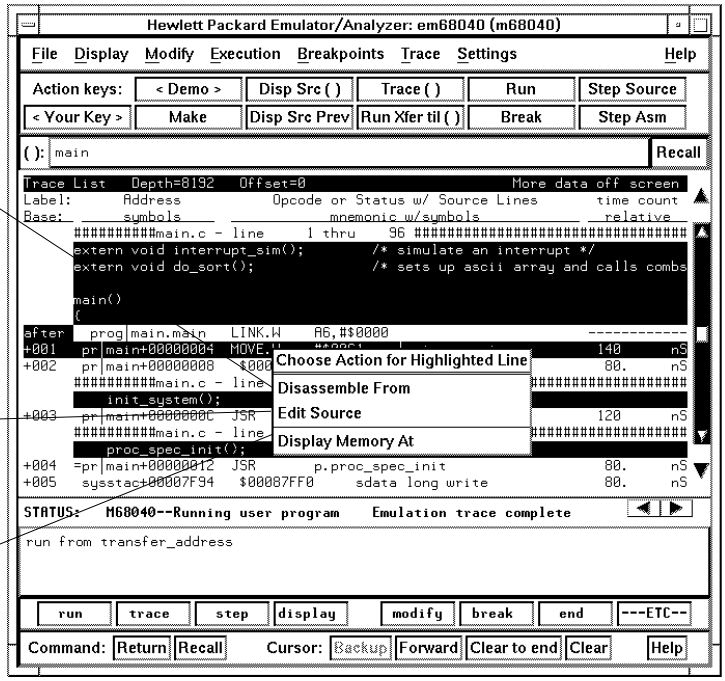
If a counter overflow occurs during a trace measurement, you may see a relative count or absolute count of negative time or negative states in the trace list. This is a normal condition. It is discussed in detail in the concepts chapter of this manual.

Examples

To use the Trace Options dialog box:



Examples To use the trace list popup menu:



Click to begin trace disassembly from the selected line, moving that line to the top of the display (if **disassemble_from_...** is available for your emulator).

Click to open an edit window into the source file that contains the address of the selected line.

Click to open a display window into memory containing the address of the selected line. Note that the format of the memory display will be mnemonic for addresses in the code segment and absolute otherwise.

PC Interface Differences

Hewlett-Packard offers two interfaces for use on personal computers: the PC Interface (older interface), and the Real-Time C Debugger Interface. The Real-Time C Debugger Interface is an upgrade and replacement for the PC Interface.

When using the deep analyzer in the PC Interface, you will see the same commands and functions as the 1K analyzer, described in your emulator/analyzer manual(s). The older PC interface does not support the deep analyzer. However, when using the deep analyzer in the compatible mode (512/1K depth), time or state counting is performed at full processor speed with all supported emulators (refer to "Compatible mode vs deep mode" in Chapter 1).

When using the Real-Time C Debugger Interface, differences between the use of the 1K analyzer and the deep analyzer are documented in the manual(s) you received with your Real-Time C Debugger product.

Terminal Interface Differences

The following pages show differences you will see when using the terminal interface with the deep analyzer instead of with the 1K analyzer described in your emulator/analyzer manual(s). The following commands are different when used with the deep analyzer from those used with the 1K analyzer:

- ta (trace activity) has no function - trace activity is not implemented.
- tcf (trace configuration) sets depth of the analyzer trace memory.
- tck (trace clock) provides a reduced set of clocking options.
- tcq (trace tag counter) supports use of an arm signal for measurement coordination.
- tf (trace format) allocates more space for showing trace memory line numbers and time or state counts.
- tgout (trigger output) provides a broad selection of trigger-generation options.
- tsck (trace slave clocks) has no function. Slave clocks are not available in the deep analyzer.
- tsq (trace sequencer) provides options for excluding sequencer-advance states from trace memory, and for selecting the initial sequencer state.

Details of the differences in the above commands are given in the remaining pages of this chapter.

Certain features of the deep analyzer can only be obtained by using the terminal interface. These features include:

- Generation of trig1/trig2 when the analyzer completes its trace.
- Generation of trig1/trig2 a specified number of states after the analyzer trigger is captured.
- Generation of trig1/trig2 a specified number of states before the end of the trace memory.

If using a workstation interface, you can access these trigger-generating options through the "pod command" method of accessing terminal interface

commands. In a PC Interface, access the terminal interface through the **S**ystem **T**erminal command. In the Real-Time C Debugger Interface, use HP ARPA Services to select TELNET.

You can specify the terminal interface commands desired, and then return to your high-level interface to continue your test procedure. Refer to the paragraph titled "To capture a continuous stream of program execution no matter how large your program" in the "Tasks that are performed more easily in the Graphical User Interface and Softkey Interface" section of Chapter 2 for an example using terminal interface commands.

Note that the current capability of the softkey user interface is "break on trigger". This sets up the analyzer to generate trig1 only when its trigger specification is satisfied.

Exclamation mark "!" in count column of PC interface and terminal interface trace lists

An exclamation mark in the count column of the trace list indicates the counter overflowed before the present state was captured. This symbol is supplied to alert you to the counter overflow. It is discussed in detail in the concepts chapter of this manual.

Counter overflow indication not seen in trace list

If a counter overflow occurred before the start of the range you specified in your "trace display" command, the counter overflow indication will not be seen in your trace list. When the analyzer reads memory to compose a trace list, it reads only the portion of memory you specify.

Negative time or state counts in trace lists

If a counter overflow occurs during a trace measurement, you may see a relative count or absolute count of negative time or negative states in the trace list. This is a normal condition. It is discussed in detail in the concepts chapter of this manual.

ta (trace activity)

The **ta** command can be accepted by the deep analyzer, but it will provide no information. The **ta** command was available on the 1K analyzer to display activity on each of the analyzer input lines.

Examples

Display current analyzer signal activity:

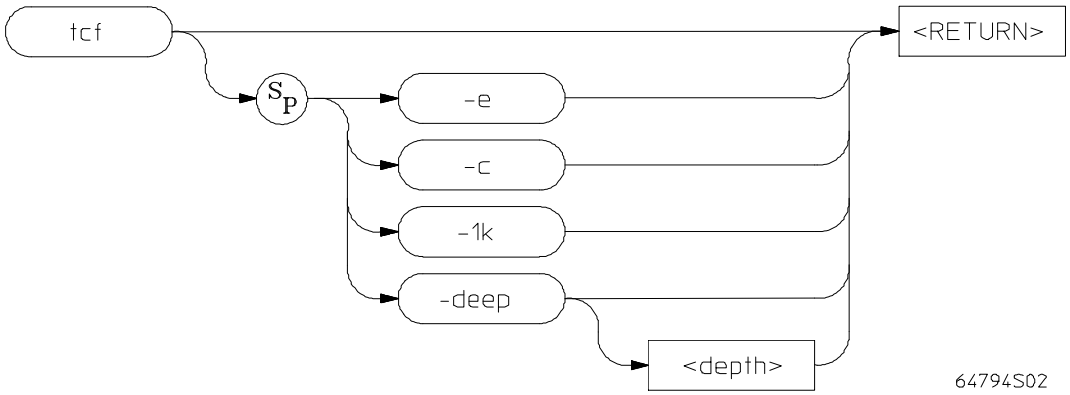
```
M> ta
```

Using the deep analyzer, you will see a display similar to the following:

```
Pod 5        = -------- --------
Pod 4        = -------- --------
Pod 3        = -------- --------
Pod 2        = -------- --------
Pod 1        = -------- --------
```

The results display indicates that signal activity cannot be determined.

tcf (trace configuration)



The **tcf** command is used to set the configuration for the emulation-bus analyzer.

The parameters are as follows:

-e — Specifying **-e** sets the analyzer to the easy configuration.

-c — Specifying **-c** sets the analyzer to the complex configuration.

-1k — Specifying **-1k** sets the analyzer to the compatible mode. In this mode, the analyzer is compatible with all emulator interfaces that were designed to work with the 1K analyzer The trace memory is 1024 states deep if you have no state or time count included in your trace specification; it is 512 states deep if a state or time count is included in your trace specification. This is the default mode at power up.

-deep [< depth>] — Specifying **-deep** sets the analyzer to the deep mode; the trace memory depth will be the maximum depth available in the analyzer. Specifying **-deep** < **depth**> sets the analyzer to have a trace memory of the depth you specify (less than the maximum depth). You may want to specify a reduced depth when making a series of traces to be sent to post-processing software.

Specify < **depth**> using a decimal number. For example, to obtain the same depth as the 1K analyzer, enter the **tcf -deep 1024** command. This gives you the same trace depth as the 1K analyzer without imposing the memory tradeoff for counting in the compatible mode (**tcf -1k**).

The **-deep** mode of the deep analyzer can be used with any HP emulator through the terminal interface. If using the terminal interface, you must select the **-deep** mode after power up by entering the **tcf -deep [depth]** command.

The **-deep** mode of the analyzer will work with most popular HP emulator/high-level interface combinations. In high-level interfaces that can use the deep mode of the analyzer with their associated emulator, the **-deep** mode is automatically selected after power up. Refer to Chapter 1 for details.

If you are using the deep analyzer with a high-level emulator interface that cannot work with the deep analyzer, the **-1k** (compatible) mode will be maintained after power up. You will still be able to take advantage of the unrestricted counting speed of the analyzer. If you need to take a trace with greater memory depth than the 1K depth of the compatible mode, you can use the **-deep** mode through the terminal interface to take your trace. Refer to the paragraph titled "To capture a continuous stream of program execution no matter how large your program" for an example of how terminal interface commands can be used in high-level interfaces.

Caution

Be sure you return the deep analyzer to the 1K (compatible) mode before returning to the high-level emulator interface that cannot work with the deep mode of the deep analyzer. Failure to do so will cause unpredictable behavior of the high-level emulation interface.

If no parameters are supplied, the current analyzer configuration is displayed. After powerup, the default analyzer configuration is **tcf -e** and **tcf -1k**. The **tinit** command will set **tcf -e**, but will not affect the **tcf -1k/-deep** specification.

Examples

Display the current analyzer configuration:

```
M> tcf
```

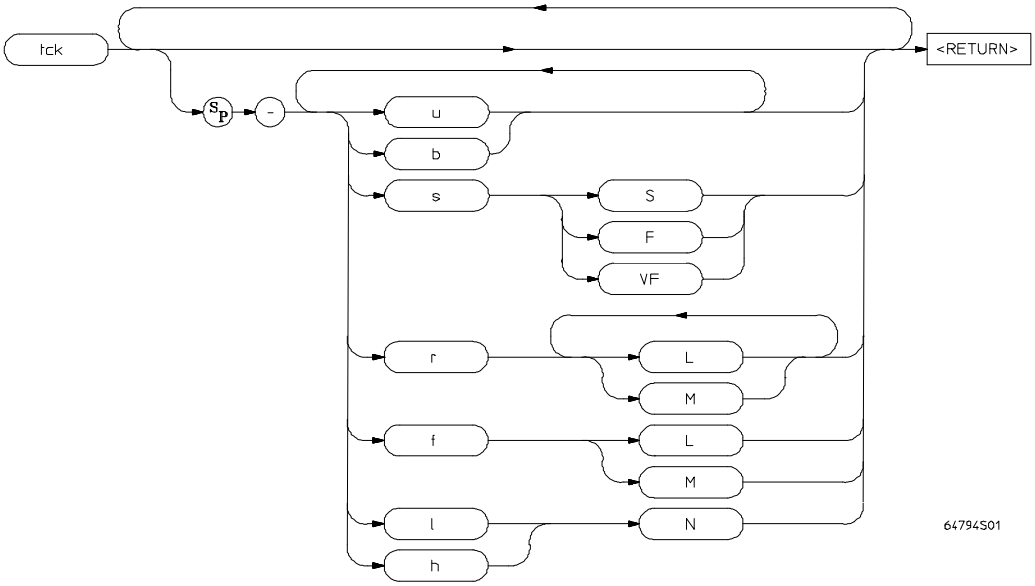
Set the analyzer to complex configuration:

```
M> tcf -c
```

Set the analyzer for a 4K memory depth:

```
tcf -deep 4096
```

tck (trace clock)



The **tck** (trace clock) command allows specification of clock qualifiers and clock edges for the master clocks used by the deep analyzer. The **tck** command also allows specification of maximum clock speed; the clock speed specifications are only included for compatibility with the 1K analyzer. Clock speed has no effect in the deep analyzer.

Note that this command is set at powerup by the emulation system; it should not normally be changed by the user.

The parameters that are different when using the deep analyzer are as follows:

r — Specifying **r** indicates that the analyzer is to be clocked on the rising edge of the indicated clock signal (either the **L** clock, the **M** clock, or both clocks). If you specify both clocks, the rising edge of either the **L** or **M** clock can clock a state.

f — Specifying **f** indicates that the analyzer is to be clocked on the falling edge of the indicated clock signal (either the **L** clock or the **M** clock). If both **L** and **M** clocks are specified, then both clock edges must be rising.

l

Specifying **l N** sets the analyzer so that it will only accept other clock signals when the **N** clock signal is low (less positive/more negative voltage). Used as a qualifier (example: clock on rising edge of **L** only if **N** is low). Note that **-l N** is the same as **-b**, which captures only background monitor execution.

h

Specifying **h N** sets the analyzer so that it will only accept other clock signals when the **N** clock signal is high (more positive/less negative voltage). Used as a qualifier (example: clock on falling edge of **M** only if **N** is high). Note that **-h N** is the same as **-u**, which captures only execution in user address space.

CLOCK SIGNALS

The **l** and **h** operators can be used on clock signal **N**.

The **r** and **f** operators may be used on clock signals **L** and **M**.

If no parameters are specified, the current clock definitions are displayed. Clock options are set at initialization and depend on the particular emulator in use.

Examples

Specify that trace data can be clocked into the analyzer on either the rising edge of the **L** clock or on the rising edge of the **M** clock, but only when the **N** clock is low:

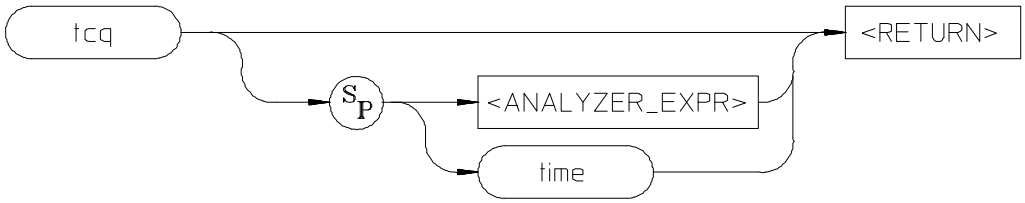
```
R> tck -r LM -l N
```

The **tck** command is included with the analyzer for internal system initialization and system control through high-level software interfaces. It is used to select the analyzer master clock(s), and clock qualifier, if desired.

The clocking options operate on three different master clock signals: **L, M** and **N**. These clocks are generated by the emulator; the emulation master clock edges are set at powerup for the particular emulator being used. You should not change them.

When the **L** and **M** clocks are specified by the **-r** option, either the rising edge of clock **L** or clock **M** can clock the trace. If clock **N** is specified as a qualifier (**l** or **h**) it is ANDed so that the trace is only clocked when the qualifier is also met.

tcq (trace tag counter)



The **tcq** command allows you to specify the type of count to be made by the emulation trace tag counter, and specify a qualification for the counter, if desired. Using this command, you can specify whether the analyzer measures time between each state it captures, counts occurrences of certain types of states you specify, or makes no count at all. Additionally, you can specify that counts be made only when the **arm** signal is asserted.

The only parameter that is different when using the deep analyzer is the **arm** parameter, listed below:

arm

If you specify **tcq arm**, the trace tag counter will make its counts only after the **arm** signal is true. The arm signal can be supplied on either trig1 or trig2, and can be asserted by either the analyzer itself, by an associated emulator or analyzer on the coordinated measurement bus, or by an instrument connected to the rear panel BNC on the instrumentation card cage.

The arm signal begins in the false state. It switches to the true state and remains true after the first false-to-true transition of the selected signal(s).

Examples

To specify that counts be made only after an external instrument provides a false-to-true transition to the rear panel BNC, type:

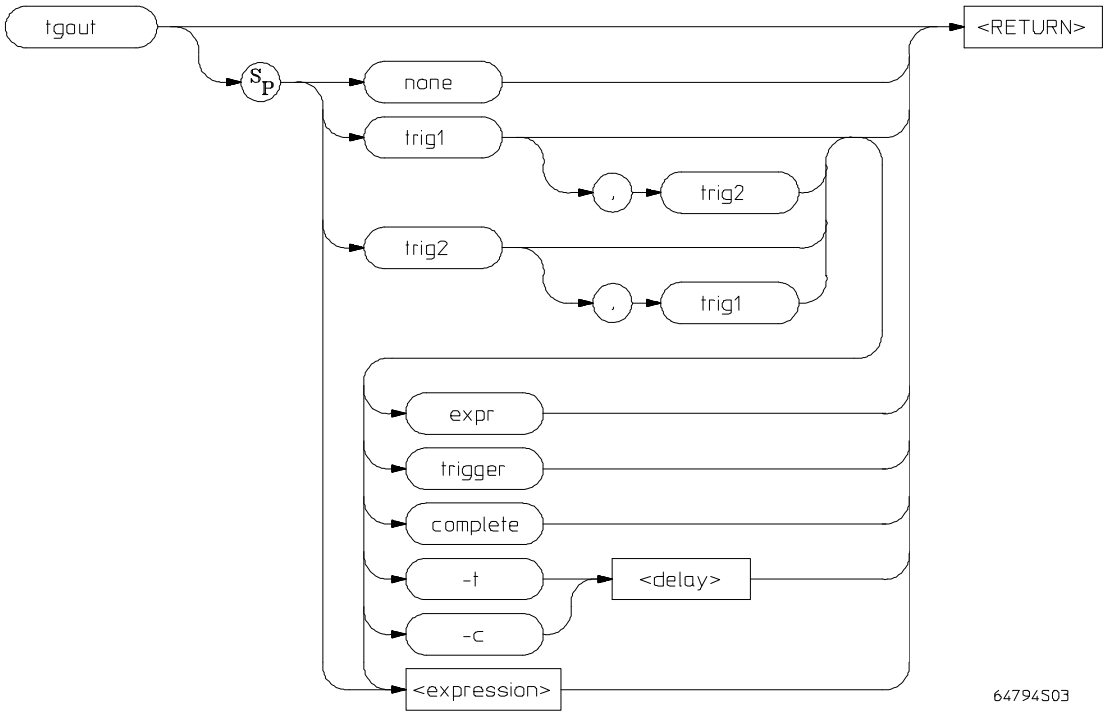
```
R>bnct -d trig1
R>tarm =trig1
R>tcq arm
```

tf (trace format)

The **tf** command allows you to specify which pieces of information from the trace memory of the deep analyzer will be displayed by **tl** (trace list) commands. You will see two differences in the trace list when using the deep analyzer:

- The trace memory line number column and the time/state count column are wider than when using the 1K analyzer.
- Double spacing between columns in the trace list has been changed to single spacing.

tgout (trigger output)



The **tgout** command allows you to specify either trig1, trig2, or both trigger signals to be driven when the emulation-bus analyzer finds the condition you specify. Trig1 and trig2 are bidirectional signal lines that can be used to coordinate measurement activity between emulators and analyzers installed in the instrumentation card cage, and instruments connected to the BNC or the CMB on the rear panel of the card cage. For details of how to configure and use trig1 and trig2, refer to the chapter on making coordinated measurements in your emulator/analyzer manual(s).

Note that there is delay in measurements that use **tgout** for measurement coordination. For example, you may specify that the emulator break to its monitor program when it receives trig1 from the analyzer. Several states may be executed in the emulator between the time the analyzer recognizes its trigger condition, generates trig1, delivers trig1 to the emulator, and the emulator responds to trig1 by breaking to its monitor program.

The parameters are as follows:

none	If **none** is specified, neither trig1 nor trig2 will be driven by the analyzer.
trig1	If **trig1** is specified, the trig1 signal will be driven by the analyzer when the condition you specify is found. If you do not specify a condition in your command, recognition of the analyzer trigger is assumed to be the specified condition.
trig2	If **trig2** is specified, the trig2 signal will be driven by the analyzer when the condition you specify is found. If you do not specify a condition in your command, recognition of the analyzer trigger is assumed to be the specified condition. To specify that both trig1 and trig2 should be driven, concatenate both options with a comma: **tgout trig1,trig2**.
trigger	Drive the selected trig1/trig2 signal(s) when the analyzer satisfies its trigger specification.
complete	Drive the selected trig1/trig2 signal(s) when the analyzer completes its measurement (captures trigger plus fills trace memory).
-t < delay>	Drive the selected trig1/trig2 signal(s) when the analyzer satisfies its trigger specification and captures the additional number of states specified in < delay> . If you are using this feature to capture a continuous stream of target program activity, you may find some stacking cycles at the end of each trace memory if your emulation processor does stacking before a break. Note that if you use **-t < delay>** or **-c < delay>** , your trace will be completed automatically when the analyzer has captured enough states to satisfy the delay specification.
-c < delay>	Drive the selected trig1/trig2 signal(s) when the analyzer captures the state that is after the trigger and is < delay> states before the end of trace memory. If you are using this feature to capture a continuous stream of target program activity, you may find some stacking cycles at the end of each trace memory if your emulation processor does stacking before a break. Note that if you use **-t < delay>** or **-c < delay>** , your trace will be completed automatically when the analyzer has captured enough states to satisfy the delay specification.
< expression>	Drive the selected trig1/trig2 signal(s) when the analyzer recognizes the state(s) that satisfies < expression> . The < expression> is a simple expression

in easy configuration, and a complex expression in complex configuration. If you have already specified a **tgout < trigger(s)> < expression>** , you can change the expression without having to reenter the **< trigger(s)>** ; simply type **tgout < new_expression>** .

expr

Drive the selected trig1/trig2 signal(s) when the analyzer captures the state that satisfies the most recently defined < expression> . This is useful if you have already defined a complex **tgout** expression, and now you want to use that same expression to drive a different trigger.

If no parameters are specified, the current state of **tgout** is displayed. Upon powerup or **tinit**, the default state is **tgout none**.

Examples

Display the state of **tgout:**

```
M> tgout
```

Set the emulator so that it will break from target program execution to monitor execution upon receipt of the analyzer trigger:

```
M> tcf -e
M> bc -e trig1
M> tgout trig1
M> tg addr=710
```

The emulator will break to its monitor program after the analyzer encounters address 710, asserts trig 1, and trig 1 is recognized by the emulator. This form of emulation break includes delay in the break response time. Therefore, it is not possible to predict which state will be executing when the emulator responds to the trig1 break signal and enters the monitor.

To generate trig1 when the analyzer detects a write to address 1000 when in easy configuration:

```
M> tgout trig1 addr=1000 and stat=write
```

To generate trig2 when the analyzer completes a trace:

```
M> tgout trig2 complete
```

To generate trig1 and trig2 when the analyzer stores the tenth state after its trigger:

```
M> tgout trig1,trig2 -t 10
```

To generate trig2 when the analyzer captures the fifth state before the end of its trace memory:

```
M> tgout trig2 -c 5
```

To generate trig1 on any access to any address from 2000 through 2010 using easy configuration:

```
M> tgout trig1 expr
M> tgout addr=2000..2010
```

or

```
M> tgout trig1 addr=2000..2010
```

To generate trig1 on any write to any address in the range from 2000..2010 while in complex configuration:

```
M> tcf -c
M> trng addr=2000..2010
M> tpat p5 stat=write
M> tgout r and p5
```

While in complex configuration, to generate trig1 if pattern p1 is found while the sequencer is at level 1, or if pattern p2 is found while the sequencer is at level 2:

```
M>tgout trig1
M>tgout 1 p1
M>tgout 2 p2
```

To define an expression, and then assign it to generate trig2:

```
M>tgout addr=100 and data=44 and stat=write
M>tgout trig2 expr
```

The most recent expression defined for the tgout command is remembered by the analyzer. Once defined, the expression can be assigned to drive either trig1, trig2, or both in a later command.

To define an expression for trig2 and then reassign it to trig1 in a later command:

```
M>tgout trig2 addr=100 and data=44 and stat=write
M>tgout trig1 expr
```

Note: To stop the analyzer from driving the trig1/trig2 line, issue the **th** (trace halt) command.

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, for synchronizing measurements that include other analyzers)

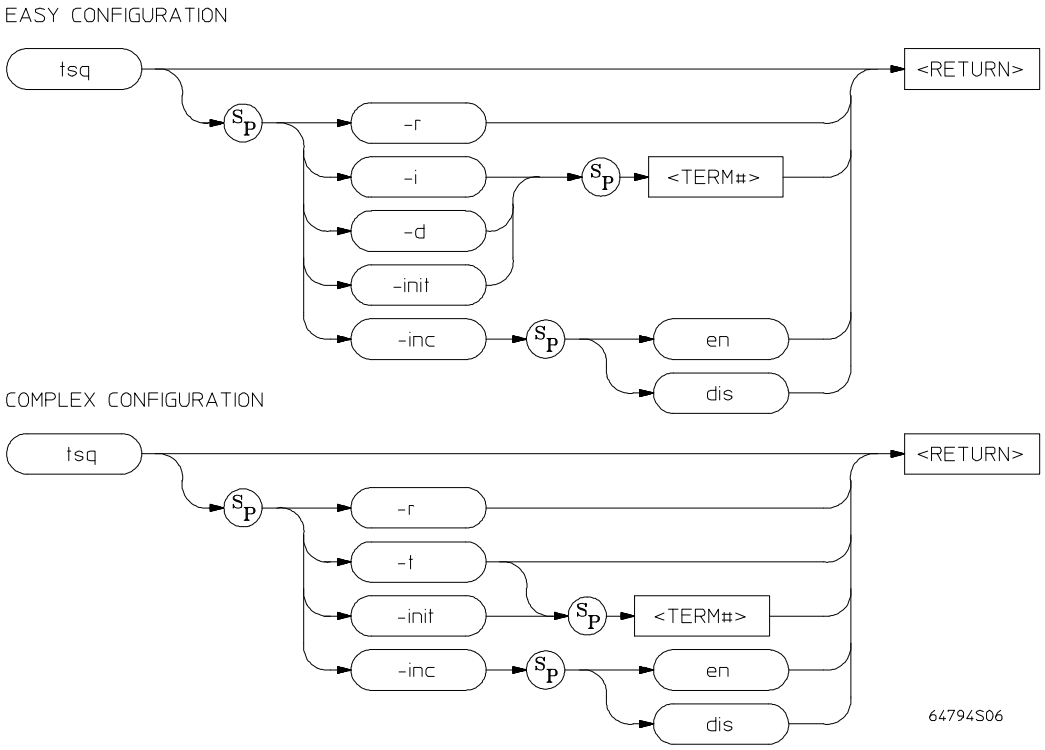
th (halts the analyzer trace and turns off any active drive of trig1/trig2)

w -m (wait_measurement_complete. The point where measurement complete is recognized is affected by any specification that includes the **-t** or **-c** options of the **tgout** command)

tsck (trace slave clocks)

The **tsck** command was used to specify slave clock edges for the emulation-bus analyzer. The deep analyzer does not support slave clocks. This command is included for the compatibility mode of the deep analyzer (the mode in which the deep analyzer appears to be a 1K analyzer to emulator interfaces that depend on use of the 1K analyzer).

tsq (trace sequencer)



The **tsq** command allows you to manipulate or display the trace sequencer. Note: the **tif** and **telif** commands are used to define each sequencer state specification.

The parameters that have been added for use with the deep analyzer are as follows:

inc

Specifying **inc** allows you to choose whether or not the states that cause the sequencer to transition from one state to another (or to the same state, in the case of restart) are qualified for storage in trace memory. By default, all states that satisfy sequencer advance specifications are stored in the trace memory (**tsq -inc en**). There may be times when an elaborate series of sequencer-advance steps is required to obtain a single traced state. You can

prevent your trace memory from being filled with sequencer-advance states by using this command.

init

Using **init** allows you to specify which sequence term will be the first active sequence term when a new trace begins. By default, term 1 is the first active sequence term when a new trace begins.

Examples

To allow all states that satisfy sequencer-advance specifications to be stored in the trace memory (the default), enter the command:

R> **tsq -inc en**

To disable storage of sequencer states in trace memory, enter the command:

R> **tsq -inc dis**

To specify that sequence term 5 will be the active term when the trace first begins, enter the command:

R> **tsq -init 5**

4

Concepts

This chapter discusses two concepts that will help you understand functions of the deep analyzer:

- Trace memory depth. Two trace memory depths are used in the deep analyzer: unload depth, and capture depth.
- The trace tag counter, and how it works. This will help you understand what is happening when it shows a negative time count or state count within a series of positive counts in a trace list.

Trace depths of the deep analyzer

The deep analyzer uses two trace depths: unload depth, and capture depth. The trace depth you use depends on the interface you use.

Unload depth

The Graphical User Interface and Softkey User Interface use unload depth. This is the depth of the trace memory that will be unloaded when new trace data is captured. Regardless of how much or how little unload depth you specify, the entire trace memory will be filled with captured states during a deep analyzer trace.

Trace data must be unloaded before it can be displayed, copied, or stored in a file. If you request display of a portion of the trace memory before the unload is complete; the analyzer will interrupt its unload process, unload the portion requested, and then return to the place where it was interrupted and continue the unload process. If you try to copy or store a portion of the trace memory before the unload process is complete, you will see an error message advising that you must wait until the unload is complete. The entire depth must be unloaded before any portion of it can be copied or stored. You can enter a new unload depth specification after a trace is complete to increase the amount of trace memory that is unloaded for display, file storage, or for copying to files and printers, if desired.

Capture depth

Terminal interface commands can be used to specify how much of the deep analyzer trace memory will be used to store trace data. When you specify a limited capture depth in the terminal interface, the deep analyzer will stop capturing new data as soon as the specified capture depth is filled. The portion of the trace memory outside the range of your capture depth specification will be unused.

Using unload depth in a workstation interface

You can specify unload of any depth from 9 states to the maximum number of states that can be stored in the trace memory of your deep analyzer. You will want to select an unload depth that is:

- Large enough to include the activity you wish to view, store, or copy.
- Small enough to unload quickly, avoiding unnecessary delay in unloading the trace memory.

Trace data is always unloaded around the trigger position. For example, assume you specified unload of 8K depth:

- If trigger is the first state in memory (**trace after**), the first 8K states in memory will be unloaded.
- If trigger is the center state in memory (**trace about**), the 4K states before trigger and the 4K states after trigger will be unloaded.
- If trigger is the last state in memory (**trace before**), the 8K states before trigger will be unloaded.

Trace memory must be unloaded before it can be displayed in a trace list, stored in a file, or copied to a printer. Trace data is unloaded in background when new trace data is available. This minimizes the waiting period required to unload the trace memory after a measurement is complete. Approximately four minutes are required to unload a full 256K trace memory with LAN. By selecting a reduced portion of the trace memory to be unloaded, your trace list will be available to store and/or copy more quickly.

You can obtain immediate display of trace memory content as long as the requested display area is within the range of your unload depth specification. The analyzer will shift to the requested location, unload it, and give you the desired display. Then it will return to the point where it interrupted its unloading process and begin unloading again.

If you specify an unload depth of 10 and then request display of line 50, the trace list will scroll as close to line 50 as possible (line 10 in this example), and stop because line 50 is not within the line range being unloaded. If you then specify an unload depth of 1000 and request display of trace list line number 50, line 50 and associated lines will appear immediately. If you then respecify an unload depth of 10, line 50 will still be available for display. Once you have

unloaded a line range from a particular trace, specifying a smaller line range will have no effect on the data available for display because it will already be unloaded.

If you specify an unload depth , the depth you specify will be shown in the heading of the trace list. For example, if you installed the optional memory modules on your deep analyzer board to obtain a depth of 256K, but you entered the command **display trace depth** 4096, then the heading line would show "Depth= 4096". If you do not specify an unload depth, the maximum trace memory depth (default depth) will be shown, and the full depth will be unloaded.

The unload depth you specify will affect how quickly the analyzer responds to a **wait measurement_complete** command. The analyzer detects measurement complete after the trigger has been captured, the trace memory has been filled to the unload depth, and the unload depth has been unloaded. Note that measurement complete is also detected if the trace has been halted and all available data has been unloaded.

Using capture depth in a terminal interface

In the terminal interface, you specify how much of the physical memory space available in the analyzer will be used to store captured states during a trace. Any unspecified memory space will be unused and ignored by the analyzer.

You may want to specify use of a reduced capture depth in order to complete your traces more quickly, or when making a series of traces to be sent to post-processing software.

The **tcf** (trace configuration) command of the terminal interface is used to set the capture depth.

Specifying **tcf -deep** sets the analyzer to the deep mode and specifies use of the maximum capture depth available in the analyzer. Specifying **-deep < depth>** sets the analyzer to the deep mode, but specifies use of only the capture depth you specify.

Note that you can enter the **tcf -deep 1024** command, if desired. This gives you the same capture depth as the 1K analyzer, but without imposing the memory tradeoff for counting (which is imposed in the compatible mode).

The deep mode of the deep analyzer can be used with any HP emulator through the terminal interface. If using the terminal interface, you may specify use of the deep mode after power up by entering the **tcf -deep [depth]** command.

The deep mode of the analyzer will work with most popular HP emulator/high-level interface combinations. High-level interfaces that can use the deep mode of the analyzer with their associated emulator select **tcf -deep** automatically after power up. Refer to Chapter 1 for details.

If you are using the deep analyzer with a high-level emulator interface that cannot work with the deep mode of the analyzer, the compatible mode (**tcf -1k**) will be maintained after power up. You will still be able to use the unrestricted counting speed of the analyzer. If you need to take a trace with greater memory depth than the 1K depth of the compatible mode, you can use the deep mode through terminal interface commands to take your trace. Refer to the chapter titled "Unique deep analyzer tasks" for an example of how terminal interface commands can be used in high-level interfaces.

Caution Be sure you return the deep analyzer to the 1K (compatible) mode before returning to the high-level emulator interface that cannot work with the deep mode of the deep analyzer. Failure to do so will cause unpredictable behavior of the high-level emulation interface.

Trace tag counter of the deep analyzer

This section describes the trace tag counter and explains the causes of certain unusual content that you may see in your trace lists.

How the counter works

The trace tag counter of the deep analyzer is a 36-bit counter. It begins counting at the start of a trace and continues until the end of the trace. In the time-count mode, it increments its count one time every 20 nsec. It reaches full count at 22.9 minutes (22 minutes and 54 seconds). When it increments again, after reaching full count, it restarts at 0. There is never a loss of resolution, however, no indication is given for the number of times the counter reached its full count and started at 0 again (counter overflow). The deep analyzer sets a flag in memory and stores it along with the first state captured after counter overflow.

Note Due to increased counter resolution, you will see time values ranging from thousands of seconds (KS)to tens of nanoseconds (nS).

During a trace, the present value of the trace tag counter is stored with each stored state. Post-processing software computes the counter value to be shown in a trace list display by subtracting the value of the reference state from the value of the present state.

- When counts are displayed in relative mode, the reference state is the preceding displayed state; you will see the difference between the counter value stored with the present state and the counter value stored with the preceding displayed state.
- When counts are displayed in absolute mode, the reference state is the trigger state; you will see the difference between the counter value stored with the present state and the counter value stored with the trigger state.

Note that the count is between displayed states in relative mode. If your trace list is inverse assembled and/or dequeued, several states may have been captured in trace memory between the present displayed state and the displayed state immediately before it. The count value shown in the trace list is obtained by subracting the count value stored with the last displayed state from the count value stored with the present state.

All time values are adjusted to center them around the trigger point. In count relative mode, the time value stored with the trigger state is subtracted from all counter values. This corrects the trigger state counter value to 0. This also explains how negative times can be shown for states captured before the trigger state.

Negative time or state counts in workstation trace lists

If counter overflow occurs during a trace measurement, you may see a count of negative time or negative states in the trace list. This indicates that the counter value stored with the reference state was greater than the counter value stored with the present state. In absolute time counts, negative times will continue to be seen until a state is captured whose counter value is greater than the trigger state counter value. In relative time counts, negative time should only be seen beside the first state captured after the counter overflows.

The next three example illustrations show effects of counter overflow in workstation trace lists. The first illustration is a wait loop routine that was run on an HP emulator for a Motorola 68040. The wait loop runs for approximately 1 second (999.98 mS), and then restarts. The deep analyzer was set up to capture only the start state of the wait loop (address 0FDCH).

```
 Memory  :mnemonic
   address   data
-------------------------------------------------------------------------------
       1
       2
       3    main()
       4    {
  00000FD2  4E560000    LINK.W    A6,#$0000
  00000FD6  2F03        MOVE.L    D3,-(A7)
  00000FD8  2F02        MOVE.L    D2,-(A7)
       5      int delaytime;
       6      int somethingtodo;
       7
       8
       9      while (1)
  00000FDA  4E71        NOP
      10      {
      11        somethingtodo = 0;
  00000FDC  7600        MOVEQ     #$00000000,D3
      12        for (delaytime = 0; delaytime  510621; delaytime++)
  00000FDE  7400        MOVEQ     #$00000000,D2
  00000FE0  4E71        NOP
      13        {
      14              somethingtodo++;
  00000FE2  5283        ADDQ.L    #1,D3
      15        }
  00000FE4  4E71        NOP
  00000FE6  5282        ADDQ.L    #1,D2
  00000FE8  4E71        NOP
  00000FEA  0C820007CA  CMPI.L    #$0007CA9D,D2
  00000FF0  6DF0        BLT.B     $00000FE2
  00000FF2  4E71        NOP
      16      }
  00000FF4  4E71        NOP
  00000FF6  60E4        BRA.B     $00000FDC
  00000FF8  4E71        NOP
      17    }
  00000FFA  4E71        NOP
  00000FFC  241F        MOVE.L    (A7)+,D2
  00000FFE  261F        MOVE.L    (A7)+,D3
  00001000  4E5E        UNLK      A6
```

The second illustration in this series (see below) is a trace list taken in the softkey user interface, and displayed with a relative count. Counter overflow occurred between state + 1369 and state + 1370. The counter value stored with state + 1370 is 1,373.4 seconds (22.89 minutes) less than the counter value stored with state + 1369. While this count does not give you useful information about state + 1370 captured from your target program, it is the only state in the trace list whose count is invalid. The next state, and states thereafter, do give useful information.

```
Trace List                Offset=0
Label:  Address   Data                Opcode or Status              time count
Base:     hex     hex                     mnemonic                  relative
------ -------- --------  ---------------------------------------- ------------
after  00023FF8 27000000   $27000000     sdata long write           ------------
+001   00000FDC 76007400   $76007400     sprog long read             38.50   uS
+002   00000FDC 76007400   $76007400     sprog long read            880      nS
+003   00000FDC 76007400   $76007400     sprog long read            999.98   mS
+004   00000FDC 76007400   $76007400     sprog long read            999.98   mS
                       .
                       .
                       .
+1366  00000FDC 76007400   $76007400     sprog long read            999.98   mS
+1367  00000FDC 76007400   $76007400     sprog long read            999.98   mS
+1368  00000FDC 76007400   $76007400     sprog long read            999.98   mS
+1369  00000FDC 76007400   $76007400     sprog long read            999.98   mS
+1370  00000FDC 76007400   $76007400     sprog long read           -  1.3734 KS
+1371  00000FDC 76007400   $76007400     sprog long read            999.98   mS
+1372  00000FDC 76007400   $76007400     sprog long read            999.98   mS
+1373  00000FDC 76007400   $76007400     sprog long read            999.98   mS
```

The third illustration in this series (see below) is a trace list of the same trace as the second illustration, but displayed with an absolute count. The count stored with the trigger state (state 0) is 6.4131 seconds greater than the count stored with state + 1370. This offset occurs because it is not possible to predict exactly how much time will pass between the start of the trace and the capture of the trigger state.

You can see that the count of time decreases after the counter overflow occurs until the count stored with a new state exceeds the count stored with the trigger state. After that, the count increases again. Without knowing exactly what counter value was stored with the trigger state, you cannot tell exactly how much time has elapsed since the trigger was captured (after the counter overflow occurs).

```
Trace List                Offset=0
Label:  Address   Data                Opcode or Status              time count
Base:     hex     hex                     mnemonic                  absolute
------ -------- --------  ---------------------------------------- ------------
after  00023FF8 27000000   $27000000     sdata long write          ------------
+001   00000FDC 76007400   $76007400     sprog long read           + 38.50   uS
+002   00000FDC 76007400   $76007400     sprog long read           + 39.38   uS
+003   00000FDC 76007400   $76007400     sprog long read           +  1.0000 S
+004   00000FDC 76007400   $76007400     sprog long read           +  2.0000 S
                         .
                         .
                         .
+1366  00000FDC 76007400   $76007400     sprog long read           +  1.3640 KS
+1367  00000FDC 76007400   $76007400     sprog long read           +  1.3650 KS
+1368  00000FDC 76007400   $76007400     sprog long read           +  1.3660 KS
+1369  00000FDC 76007400   $76007400     sprog long read           +  1.3670 KS
+1370  00000FDC 76007400   $76007400     sprog long read           -  6.4131 S
+1371  00000FDC 76007400   $76007400     sprog long read           -  5.4131 S
+1372  00000FDC 76007400   $76007400     sprog long read           -  4.4131 S
+1373  00000FDC 76007400   $76007400     sprog long read           -  3.4132 S
+1374  00000FDC 76007400   $76007400     sprog long read           -  2.4132 S
+1375  00000FDC 76007400   $76007400     sprog long read           -  1.4132 S
+1376  00000FDC 76007400   $76007400     sprog long read           -413.21   mS
+1377  00000FDC 76007400   $76007400     sprog long read           +586.77   mS
+1378  00000FDC 76007400   $76007400     sprog long read           +  1.5868 S
+1379  00000FDC 76007400   $76007400     sprog long read           +  2.5867 S
+1380  00000FDC 76007400   $76007400     sprog long read           +  3.5867 S
+1381  00000FDC 76007400   $76007400     sprog long read           +  4.5867 S
+1382  00000FDC 76007400   $76007400     sprog long read           +  5.5867 S
```

Exclamation marks "!" in count columns of PC interface and terminal interface trace lists

An exclamation mark in the count column of the PC interface or terminal interface trace list indicates the counter overflowed (began again at 0) before the present state was captured. The exclamation mark warns you that the counter value may not be accurate because the analyzer is unable to determine how many times the counter overflowed between the preceding state and the state where the exclamation mark is shown.

If you were to scroll through a trace list of the entire trace memory in relative count mode, a "!" would be seen beside the first state after each occurrence of counter overflow (each 22.9 minutes). If you were to scroll through the entire trace memory in absolute count, the "!" would be seen beside every state after the first occurrence of counter overflow. Refer to the last illustrations in this chapter for an example of a terminal interface trace list with exclamation marks.

Counter overflow indication not seen in trace list

If you scroll through a reduced portion of the trace memory, one that contains no counter overflow, no counter overflow indication will be seen, even if counter overflow occurred before the line range you specified in your **display/store/copy** command. The routine that reads trace memory to compose a trace list only reads the portion of the trace memory you specify in your **display/store/copy** command.

Negative time or state counts in terminal interface trace lists

If counter overflow occurs during a trace measurement, you may see a count of negative time or negative states in the trace list. This indicates that the counter value stored with the reference state was greater than the counter value stored with the present state. In absolute time counts, negative times will continue to be seen until a state is captured whose counter value is greater than the trigger state counter value. In relative time counts, the counter value is corrected so no negative time is seen.

The last three example illustrations in this chapter show effects of counter overflow in terminal interface trace lists. The first illustration shows the terminal interface setup that was used to capture wait loop execution. This is the same wait loop that was used to demonstrate counter overflow in the workstation interface. This illustration also shows the assembly language listing of the wait loop program. The wait loop runs for approximately 1 second (999.98 mS), and then restarts.

```
  tif 1 any 2
  tif 2 never
  tsq -t 2
  tsto 1 p5
  tsto 2 p5
  tpat p5 addr=0FDCH
U$
U$m -dm 0fd2..1000
  000000fd2  -                LINK.W    A6,#$0000
  000000fd6  -                MOVE.L    D3,-(A7)
  000000fd8  -                MOVE.L    D2,-(A7)
  000000fda  -                NOP
  000000fdc  -                MOVEQ     #$00000000,D3
  000000fde  -                MOVEQ     #$00000000,D2
  000000fe0  -                NOP
  000000fe2  -                ADDQ.L    #1,D3
  000000fe4  -                NOP
  000000fe6  -                ADDQ.L    #1,D2
  000000fe8  -                NOP
  000000fea  -                CMPI.L    #$0007CA9D,D2
  000000ff0  -                BLT.B     $00000FE2
  000000ff2  -                NOP
  000000ff4  -                NOP
  000000ff6  -                BRA.B     $00000FDC
  000000ff8  -                NOP
  000000ffa  -                NOP
  000000ffc  -                MOVE.L    (A7)+,D2
  000000ffe  -                MOVE.L    (A7)+,D3
  000001000  -                UNLK      A6
U$
```

Trace tag counter of the deep analyzer

The setup of the deep analyzer (illustration on preceding page) was edited to retain only the specifications that were required to capture the demonstration illustrations. The deep analyzer was set up to transition to sequence state 2 after the capture of any state, and accept transition to sequence state 2 as the trigger. Storage was qualified only for the start state of the wait loop (address 0FDCH), using tpat p5.

The second illustration (see below) is a trace list displayed with a relative count. Counter overflow occurred between state + 1369 and state + 1370. An exclamation mark is shown in the first character space of the count column. The counter value stored with state + 1370 is the same as all the preceding counter values because it is corrected by software in the terminal interface. Therefore it gives useful information about the program being traced. This number would not be correct if the counter had overflowed two or more times between state + 1369 and state + 1370.

```
U$tf
  tf addr,h mne count,r
U$tl 0..1090
     Line  addr,H   68040 Mnemonic                                    count,R
  -------  -------- ---------------------------------------- -------------
        0  00023ff8  $27000000     sdata long write            -------------
        1  00000fdc  $76007400     sprog long read                   38.50uS
        2  00000fdc  $76007400     sprog long read                    0.88uS
        3  00000fdc  $76007400     sprog long read                 999.98274mS
        4  00000fdc  $76007400     sprog long read                 999.98280mS
        5  00000fdc  $76007400     sprog long read                 999.98282mS
                                .
                                .
                                .
     1367  00000fdc  $76007400     sprog long read                 999.98278mS
     1368  00000fdc  $76007400     sprog long read                 999.98272mS
     1369  00000fdc  $76007400     sprog long read                 999.98270mS
     1370  00000fdc  $76007400     sprog long read               ! 999.98270mS
     1371  00000fdc  $76007400     sprog long read                 999.98290mS
     1372  00000fdc  $76007400     sprog long read                 999.98290mS
     1373  00000fdc  $76007400     sprog long read                 999.98290mS
     1374  00000fdc  $76007400     sprog long read                 999.98290mS
```

The third illustration (see below) is a trace list of the same trace as in the second illustration, but displayed with an absolute count. The count stored with the trigger state (state 0) is 6.41311124 seconds greater than the count stored with state + 1370. You can see the count reducing as the count stored with each state approaches, and then exceeds, the count stored with the trigger state. An exclamation mark is shown preceding every state after the counter overflow to indicate that the count shown may not indicate the true time between the trigger state and the state shown in the trace list.

Line	addr,H	68040 Mnemonic		count,A
0	00023ff8	$27000000	sdata long write	0
1	00000fdc	$76007400	sprog long read	38.50uS
2	00000fdc	$76007400	sprog long read	39.38uS
3	00000fdc	$76007400	sprog long read	1.00002212S
4	00000fdc	$76007400	sprog long read	2.00000492S
5	00000fdc	$76007400	sprog long read	2.99998774S
.				
.				
.				
1368	00000fdc	$76007400	sprog long read	1.3659764KS
1369	00000fdc	$76007400	sprog long read	1.3669764KS
1370	00000fdc	$76007400	sprog long read	!-6.41311124S
1371	00000fdc	$76007400	sprog long read	!-5.41312852S
1372	00000fdc	$76007400	sprog long read	!-4.41314576S
1373	00000fdc	$76007400	sprog long read	!-3.41316298S
1374	00000fdc	$76007400	sprog long read	!-2.41318018S
1375	00000fdc	$76007400	sprog long read	!-1.41319738S
1376	00000fdc	$76007400	sprog long read	!-413.21458mS
1377	00000fdc	$76007400	sprog long read	! 586.76820mS
1378	00000fdc	$76007400	sprog long read	! 1.58675100S
1379	00000fdc	$76007400	sprog long read	! 2.58673382S
1380	00000fdc	$76007400	sprog long read	! 3.58671664S
1381	00000fdc	$76007400	sprog long read	! 4.58669944S
1382	00000fdc	$76007400	sprog long read	! 5.58668224S
1383	00000fdc	$76007400	sprog long read	! 6.58666504S

5

Error and Status Messages

Error and Status Messages

This chapter contains descriptions of error messages and status messages that you may see while using the deep analyzer. The error and status messages are listed in in numerical order by their error message numbers. Only error and status messages that are unique to the deep analyzer are listed in this chapter. If you do not see the message you are looking for in this chapter, refer to the manual(s) you received with your emulation system.

If you are using a workstation interface or a PC interface, you may not see the error message number along with the message on the status line. You can view the error log display of your interface to see the error message number, if desired.

1002 **!STATUS 1002! Analyzer SIMMs are not all the same size; using smallest size**

Cause: Plug-in SIMMs are used to expand the trace depth to 64k or 256k states. Four SIMMs, all of the same size must be used. If they are not all the same size, the smallest SIMM size in the set of four will be used for trace depth.

Action: No action necessary.

1203 **!STATUS 1203! Trigger position changed to accomodate trig1, trig2 delay spec**

Cause: The terminal interface **tgout** (trigger output) command provides a delay feature that allows for driving of the trig1 and/or trig2 signals a specified number of states after trigger or before trace complete. The setup of this delay feature interacts with the trigger position specification. The trigger position specification may be automatically modified by the deep analyzer in order to make the delay feature work in the expected manner.

Action: You can use the terminal interface command **tp** (trigger position) to examine the new trigger position value.

1254 **!ERROR 1254! Incompatibile signal out events: < Incompatible Event Name>**

Cause: The terminal interface **tgout** (trigger output) command may be used to drive the trig1 and/or trig2 signals to the emulator in response to several different events. The events are trigger recognition, measurement complete, finding a specified expression, delay after trigger recognition, and delay before measurement complete. Some of these events may be ORed together, but a delay specification may not be ORed with trigger recognition or mesaurement complete events.

Action: Examine your **tgout** specification and modify it to remove ORing of delay specifications with trigger recognition or measurement complete events.

1255 **!ERROR 1255! Trig1, trig2 delay spec out of bounds: < Entered Numeric Value>**

Cause: The terminal interface **tgout** (trigger output) command provides a delay feature that allows for driving of the trig1 and/or trig2 signals a specified number of states after trigger or before trace complete. The delay value must be in the range 0 through "current analyzer depth - 1". The current analyzer depth is controlled by the terminal interface command **tcf**. Note: Use of this delay feature may cause modification of the current trigger position value.

Action: Correct the delay value in your specification so that it is within the range of 0 through "current analyzer depth -1".

1256 **!ERROR 1256! Event "expr" cannot be combined with expression definition**

Cause: The terminal interface **tgout** (trigger output) command may use an arbitrary expression as an event to drive the trig1 and/or trig2 signals to the emulator. This expression can be set up two ways. One way uses two **tgout** commands; the first command defines the signals and type of events, and the second command defines the expression. This is most useful when defining complicated expressions. The other way uses one **tgout** command which defines the expression as the event. This error message indicates that you have tried to combine the two methods.

Action: Reenter your **tgout** command using the correct format for the command. Refer to the **tgout** command description in the chapter titled "Interfaces of the Deep Analyzer" for correct formats for the **tgout** command.

IL# in trace list Mnemonic column

The notation # # IL# may appear in place of an inverse-assembled instruction in the Mnemonic column of a trace list. This notation indicates the inverse assembler was unable to find the information it needed to complete a trace list line. Probably, the information was not available in the trace memory.

For example, you would see the # # IL# notation in your trace list if you were tracing an Intel processor and you qualified capture of only instruction execution cycles during your trace. The inverse assembler would look for an opcode to associate with each instruction execution, but it would find none. When its search for an opcode timed out, it would place # # IL# in the Mnemonic column of the trace list.

The following trace list was obtained to show the appearance of # # IL# in a trace list Mnemonic column. The example trace list was obtained from an 80186 emulator using its terminal interface.

```
   Line  addr,H 8018x mnemonic,H                          count,R seq
-------  ------ ------------------------------------ ------------- ---
 26200  81991   ##IL#                                    3.00uS  .
 26201  81994   ##IL#                                    3.00uS  .
 26202  81996   ##IL#                                    1.00uS  .
 26203  8197f   ##IL#                                    3.26uS  .
 26204  81984   ##IL#                                    4.98uS  .
 26205  81985   ##IL#                                    1.02uS  .
 26206  8198a   ##IL#                                    4.00uS  .
```

The analyzer takes a long time to compose a trace list when the inverse assembler times out before it places each line on screen. Typically, the analyzer takes 2 seconds to place each line on screen when it fails to find the information it needs and places # # IL# on screen, instead.

6

Installation and Service

This chapter shows you how to install the analysis hardware. It also shows you how to verify installation by using the performance verification procedure. These installation tasks are described in the following sections:

The following information is covered in this chapter:

- Steps to follow when installing analyzer hardware.
- How to verify proper installation and operation of analyzer hardware.
- A complete list of replaceable parts for the analyzer, and instructions on how to order replaceable parts.

Installing hardware

This section shows you how to install the analysis hardware, and how to connect to the emulator hardware and optional software performance analyzer.

Equipment supplied

The minimum system contains:

- HP 64794 Emulation-Bus Analyzer (deep analyzer) card.
- Ribbon cable.
- HP Emulation Control card.
- HP 64700A Card Cage.

Optional parts are:

- HP 64172A 256-Kbyte Memory Modules (for additional memory depth).
- HP 64172B 1-Mbyte Memory Modules (for additional memory depth).
- HP 64708A Software Performance Analyzer.

Equipment and tools needed

If you must remove circuit cards from the card cage before you can install the deep analyzer hardware, you need:

- Flat-blade screwdriver with shaft at least 5 inches long (13 mm approx).

Installation overview

The steps in the installation process are:

1 Install the optional memory modules on the deep analyzer card, if desired.

2 Install the deep analyzer card in the HP 64700A Card Cage.

3 Connect the ribbon cable from the deep analyzer card to the emulation control card, if required. Some emulators do not have a connection for this cable.

4 Apply power to the HP 64700A Card Cage.

Your emulation and analysis system may already be installed, depending on how parts of the system were ordered.

Antistatic precautions

Printed-circuit cards contain electrical components that are easily damaged by small amounts of static electricity. To avoid damage to the emulator and analyzer cards, follow these guidelines:

- If possible, work at a static-free workstation.
- Handle the cards only by the edges; do not touch components or traces.
- Use a grounding wrist strap that is connected to the HP 64700A chassis.

Note

If you already have a modular HP 64700A Card Cage and want to remove the 1K analyzer and install the deep analyzer in its place, the analyzer firmware will be updated by your installation because the analyzer firmware is contained on the analyzer card.

Checking hardware installation

After hardware installation, run a performance test to verify that the analyzer and emulator are working properly. The performance verification procedure is described under "Verifying the installation," later in this chapter.

Service information

Use this chapter when removing and installing hardware, running performance verification, and ordering parts. See the HP 64700 Series Installation/Service Guide for information on system configurations, installing product software, software updates, and ordering parts for the card cage. Turn off power to the card cage before removing or installing hardware.

Step 1. Install optional memory modules, if desired

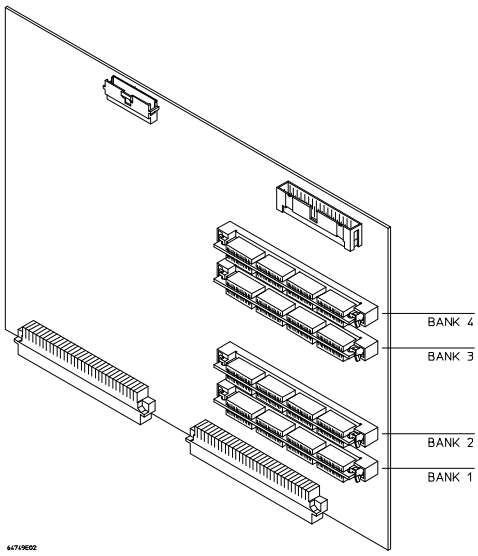
Observe antistatic precautions

With no optional memory modules installed on the deep analyzer card, the trace memory depth is 8K. If you are going to use the analyzer with this default trace memory depth, skip this step and proceed to Step 2 of this installation procedure.

1 Determine placement of the optional memory modules. Two types of modules may be installed: 256-Kbyte (HP 64172A), and 1-Mbyte (HP 64172B). Either module type may be installed in the banks on the analyzer card. Do not use HP 64171A/B memory modules; they are too slow.

If you install no memory modules, the deep analyzer will have 8K maximum memory depth.
If you install four 256-Kbyte memory modules, the analyzer will have 64K maximum memory depth.
If you install four 1-Mbyte memory modules, the analyzer will have 256K maximum memory depth.

If you install a combination of 256-Kbyte memory modules and 1-Mbyte memory modules, the analyzer will have 64K maximum memory depth. All four connectors must have memory modules installed before the analyzer depth will be increased.



Step 1. Install optional memory modules, if desired

2 Install optional memory modules on the deep analyzer card. There is a cutout at one end of the memory modules so they can only be installed the correct way.

To install a memory module:

Align the groove in the memory module with the alignment rib in the connector.

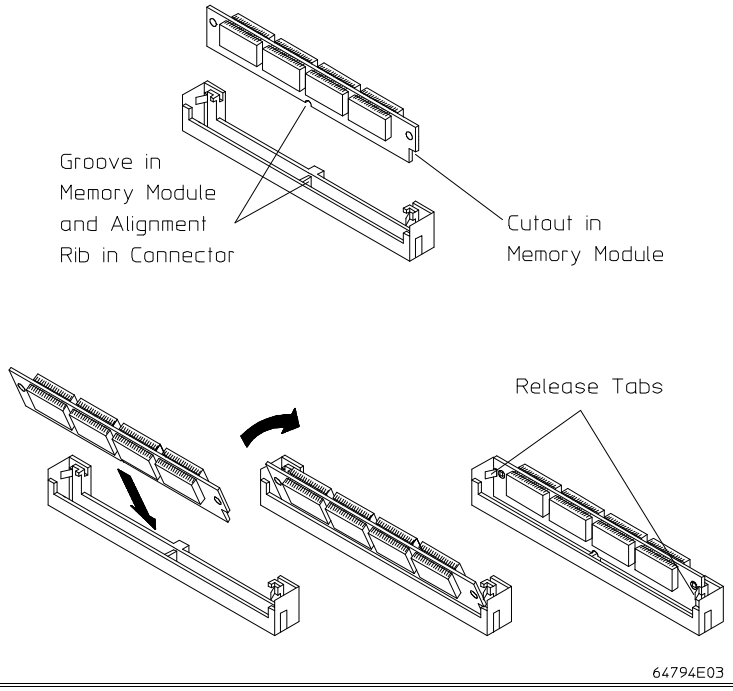
Align the cutout in the memory module with the projection in the connector.

Place the memory module into the connector groove at an angle.

Firmly press the memory module into the connector and make sure it is completely seated.

Rotate the memory module forward so that the pegs on the connector fit into the holes on the memory module.

Make sure the release tabs at each end of the connector snap around the memory module to hold it in place.

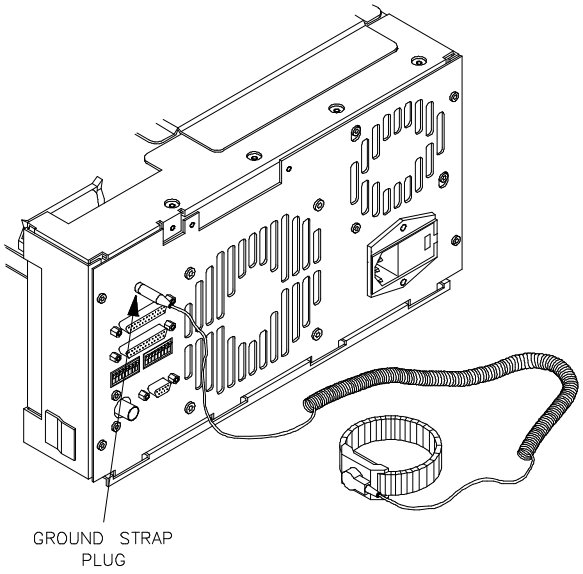


Step 2. Install the deep analyzer card in the HP 64700A card cage

WARNING **Before removing or installing parts in the HP 64700A card cage, make sure the card cage power is off and the power cord is disconnected.**

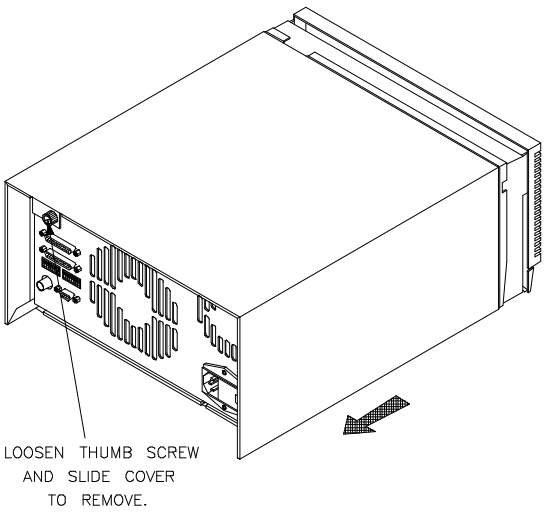
CAUTION Do NOT stand the HP 64700A card cage on the rear panel. You might damage the rear panel ports and connectors.

1 Use a ground strap when removing or installing cards in the HP 64700A card cage to reduce the risk of damage to the circuit cards from static discharge. A jack on the rear panel of the HP 64700A card cage is provided for this purpose.

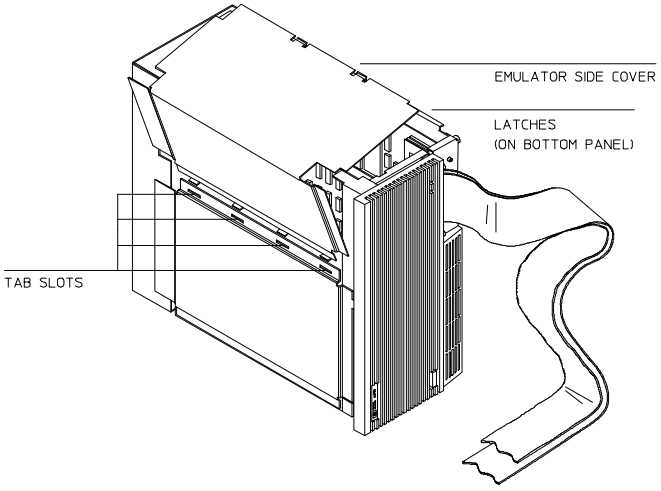


Step 2. Install the deep analyzer card in the HP 64700A card cage

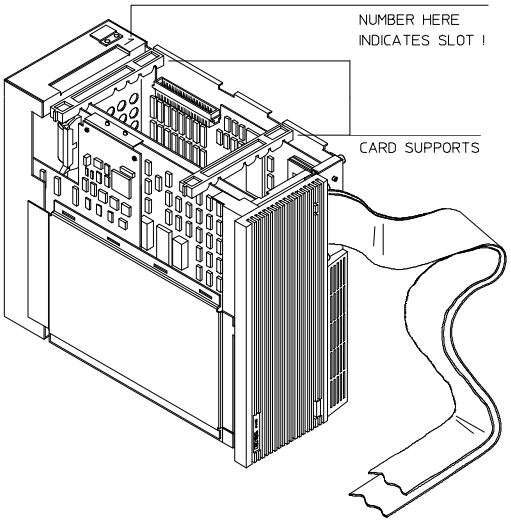
2 Turn the thumb screw and remove the top cover by sliding the cover toward the rear and up.



3 Remove the side cover by unsnapping the two latches and lifting off.



4 Remove the card supports.



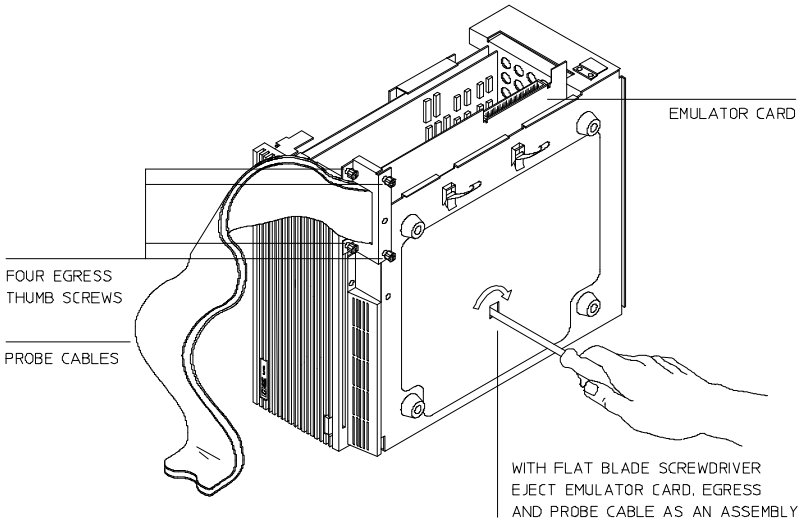
Step 2. Install the deep analyzer card in the HP 64700A card cage

5 If you do not need to remove an existing analyzer card from the HP 64700A Card Cage, skip this step and the next two steps. Go to step number 8 in this procedure.

Remove any connections across the tops of the cards.

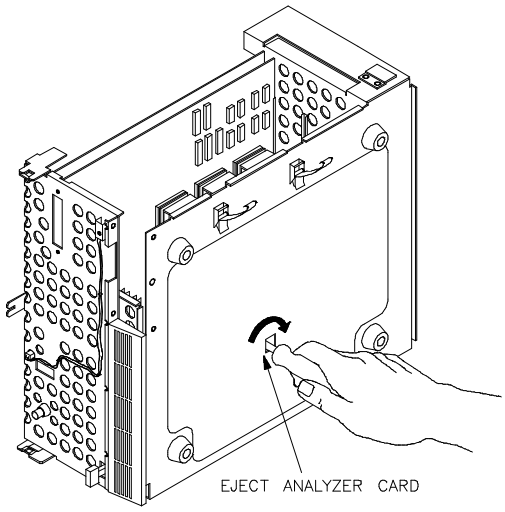
The emulator card(s) must be removed before the analyzer card can be removed. To remove the emulator card(s), first, completely loosen the four egress thumb screws.

Insert a flat blade screwdriver in the access hole and eject each card, one at a time, by rotating the screwdriver.



6 Remove the software performance analyzer, if installed.

7 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 card. This card is used in all HP 64700 emulation and analysis systems.

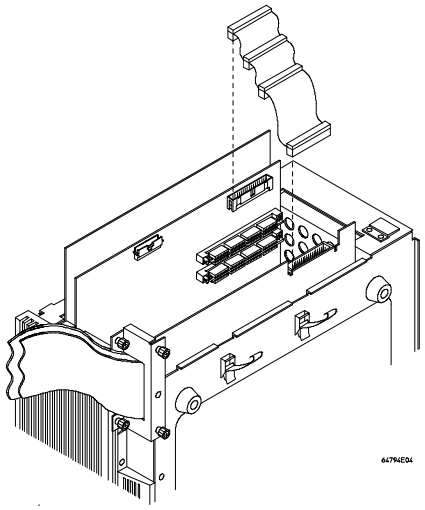
Step 2. Install the deep analyzer card in the HP 64700A card cage

8 Install the deep analyzer card and the emulation control card(s). The deep analyzer card is installed in the slot next to the system controller card. The emulation control card(s) is installed in the first slot at the bottom of the HP 64700A Card Cage. The software performance analyzer card may occupy any slot between the deep analyzer and the emulation control card(s). These cards are identified with labels that show their model numbers and serial numbers. Note that components on the analyzer card face the opposite direction to the other cards.

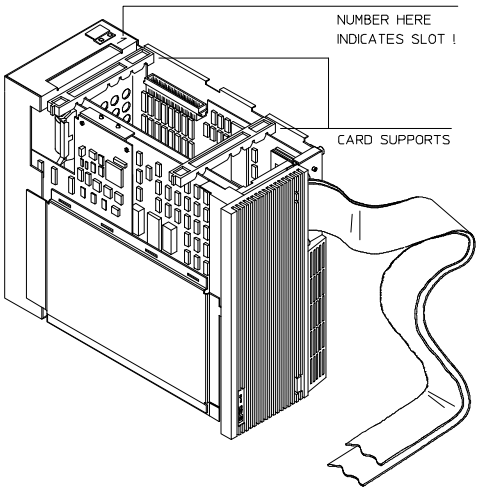
To install a card, insert it into the plastic guides. Make sure the connectors are properly aligned; then, press the card into the mother board socket. Ensure that each card is seated all the way into its mother board socket. If a card can be removed with your fingers, it is NOT seated all the way into its mother board socket.

Tighten the thumbscrews that hold the emulation control card to the HP 64700A Card Cage frame.

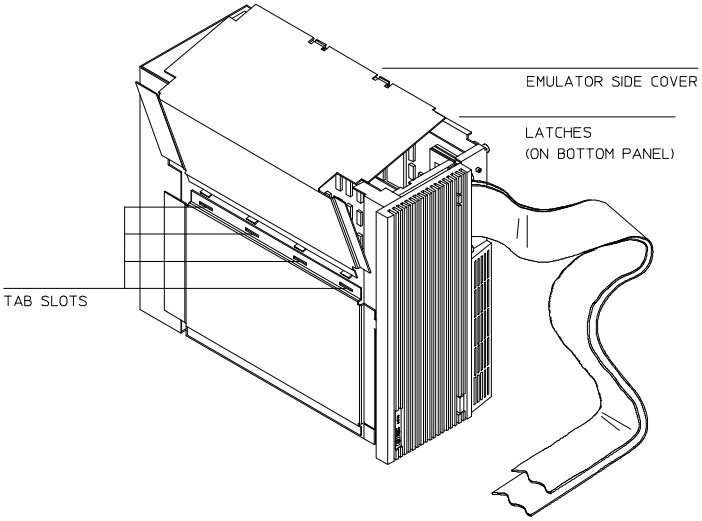
Attach the ribbon cable from the deep analyzer card to the emulation control card (if there is a connector on the emulation control card), and to the software performance analyzer card, if installed.



9 Install the card supports.

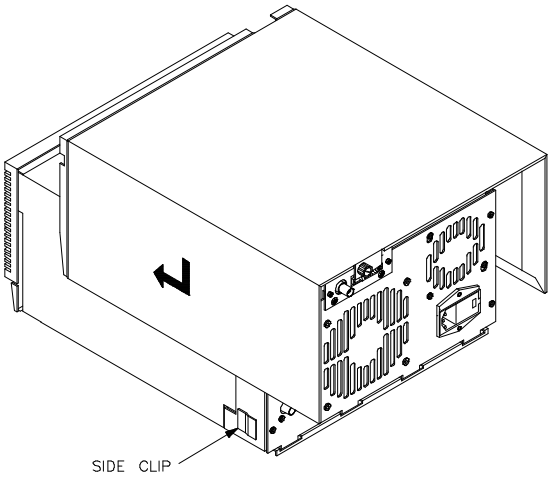


10 To install the side cover, insert the side cover into the tab slots and fasten the two latches.



Step 2. Install the deep analyzer card in the HP 64700A card cage

11 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 3. Turn on power

Connect the power cord to the rear panel of the HP 64700A Card Cage, and turn on power. The line power switch is a pushbutton located at the lower, left-hand corner of the front panel. To turn ON power, push the line switch button in to the ON position. The power light at the lower, right-hand corner of the front panel will be illuminated.

Verifying the installation

Verify installation by running the performance verification procedure for your emulator/analyzer. The exact performance verification procedure to follow depends on which emulator you have installed in your card cage, and which emulation interface you are using. Follow the procedure for your emulator/analyzer and its interface, as outlined in your emulator/analyzer manual.

If your installation is correct, the performance verification procedure should give you a display similar to the following:

```
Testing:  HP64783A Motorola 68040 Emulator
     PASSED:
  Number of tests:  1           Number of failures:  0
Testing:  HP64740 Compatible (PPN: 64794A) Deep Emulation 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.

  HP64700 Series Emulation System
  Version    A.04.00 22Oct92
HP64783A Motorola 68040 Emulator
HP64740 Compatible (PPN: 64794A) Deep Emulation Analyzer
HP 64701A LAN Interface
```

If you have an emulation failure, you can replace the assembly that failed through your local Hewlett-Packard representative, and through the Support Materials Organization (SMO). Refer to the list of replacable parts at the end of this chapter.

To verify installation of the optional memory modules on the deep analyzer card, type the **ver** command, as follows:

M> **ver**

```
HP64740 Compatible (PPN: 64794A) Deep Emulation Analyzer
  Version:   A.03.00 25Jun93 Unreleased
  PC Board:  794-01-A1
  Depth:     80ch X 1K states selected, 80ch X 64K states available
    Bank 0:    HP64172A (20ns) 256 Kbyte Module
    Bank 1:    HP64172A (20ns) 256 Kbyte Module
    Bank 2:    HP64172A (20ns) 256 Kbyte Module
    Bank 3:    HP64172A (20ns) 256 Kbyte Module
```

What is pv doing to the analyzer?

The performance verification procedure provides a thorough check of the functionality of all of the products installed in the HP 64700A Card Cage. The exact set of test modules contained in the performance verification procedure varies from one product to another. After the performance verification is run, all products are set to their power on initialization states.

Troubleshooting

The test results for all of the performance verification modules are indicated by a simple PASS/FAIL message. The PASSED message gives a high level of confidence that all major functions and signals are operating properly.

A FAIL message indicates that one or more of the tested functions is NOT working. In this event, an HP field representative can either swap assemblies to isolate the failure, or replace all of the major assemblies. The optional memory modules are not part of the analyzer card assembly. These memory modules must be ordered separately. Remove these memory modules from the analyzer card before replacing the card assembly.

Parts list

What is an exchange part?

Exchange parts are shown on the parts list. A defective part can be returned to HP for repair in exchange for a rebuilt part.

Analyzer card (exchange)

To replace the analyzer card on the exchange program, you must remove the parts listed below, and return only the analyzer card. Remove:

- the ribbon cable that connects the emulation control card to the analyzer card.
- the optional memory modules installed in BANK1 through BANK4.

Main Assembly **Component Part**	**New**	**Exchange**
HP 64794A Emulation-Bus Analyzer (deep) Card	64794-66501	64794-69501
34-pin ribbon cable	64708-61601	
HP 64172A 256-Kbyte SRAM Module	64172A	64172-69501
HP 64172B 1-Mbyte SRAM Module	64172B	64172-69502

Glossary

absolute count An absolute count in the trace list count column indicates the total count accumulated between the displayed state and the trigger state. For example, an absolute time count shown beside trace memory line number 100 indicates the elapsed time between capture of the trigger state and capture of state 100.

analyzer clock speed The analyzer clock speed is defined as the bus cycle rate of the emulation processor. If the emulation processor is running at 21 MHz and the fastest bus cycle requires 3 clocks, then the analyzer clock speed (bus cycle rate) is 21/3 = 7 MHz.

compatible mode The compatible mode of the deep analyzer configures the analyzer to provide the same memory depth as the 1K analyzer: 1024 states deep when the analyzer is not configured to make a count of states or time during a measurement, and 512 states deep when the analyzer is configured to make a count of states or time during a measurement. If the emulator interface you are using along with the deep analyzer requires that you use the compatible mode, the deep analyzer will still be able to provide one of its benefits for your measurement; you will be able to make your counts of states or time at full emulator clock speed.

counter overflow This is the condition when the counter reaches maximum count and begins a new count from zero. The counter simply counts continuously once a trace begins; it increments its count every 20 ns, and reaches maximum count in about 22.9 minutes (22 minutes and 54 seconds). The deep analyzer sets a flag in memory and stores it along with the first state that is captured after the counter overflow occurs (first state captured after the counter begins again at zero).

deep analyzer In this manual, the term "deep analyzer" refers to the HP 64794 Emulation-Bus Analyzer with deep trace memory.

expression In the terminal interface of your emulator, a simple expression is the information that can fit into a single pattern or a single range (such as, **addr= 2105**, **data!= 15**, and **addr= 4012..401a**). A complex expression is made up of pattern, range, and arm labels joined together by various operators that define the specific condition. Each of the pattern and range labels must be previously assigned to specific simple expressions using the terminal interface commands: **tpat** and **trng**.

label A label identifies a set of one or more analyzer channels. Example, the label "addr" is used to identify the analyzer channels connected to the address bus of the emulation processor.

overflow See counter overflow.

prestore qualifier A specification that must be met by a state before it can be saved in the analyzer prestore memory.

qualifier A specification that must be met before an action can be taken by the analyzer. For example, a store qualifier is a specification that must be met by an incoming state before it can be stored in the trace memory. The "arm" condition can be used as an additional qualifier. For example, an external analyzer may be set up to supply a true signal to the rear panel BNC connector on the card cage when it detects a true condition in the target system. Then the analyzer can be set up to store qualify a certain kind of state, but only when the arm signal from the BNC is true.

relative count A relative count in the trace list count column shows the count between the present displayed state and the state displayed immediately before it. Relative time count, for example, shows the elapsed time between the previous displayed state and the present state. Note that the count is between displayed states. If your trace list is inverse assembled and/or dequeued, several states may have been captured in memory between the present displayed state and the displayed state immediately before it.

store qualifier A specification that must be met by a state before it can be saved in the analyzer trace memory.

trigger The trigger signals are called trig1 and trig2. They are bidirectional signal lines that can be used to coordinate measurement activity between emulators and analyzers installed in the instrumentation card cage, and between instruments connected to the BNC on the rear panel of the card cage. For details of how to configure and use trig1 and trig2, refer to the chapter on "Tasks you can do with the deep analyzer" in this manual, and the chapter on making coordinated measurements in your emulator/analyzer manual(s).

Note that there is delay when you use trig1 and/or trig2 for measurement coordination. For example, you may specify that the emulator break to its monitor program when it receives trig1 from the analyzer. Several states may be executed in the emulator between the time the analyzer recognizes its trigger condition, generates trig1, delivers trig1 to the emulator, and the emulator responds to trig1 by breaking to its monitor program.

1K analyzer In this manual, the term "1K analyzer" refers to the HP 64703, HP 64704, and HP 64706 Emulation-Bus analyzers with 1K trace memories.

! When shown in the trace list count column of the terminal interface or the PC interface, the exclamation mark "!" indicates counter overflow.

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. 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.

Designed to Meet Requirements of IEC Publication 348

This apparatus has been designed and tested in accordance with IEC Publication 348, safety requirements for electronic measuring apparatus, and has been supplied in a safe condition. The present instruction manual contains some information and warnings which have to be followed by the user to ensure safe operation and to retain the apparatus in safe condition.

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.

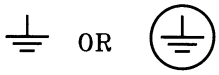
Safety Symbols Used In Manuals

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.

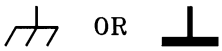
Hot Surface. This symbol means the part or surface is hot and should not be touched.

Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).



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.



rame or chassis terminal. A connection to the frame (chassis) of the quipment 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.