User's Guide

Software Performance Analyzer (HP B1487A)

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.

HP is a trademark of Hewlett-Packard Company.
UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.
SunOS, SPARCsystem, Open Windows, and Sun View are trademarks of Sun Microsystems, Inc.
Microtec is a registered trademark of Microtec Research, Inc.
TORX is a registered trademark of the Camcar Division of Textron, Inc.

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. The date on the title page changes only when a new edition is published.

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

Edition 1	B1487-97000	October 1991
Edition 2	B1487-97001	November 1991
Edition 3	B1487-97002	October 1992
Edition 4	B1487-97003	April 1993
Edition 5	**B1487-97004**	**January 1994**

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 book documents the HP B1487A Software Performance Analyzer. It is the only book you will need to operate and service the analyzer. It contains:

Part 1. Quick Start Guide

Chapter 1 presents an overview of the Software Performance Analyzer and quickly shows you how to use it.

Part 2. Making Measurements with The Software Performance Analyzer

Chapter 2 shows how to prepare to make measurements.
Chapter 3 shows how to make measurements.
Chapter 4 shows how to obtain and interpret measurement results.
Chapter 5 shows how to perform miscellaneous tasks, such as saving measurement specifications, and turning off event numbers.
Chapter 6 lists problems you may see and ways to clear the problems.

Part 3. Concepts

Chapter 7 lists and describes the kinds of measurements that are made. It also shows the kinds of problems you can solve with the Software Performance Analyzer, and the effects of caches, prefetching, and recursive functions.
Chapter 8 shows the equations used by the Software Performance Analyzer.

Part 4. Reference

Chapter 9 shows details of the Softkey and Graphical User Interfaces.
Chapter 10 shows the syntax of analyzer commands, and the command tokens.
Chapter 11 lists error messages, their causes and corrections .
Chapter 12 interprets the events list and helps you understand events.
Chapter 13 shows how to interpret displays of measurement setups and results.
Chapter 14 shows how to use triggers with the host emulator and/or emulation bus analyzer.
Chapter 15 lists commands that do not appear on softkeys, and shows their use.
Chapter 16 lists the specifications of the Software Performance Analyzer.

Part 5. Installation and Service

Chapter 17 shows how to install and service the Software Performance Analyzer .
Chapter 18 shows how to install and update firmware into the Flash EPROM memory of the analyzer card using "progflash".

Contents

Part 2 Making Measurements With The Software Performance Analyzer

2 Preparing the Software Performance Analyzer to Make Measurements

3 Controlling the Profile Measurement

4 Managing the Display of Measurement Results

5 Supporting Tasks that Add Flexibility to Performance Measurements

Part 3 Measurement Concepts

7 Software Performance Measurement Techniques and Difficulties

Part 4 Reference

9 The User Interface

10 Syntax of the Software Performance Analyzer Commands

15 Hidden Commands of the Software Performance Analyzer

16 Software Performance Analyzer Specifications

Part 5 Installation and Service

17 Installation and Service

Part 1

Quick Start Guide

Part 1

This part of the manual contains the following chapter:

Chapter 1. Quick start guide

1

Quick Start Guide

In this chapter, you will see typical measurements that will help you become familiar with the Software Performance Analyzer. At the end of this chapter you will see how to use on-line help to answer many of your questions on screen. Install the hardware and software according to Chapter 17 before performing these procedures.

The Software Performance Analyzer - At a Glance

The Software Performance Analyzer helps you understand the execution of software modules in an executable file. It measures execution of software modules, interation between software modules, and usage of data points and I/O ports. It answers questions, such as:

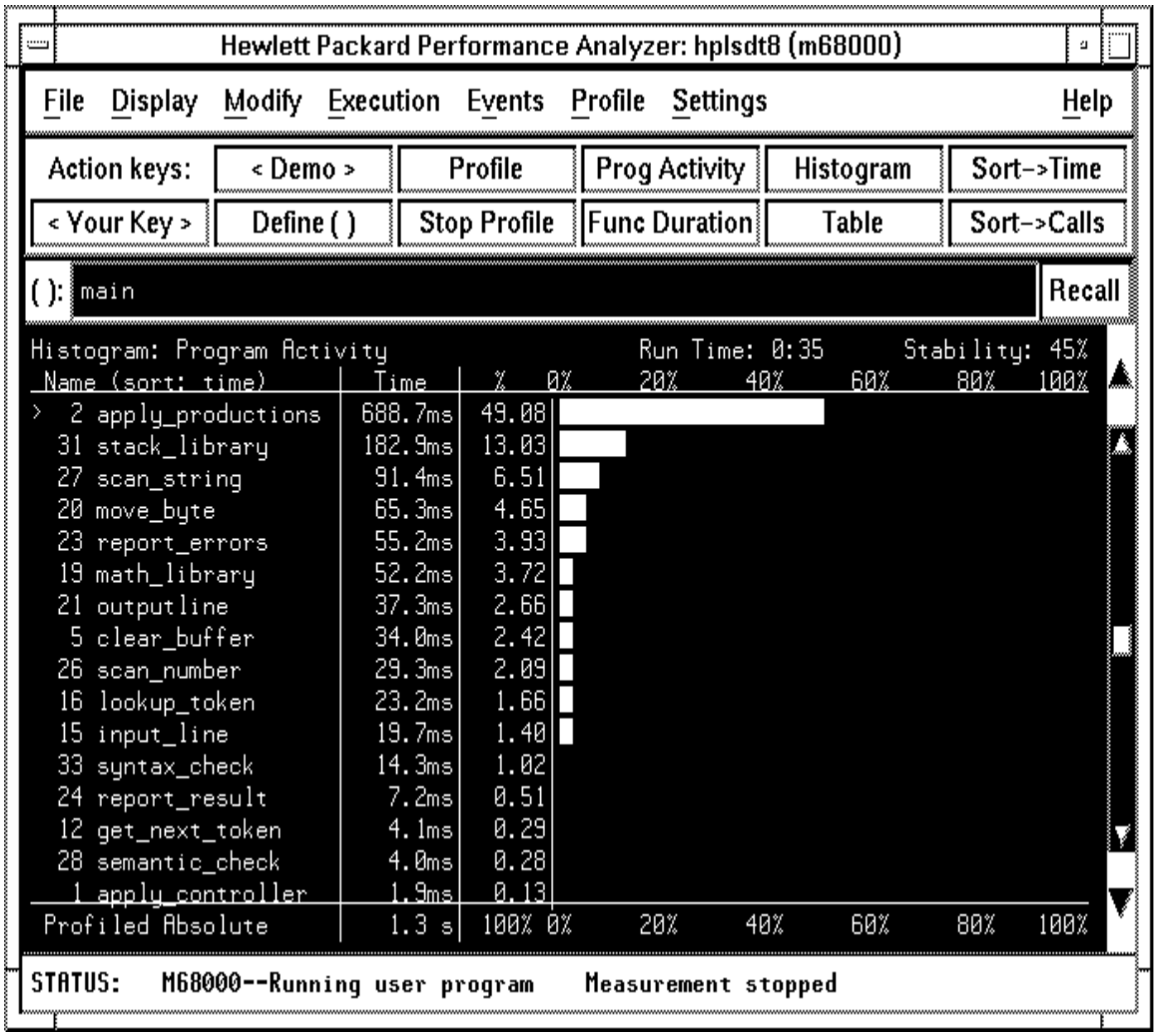
"Why does it take so long to execute my program?"
"Which modules are slowing down overall program execution?"
"How intensive is my use of this data or I/O port?"
"How much time is spent getting between various points in my program?"
"Which function is called most often?"
"Which functions are taking the most processor time?"
"What is the time distribution of calls to my functions?"

The Software Performance Analyzer consists of measurement software that runs on an analyzer circuit board. The Software Performance Analyzer can operate through one of two user interfaces: (1) the Softkey User Interface shown on this page, or (2) the Graphical User Interface shown on the next page. The Softkey User Interface requires you to either press softkeys or type commands on the command line. The Graphical User Interface allows you to compose commands by moving a cursor with a mouse to select desired options or softkeys, and then clicking the mouse button. If you have a display capable of running X windows,

```
Histogram: Function Duration include calls    Run Time: 0:41      Stability: 99%
_Name_(sort:_time)_____|__Time__|__%___0%_____20%_____40%_____60%_____80%___100%
  22 parse_command     |  37.9 s| 90.68|**********************************
   1 apply_controller  |  31.9 s| 76.22|****************************
   2 apply_productions |  27.2 s| 65.01|************************
  12 get_next_token    |   6.2 s| 14.83|******
  31 stack_library     |   6.1 s| 14.59|******
  16 lookup_token      |   5.3 s| 12.62|*****
  28 semantic_check    |   3.8 s|  9.08|****
  27 scan_string       |   2.9 s|  7.05|***
  25 request_command   |   2.1 s|  4.90|**
  14 initialize        |   2.0 s|  4.80|**
  19 math_library      |   1.9 s|  4.64|**
  20 move_byte         |   1.8 s|  4.38|**
  23 report_errors     |   1.8 s|  4.37|**
  24 report_result     |   1.5 s|  3.52|*
___5_clear_buffer_____|___1.2_s|__2.90|*_____________________________________________
 Profiled Absolute     |  41.8 s| 100% 0%     20%     40%     60%     80%   100%

STATUS:   M68000--Running user program    Measurement in process________........
profile function_duration include_calls


profile   define   setup   display          EXPAND   delete    end    ---ETC--
```

you can use either interface. If you are using a terminal, you must use the Softkey User Interface.

In order to make a performance measurement of an address or range of addresses in the source file, the Software Performance Analyzer must have an event defined to represent that address or range of addresses. The Software Performance Analyzer can automatically define the events it needs for measurements. Simply decide which measurements you want to make. The Software Performance Analyzer will



define the events, run the measurement, and present the results. Of course, you can also define events yourself if you have special requirements.

The following four kinds of events can be defined for measurements by the Software Performance Analyzer:

- *functions*, and *recursive functions*: events that represent functions in the program memory.
- *static variables*: events that represent variables at fixed addresses within the program (not variables in stack space).
- *ranges*: events that represent a contiguous group of functions, static variables, or memory locations.
- *intervals*: events that represent a pair of addresses (an interval-start address and an interval-stop address).

The Software Performance Analyzer can make the following four profile measurements of the event types (italics) listed above:

- Program Activity. This records instruction execution within *functions* or *ranges* of program memory.
- Memory and I/O Activity. This records reads and/or writes to *static variables* or *ranges* of addresses in data memory and I/O space.
- Function Duration. This records the execution periods of selected source-file *functions*.
- Interval Duration. This records periods between occurrence of a start address and an end address of an *interval*. Any addresses can be specified to be the start and end addresses of an *interval* (as long as they are not also the start and/or end address of another *interval*).

When the Software Performance Analyzer makes a measurement, it displays the results in two formats: histogram, and table. Histograms show graphic bars beside each of the events in the measurement. The lengths of the bars are proportional to the information measured for the associated events. Tables show columns of numbers beside each of the events. These columns give statistical details about executions of each of the events.

Once the Software Performance Analyzer has identified the functions (or other software structures) that are slowing down the performance of your system, you can modify their source code to make them run faster.

To use the interface in this demonstration

The procedures in this chapter are written to help you use either the Graphical User Interface or the Softkey User Interface. Each procedure shows command forms to use in both interfaces.

The command form to use in the Graphical User Interface is given first. A typical command will be: **Display→Sort_Events→Time**. To execute this command, place the cursor on **Display** in the menu bar (top line of the interface) and press the *command select* mouse button. A pull-down will appear. Move the cursor to **Sort_Events** in the pull-down. A small menu will appear. In the small menu, move the cursor to **Time** and release the mouse button.

The second command form in each procedure will show how to enter commands in the Softkey User Interface. In this interface, enter commands by either typing them on the command line, or by pressing softkeys. To enter the command in the above example, you might press the following softkeys: **display, sort, time**, and Return.

Step 1. Obtain the demonstration program

- Enter the following command:

cd $HP64000/demo/spa/demo1

Where **$HP64000** is the path to your hp64000 directory. This directory contains a demonstration program that is precompiled to run with your emulator. The source of this demonstration program is listed in the file runtest.c. An example of the Makefile that was used to make the executable is contained in the file Makefile. In this getting started example, you will load a configuration file that is appropriate for your emulator and the appropriate executable. The Software Performance Analyzer will make measurements on the executable as it runs on your emulator.

The executable for the 80960 Sx emulator (r80960.x) was made with MRI Compiler tools and linked in the appropriate initialization routines to run with the 80960 Sx emulator. Refer to the 80960Sx debug environment demo directory ($HP64000/demo/debug_env/hp64760) for an example Makefile. Remember to compile all files you want to analyze with the "-Kt" compiler option.

Step 2. Start your emulator and run the demonstration program

- If using the Graphical User Interface, enter the following commands:

  1 **emul700 <logical name>**

  2 **File→Load→Emulator Config...**. A dialog box opens. Click on **<directory>/c<processor_number>.EA** and **OK** in the dialogue box.

  3 **File→Load→Executable...**. A dialog box opens. Click on **<directory>/r<processor_number>.x** and **OK** in the dialog box.

  4 **Execution→Run→from Transfer Address** (for most emulators. See exceptions below.)

- If using the Softkey User Interface, enter the following commands:

  1 **emul700 -u skemul <logical name>**

  2 **load configuration c<processor_number>.EA**

  3 **load r<processor_number>.x**

  4 **run from transfer_address** (for most emulators. See exceptions below.)

  - Notes for the above procedures:

    <logical name> = the name you assigned to identify your emulation system.

    <processor_number> identifies your emulation processor (for example, 68000 for an M68000 emulation processor, 80960 for an Intel 80960 Sx emulation processor, etc.).

  - Exceptions for Step 4 above:

    If you are using the Software Performance Analyzer with a 68340 Emulator, use **run from reset**, and then **run from transfer_address**.

    If you are using the Software Performance Analyzer with an 80960 Emulator, use **run from reset**.

The above commands set up your emulator so that the Software Performance Analyzer can analyze the demonstration program. These commands loaded a configuration file, loaded an executable, and started the executable running in emulation. In general, you will follow the same steps for any programs you wish to analyzer with the Software Performance Analyzer.

Step 3. Start the Software Performance Analyzer

- Enter one of the following two commands.

  - If using the Graphical User Interface, choose:

    File→Emul700→Graphic Windows, Performance Analyzer...

  - If using the Softkey User Interface, enter:

    !emul700 -u skperf <logical name>

    The above commands call the Software Performance Analyzer to operate through the desired interface. You will find the Graphical User Interface more user-friendly. The Softkey User Interface can can provide faster response on smaller HP 9000 Series 300 systems.

Step 4. Start a performance measurement

- If you are using the Graphical User Interface, select one of the two actions below:
  - Click on the **Profile** action key (second line of the interface).
  - Move the cursor into the keywords line (top line of the display), and choose **Profile→Profile Again**.
- If you are using the Softkey User Interface, enter the command: **profile**

  This causes the Software Performance Analyzer to make a default measurement of program activity. A histogram of program activity will be shown on the screen.

```
Histogram: Program Activity                    Run Time: 1:13      Stability: 57%
_Name_____________________|__Time__|__%___0%_____20%_____40%_____60%_____80%___100%
   1 apply_controller     |   3.2ms|  0.11|
   2 apply_productions    |   1.4 s| 49.88|********************
   3 atexit               |   0.0us|  0.00|
   4 calculate_answer     |   2.7ms|  0.09|
   5 clear_buffer         |  73.4ms|  2.53|*
   9 endcommand           |  27.5us|  0.00|
  11 format_result        |   1.5ms|  0.05|
  12 get_next_token       |   6.4ms|  0.22|
  14 initialize           |   1.7ms|  0.06|
  15 input_line           |  55.5ms|  1.92|*
  16 lookup_token         |  56.6ms|  1.96|*
  17 main                 | 196.0us|  0.01|
  19 math_library         |  98.5ms|  3.40|*
  20 move_byte            | 134.8ms|  4.65|**
__21_outputline__________|__81.4ms|__2.81|*_____________________________________
 Totals Absolute          |   2.8 s| 100% 0%     20%     40%     60%     80%   100%
```

Here's what happened when you entered the "profile" command:

1 The Software Performance Analyzer checked its events list to see if any events had been defined for a measurement. Since no events had been defined, the Software Performance Analyzer defined a set of events as outlined in the next step.

2 The Software Performance Analyzer accessed the symbols data base for the absolute file running in emulation, and defined events to represent those symbols, as follows:

Up to 1000 events were defined to represent the first 1000 functions or static variables found in the symbols data base. Any symbol beginning with "_" was ignored (compilers generate symbols whose names begin with "_").

Each Software Performance Analyzer event represents one source-file symbol (either a function, or a variable).

The Software Performance Analyzer assigned a name and address to each event:
(1) the event name is the same as the symbol name it represents.
(2) the address is the same as the address or address range occupied by the code associated with the source-file symbol.

3 The Software Performance Analyzer performed a measurement of program activity (the default measurement). The Software Performance Analyzer can include up to 254 events in a single program-activity measurement. The first 254 functions in the events list were included.

4 The Software Performance Analyzer presented a histogram of the program activity it found for each of the selected events. The lengths of the bars on the histogram are proportional to the totals of the times measured for the associated events.

Activity measurements record information on up to 254 events. The Software Performance Analyzer can make two kinds of activity measurements: program activity, and memory and I/O activity. Program activity measurements obtain information about program execution. Memory and I/O activity measurements record usage of static variables and I/O addresses.

The Software Performance Analyzer makes activity measurements by sampling. That is, it activates one event in its events list at a time. The Software Performance Analyzer records incoming bus cycles if they meet the specifications of the active event. If not, the bus cycles are ignored.

After about 2.5 ms, the Software Performance Analyzer deactivates its present event (stops recording bus cycles for it), and activates the next event in the list (starts recording bus cycles for it). When the Software Performance Analyzer has sampled activity for each of the events in the list, it starts over again with the first event. The Software Performance Analyzer continues to sample and record incoming activity until the measurement ends.

Step 5. Sort the histogram to show events that used the most execution time

- If you are using the Graphical User Interface, select one of the two actions below:
  - Click on the **Sort->Time** action key.
  - Move the cursor into the keywords line (top line of the display), and choose **Display→Sort_Events→Time**.
- If you are using the Softkey User Interface, enter the command:
  display histogram sort_events time

  The functions that used the most system time are shown at the top of the histogram display.

Step 6. Look at the table of statistical information recorded in the measurement

- If you are using the Graphical User Interface, select one of the two actions below:
  - Click on the **Table** action key.
  - Move the cursor into the keywords line, and choose **Display→Table**.
- If you are using the Softkey User Interface, enter the command: **display table**

  The table display shows the event measurement in greater detail. The following information is shown:
  - Event name and event number.
  - Cycles. This is a count of the number of bus cycles that were associated with this event during the measurement.
  - Time. This is a record of the total time of all bus cycles that were associated with the event.

```
Table: Program Activity                          Run Time: 1:13      Stability: 57%
_Name_(sort:_time)_____|_Cycles_|__Time__|_Time_%_|_Mean(1s)|_StDv(1s)|_Time/cyc
   2 apply_productions | 2.64E06|   1.4 s|  49.88 |  498.8ms|  446.5ms|  546.9ns
  31 stack_library     |  782034| 419.8ms|  14.49 |  144.9ms|  248.5ms|  536.8ns
  27 scan_string       |  404891| 214.7ms|   7.41 |   74.1ms|  208.5ms|  530.2ns
  20 move_byte         |  251109| 134.8ms|   4.65 |   46.5ms|  141.5ms|  536.9ns
  23 report_errors     |  250261| 134.3ms|   4.64 |   46.4ms|  140.8ms|  536.8ns
  19 math_library      |  180832|  98.5ms|   3.40 |   34.0ms|  101.3ms|  544.8ns
  21 outputline        |  156125|  81.4ms|   2.81 |   28.1ms|  158.9ms|  521.2ns
   5 clear_buffer      |  134565|  73.4ms|   2.53 |   25.3ms|  142.0ms|  545.6ns
  16 lookup_token      |  102740|  56.6ms|   1.96 |   19.6ms|   62.7ms|  551.1ns
  15 input_line        |  106926|  55.5ms|   1.92 |   19.2ms|  129.2ms|  519.2ns
  26 scan_number       |   57989|  31.1ms|   1.07 |   10.7ms|   90.0ms|  536.8ns
  24 report_result     |   52309|  28.5ms|   0.99 |    9.9ms|   83.9ms|  545.7ns
  33 syntax_check      |   39934|  21.9ms|   0.76 |    7.6ms|   47.5ms|  548.5ns
  28 semantic_check    |   16318|   9.4ms|   0.33 |    3.3ms|    9.8ms|  577.0ns
__12_get_next_token____|___11475|___6.4ms|___0.22 |____2.2ms|___18.4ms|__561.8ns
 Totals Absolute       | 5.22E06|   2.8 s|  100%  |
```

- Time_%. This is an expression of the time recorded for this event as a percent of the total execution time that was recorded for all events.
- Mean(1s). This indicates the average time this event would be active during any given second.
- StDv(1s). This indicates the standard deviation of the mean. A standard deviation value that is greater than the mean indicates that there are large fluctuations about the mean.
- Time/cyc. This indicates the average time required to complete one bus cycle associated with this event.

Step 7. Make a performance measurement of function durations

- If using the Graphical User Interface:

  1 Place the cursor in the menu bar and choose **Profile→Profile...**. A dialog box opens.

  2 In the dialog box, click on **Function Duration Including All Calls**, and click **OK**.

  3 Again, sort the functions with either the **Sort->Time** action key or by choosing **Display→Sort_Events→Time** in the pull-down menus.

- If using the Softkey User Interface, enter the following commands:
  profile function_duration include_calls
  display sort_events time

  The Software Performance Analyzer can make two duration measurements: function_duration, and interval_duration. In a function_duration measurement, only events that represent functions in the source file can be included. The table display now shows data about the functions defined in the source file.

```
Table: Function Duration include calls        Run Time: 1:05      Stability: 99%
_Name_(sort:_time)_____|_Calls_|__Time__|_Time_%|__Max__|__Min__|_Mean__|Std_Dev
  22 parse_command     |    145|  59.8 s|  91.11|412.3ms|412.3ms|412.3ms|  0.0us
   1 apply_controller  |    436|  50.2 s|  76.49|115.1ms|115.1ms|115.1ms|  0.0us
   2 apply_productions |   3933|  42.6 s|  65.00| 11.7ms| 10.0ms| 10.8ms|525.3us
  12 get_next_token    |    291|   9.7 s|  14.79| 33.3ms| 33.3ms| 33.3ms|  0.0us
  31 stack_library     |  13109|   9.6 s|  14.59|730.3us|730.2us|730.2us|  0.0us
  16 lookup_token      |   1748|   8.3 s|  12.61|  4.7ms|  4.7ms|  4.7ms|  0.0us
  28 semantic_check    |   2621|   6.0 s|   9.09|  2.3ms|  2.3ms|  2.3ms|  0.0us
  27 scan_string       |   5245|   4.6 s|   7.04|880.3us|880.2us|880.2us|  0.0us
  25 request_command   |    146|   3.2 s|   4.91| 22.1ms| 22.1ms| 22.1ms|  0.0us
  14 initialize        |    146|   3.1 s|   4.80| 21.6ms| 21.6ms| 21.6ms|  0.0us
  19 math_library      |  35746|   3.0 s|   4.64|188.0us| 19.0us| 85.2us| 44.6us
  20 move_byte         |   3936|   2.9 s|   4.38|730.3us|730.2us|730.2us|  0.0us
  23 report_errors     |   3929|   2.9 s|   4.37|730.3us|730.2us|730.2us|  0.0us
  24 report_result     |    146|   2.3 s|   3.55| 19.1ms|  8.8ms| 16.0ms|  4.2ms
___5_clear_buffer_____|____438|___1.9_s|___2.90|__4.3ms|__4.3ms|__4.3ms|__0.0us
 Totals Absolute      |  76497|  65.6 s|  100% |
```

Here's what happened when the profile measurement was made:

1 The Software Performance Analyzer checked its events list to see if any events had been defined for a measurement. Events were defined so the Software Performance Analyzer did the next step.

2 The Software Performance Analyzer accepted the first 84 functions that were selected in the events list.

3 The Software Performance Analyzer performed a measurement of function durations on the events it accepted.

Duration measurements continuously record information for all the selected events in the measurement (no sampling is done in duration measurements). Duration measurements can record information for up to 84 events. When the Software Performance Analyzer makes a duration measurement, all of the selected events are active throughout the measurement.

Step 8. Look at a histogram of calls

- If using the Graphical User Interface, place the cursor in the menu bar and choose the following commands:

  Display→Histogram

  Either click on the **Sort->Calls** action key, or choose the following:

  Display→Performance Data→Calls

  Display→Sort_Events→Calls

- If using the Softkey User Interface, enter the following commands:

  display histogram data calls

  display histogram sort_events calls

```
Histogram: Function Duration include calls    Run Time: 1:05      Stability: 99%
_Name_(sort:_calls)____|__Calls_|__%___0%_____20%_____40%_____60%_____80%___100%
  19 math_library      |   35746| 46.73|******************
  31 stack_library     |   13109| 17.14|*******
  27 scan_string       |    5245|  6.86|***
  20 move_byte         |    3936|  5.15|**
   2 apply_productions |    3933|  5.14|**
  23 report_errors     |    3929|  5.14|**
  28 semantic_check    |    2621|  3.43|*
  16 lookup_token      |    1748|  2.29|*
  26 scan_number       |    1746|  2.28|*
  33 syntax_check      |    1308|  1.71|*
   5 clear_buffer      |     438|  0.57|
  15 input_line        |     438|  0.57|
   1 apply_controller  |     436|  0.57|
  11 format_result     |     349|  0.46|
__21_outputline_______|_____349|__0.46|_____________________________________
 Totals                |   76497| 100% 0%     20%     40%     60%     80%   100%
```

The histogram of calls is sorted so the events that were called most are shown first. The display shows:

- How many calls were recorded for each event.
- The percentage of calls that were made to the associated event as a percent of all recorded calls.
- A histogram bar expressing the percent of all calls that were calls to the associated event.

Step 9. See statistical information for each event in the function_duration measurement

- If using the Graphical User Interface, either click on the **Table** action key, or use the pull-down menus to choose **Display→Table**.

- If using the Softkey User Interface, enter the command: **display table**

  The table display is sorted to place events with the most calls first. It shows:

  - How many calls were made to the associated function.
  - How much execution time was spent in the associated function.
  - The percent of all execution time that was spent executing this function.
  - What was the longest single execution time measured for this function.
  - What was the shortest single execution time measured for this function.
  - What was the average time of all executions of this function.
  - What was the standard deviation of the execution times of this function.

```
Table: Function Duration include calls        Run Time: 1:05      Stability: 99%
_Name_(sort:_calls)____|_Calls_|__Time__|_Time_%|__Max__|__Min__|_Mean__|Std_Dev
  19 math_library      |  35746|    3.0 s|   4.64|188.0us| 19.0us| 85.2us| 44.6us
  31 stack_library     |  13109|    9.6 s|  14.59|730.3us|730.2us|730.2us|  0.0us
  27 scan_string       |   5245|    4.6 s|   7.04|880.3us|880.2us|880.2us|  0.0us
  20 move_byte         |   3936|    2.9 s|   4.38|730.3us|730.2us|730.2us|  0.0us
   2 apply_productions |   3933|   42.6 s|  65.00| 11.7ms| 10.0ms| 10.8ms|525.3us
  23 report_errors     |   3929|    2.9 s|   4.37|730.3us|730.2us|730.2us|  0.0us
  28 semantic_check    |   2621|    6.0 s|   9.09|  2.3ms|  2.3ms|  2.3ms|  0.0us
  16 lookup_token      |   1748|    8.3 s|  12.61|  4.7ms|  4.7ms|  4.7ms|  0.0us
  26 scan_number       |   1746|    1.3 s|   1.94|730.3us|730.2us|730.2us|  0.0us
  33 syntax_check      |   1308|    1.6 s|   2.41|  1.2ms|  1.2ms|  1.2ms|  0.0us
   5 clear_buffer      |    438|    1.9 s|   2.90|  4.3ms|  4.3ms|  4.3ms|  0.0us
  15 input_line        |    438|    1.2 s|   1.81|  2.7ms|  2.7ms|  2.7ms|  0.0us
   1 apply_controller  |    436|   50.2 s|  76.49|115.1ms|115.1ms|115.1ms|  0.0us
  11 format_result     |    349|    1.7 s|   2.66|  5.0ms|  5.0ms|  5.0ms|  0.0us
__21_outputline_______|____349|___1.7_s|___2.61|__4.9ms|__4.9ms|__4.9ms|__0.0us
 Totals Absolute       |  76497|   65.6 s|  100% |
```

Step 10. Expand an event to see greater details of its execution

- If using the Graphical User Interface, select one of the two actions below:
  - Place the cursor on the math_library event and click the *select* mouse button.
  - Place the cursor on the math_library event and press and hold the *select* mouse button until a popup menu appearrs. Click on **Expand Event Toggle** in the popup menu.
- If using the Softkey User Interface, position the cursor (">") beside the event named, "math_library" in the Name column by using the arrow keys on your keyboard (if not already there). Now press the **EXPAND** softkey.

  An expanded window is opened under the math_library event. It shows six time ranges. The columns of information beside the time ranges are set to zero when the time ranges are first opened. Now the duration of each execution of math_library will update two locations as the measurement progresses:
  - The duration will be added to the total time shown in the line beside the event name, as before.

```
Table: Function Duration include calls        Run Time: 0:49      Stability: 99%
_Name_(sort:_calls)____|_Calls_|__Time__|_Time_%|__Max__|__Min__|_Mean__|Std_Dev
  19 math_library      |  27244|   2.3 s|   4.64|188.0us| 19.0us| 85.2us| 44.6us
      1.00us - 10.0us  |      0|   0.0us|   0.00|  0.0us|  0.0us|  0.0us|  0.0us
      10.0+us- 100us   |  17919|   1.1 s|   2.11| 97.0us| 19.0us| 58.8us| 27.0us
      100+us - 1.00ms  |   9325|   1.3 s|   2.54|188.0us|110.0us|136.0us| 22.5us
      1.00+ms- 10.0ms  |      0|   0.0us|   0.00|  0.0us|  0.0us|  0.0us|  0.0us
      10.0+ms- 100ms   |      0|   0.0us|   0.00|  0.0us|  0.0us|  0.0us|  0.0us
      100+ms - 1.00s   |      0|   0.0us|   0.00|  0.0us|  0.0us|  0.0us|  0.0us
      non_range        |      0|   0.0us|   0.00|  0.0us|  0.0us|  0.0us|  0.0us
 ______________________|_______|________|_______|_______|_______|_______|_______
  31 stack_library     |   9992|   7.3 s|  14.59|730.3us|730.2us|730.2us|  0.0us
  27 scan_string       |   3996|   3.5 s|   7.04|880.3us|880.2us|880.2us|  0.0us
   2 apply_productions |   2997|  32.5 s|  65.00| 11.7ms| 10.0ms| 10.8ms|525.4us
  20 move_byte         |   2997|   2.2 s|   4.38|730.3us|730.2us|730.2us|  0.0us
  23 report_errors     |   2997|   2.2 s|   4.38|730.3us|730.2us|730.2us|  0.0us
__28_semantic_check____|__1998|___4.5_s|___9.09|__2.3ms|__2.3ms|__2.3ms|__0.0us
 Totals Absolute       |  58299|  50.0 s|  100% |
```

- The duration will also be added in the appropriate time-range column. For example, if one of the time ranges is 100+us - 1.00 ms, and an execution of math_library is completed in 254 usec, then that duration will be added in the table beside the 100+us - 1.00 ms time range.

The Software Performance Analyzer can "EXPAND" up to ten events at the same time during a measurement, and each expanded event can have up to ten time ranges shown below it. Additionally, a non-range entry will be included along with the time ranges; it will show executions that did not fall within any of the time ranges.

The EXPAND feature provides a look at the time distribution of the selected event. The time ranges let you see how many of the executions of a particular event were completed within each of the time ranges. This information might show an event that normally completes its execution in one time range, but occasionally takes ten times as long to complete its execution. This could help you identify an intermittent problem in a function.

The **EXPAND** feature is simply a switch. If you click the *select* mouse button again on math_library, or if you press **EXPAND** again, the set of time ranges under the expanded function will disappear (and their values will be reset to zero).

- Unexpand the display by clicking the mouse on math_library, or by pressing the **EXPAND** softkey .

Step 11. Measure performance of selected intervals

- If using the Graphical User Interface, enter the following commands:

  1 Click on the action key labeled, **Stop Profile**.

  2 In the menu bar, choose: **Display→Events**.

  3 In the menu bar, choose: **Events→Define Single Event ...**. This opens a dialog box. Enter the following in the dialog box:

  - Select Interval.
  - Type count in the Start Address entry field.
  - Type count in the End Address entry field.
  - Click Apply.

  4 Enter the following in the dialog box:

  - Type tasknumber in the Start Address entry field.
  - Type tasknumber in the End Address entry field.
  - Click OK.

  Note that steps 3 and 4 defined two interval events. When you clicked Apply in step 3, the event was defined and the dialog box was left open so that you could define another event. When you clicked OK in step 4, the event was defined and the dialog box was closed.

  5 In the menu bar, enter **Profile→Profile...**. A dialog box will open.

  In the dialog box, click on **Interval Duration**. Beside Status Qualification, select **data_write**. Now click **OK**.

- If using the Softkey User Interface, enter the following commands:

  stop_profile

  display events

  define single_event interval count thru count

  define single_event interval tasknumber thru tasknumber

  profile interval_duration status data_write

  In an interval_duration measurement, only events that represent intervals can be included. The interval events must be uniquely defined before an interval_duration measurement can be made. Interval events can have start addresses that are lower or higher than their end addresses, or be the same as their end addresses (as in this example). The table display below shows how often the count variable is accessed for a write transaction and how often the tasknumber variable is accessed for a write transaction.

```
Table: Interval Duration                        Run Time: 0:47      Stability: 93%
_Name_(sort:_calls)____|_Calls_|__Time__|_Time_%|__Max__|__Min__|_Mean__|Std_Dev
  35 count__count      |    167|  46.8 s|  97.61|453.5ms|  4.0us|280.5ms|219.0ms
  36 tasknumber__tasknu|    105|  46.8 s|  97.62|453.5ms|  5.8us|446.2ms| 44.2ms
_Undefined_Addresses___|______?|_______?|______?|_______|_______|_______|_______
 Totals Absolute       |    272|  48.0 s|  100% |
```

Step 12. Get help when measuring software performance

- If using the Graphical User Interface, two groups of help displays are available: **Help→General Topic ...**, and **Help→Command Line ...**. Place the cursor in the keywords line and select **Help→Command Line ...**. A dialog box opens.

- Use the scroll bar to bring **time_ranges** into view. Click on **time_ranges** and click **OK**, or double-click on **time_ranges**.

  or

- If using the Softkey User Interface, on the command line, type: **help**

- Press the **---ETC--** softkey to see the names of the help files available.

- Press the softkey labeled: **time_rng**, or type the command: **time_ranges**.

  The softkeys name subjects for which the Software Performance Analyzer can give helpful information. Help is offered for each of the first-level softkeys, and additionally, for tasks you may want to do that don't appear on the first-level softkeys, such as setting up your own time ranges, sorting events, or understanding absolute and relative values on the display.

  Select other help files from the Graphical User Interface Help selections, or from the softkeys of the Softkey User Interface to see the kinds of online help available in the Software Performance Analyzer.

  Refer to the General Topics Help information available in the interface screen (**Help→General Topic ...**) titled "Duration and Activity Distinctions" for a discussion of the basic measurements performed by the Software Performance Analyzer.

```
---Syntax---

display  time_ranges   <TIME>, (TIME>, ...
                       <TIME> thru <TIME>, <TIME thru <TIME>, ...

                       start_at <TIME>

                             end_at <TIME> [number_divisions #DIVS]
                                           [linear/logarithmic]
                             plus_increment <INC> [number_divisions #DIVS]
                             multiply_increment <INC> [number_divisions #DIVS]


---Function---

time_ranges lets you specify the time ranges to be displayed below an
expanded event in a histogram or list.  You can see the present set of
time ranges by displaying the measurement specification.

You can specify each individual time range desired, or you can specify
one overall time period and the Software Performance Analyzer will
divide it into a set of equal time ranges.  Up to 10 time ranges
---More---28%
STATUS:   M68000--Running user program    Measurement in process________........
help  time_range


profile   define   setup   display          EXPAND   delete    end    ---ETC--
```

Measurement Problems

If you have problems while running the procedures in this quick-start guide, refer to Chapter 6, which has a complete list of possible problems and checks you can make to solve those problems.

Running the Debug Environment demo program

If you have the Debug Environment for your emulator, you may want to examine the debug environment demo. The demo program, "ecs.x", implements a hypothetical (E)nvironmental (C)ontrol (S)ystem for a computer room. To run this demo, enter the following commands:

1 **cd $HP64000/demo/debug_env/hp<XXXXX>**

2 **Startall <logical name>**

3 **Execution→Run→from Transfer Address** (for most emulators. See exceptions below.)

Notes:

<XXXXX> is the product number of your emulator (for example, hp64742 is the product number of a 68000 emulator and hp64748 is the product number of a 68020 emulator.).

<logical name> is the name you assigned to identify your emulation system.

Exceptions for Step 3 above:

If you are using the Software Performance Analyzer with an HP MC68340 Emulator, use **run from reset**, and then **run from transfer_address**.

If you are using the Software Performance Analyzer with an HP I80960 Emulator, use **run from reset**.

About the Debug Environment demo

The Software Performance Analyzer in this demo uses a different set of action keys. This set of action keys has been designed for use in sales demonstrations; it shows a few of the features of the Software Performance Analyzer. Some of the action keys do more than you might want to do during normal use of the analyzer. The additional tasks performed by an action key might lead to confusion about the measurements of the analyzer.

The Break Dur () action key sends a break command to the emulator when the duration of a specified function exceeds a specified time. This works as it should, but the action key also forces a measurement of function durations, excluding all calls. While the durations presented exclude all calls, the duration that is sent out as a trigger is based on a measurement of function durations, including all calls. Therefore, it is impossible to correlate the duration of an event on the display with the duration of the event that causes the trigger. A second problem with the Break Dur () action key is that it is assumed that the emulator has been set up to receive the trig2 signal to cause a break. This is not the default case of an emulator configuration; it must be set up separately.

Another point that can cause confusion when using these action keys is that they do multiple commands. The Prog Activity, Var Activity and Func Durations action keys will sort the histogram displays after they start their measurements. In addition, the Func Duration action key only makes measurements of function durations, excluding all calls.

Therefore, when examining this demo, it might be best to avoid using the action keys altogether; use only the pull-down menu commands and window popups.

You may want to repeat steps 4 through 10 of this chapter to make sure you understand the general operation of the Software Performance Analyzer.

Using markers in MC68040 and MC68030 measurements

Markers let you make measurements when the address and data caches of your emulation processor are turned on. A demonstration program has been developed to let you see the effects of using markers with the caches enabled. This demonstration program is set up for the MC68040 and MC68030 emulation processors. To see this demonstration, change your directory to **$HP64000/demo/spa/demomt**, and follow the procedure given in the README file. To find out more about marker based measurements, refer to the section titled "Markers and how to use them..." in Chapter 7.

Part 2

Making Measurements With The Software Performance Analyzer

Part 2

This part of the manual contains the following chapters:

Chapter 2. Preparing the Software Performance Analyzer to make measurements

Chapter 3. Controlling the profile measurement

Chapter 4. Managing the display of measurement results

Chapter 5. Miscellaneous tasks performed when using the Software Performance Analyzer

Chapter 6. Measurement problems

2

Preparing the Software Performance Analyzer to Make Measurements

The Software Performance Analyzer makes its measurements on an executable file running in emulation. This chapter shows you how to set up the Software

Performance Analyzer and prepare it to make measurements. The following topics are covered in this chapter:

- Preparing your program for Software Performance Analysis.
- Preparing the emulator to accept the Software Performance Analyzer.
- Starting the Software Performance Analyzer.
- Defining events.
- Defining events for a class of symbols in the symbols data base.
- Having events automatically defined.
- Defining events from the keyboard.
- Modifying the specification of an event.
- Rules to follow when naming an event.
- Defining events to be fetched on byte, word, or long-word boundaries.
- Selecting events to be included in the next measurement.
- Unselecting events to keep them out of the next measurement.
- Deleting events.
- Preparing to make duration measurements within limited regions of execution.
- Measuring activity following an enable condition.
- Holding off measurement start until after a trigger signal is received from an associated emulator or analyzer.
- Setting up a measurement in which the emulator will break to its monitor program when an event runs too long.
- Setting up a measurement in which the emulation-bus analyzer will take a trace of state flow when an event runs too long.
- Setting up a trace when an expanded event hits a certain time range.
- Setting up a measurement that ends after a specified period of time, or after obtaining a desired stability of measured data.

- Qualifying a measurement on processor status, such as user or supervisor.
- Positioning the cursor beside a new event on screen.

To prepare your program for Software Performance Analysis

There are a variety of compiler issues that must be considered when preparing a program to make it work with the Software Performance Analyzer. If you are using an HP AxLS compiler, your program can be prepared correctly and you will have to be concerned about only the last three issues listed below. If you are not using an HP AxLS compiler, depending on which compiler you are using, the items of concern will be:

- Ensuring source file symbols are available to the emulation system.
- Ensuring proper function entry and exit points in your program.
- Overcoming complications caused by multibyte return instructions.
- Avoiding function address overlaps.
- Instrumenting markers in your program to allow measurements when the instruction and/or data caches are turned on.
- Instrumenting markers in your program to allow function-duration measurements in Motorola 68040 and Intel 80960 microprocessors.

For detailed discussions about each of the above issues, refer to the paragraph titled "Preparing your program for Software Performance Analysis" in Chapter 7 of this manual.

To prepare your emulator to accept the Software Performance Analyzer

1 Start your emulator.

2 Load your emulator configuration file.

3 Load the absolute file.

4 Start your absolute file running in emulation.

If you are using the Softkey Interface of your emulator, you may want to end your emulation session with **end** <RETURN>. This will leave your program running and remove the emulation interface from your screen. The Software Performance Analyzer will make its measurements on the code that the emulator is running. The Software Performance Analyzer will gain access to the symbols data base within the emulator.

To start the Software Performance Analyzer

- If you are using the graphical user interface, click on:

  File→Emul700→Graphic Windows→Performance Analyzer...

- If you are using the softkey interface, enter the command:

  emul700 -u skperf <logical name>

  Replace **<logical name>** with the name of the emulation system that is running the file to be analyzed. The above commands call the Software Performance Analyzer to operate on your emulator through the appropriate interface.

To define events

There are four kinds of events that can be defined for the Software Performance Analyzer:

- functions
- static variables
- ranges
- intervals

The Software Performance Analyzer can automatically define events to represent functions and static variables in your symbols data base. If you want events that represent ranges or intervals, you will need to define them, yourself. The following paragraphs show you how to define events.

Note that the static variables are not properly defined if using the "-h" option to your Hewlett-Packard compiler. In general, static variables are available if you are using IEEE or OMF format symbol files.

To define events for a class of symbols in the symbols data base

- Using the command line, enter the **set symbols** command.

The symbols setting you select will determine which symbols from the symbols data base will have events defined for them in the Software Performance Analyzer. If you set symbols high (the default symbols setting), the Software Performance Analyzer will define events to represent the high level source-file symbols. If you set symbols low, events will be defined to represent the assembly level symbols in the symbols data base. If you set symbols all, then events will be defined to represent the source-file symbols and the assembly level symbols.

Examples

To have symbols defined for only the high level source-file symbols:

set symbols high

To have symbols defined for only the assembly level symbols in the symbols data base:

set symbols low

To have symbols defined for the high level source-file symbols and the assembly level symbols:

set symbols all

To have events automatically defined for you

- Choose **Profile→Profile Again.**

- Choose **Profile→Profile ...**, and select the desired profile type and status qualification (if applicable) from the dialog box. Then click OK or Apply.

- Using the command line, enter the command: **profile**

  When you enter any form of the "profile" command, the Software Performance Analyzer first checks to see if events have been defined in the events list. If no events have been defined, the Software Performance Analyzer delays the making of the measurement while it defines a list of events.

  The Software Performance Analyzer accesses the symbols data base for the absolute file running in emulation, and it defines events to represent each of the symbols it finds. The specifications of the automatic-definition process are as follows:

  - Up to 1000 events can be defined in the events list (for the first 1000 symbols found in the emulator symbol data base).

- Each event represents one function or one static variable found in the symbols data base.
- The Software Performance Analyzer assigns a name and address to each event:
  - The event name is the same as the symbol name it represents, except that special characters in the symbol name will be replaced by underscores "_" in the event name.
  - The address is the same as the address or address range occupied by the code associated with the symbol. The addresses of functions will be aligned according to the current alignment setting (long, word, or byte).
  - Symbol names beginning with "_" are ignored during automatic definition (these are the compiler-generated and assembler-generated symbols).

When an event is defined to represent a static variable, it may represent a single address, or it may represent the address of an array or structure in your absolute file.

If you are using markers to represent function-start and function-end locations, the Software Performance Analyzer will define events that show both the address range of the function events and the addresses of the event markers. Refer to Chapter 7 for a complete description of markers.

To define a set of desired events

- Choose **Events→Define Events ...**, and from the dialog box, select the desired type of events, pattern to be matched or not matched by symbol names, and symbolic filter desired. Then click OK or Apply.

- Using the command line, enter the **define multiple_events** command.

  You can define groups of events by specifying the characteristics of the events you want defined. Multiple events can be defined for functions and/or variables that

reside in the absolute file under test and have symbols in the symbols data base in the emulator. The Software Performance Analyzer accesses the symbols data base and defines events for symbols it finds there.

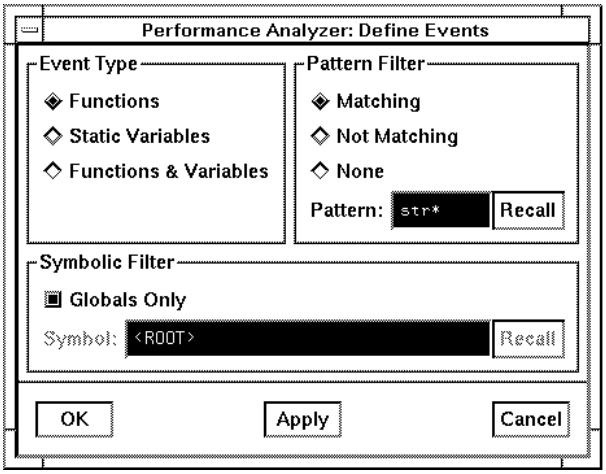
There are cases where the process of automatic definition takes a long time. The Software Performance Analyzer can check your specification against approximately 50 symbols in the symbols data base every second. If you use a **define multiple_events** command to define events for a file that has 50,000 symbols, execution of the command could take 15 to 20 minutes. There are ways you can use qualifiers, file names, or module names to limit the regions that are accessed for creation of the events. If your absolute file had seven principle files, but you know that the events you want to measure all exist in one or two of those principle files, you can include the names of those files in the Symbolic Filter of the dialog box, or in your command on the command line.

If you want to define events to represent ranges or intervals, refer to the next discussion. Ranges and intervals must be defined as single events.

Examples

To define events for all global functions whose names begin with the characters str:

Choose **Events→Define Events ...**, and set up the dialog box as shown.



or on the command line, enter:

define multiple_events functions globals_only matching "str*"

To define events for symbols in a particular file or module of the program under test:

Choose **Events→Define Events ...**, and set up the Symbolic Filter in the dialog box to contain the file or module name desired. Click on Globals Only if necessary to activate the Symbol entry area. Then click OK or Apply. You can enter multiple module names by separating them with a comma.

define multiple_events funcs_and_vars_static myfile.c:, hisfile.c:, herfile.c:
define 1 **thru** 12 **multiple_events functions** file1.c:
define multiple_events functions mymodule(module)

To define events for all functions except those whose names begin with myvar:

Choose **Events→Define Events ...**, and set up the Pattern Filter in the dialog box to activate "Not Matching", and type myvar* in the Pattern entry field. Then click OK or Apply.

define multiple_events variables_static not_matching "myvar*"

To define events for all global functions in your executable:

Choose **Events→Define Events ...**, and select Event Type Functions, and set the Symbolic Filter to Globals Only in the dialog box. The Symbol line will be grayscaled and unresponsive. Then click OK or Apply.

define multiple_events functions globals_only

To define a single event

- Choose **Events→Define Single Event ...**, and in the dialog box, select the desired event type, and enter the event number and/or name, and event address range. Then click OK or Apply.

- In the entry buffer, type the name of the symbol for which you want to create an event. Then choose **Events→Define Single Event ()**.

To define a single event

- Place the global or local symbols display on screen. Place the mouse pointer beside the symbol for which you want to define an event. Press the *select* mouse button to obtain the popup menu. Then click on **Define Single Event** in the popup menu.

- Using the command line, enter the **define single_event** command.

  There are four types of events: functions (inclucing recursive functions), static variables, intervals, and ranges. The Software Performance Analyzer can define an event to represent a single function in your source file. The Software Performance Analyzer can also define an event to represent a static variable. In either case, the Software Performance Analyzer will access the emulation data base to obtain the symbol name and corresponding addresses.

  You can define a single event to represent a range of functions (function1 thru function9), a range of variables (var1 thru var27), or a simple range of address space (1000H thru 2000H). The Software Performance Analyzer will find the lowest address and the highest address in your definition, and define your event to include all addresses within the range.

  If you define a single event to represent a range of addresses, all addresses within the range will be included. You might define an event to represent all of the addresses from the start of FunctionX to the end of FunctionY. This allows you to make a measurement that records all activity within two or more functions, and provides just one number to represent all of the activity. Check to see what will be included when making such a definition. A compiler that is optimizing your code might place additional functions within the address range represented by your event.

The Software Performance Analyzer does not allow overlapping events (two events representing the same address). Before it accepts your definition for a function, variable, or range event, the Software Performance Analyzer will check if any event already represents an address in your definition. If it finds that your new definition will overlap one or more events, the Software Performance Analyzer will show you one of the overlapping events and ask if you want to delete all of the overlapping events.

- If you press 'yes', all of the overlapping events will be deleted and your new event will be defined.
- If you press 'no', none of the overlapping events will be deleted, and your new event will not be defined.

To see the overlapping events that will be deleted by your new definition, display the events list and sort it by address.

An interval is any address space defined by a starting address and an ending address. The starting address may have a lower value or higher value than the ending address. The starting and ending addresses may have the same value. An interval with the same starting and ending addresses is useful to measure how often a static variable is accessed for a write transaction.

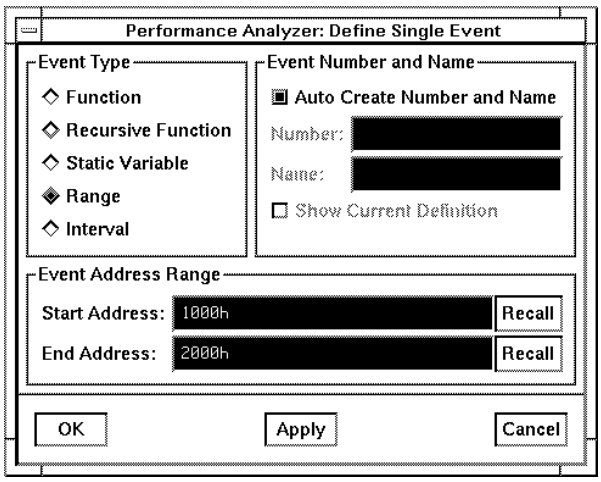
Interval event specifications are separate from function, variable, and range events. An interval event cannot have the same start or end address as another interval event. Otherwise, there are no restrictions on addresses within the ranges of the intervals. If you try to define an interval event that has a start or end address that is the same as the start or end address of another interval event, an error message will ask if you wish to delete all of the overlapping events.

- If you press 'yes', all of the overlapping events will be deleted and your new event will be defined.
- If you press 'no', none of the overlapping events will be deleted, and your new event will not be defined.

Examples

To define an event to represent a range of address space:

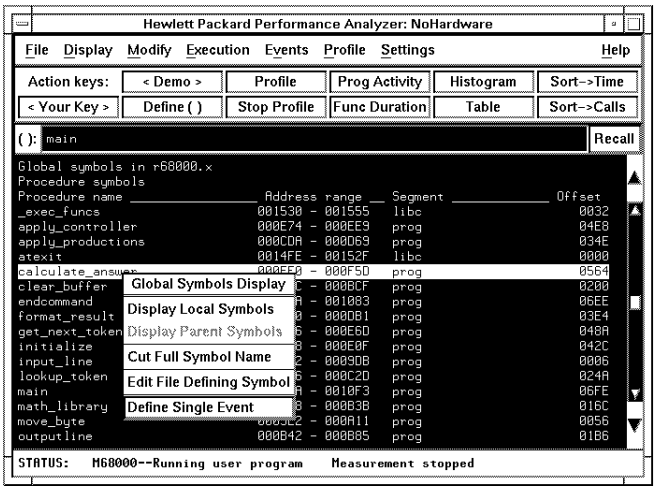
Choose **Events→Define Single Event ...**, and set up the dialog box. The example dialog box is set up to define a range event to represent addresses from 1000h through 2000h. The Software Performance Analyzer will name the event using its naming conventions.



or on the command line, enter:

define single_event range 1000h **thru** 2000h
define single_event range function1 **thru** function4 end

To define an event to represent a function or static variable, place the mouse pointer beside a symbol name in the symbols display. Press the *select* mouse button to obtain the popup menu. Click on **Define Single Event**.



To define an event to represent an interval between the start address of Symbol2 and the start address of Symbol9:

Choose **Events→Define Single Event ...**, and set up the dialog box to select Event Type Interval, and type Symbol2 beside Start Address, and Symbol9 beside End Address. Then click OK or Apply.

define single_event interval Symbol2 **thru** Symbol9

To define an event to represent an interval between the start address of Symbol2 and the end address of Symbol9:

Choose **Events→Define Single Event ...**, and set up the dialog box to select Event Type Interval, and type Symbol2 beside Start Address, and Symbol9 end beside End Address. Then click OK or Apply.

define single_event interval Symbol2 **thru** Symbol9 end

To define and name an event to represent an array variable:

Choose **Events→Define Single Event ...**, and set up the dialog box to select Event Type Static Variable, click Auto Create Number and Name to activate the associated fields and type data1 beside Name, type 01234h beside Start Address, and +010h beside End Address. Then click OK or Apply.

define single_event named data1 **variable_static** 01234h **thru** +010h

To define an event to represent a recursive function

Choose **Events→Define Single Event ...**, and set up the dialog box to select Event Type Recursive Function. Enter the name of the function (e.g. function2) in the Start Address field, and leave the End Address field blank. Then click OK or Apply.

define single_event function Function2 recursive_function

When you define an event to represent a function and specify that the function is recursive, prefetch correction is turned off for that function. The prefetch correction algorithm may concatinate repetitive calls of a recursive function into a single call.

To modify the specification of a single event

- Create the modified event specification by clicking Auto Create Number and Name to activate the Number and Name text entry areas. Enter the event number or name in the appropriate box and click on Show Current Definition. Now modify the definition, as required, and click OK or Apply.

- Using the command line, enter **define** <EVENT> **modify_command**

  The above command calls the present definition for <EVENT> to the command line where you can modify it and reenter it. The Software Peformance Analyzer will accept the changes you make and redefine the selected event, as long as the address range in your modified definition does not overlap an address in an existing event.

Examples

To modify the definition of a single event:

define 4 **modify_command**
define clear_buffer **modify_command**

To name an event

- Choose **Events→Define Single Event ...**, and in the dialog box, enter the desired Event Number and/or Name and associated Event Type and Event Address Range. Then click OK or Apply. If this will replace an existing event, the Software Performance Analyzer will ask if you wish to delete the overlapped events. Click Yes.

- Using the command line, enter the **define single_event named <NAME> ...** command.

- Use the **define** <EVENT> **modify_command** command and modify the definition of the event on the command line.

  The name of an event can be 40 or more characters long. If you define an event without giving it a name, the Software Performance Analyzer will name the event for you. Event names will be identical to the symbol names of the functions or variables they represent. If the event symbol contains non-alphanumeric characters, like "?A5", the event name will have to be placed in quotes to use the name in a command.

  If an event represents a numeric address range, the Software Performance Analyzer will create a name that includes the address range of the event (example: _1000h__1010h). Leading underscores will be added to all numeric-named events. If you want to assign a new name to one of the numeric-named events, or if you want to delete a numeric-named event from your events list, be sure to include the leading underscore in your command, or use the event number.

To define events that are fetched on byte, word, or long-word boundaries

- Use the **set byte_alignment** command on the command line.

The addresses used in event definitions for symbols in program memory will be aligned according to the normal addressing scheme of your target microprocessor (by default). If you need a different alignment for addressing some of the events in program memory, you must specify the alignment before you enter the definitions of your events. You can define events whose addresses will be fetched on byte, word, or long-word boundaries.

You can define some function events to be fetched on long-word boundaries (every 4th byte), and others in the same measurement to be fetched on word boundaries (fetch even bytes), or byte boundaries (fetch every byte).

Events defined to be functions will be aligned according to your **set byte_alignment** specification. Events defined to be variables and ranges will not be aligned. Events defined to be intervals will be aligned if the symbols in your definition are function symbols. Otherwise, no alignment will occur.

Examples:

To define a set of events whose addresses are aligned on long-word boundaries:

```
set byte_alignment long
define multiple_events functions file1.c:
define single_event function myfunction
define interval myfunction
```

To define a set of events whose addresses are aligned on word boundaries:

```
set byte_alignment word
define multiple_events functions file2.c:
define single_event interval myfunction1 thru myfunction2
```

To select events from the events list to be included in the next measurement

- Choose **Events→Select Events ...**, and in the dialog box, select the desired Event Operation, event type to be selected, pattern to be matched or not matched by the event names, and range of events within the events list. Then click OK or Apply.

- Place the mouse cursor beside the event to be selected. Press the *select* mouse button, and click on **Select Event Toggle** or **Select Events Thru End** in the popup menu.

- Using the command line:

  1 Enter the command: **display events**

  2 Place the cursor beside the event number to be included, and press: **SELECT**

- Use the **select_events** .... command

  You can select individual events, and you can select classes of events, such as functions or intervals. An event is either selected or not selected. If a symbol appears beside the event name in the events list (*, ?, or r), it is selected. If no symbol appears, the event is not selected. Interpret these symbols as follows:

  If "*" is shown beside the name of an event in the events list, the event is selected for the next profile measurement. If a performance measurement has already been made, the "*" beside an event name shows that the event was included in the last profile measurement.

  If "?" is shown beside the name of an event, the event was selected but not qualified to be in the most recent profile measurement (for example, static variables are not qualified to be in function_duration measurements). If the next profile measurement you perform is one for which this event is qualified, it will be included, and it will show "*" beside its name in the events list.

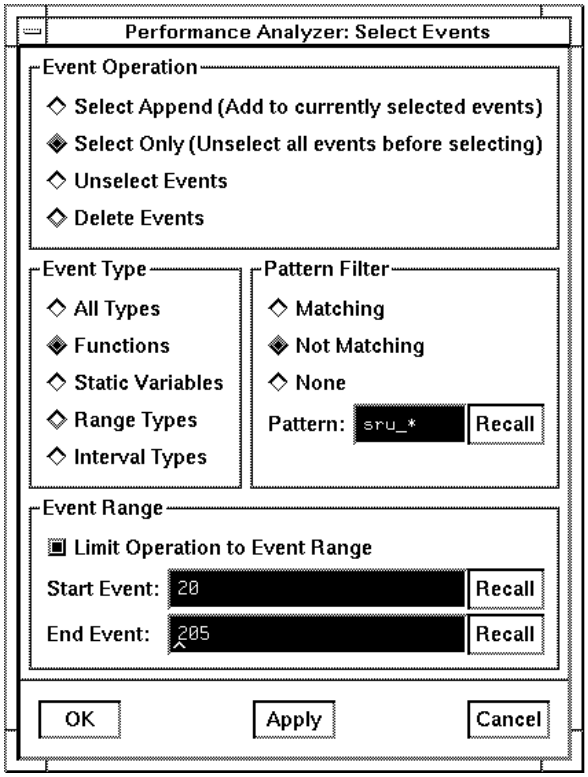
  If "r" is shown beside the name of an event, the event was selected and qualified to be in the most recent profile measurement, but it was not included because of resource limitations. The Software Performance Analyzer can include up to 254

events in a program-activity measurement (the first 254 functions or ranges selected in the events list). Event 255 may also be selected and qualified for the measurement, but the Software Performance Analyzer will not have enough resources to include it in the measurement. Therefore, the "r" will appear beside its name in the events list. If you unselect or delete enough events preceding this event in the list, there will be resources available to include this event in the next profile measurement.

Examples

To select events to be included in the next measurement:

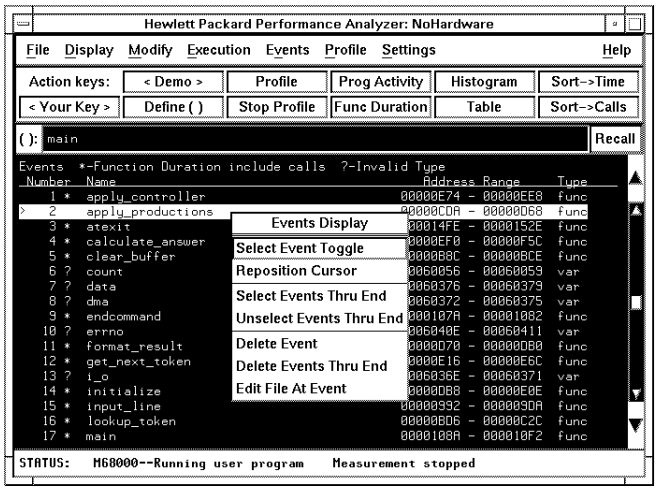
Choose **Events→Select Events ...**, and set up the dialog box. The example dialog box will select function events whose names do not begin with "sru_", and whose locations in the events list are between numbers 20 and 205:



To select events from the events list to be included in the next measurement

Place the mouse cursor beside the event to be selected, and click the *select* mouse button. This will toggle the event from unselected to selected.

Place the mouse cursor beside the event to be selected, and press the *select* mouse button. Then click on **Select Event Toggle** or **Select Events Thru End**.



or on the command line enter:

select_events functions
select_events append functions matching "mem*"
select events variables_static
select events 20 **thru** 205 **notmatching** "sru_*"

To unselect events, but keep them available for future measurements

- Choose **Events→Select Events ...**, and in the dialog box, click on Unselect Events in the Event Operation area. Then choose the desired event type to be unselected, pattern to be matched or not matched by the event names, and range of events within the events list. Then click OK or Apply.

- Place the mouse cursor beside the event to be unselected. Press the *select* mouse button and click on **Select Event Toggle**, **Unselect Event**, or **Unselect Events Thru End** in the popup menu.

- Using the command line:

  1 Enter the command: **display events**

  2 Place the cursor beside the event number to be unselected, and press **SELECT**

- Enter an appropriate command beginning: **unselect_events ....**

  You can unselect a single events or groups of events. Characteristics you can specify when unselecting groups of events are:

  - functions
  - variables
  - ranges
  - intervals
  - ranges of event numbers
  - events whose names begin with (match), or do not begin with, (not match) a pattern of characters

To delete events

- Choose **Events→Delete All Events.**

- Choose **Events→Select Events ...**, and in the dialog box, click on the Delete Events Event Operation. Then specify the Event Type, Pattern Filter, and range in the events list where events are to be deleted. Then click OK or Apply.

- Place the mouse cursor beside the event to be deleted, or the first event to be deleted, in the events list. Press the *select* mouse button and click on **Delete Event** or **Delete Events Thru End** in the popup menu.

- Using the command line, enter the **delete_events** command.

  The delete_events methods let you delete events you do not need. You may want to delete events that show low usage during a measurement. Using the command line, simply move the cursor down the list of events on display to the first event with low usage and enter a command that deletes events through the end of the list.

  You can delete events from the cursor position through the start of the list, if desired, by using a command-line entry.

  You can delete events when a histogram or table is on screen. By default, only events shown on the present display will be deleted. For example, if a histogram of function durations is on screen, only events that represent functions will be present. Therefore, function events will be deleted, but events that represent variables, I/O, intervals, and ranges will remain in the events list.

  In any display, you can delete specific types of events (functions variables, etc.), or you can delete events whose names begin with certain characters (or whose names don’t begin with certain characters).

Examples

To delete events from the present display:

Choose **Events→Delete All Events**, or **Events→Select Events ...**, and set up the dialog box to identify the events to be deleted. Then click OK or Apply.

Obtain the popup menu and click on **Delete Events Thru End**.

or on the command line enter:

delete_events
delete_events functions
delete_events range_types matching "RNG3*"
delete_events 10 **thru** 105
delete_events interval_types 44 **thru** start
delete_events functions thru 205 **notmatching** "sru_*"

To set up a measurement of activity following an enable condition

- Using the pulldown menus:

  1 Choose **Display→Measurement Spec**.

  2 Choose **Modify→Setup→Enable/Disable ...**, and in the dialog box, enter the name or number of the event to enable, and an enable status qualification, if desired. Then click OK.

- Using the command line, enter the commands:

  display measurement_spec

  setup_measurement enable start_address <enable event name>

  profile program_activity

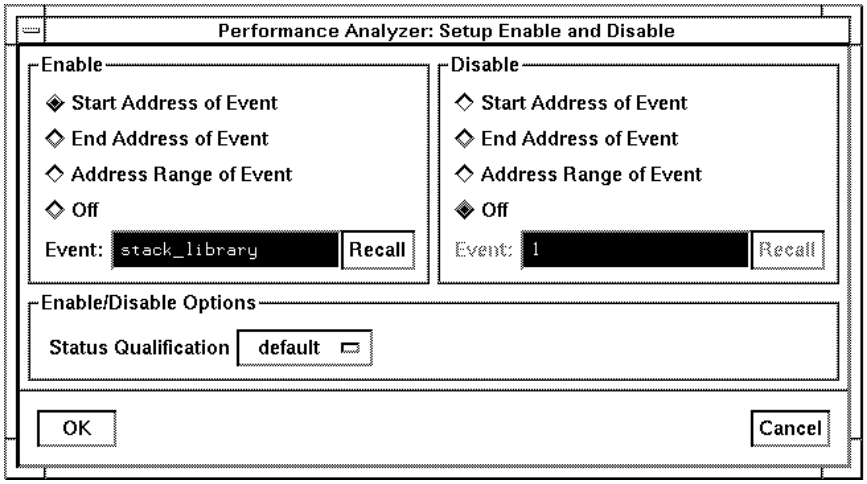
  With the above commands, you have set up an enable condition. The Software Performance Analyzer will not collect data until it finds the state that satisfies the enable condition. When this address is found, the Software Performance Analyzer

will begin making its measurement. This method can also be used to enable duration measurements.

Examples

To specify enable events for an activity measurement:

Choose **Modify→Setup→Enable/Disable ...**, and set up the dialog box as shown. The example dialog box will enable the measurement on the start address of stack_library:



or on the command line enter:

setup_measurement enable start_address initialize
setup_measurement enable end_address move_byte
setup_measurement enable any_address stack_library

To prepare for a measurement of durations within limited regions of execution

- Using the pulldown menus:

  1 Choose **Display→Measurement Spec**.

  2 Choose **Modify→Setup→Enable/Disable ...**, and in the dialog box, enter the names or numbers of the events to enable and disable the profile measurement; click on the event starting address, ending address, or any address within the event range. In the Enable/Disable Options block, you can specify an enable/disable status qualification, if desired. Then click OK.

- Using the command line, enter the following commands:

  display measurement_spec

  setup_measurement enable start_address <start event name>

  setup_measurement disable end_address <end event name>

  profile function_duration

  Enable/disable (measurement) windows can only be used in duration measurements. The above commands set up a measurement window. The Software Performance Analyzer will suspend its collection of data until it sees the address that represents the enable condition. Collection of data will proceed according to the specifications entered in the Software Performance Analyzer until appearance of the address that represents the disable condition. At that point, all data collection will be suspended. The measurement will be suspended until the address that represents the enable condition appears again.

  The **enable** and **disable** specification can be used to include or exclude regions of program execution. You can set up the enable and disable occurrences to be true on the start address or end address of any event in the events list. You can also set up the enable or disable to be recognized on any address within the range represented by an event, provided that event is not an interval event. You can further qualify your enable or disable specifications with a status assignment (such as writes).

The following limitations apply when setting up measurement enable and disable specifications:

- You cannot use the same address for both the enable and disable specifications.
- Events used to enable and/or disable a measurement cannot be part of the profile measurement.
- Activity measurements cannot recognize disable specifications; therefore, activity measurements cannot be windowed.

Examples

To qualify regions of execution for performance measurements:

Choose **Modify→Setup→Enable/Disable ...**, and in the dialog box, select Start Address of Event parse_command, and Disable End Address of Event parse_command.

setup_measurement enable start_address initialize
setup_measurement disable end_address move_byte
setup_measurement disable any_address main

To hold off measurement start until a trigger is received from an emulator or analyzer

- Using the command line, enter the command: **setup_measurement start after_receiving_trig1**

trig1 is the only external trigger that can be used to delay the start of measurements in the Software Performance Analyzer. With the above command, the Software Performance Analyzer will wait until trig1 is received before it begins its profile measurement. Using trig1, the emulator or emulation bus analyzer can hold off the performance measurement until a particular software condition has occurred. Typical uses for this measurement setup include:

- Holding off the measurement until a specified initialization routine has completed so that measurement results are not affected by executions within the initialization routine.

- Measuring performance of a set of functions, but only after your emulator or emulation bus analyzer detects completion of a sequence of software execution in the absolute file under test.

No provision is made to specify trig1 usage in the pulldown menus of the Software Performance Analyzer.

To cause the emulator to break to its monitor when an event runs too long

- Choose **Modify→Setup Trig2 ...**, and in the dialog box, select Drive After Time Exceeded and enter the Trig2 time and event name and/or number. Then click OK.

- Using the command line, enter the **setup_measurement drive trig2_after** command.

The above command sets up the Software Performance Analyzer to generate a signal on trig2 when an event you name runs continuously for too long a period of time. You can set up the emulator to break to its monitor program when it receives trig2. By using the **drive trig2_after** command, you can use the emulator to review the state of your microprocessor and examine program execution when a particular function runs longer than you think it should.

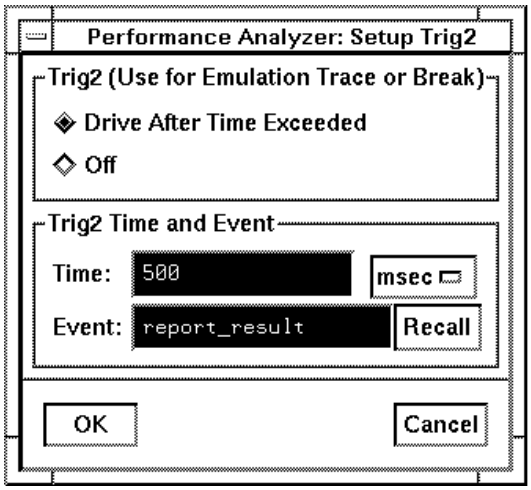
The **setup_measurement drive trig2_after** function is only available when making function-duration and interval-duration measurements.

Note that the trigger command always calculates its interval as an including_calls time (even if the present measurement is function_duration excluding_calls). In addition, the trigger command is not corrected for prefetch conditions. Prefetch correction is discussed in detail in Chapter 7.

Examples

To specify a measurement that causes an emulation break to occur when an event runs too long:

Choose **Modify→Setup Trig2 ...**, and set up the dialog box as shown. The example dialog box sets up generation of Trig2 when report_result runs for 500 msec:



or on the command line enter:

setup_measurement drive trig2_after 500 **msec** report_result
setup_measurement drive trig2_after 100 **msec** int_loop

To set up the emulator to respond to the trig2 signal:

Choose **Modify→Emulator Config ...**, and in the dialog box, click on **Interactive Measurement Specification**, and **Modify Section**. Then advance to the question: Should Emulator break receive Trig2? **yes**. Answer the remaining questions, and click on Apply to Emulator. Select a configuration file name, and click OK or Apply. Finally, in the Emulator Configuration Main Menu, click on Exit Window.

or on the command line enter:

modify configuration
modify interactive measurement specification? **yes**
should emulator break receive trig2? **yes**

To set up taking a trace in the emulation-bus analyzer when an event runs too long

- Choose **Modify→Setup Trig2 ...**, and in the dialog box, select Drive After Time Exceeded and enter the Trig2 time and event name or number. Then click OK.

- Using the command line, enter the **setup_measurement drive trig2_after** command.

The above command sets up the Software Performance Analyzer to generate a signal on trig2 when an event you name runs continuously for too long a period of time. You can set up the emulation-bus analyzer to capture a trace of state execution when it receives trig2. By using the **drive trig2_after** command with the emulation-bus analyzer, you can see the states being executed when your program runs longer than you think it should.

The **setup_measurement drive trig2_after** function is only available when making function-duration and interval-duration measurements.

Note that the trigger command always calculates its interval as an including_calls time (even if the present measurement is function_duration excluding_calls). In addition, the trigger command is not corrected for prefetch conditions. Prefetch correction is discussed in detail in Chapter 7.

Examples

To specify a measurement that starts after an event runs too long:

Choose **Modify→Setup Trig2 ...**, and in the dialog box, select Drive After Time Exceeded, enter a Time of 500 msec, and an Event name of report_result. Then click OK.

setup_measurement drive trig2_after 500 **msec** report_result
setup_measurement drive trig2_after 100 **msec** int_loop

To set up the emulation-bus analyzer to respond to the trig2 signal:

Choose **Modify→Emulator Config ...**, and in the dialog box, click on **Interactive Measurement Specification**, and Modify Section. Then advance to the question: Should Analyzer drive or receive Trig2? **receive**. Answer the remaining questions,

and click on Apply to Emulator. Select a configuration file name, and click OK or Apply. Finally, in the Emulator Configuration Main Menu, click on Exit Window.

or on the command line enter:

modify configuration
modify interactive measurement specification? **yes**
Should Analyzer drive or receive trig2? **receive**

A useful emulation trace to capture all states before the trigger would be:
trace arm_trig2 before address 0xxxxxxxh
OR
trace arm_trig2 before anystate

To set up taking a trace when an expanded event executes in an abnormal time range

1 Stop the duration measurement.

2 Set up your emulation-bus analyzer to start a trace when it receives the trig2 signal, as discussed in the paragraph titled, "To set up taking a trace in the emulation-bus analyzer when an event runs too long."

3 Place the mouse pointer on the abnormal time range.

4 Press the *select* mouse button to obtain the popup menu, and click on **Drive TRIG2 on Range Min**.

5 Start a new duration measurement.

The above procedure causes trig2 to be generated when the expanded event runs long enough to record an execution in the time range you selected with your mouse pointer. This is useful if you notice that most executions of an expanded event occur within one or two time ranges, but on rare occasions, an execution of the event takes a much longer time (is recorded in a longer time range). This feature

lets you take a trace of processor activity when the event runs that abnormally long time.

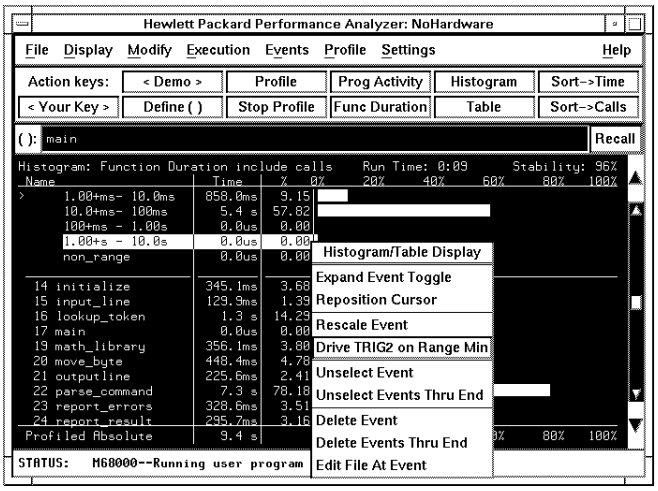
Note that the setup for this measurement can only be made when no profile measurement is running.

Examples

To trigger a trace when the expanded event runs for more than 1 second:

Set up the emulation-bus analyzer to start a trace when it receives trig2.

Place the mouse pointer on the 1.00+s - 10.0s time range, obtain the popup menu, and click on **Drive TRIG2 on Range Min**. Then start a new measurement.



To set up a measurement that ends after a time period, or after obtaining a data stability

- Use the command line to enter the **setup_measurement termination** command.

Commands like those above let you set up a measurement that will run until some desired condition is reached by the Software Performance Analyzer. You can set up your measurement to run continuously for a period of time. You can set up your measurement to run until a desired stability of measured data is achieved (refer to discussions of stability and confidence in Chapter 8 for further information). You can set up your measurement to run for either a period of time or level of stability, whichever happens first.

Stability can be specified between 51 and 99 percent. Termination time can be specified between 5 seconds and 100 hours. Your measurement will end when your specification is satisfied in the Software Performance Analyzer.

Examples

To stop a measurement automatically when a condition is reached:

setup_measurement termination run_time 10 **minutes**
setup_measurement termination stability 95 **percent**
setup_measurement termination stability 80 **percent or** 600 **seconds**

To qualify your measurement on processor status like user or supervisor

- Choose **Settings→Status Qualifications→**<desired status qualification>.

- Use the **set status_qualification** command.

  If the emulation microprocessor you are using has user/supervisor modes, or similar modes, the Software Performance Analyzer will offer this command form. You can use it to qualify your measurement on processor status. This command form lets you ensure that the data in your measurement results will only be taken when your emulation processor is executing in the desired status.

Examples

To qualify measurement data to be taken only during selected processor status:

Settings→Status Qualifications→supervisor

```
set status_qualification supervisor
set status_qualification user
set status_qualification any
```

To position the cursor beside a new event on screen

- In the Graphical User Interface, place the cursor beside the desired event and hold the *select* mouse button to obtain the popup menu. Choose **Reposition Cursor** from the popup menu.

- Use the keyboard arrow keys to move the cursor.

3

Controlling the Profile Measurement

The Software Performance Analyzer makes profile measurements using two basic measurement modes: activity and duration. The activity measurement mode records activity in regions of program or memory address space. Activity mode uses a sampling technique; it samples activity in each memory region or event,

sequentially. Only one event is active (being sampled) at a time. The activity measurement mode can accept up to 254 events in a single measurement.

The duration measurement mode operates in a real-time manner. Only events that represent functions or intervals can be active in a duration measurement. All selected events are active throughout the measurement. In the duration mode, the entry and exit points of each event are recorded. The Software Performance Analyzer matches the entry and exit points with each other and records the durations between them. Up to 84 functions or intervals can be included in a single, duration measurement.

Topics covered in this chapter include:

- Making the most simple profile measurement.
- Obtaining a profile of the activity of your program code.
- Obtaining a profile of the activity of variables and I/O ports.
- Obtaining a profile of durations of functions defined in your source files.
- Obtaining a profile of durations of intervals.
- Obtaining time-range details under a selected event.
- Creating your own set of time ranges for expanding events.
- Stopping the present profile measurement.

To make the most simple profile measurement

- Choose **Profile→Profile Again**.

- Using the command line, enter the command: **profile**

  With this command, the Software Performance Analyzer will first check to see if events have been defined. If not, events will be defined as described in Chapter 2. With events defined, the Software Performance Analyzer will begin a new measurement of the same type as performed last. If this is the first "profile" to be taken, the Software Performance Analyzer will profile program activity. If a previous measurement is being repeated, the former results display will be placed on screen (either a histogram or table). If this is the first profile measurement (a default measurement of program activity), a histogram will be placed on screen.

To obtain a profile of the activity of your program code

- Choose **Profile→Profile ...**, and in the dialog box, click on Program Activity. Then click OK or Apply.

- Using the command line, enter the command: **profile program_activity**

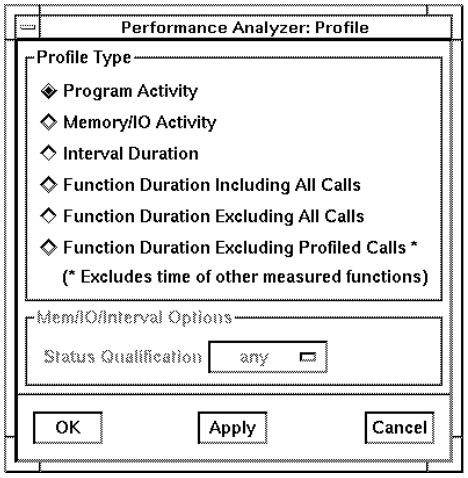
  With this command, the Software Performance Analyzer will first check to see if events have been defined. If not, events will be defined as described in Chapter 2. Then the Software Performance Analyzer will begin sampling activity for all of the qualified events. To qualify for a program activity profile:

  - An event must represent program address space (either a function or a range of functions).
  - An event must be "selected" in the events list.

A results display will appear on screen. It will be a histogram, unless you selected a table display in your last measurement. The data shown in the display of measurement results will be updated continuously until you stop the measurement. To select a different display of measurement results, use the **display** command discussed in Chapter 4.

Examples

To begin a profile of program activity, choose **Profile→Profile ...**, and set up the dialog box as shown:



or on the command line enter:

profile program_activity

To obtain a profile of the activity of variables and I/O ports

- Choose **Profile→Profile ...**, and in the dialog box, click on Memory/IO Activity. You can specify the type of processor status that will qualify the addresses of the variables and I/O ports, if desired. Then click OK or Apply.

- Using the command line, enter the command: **profile memory_and_io_activity**

  With this command, the Software Performance Analyzer will first check to see if events have been defined. If not, events will be defined as described in Chapter 2. Then the Software Performance Analyzer will begin sampling activity for qualified events. To qualify for a memory_and_io_activity profile:

  - An event must represent memory address space (either a static variable, an I/O port, or a range of variables).
  - An event must be "selected" in the events list.

  You can further qualify the types of cycles that the Software Performance Analyzer will record by specifying status conditions. If you specify status any, all cycles that match the active event will be recorded. If you specify a particular kind of status, only cycles that match the active event and are of the specified type will be recorded.

  A results display will appear on screen. It will be a histogram, unless you selected a table display in your last measurement. The data shown in the histogram will be updated continuously until you stop the measurement. To select a different display of measurement results, use the **display** command discussed in Chapter 4.

Examples

To profile activity of variables and I/O ports:

Choose **Profile→Profile ...**, and make desired selections in the dialog box.

profile memory_and_io_activity
profile memory_and_io_activity status data_write
profile memory_and_io_activity status any

To obtain a profile of durations of functions defined in your source files

- Choose **Profile→Profile ...**, and in the dialog box, click on one of the profile types beginning with Function Duration. Then click OK or Apply.

- Using the command line, enter the **profile function_duration** command.

  With this command, the Software Performance Analyzer will first check to see if events have been defined. If not, events will be defined as described in Chapter 2. Then the Software Performance Analyzer will begin making real-time records of the durations of the qualified events. To be qualified for function_duration measurements:

  - An event must be defined as a function in the events list (representing a function in the source file). No intervals, ranges, or static variables will be accepted in this measurement.
  - An event must be "selected" in the events list.

  The function-duration measurement offers three measurement modes: including all calls, excluding all calls, and excluding profiled calls. The including all calls mode includes in the function-duration, all time associated with calls made by the function and all time taken by interrupts to the function.

  The excluding all calls mode excludes from the function duration the time associated with any call to any function (whether measured or not). The measured time is strictly the time required to execute the code in the address range of the function.

  The excluding profiled calls mode is similar to the including calls mode, except that any call to another profiled function (one that is also being profiled in the present measurement) will be excluded from the time of the current function. Interrupts, if not being profiled, will be included into the durations of the functions. Interrupts that are being profiled as functions in the present measurement will be excluded from the measured function durations.

  You can set up an enable/disable specification to exclude interrupt processing time, if desired.

To obtain a profile of durations of functions defined in your source files

A results display will appear on screen. It will be a histogram, unless you selected a table display in your last measurement. The data shown in the histogram will be updated continuously until you stop the measurement. To select a different display of measurement results, use the **display** command discussed in Chapter 4.

Examples

To profile the durations of functions and include time spent executing code in other functions called by the selected functions, and time spent processing interrupts that occur while the event is active:

Choose **Profile→Profile...**, and click on Function Duration Including All Calls

profile function_duration include_calls

To profile the durations of functions, but record none of the time spent executing code in other functions called by the selected functions, and none of the time spent processing interrupts:

Choose **Profile→Profile...**, and click on Function Duration Excluding All Calls

profile function_duration exclude_calls

To profile the durations of functions, but:

- record none of the time spent executing code in other functions that are presently included in the measurement.
- include all of the time spent processing interrupts and executing code in functions that are not included in this measurement:

Choose **Profile→Profile...**, and click on Function Duration Excluding Profiled Calls

profile function_duration exclude_profiled

To obtain a profile of durations of intervals

- Choose **Profile→Profile ...**, and in the dialog box, click on Interval Duration. You can specify the type of processor status that will qualify the interval addresses, if desired. Then click OK or Apply.

- Using the command line, enter the **profile interval_duration command.**

  With these commands, the Software Performance Analyzer will begin making real-time records of the durations of qualified interval events. To qualify an event for an inverval_duration profile:

  - An event must be defined as an interval in the events list. Refer to Chapter 2 for instructions on how to create single events for intervals. No functions, ranges, or static variables will be accepted in this measurement.
  - An event must be "selected" in the events list.

  Interval measurements can include "status". That is, an interval can begin and end on a write transaction to an address; with this specification, read transactions to that same address would be ignored during the measurement.

  A results display (histogram unless you selected table in your last measurement) will appear on screen. The data in the histogram will be updated continuously until you stop the measurement. To select a different display, use the **display** command discussed in Chapter 4.

  Note that interval_duration measurements are not corrected for unused prefetches. Refer to interval_duration measurements without prefetch correction in Chapter 7.

Examples

To define interval events for measuring durations between two points:

Choose **Events→Define Single Event ...**, and click on Interval, and type myfunction1 beside Start Address, and myfunction2 beside End Address.

define single_event interval myfunction1 **thru** myfunction2

To define interval events for measuring durations between transactions to a single point (such as writes to a variable):

Choose **Events→Define Single Event ...**, and click on Interval, and type myvariable beside Start Address and myvariable beside End Address.

define single_event interval myvariable **thru** myvariable

To execute interval_duration measurements:

Choose **Profile→Profile ...**, and click on Interval Duration. Select a Status Qualification, if desired.

profile interval_duration
profile interval_duration status data_write

To see the time-range details under a selected event

- Place the mouse pointer on the desired event Click the *select* mouse button, or use the *select* mouse button to obtain the popup menu, and click on **Expand Event Toggle**.

- If using the command line:

  1 Place the cursor beside the event of interest in the Name column of the table or histogram.

  2 Press the softkey named: **EXPAND**

  Expanding an event to see its time-range details is only available in interval_duration and function_duration measurements. A set of time ranges will appear under the event name. Initially, the time ranges will show values of zero. Each new duration measurement will be recorded in two places:

  - It will be added to the value of duration shown beside the event name.
  - It will be shown beside the time range where it fit.

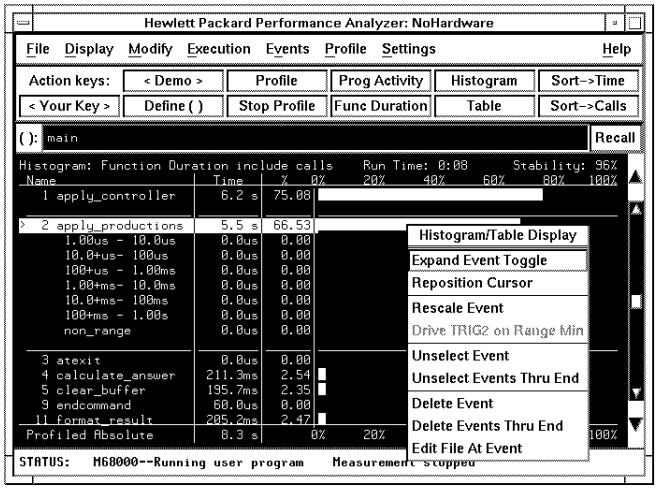
  If a duration of 720 usec is recorded for an expanded event, and if you have a 100 us - 1 ms time range under that event, the 720-us duration will be shown beside it. Otherwise, it will be added to the time range labeled non-range.

  You can define a different set of time ranges, if desired. Use a command beginning with: **display time_ranges ....** The next paragraph in this chapter explains how to set up your own set of time ranges. This information is also available in the help screens of the Software Performance Analyzer.

Examples

To obtain expanded information for an event:

Place the mouse pointer on the event you wish to expand. Obtain the popup menu, and click on **Expand Event Toggle**.



To create your own time ranges to be used under an event in a duration measurement

- Choose **Display→Time Ranges ...**, and in the dialog box, specify the start and end limits of the time ranges, the number of ranges desired, and whether those ranges will be logarithmic or linear. Then click OK or Apply.

- Using the command line, enter the **display time_ranges** command.

  The present set of time ranges are shown in the measurement specification display. There are several ways you can define a new set of time ranges to be shown under an expanded event during a measurement. You can enter a single range of time and let the Software Performance Analyzer divide it into the number of ranges you desire, or you can specify the exact time ranges desired.

  Up to 10 time ranges can be created to show below expanded events in histograms and tables.

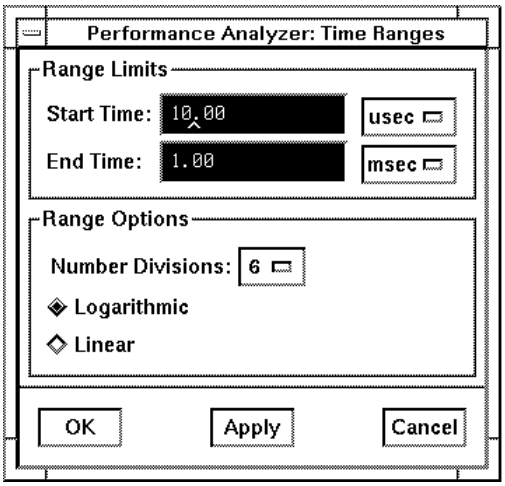
  You can only expand events to see durations in your set of time ranges during function_duration and interval_duration measurements. Durations of events are not recorded in activity measurements.

Examples

To specify an overall range to be divided into a set of time ranges, choose **Display→Time Ranges ...**, and set up the dialog box as shown:

The example dialog box defines a single, overall range; it will be divided automatically into time ranges by the Software Performance Analyzer. The time ranges will be divided logarithmically. The overall range will be divided into six time ranges.

To create your own time ranges to be used under an event in a duration measurement



or on the command line enter:

display time_ranges start_at 10 usec end_at 1 msec logarithmic number_divisions 6

To specify an overall range that begins at 1 usec, is divided into 10 time ranges, and each time range is 50 usec wide, use the command line below. Note that "plus_increment" accepts values from 1 to 100,000. Ten is the maximum (and default) number of time ranges that can be defined.

display time_ranges start_at 1 usec plus_increment 50 number_divisions 10

To specify an overall range that begins at 10 usec, and the length of each time range is equal to the lowest value of the range multiplied by 5, use the command line below. Note that "multiply_increment" accepts values from 2 to 1000. Ten time ranges will be created, by default. The list below the command line shows the time ranges that will be created by the example command:

display time_ranges start_at 10 usec multiply_increment 5

```
10.0 us - 50.0us
50.0+us- 250us
250+us - 1.25ms
1.25+ms- 6.25ms
6.25+ms- 31.2ms
31.2+ms- 156ms
156+ms - 781ms
781+ms - 3.91s
3.91+s - 19.5s
19.5+s - 97.7s
```

To specify four time ranges by specifying range ending values, enter the following command on the command line. Note that each number in the command represents the end of a time range. Each range begins at the lowest possible number. The first range, by default, begins at 1 usec; it runs from 1 usec to 10 usec. The second range runs from 10+ usec to 40 usec. The plus sign indicates the second time range does not overlap the first; it begins at a point just beyond the first time range. You can specify time ranges from 1.00 usec through 999.0 seconds.

display time_ranges 10 usec, 40 usec, 100 usec, 1 msec

To specify three time ranges by specifying beginning and ending values, use the command line to enter the following command. This example command specifically defines the beginning and ending values of each time range. Again, the beginning values of all ranges after the first range will carry "+" signs to show they do not overlap the ranges preceding them.:

display time_ranges 10 usec thru 100 usec, 100 usec thru 1 msec, 1 msec thru 1 sec

To stop the present profile measurement

- Choose **Profile→Stop Profile**.

- Using the command line, enter the command: **stop_profile**

  With the above command, the present profile measurement will end and all data capture and display updates will end.

4

Managing the Display of Measurement Results

Histograms and tables are used to show the results of measurements made by the Software Performance Analyzer. This chapter shows how to obtain and control those displays.

The following information is discussed in this chapter:

- Displaying a histogram of time periods measured.
- Displaying a histogram of cycles or calls measured.
- Changing the scale of the histogram display
- Interpreting a table of time and cycles from an activity measurement.
- Interpreting a table of time and calls from a duration measurement.
- Sorting events on the display.
- Obtaining a list the most active events in a file.
- Obtaining three significant digits of information in columns of table and histogram displays.
- Printing a copy of measurement results.

To display a histogram of time periods measured

- Choose **Display→Histogram**. Then choose **Display→Performance Data→Time**.

- Using the command line, enter the **display histogram** command.

You can display a histogram of time periods measured for each of the events selected in the measurement. You can add **absolute** or **relative** to the command to obtain the desired type of time values in the display. Your selection will determine the lengths of the histogram bars and the percentages beside each event, as follows:

In absolute displays, the percentages and histogram bars will be adjusted by the value of the "Undefined Addresses" event, which represents all address space that was not represented by any other event in the measurement.

In relative displays, the percentages and histogram bars will be calculated on the totals of all events included in the measurement, ignoring executions in address space that was unrepresented by an event.

Examples:

To display a histogram of time periods measured:

Choose **Display→Histogram**, then **Display→Performance Data→Time**, and finally **Display→Performance Data→Absolute** or **Relative**.

On the command line:

display histogram data time
display histogram data time absolute
display histogram data time relative

To display a histogram of cycles or calls measured

- Choose **Display→Histogram**. Then choose **Display→Performance Data→Calls** or **Cycles**.

- Using the command line, enter the **display histogram data** command.

A histogram of cycles is available in activity measurements; it shows the number of cycles recorded for each active event. A histogram of calls is available in duration measurements; it shows the number of calls that were made to each event in the measurement. You can specify that the numbers recorded for each of the events be in absolute or relative values. Your selection will determine the lengths of the histogram bars and the percentages beside each event, as follows:

In displays of absolute values, the percentages and histogram bars will be adjusted by the value of the "Undefined Addresses" event, which shows cycles in address space that was not represented by any other event in the measurement.

In displays of relative values, the percentages and histogram bars show the total of calls or cycles for each of the events included in the measurement, ignoring cycles in address space that was not represented by any event.

Examples:

To display a histogram of cycles or calls:

Choose **Display→Histogram**, then **Display→Performance Data→Calls** or **Cycles**, and finally **Display→Performance Data→Absolute** or **Relative**.

On the command line:

display histogram data cycles
display histogram data cycles relative
display histogram data calls absolute

To change the scale of the histogram display

- Place the mouse pointer on an event for which you would like the histogram rescaled. Then press the *select* mouse pushbutton, and in the popup menu, click on **Rescale Event**.

- To rescale the histogram back to 100%, place the mouse pointer on the line that shows the percentages above the Histogram. Press the *select* mouse pushbutton, and in the popup menu, click on **Rescale Event**.

- Using the command line, enter the **display histogram rescale** command.

  You can change the scale of the histogram bars to obtain the best resolution of the data. If you specify a percentage on the command line, the display of histogram bars will be adjusted to fit your scale. Histogram bars that exceed the percentage you specify will be shown across the entire display. No indication will be given to show how much they exceed the percentage you specified in your command.

  If you rescale the histogram to **current_max** on the command line, the scale of the histogram will be set to the percentage of the longest histogram bar. No automatic rescaling of the histogram display takes place while the measurement is in progress. If the longest histogram bar gets a higher value during the measurement, you may want to enter the command again to rescale the display to the new **current_max**.

  When the measurement is stopped with rescale set to **current_max**, the histogram will be rescaled, automatically.

  You can specify rescale values of 0.1 percent to 100 percent.

Examples

To rescale the histogram display:

Place the mouse pointer on an event to be rescaled. Obtain the popup menu, and click on **Rescale Event**.

On the command line:

display histogram rescale current_max
display histogram rescale to_max 60 **percent**

To interpret a table of time and cycles from an activity measurement

- Choose **Display→Table.**

- Using the command line, enter the command: **display table**

  The display shows the details of information obtained in the present, or most recent, program_activity or memory_and_io_activity measurement. The columns of the display show the following:

  - Name and number of the event.
  - Number of cycles that were associated with the event during periods when the event was actively being sampled.
  - Total amount of time recorded for all bus cycles associated with the event while the event was being sampled.
  - Percentage of time recorded during the measurement that was recorded for this event (as a percent of profiled time). The percentage shown will be less when you show absolute values because the calculation will include the value of the "Undefined Addresses" event, which represents all addresses that were not represented by any of the events included in the measurement.
  - The Mean(1s) is the average time the associated event is likely to be active during any given second of program execution.
  - The standard deviation is the variation between executions of the event at the point of one standard deviation. The StDv(1s) is the value of one standard deviation of the event during any given second of program execution. A standard deviation that is greater than the mean indicates there are large fluctions about the mean.
  - The last column shows the average time required to complete one bus cycle associated with the event.

```
Table: Program Activity                        Run Time: 1:13      Stability: 57%
_Name_(sort:_time)_____|_Cycles_|__Time__|_Time_%_|_Mean(1s)|_StDv(1s)|_Time/cyc
   2 apply_productions | 2.64E06|   1.4 s|  49.88 |  498.8ms|  446.5ms|  546.9ns
  31 stack_library     |  782034| 419.8ms|  14.49 |  144.9ms|  248.5ms|  536.8ns
  27 scan_string       |  404891| 214.7ms|   7.41 |   74.1ms|  208.5ms|  530.2ns
  20 move_byte         |  251109| 134.8ms|   4.65 |   46.5ms|  141.5ms|  536.9ns
  23 report_errors     |  250261| 134.3ms|   4.64 |   46.4ms|  140.8ms|  536.8ns
  19 math_library      |  180832|  98.5ms|   3.40 |   34.0ms|  101.3ms|  544.8ns
  21 outputline        |  156125|  81.4ms|   2.81 |   28.1ms|  158.9ms|  521.2ns
   5 clear_buffer      |  134565|  73.4ms|   2.53 |   25.3ms|  142.0ms|  545.6ns
  16 lookup_token      |  102740|  56.6ms|   1.96 |   19.6ms|   62.7ms|  551.1ns
  15 input_line        |  106926|  55.5ms|   1.92 |   19.2ms|  129.2ms|  519.2ns
  26 scan_number       |   57989|  31.1ms|   1.07 |   10.7ms|   90.0ms|  536.8ns
  24 report_result     |   52309|  28.5ms|   0.99 |    9.9ms|   83.9ms|  545.7ns
  33 syntax_check      |   39934|  21.9ms|   0.76 |    7.6ms|   47.5ms|  548.5ns
  28 semantic_check    |   16318|   9.4ms|   0.33 |    3.3ms|    9.8ms|  577.0ns
__12_get_next_token____|___11475|___6.4ms|___0.22 |____2.2ms|___18.4ms|__561.8ns
 Profiled Absolute     | 5.22E06|   2.8 s|  100%  |
```

To interpret a table of time and calls from a duration measurement

- Choose **Display→Table.**

- Using the command line, enter the command: **display table**

  The display shows the details of information obtained in the present, or most recent, interval_duration or function_duration measurement. The columns of the display show the following:

  - Name and number of the event.
  - Number of calls made to the event.
  - Total amount of execution time spent in the event during the measurement.
  - Percentage of execution time recorded for this event (as a percent of the profiled time). The percentage shown will be less in a display of absolute values because the calculation of percentage will include the value of the "Undefined Addresses", which represents all addresses that were not represented by events included in the measurement.
  - The longest single duration measured during any execution of this event.

To interpret a table of time and calls from a duration measurement

- The shortest single duration measured during any execution of this event.
- The average execution of the event is shown under Mean. For a duration measurement, the Mean is the true average duration of one execution of the event.
- The standard deviation is the variation between executions of the event (it is the value of one standard deviation of the event).

```
Table: Function Duration include calls        Run Time: 1:05      Stability: 99%
_Name_(sort:_time)_____|_Calls_|__Time__|_Time_%|__Max__|__Min__|_Mean__|Std_Dev
  22 parse_command     |    145|  59.8 s|  91.11|412.3ms|412.3ms|412.3ms|  0.0us
   1 apply_controller  |    436|  50.2 s|  76.49|115.1ms|115.1ms|115.1ms|  0.0us
   2 apply_productions |   3933|  42.6 s|  65.00| 11.7ms| 10.0ms| 10.8ms|525.3us
  12 get_next_token    |    291|   9.7 s|  14.79| 33.3ms| 33.3ms| 33.3ms|  0.0us
  31 stack_library     |  13109|   9.6 s|  14.59|730.3us|730.2us|730.2us|  0.0us
  16 lookup_token      |   1748|   8.3 s|  12.61|  4.7ms|  4.7ms|  4.7ms|  0.0us
  28 semantic_check    |   2621|   6.0 s|   9.09|  2.3ms|  2.3ms|  2.3ms|  0.0us
  27 scan_string       |   5245|   4.6 s|   7.04|880.3us|880.2us|880.2us|  0.0us
  25 request_command   |    146|   3.2 s|   4.91| 22.1ms| 22.1ms| 22.1ms|  0.0us
  14 initialize        |    146|   3.1 s|   4.80| 21.6ms| 21.6ms| 21.6ms|  0.0us
  19 math_library      |  35746|   3.0 s|   4.64|188.0us| 19.0us| 85.2us| 44.6us
  20 move_byte         |   3936|   2.9 s|   4.38|730.3us|730.2us|730.2us|  0.0us
  23 report_errors     |   3929|   2.9 s|   4.37|730.3us|730.2us|730.2us|  0.0us
  24 report_result     |    146|   2.3 s|   3.55| 19.1ms|  8.8ms| 16.0ms|  4.2ms
___5_clear_buffer_____|____438|___1.9_s|___2.90|__4.3ms|__4.3ms|__4.3ms|__0.0us
 Profiled Absolute    |  76497|  65.6 s|       |
```

To sort the events on the display

- Choose **Display→Sort Events→**<sort criterion>

- Using the command line, enter the **display histogram** (or **table**) **sort_events** command.

  You may want to sort the events before removing those with low usage, or before creating a range event that overlaps several existing events. You can sort events in the histogram, table, or events list in one of five orders:

  - **time**, which places the events that used the most system time first, and the events that used the least system time last.
  - **cycles** or **calls**, which places the events that recorded the most cycles or calls first.
  - **address** which places the events that represent the lowest addresses first.
  - **alphabetical** order which arranges the display in the order of the event names.
  - **definitions**, which places the events in the order in which they were defined (order of event numbers).

  The sort you choose affects all displays. If you sort the histogram by time, the table and events list will also be sorted by time.

Examples

To sort events on the display:

Choose **Display→Sort Events→Time**

On the command line:

display events sort_events address
display table sort_events time
display histogram sort_events cycles
display events sort_events defined

To obtain a list of the most active events in a file (even a file having thousands of events)

1 Make the profile measurement.

2 Choose **Display→Histogram**, or on the command line, enter: **display histogram**.

3 Choose **Display→Performance Data→Time**, or on the command line, enter: **display data time**.

4 Choose **Display→Sort Events→Time**, or on the command line, enter: **display sort_events time**.

5 Place the mouse pointer beside the first event that recorded a small amount of time.

6 Press the *select* mouse button and click **Delete Events Thru End** in the popup menu, or on the command line, enter: **delete_events thru end**.

7 Make another profile measurement. This loads more events from the events list.

8 Repeat the above process until all events from the events list have been profiled, and only the most active events remain.

The above commands obtain a profile of the events and sort them in order, placing the event that uses the most system time at the top of the list. The last command deletes all the events that use little system time.

When you make a new profile measurement, new events from the events list will be included in the measurement. The new events are profiled against the list of active events from the preceding measurement. After several profile measurements are made, you obtain a histogram or table of the most active events from the events list.

This method can be used to accumulate a histogram or table of the most active events from a file having thousands of events. Simply make measurements, delete the inactive events (as above), define more events from the file under test, and rerun the measurements.

To obtain three significant digits in the columns of the table or histogram

- Choose **Settings→Decimal Alignment**.
- Enter the command: **set decimal_alignment off**

The above commands cause the values shown in the columns to be displayed to three significant digits. (The first command is a toggle.) With these selections, the values in the columns might not have their decimals aligned vertically.

To print a copy of measurement results

- Define the shell variable, HP64PRINTER.

Before you can specify the printer as the destination device for your copy command, you must define HP64PRINTER as a shell variable. Make this definition at the shell prompt ("$" symbol).

Example

To define the shell variable, HP64PRINTER:

If using sh(1) or ksh(1), enter:

```
$ HP64PRINTER = lp
$ export HP64PRINTER
```

If using csh(1),

```
setenv HP64PRINTER lp
```

Example

To keep the print message from appearing or from overwriting the softkey command line, execute:

If using sh(1) or ksh(1), enter:

```
$ HP64PRINTER = "lp -s"
$ export HP64PRINTER
```

If using csh(1),

```
setenv HP64PRINTER "lp -s"
```

The **-s** above makes the print message silent.

- Print measurement results.

You can print a copy of the present measurement results by using the **copy** command. The **copy** command will print the entire content of the histogram, table, measurement specification, or events list to the printer. The inverse-video bars of the histogram display will be automatically replaced by asterisks (*) when you use **copy**.

To obtain a copy of only the present display on screen (instead of the entire content of a histogram, table, measurement_spec, or events list), use the **copy display** command. If your display is a histogram that uses inverse-video bars, the bars will not be copied. Use the **set histogram_character** command to choose an ASCII character for the histogram bars before you execute the **copy display** command.

Examples

To print copies of analyzer displays:

Choose **File→Copy→<display_name>...**, and in the File Selection dialog box, enter the name of the destination file under "copy <display_name> to". Then click OK. Filenames in other directories can be obtained by entering the desired directory under File Filter and clicking on the Filter pushbutton in the dialog box.

```
copy histogram to printer
copy table to printer
copy measurement_spec to printer
copy events to printer
copy display to printer
```

5

Supporting Tasks that Add Flexibility to Performance Measurements

The tasks described in this chapter will be used occasionally during operation of the Software Performance Analyzer. They add flexibility to the analysis process

offered by the Software Perfformance Analyzer. The information in this chapter shows you how to do the following:

- save and reload a profile specification with present measured data.
- bring help screens to the display to answer questions during operation.
- see the software version number of the Software Performance Analyzer.
- edit a source file while using the Graphical User Interface.
- set the rate of recalculating stability to improve display-updates, or turn it on or off.
- specify a desired confidence in the data obtained by analysis.
- select histogram character to be used as histogram bars on display copies.
- turn off event numbers to get more space for event names.
- increase size of the display window to get more space for event names.
- defining action keys for the Graphical User Interface.
- placing information strings in the entry buffer and command line.
- copying event names to a dialog box.
- using Software Performance Analyzer with C++ programs.

To save and reload a profile specification with measured data

- Choose **File→Store→Profile Spec ...**, and in the dialog box, enter the name of the file to store your profile specification.

- Choose **File→Load→Profile Spec ...**, and in the dialog box, select the file that contains the profile specification you wish to load.

- Using the command line, enter the **store profile_spec** and/or **load profile_spec** commands.

  The **store** command in the following examples will cause the Software Performance Analyzer to store all setup commands, display specifications, and present measurement data in a file named MYfile2. If you changed any specification since the last measurement, the data will not be stored. File names can include numbers and letters; they must begin with a letter. The stored file will be placed in your current working directory.

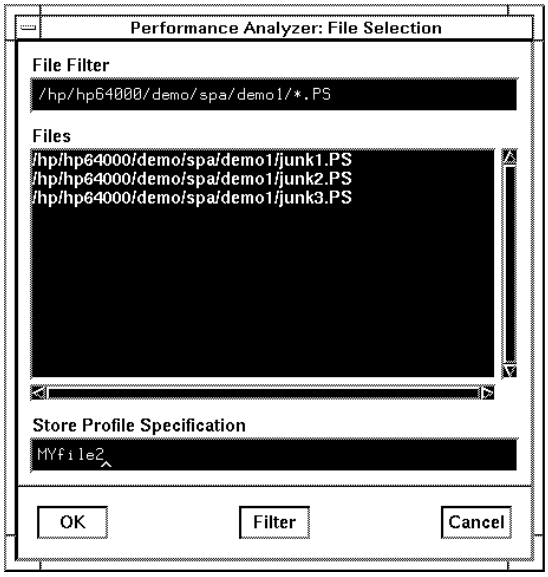
  The **load** command in the examples below will cause the Software Performance Analyzer to configure itself according to the measurement setup that was present when the file named test24 was created. The measured data that was present when the file was created will be loaded into the analyzer memory. You can obtain histograms and tables of that data, and use it for comparison against data you obtain in future measurements with the setup.

Examples

To store new profile specifications and/or load existing profile specifications (with measurement data):

Choose **File→Store→Profile Spec ...**, and set up the dialog box to specify the file name to contain the profile specification. The example dialog box is set up to store MYfile2 in the directory shown in the File Filter entry area.

Choose **File→Load→Profile Spec ...**, and set up the dialog box. The only difference in the Load and Store dialog boxes is the word Load or Store above the Profile Specification entry area.



or on the command line, enter:

store profile_spec MYfile2
load profile_spec test24

To obtain help screens for making performance measurements

- Choose **Help→Command Line ...**, and in the dialog box, click on the command you wish to review. Then click OK or Apply.

- Use the **help** or **?** command on the command line.

  Help screens are provided for each of the Software Performance Analyzer commands, and additionally for tasks you may need to do that do not appear on first-level commands, such as defining time ranges, specifying display stability, and rescaling the histogram. The help screens show command syntax, and give detailed information about the function requested, as well as example commands, default values, and descriptions of command parameters.

  The help screens answer questions you may have while working with the Software Performance Analyzer. This manual provides the information you need to fully understand the Software Performance Analyzer so you can use it to its full capability.

Examples

To obtain help screens written for the Software Performance Analyzer:

Choose **Help→Command Line...→SELECT_key**

help time_range
? time_range
help SELECT

To obtain help screens of general information about the Graphical User Interface

- Choose **Help→General Topic ...**, and in the dialog box, click on the topic you wish to review. Then click OK or Apply.

- Click on the Help button at the bottom of the Graphical User Interface screen.

  These help screens explain details about topics related to use of the Graphical User Interface implementation for the Software Performance Analyzer.

To see the software version number of the Software Performance Analyzer

- Choose **Help→Version ...**

- Using the command line, type: **version**

  If you clicked on **Version...** in the Graphical User Interface, a dialog box will appear that shows the version number. The software version number of the Software Performance Analyzer will also appear on the status line if you entered **version** on the command line.

To get help in controlling the appearance and operation of the Graphical User Interface

- Choose **Help→X Resource Names ...**

  The dialog box gives you information you need to set the X resources that control the appearance and operation of the Graphical User Interface. You can click on the Help button in the dialog box to get a brief description of the X windows concepts necessary to understand how to set the interface resources.

To edit a source file during a measurement when using the Graphical User Interface

1 Place the cursor (pointing finger) on an event in a histogram, table, or events list.

2 Press and hold the *select* mouse button until a popup menu appears.

3 Click on **Edit File at Event** in the popup menu.

4 Edit your source file in the new window that appears.

5 Quit the edit file window when you finish.

6 Recompile your executable file and test the results of your changes.

The edit window that appears when you perform the above steps shows your source file at the point where the event you selected in Step 1 is defined. By default, the vi editor is active in the edit window. You can change the editor in this window to an editor of your choice by changing the application defaults.

To control the stability calculation

- Use the **display stability** command**.**

The present stability of the data on display is shown in the upper, right-hand corner of the histogram or table. The calculation of stability includes the values of the mean and standard deviation (Chapter 8 of this manual discusses these parameters under a paragraph that shows you how to determine the validity of statistical measurements).

You can select any interval of time (in seconds) between recalculations of the stability of measured data. Each time the Software Performance Analyzer recalculates the stability, it suspends the display-update process. The more events that are included in your measurement, the longer the display-update process will be suspended. If you have included many events in your measurement, the process of recalculating stability may suspend the display-update process for several seconds.

By default, stability is recalculated every 30 seconds. You can have stability recalculated less often if suspending the display-update process causes problems for your display, or you can turn off the recalculation altogether.

You can set up a measurement to end automatically when a desired stability is reached. For this measurement, the stability calculation must be turned on; if stability is turned off, in this case, it will turn on automatically and be recalculated every 30 seconds, by default.

No method is available in the pulldown menus for setting the rate of recalculating stability.

Examples

To have stability recalculated every 120 seconds, enter:

display stability after_every 120 **seconds**

To turn off the stability calculation, enter:

display stability off

To have your measurement end when 98% stability is obtained, enter:

setup_measurement termination stability 98 **percent**

To specify a desired confidence in the stability of the measurement data

- Use the **set confidence** command.

  By default, the Software Performance Analyzer has a setting of 95% confidence. This indicates the percent of confidence you require in the value of stability that is returned.

  The confidence you set in your measurement specification will determine how long your measurement must run. Stability specifications (discussed in Chaper 8) interact with confidence specifications. If, for example, you specify that your measurement should run until 98% stability is achieved in the measurement results, then the Software Performance Analyzer will run until it has captured enough data to be 95% confident (with the example specification below) that its data has reached a stability of 98%. Refer to Chapter 8 for further information.

  No method is available in the pulldown menus for setting confidence in the stability of measurement data.

Example

To specify desired confidence in measurement data:

```
set confidence 95 percent
```

To change the histogram character

- Use the **set histogram_character** command.

  By default, the histogram_character is an inverse-video bar. You might want to use this command to select a different histogram character when copying your display to a printer.

  The inverse-video bars will be replaced with asterisks (*) in the file that is created by execution of the **copy histogram** command.

  When you use the **copy display** command with a histogram on screen, the inverse-video bars are not copied at all. You must use the **set histogram_character** command to define a histogram character before you enter the **copy display** command.

  The **set histogram_character** command accepts decimal values between 33 and 127. The value you choose will specify the corresponding ASCII character. A decimal value of 127 makes a good histogram bar with most fonts. You can also enter the character you want to use by placing it in quotes in the command.

  No method is available in the pulldown menus for setting the histogram character.

Examples:

To specify a histogram character to be used when printing histogram copies:

set histogram_character 127
set histogram_character "#"
set histogram_character "*"

To turn on or off the event numbers

- Choose **Settings→Event Numbers**

- Using the command line, enter the **set event_numbers** command.

You might want to turn off the event numbers in your display to obtain additional character space for the names of your events. By turning off the event numbers, four additional character spaces will be available.

The event numbers, by default, show the order in which events were defined. To reassign event numbers, sort the events in the desired order and choose **Events→Renumber Events**, or enter the command: **renumber_events**.

Examples:

To turn on and turn off the event numbers:

Choose **Settings→Event Numbers**

on the command line:

set event_numbers on
set event_numbers off

To resize the display window

- To obtain a new size for the display window allocated to the Graphical User Interface, add the following lines to your .Xdefaults file:

  HP64_Softkey.lines: <HEIGHT>
  HP64_Softkey.columns: <WIDTH>

  or, for only the Software Performance Analyzer:

  perf.lines: <HEIGHT>
  perf.columns: <WIDTH>

- If you are going to use the Softkey User Interface to the Software Performance Analyzer, the following steps will resize a display window:

  1. On your workstation, place the mouse pointer on the window edge you wish to move. (The Software Performance Analyzer must not be running in the window at this time.) Press and hold the *pushbutton select* mouse button and drag the window edge to the desired size.
  2. Enter: **eval ‘resize‘**
  3. Start the Software Performance Analyzer with the command:

     emul700 -u skperf <logical name>

  In entry above, the left-hand single-quote mark is used on both sides of **‘resize‘**.

  By increasing the width of the display window that contains the Software Performance Analyzer, more space will be allocated to display event names. This can be helpful if you are using very long event names. Note that this will not increase the space allocated to display of the histogram lines, display of data in table columns, or display of the command line.

To define action keys

1 In your .Xdefaults file, find the structure that looks like the following:

```
!---------------------------------------------------------------
! Action Key Definitions (See also XcHotkey discussion above)
perf*actionKeysSub.keyDefs:\
    "< Demo >"       "!telldemoHP! in_browser" \
    "Profile"        "profile" \
    "Prog Activity"  "profile program_activity" \
    "Histogram"      "display histogram" \
    "Sort->Time"     "display data time; display sort_events time"\
    "< Your Key >"   "!tellkeysHP! in_browser" \
    "Define ()"      "define single_event ()" \
    "Stop Profile"   "stop_profile" \
    "Func Duration"  "profile function_duration" \
    "Table"          "display table" \
    "Sort->Calls"    "display data calls; display sort_events calls"
```

2 Edit the above structure to add:

```
    "Sort->Calls"    "display data calls; display sort_events calls"\
    "<label string>" "<action string>" \
    "<label string>" "<action string>"
```

3 Restart the interface.

You can define action keys to appear in the Action Keys line of the Graphical User Interface and set them up to perform tasks you desire. An action key definition consists of a label string and an action string. If either string contains a blank space, quote the string. Place each label-string/action-string pair on a separate line for readability. Separate the label string from the action string by at least one space.

The .Xdefaults file is read when the interface starts. New action keys will appear. They will have the names you assigned in <label string>. When you click on one of them, the associated command (<action string>) will be executed.

Note that if these definitions are not in your .Xdefaults file, you can copy them from the HP64softkeys file in the app-defaults directory.

Example

To enter additional action-key definitions in the .Xdefaults file:

```
"Sort->Calls"    "display data calls; display sort_events calls"\
"Mem/IO"         "profile memory_and_io_activity" \
"Delete()->End"  "delete_events () thru end"
```

To define action keys that run profile measurements

- Enter the following label-string/action-string pairs in the .Xdefaults file:

```
"<action key name>"   "<action string that is a profile command>"
```

The action-key feature is useful if you want to run a series of measurements that require you to switch back and forth between measurement types. You can simply click on the action key that represents the measurement desired; the associated profile command will be executed.

For general information on defining action keys, refer to the previous paragraph titled, "To define action keys."

Example

To define action keys that run profile measurements:

```
"Func_Inc"       "profile function_duration include_calls" \
"Func_Exc"       "profile function_duration exclude_calls" \
"Mem/IO"         "profile memory_and_io_activity"
```

To define an action key that deletes low-usage events

- For general information on defining action keys, refer to the paragraph titled, "To define action keys" in this chapter.

- Enter the following label-string/action-string pair in the .Xdefaults file:

```
"Delete()->End"  "delete_events () thru end"
```

The above step creates an action key that will delete low-usage events from the events list. When you restart the interface, the Delete()->End key will appear in the action keys line. Use it after each new performance measurement, as follows:

1 Sort the histogram, table, or events list by time, cycles, or calls (as appropriate).

2 Click on the first event to have low usage. This places its event name in the entry buffer text area.

3 Click on the Delete()->End action key. All events will be deleted from the first low-usage event through the end of the display.

The content of the entry buffer string will replace the "()" in your action string. It's a good idea to include the "()" symbol in your action key name to remind you to place the appropriate information in the entry buffer before you click on the action key.

To define an action key that runs a command file

- For general information on defining action keys, refer to the task module titled, "To define action keys" in this chapter.

- Enter the following label-string/action-string pair in the .Xdefaults file:

  ```
  "<label string>"   "<command file name>"
  ```

  You can define an action key to run a command file. This way, you can execute an action that requires a series of commands.

To define two or more lines of action keys

- Enter the following resources in your .Xdefaults file:

  ```
  perf*actionkeys.packing:PACK_COLUMNS
  perf*actionKeys.numColumns:<number of lines desired>
  ```

  The above lines enable a multi-line action key display. You can define two or more lines of action keys. You will have to experiment with this entry because of differing lengths of key names in your label strings.

Example

To define three lines of action keys:

```
perf*actionkeys.packing:PACK_COLUMNS
perf*actionKeys.numColumns:3
```

Note that normally columns are vertical and rows are horizontal. In this case, these are columns rotated 90 degrees.

To place information strings in the entry buffer of the Graphical User Interface

- Place the cursor in the field of the entry buffer and type in the desired information string.

- Cut words or lines from a histogram, table, events list, or any other display by highlighting the desired text using the left mouse button.

- Click on the Recall button to obtain the Recall dialog box. Click on the information string desired in the Recall dialog box. Now click on APPLY or OK.

  If you obtain information strings from the Recall dialog box, you can click on either APPLY or OK to enter the information in the entry buffer. If you click on APPLY, the dialog box will remain on screen. If you click on OK, the dialog box will close. You can also double click on the desired line; this is the same as clicking OK.

  Use the entry buffer to create information strings for commands. You can copy the content of the entry buffer into the command line. Action keys can be defined to include the present content of the entry buffer in their commands.

  If you try to cut words or lines from a histogram or table display while a measurement is in progress, you may get unexpected results because the display is updated constantly during the measurement. It is best to stop the profile measurement before cutting from the histogram or table to the entry buffer.

To copy the entry buffer to the command line of the Graphical User Interface

1 First place the desired information in the entry buffer. See the paragraph titled "To place information strings in the entry buffer of the Graphical User Interface."

2 Place the cursor at the desired point in the command line and click the *command paste* mouse button.

To copy an event name to a dialog box

1 First place the desired information in the entry buffer. See the paragraph titled "To place information strings in the entry buffer of the Graphical User Interface."

2 Double click the left mouse button in the entry buffer to highlight the entry buffer content.

3 Move the cursor to the destination field in the dialog box and click the *command select* mouse button.

To use the Software Performance Analyzer with C++ Programs

The Software Performance Analyzer has been designed to work with C programs. If your C++ compiler translates the C++ programs to C before compiling, you should be able to use the Software Performance Analyzer with it.

The Software Performance Analyzer has been tested with the MRI C++ compiler. The MRI C++ compiler uses a translator to translate the C++ files to C files. The translation process creates a variety of C functions, each of which corresponds to one of the member functions of the C++ program. The Software Performance Analyzer can measure the duration and activity of these created functions just as it would a C function.

When using the MRI compiler, set up the needed function start and end prefetch adjustment (HPSPAADJUST), or use markers (HPSPAMARKERS). Refer to Chapter 7 or to the "prefetch" and "markers" help page.

As an example, using a Motorola 68000 microprocessor, and the MRI C++ compiler, set the shell variable HPSPAADJUST="2 0". With this setting, a function duration measurement will yield the following results. Note that the "perf.columns:100" resource has been set in the Xdefaults file to get a wider Sofware Performance Analyzer window.

```
Histogram: Function Duration exclude profiled                      Run Time: 0:54
_Name_(sort?_time)_____________________________|__Time__|__%___0%_____20%_____40%___
  15 fnIdle__Fv                                |  19.8 s| 35.98|**************
   7 d_move__7DisplayFiT1                      |  14.2 s| 25.75|**********
  10 d_printf2s__7DisplayFPcT1                 |  10.8 s| 19.55|********
  42 showBuf__7DisplayFPcT1                    |   9.0 s| 16.38|*******
      :
_Undefined_Addresses___________________________|__61.7ms|__2.11|____________________
 Profiled Absolute                             |  55.0 s| 100% 0%     20%     40%
```

If the above result is passed through the Host C++ utility **c++filt(1)**, we can get a C++ description of the function names. Use the commands below to get the next display.

copy histogram to /tmp/SPA_results noappend
!cat /tmp/SPA_results | c++filt !in_browser

Note that column alignment is affected after using the above commands.

```
Software Performance Analyzer                 Wed Aug 11 14:02:58 1993

Histogram: Function Duration exclude profiled                     Run Time: 0:54
_Name_(sort?_time)_____________________________|__Time__|__%___0%_____20%_____40%___
  15 fnIdle(void)                              |  19.8 s| 35.98|**************
   7 Display::d_move(int,int)                   |  14.2 s| 25.75|**********
  10 Display::d_printf2s(char*,char*)           |  10.8 s| 19.55|********
  42 Display::showBuf(char*,char*)              |   9.0 s| 16.38|*******
_Undefined_Addresses___________________________|__61.7ms|__2.11|____________________
 Profiled Absolute                             |  55.0 s| 100% 0%     20%     40%
```

Now run the following commands to reformat the display:

copy histogram to /tmp/SPA_results noappend
!cat /tmp/SPA_results | c++filt | cut -c1-42 >/tmp/SPA.start! in_browser
!cat /tmp/SPA_results | c++filt | cut -f2,3,4 -d\| >/tmp/SPA.end! in_browser
!paste /tmp/SPA.start /tmp/SPA.end !in_browser

The display will be rearranged to look something like this:

```
Software Performance Analyzer                 Software Performance Analyzer
             Wed Aug 11 14:02:58 1993

Histogram: Function Duration exclude profi    Histogram: Function Duration exc
_Name_(sort?_time)_____________________       __Time__|__%___0%_____20%_____40
  15 fnIdle(void)                               19.8 s| 35.98|**************
   7 Display::d_move(int,int)                   14.2 s| 25.75|**********
  10 Display::d_printf2s(char*,char*)           10.8 s| 19.55|********
  42 Display::showBuf(char*,char*)               9.0 s| 16.38|*******
_Undefined_Addresses___________________       __61.7ms|__2.11|_______________
 Profiled Absolute                             55.0 s| 100% 0%     20%     40
```

In summary, if you have a Native **c++filt(1)**, or a command line utility to reformat the C++ names, then you can add the above commands into a command file. An action key can be used to run this command file to create displays that are easier to read.

Defining C++ functions

To selectively define events (functions to measure), you can use features of the Define Events dialog box, obtained by entering: **Events→Define Events...**. You can specify the file names from which to extract functions, or you can enter a pattern to be matched for the base class name from which to look for functions.

For example, to define events for all of the functions in "main.cxx", unselect the "Globals Only" toggle and enter "main.cxx:" in the "Symbol:" entry field in the Symbolic Filter region.

To get all functions of the base class "Display", select the "Matching" option in the "Pattern Filter" region and then enter "*DisplayF*" in the "Pattern:" entry field. Be sure to clear the "Symbol:" option or select "Globals Only" in the Symbolic Filter region.

For additional help, refer to the description of the Define Events dialog box in Chapter 9, or to the help screen for "Dialog Box: Define Events".

6

Measurement Problems

The problem-solving information in this chapter is to help you with problems you may encounter while trying to run the Software Performance Analyzer. Problem-solving information to help a service technician is presented in Chapter 17 of this manual.

Problems discussed in this chapter include:

- Software Performance Analyzer won’t turn on.
- Symbols not loaded in emulation data base.
- Software Performance Analyzer won’t make a measurement.
- Incorrect measurement results.
- Data not saved with profile specification.
- Appearance of the Event Rate Overflow message.
- Appearance of the Stack Overflow message.
- Appearance of the Event Rate Underflow message.
- Histogram or table updated in small blocks during the measurement.
- Display often freezes for several seconds during a measurement.
- When events are defined for some functions, but not for others.
- When the Time and Time% columns do not total 100%.
- If the trigger (trig2) does not seem to work correctly.
- If "XSigServe" runs after you exit from the Graphical User Interface.
- Drag-thru menu selection is too slow with Graphical User Interface.
- Help screens cover display window with Graphical User Interface.

If the Software Performance Analyzer won't turn on

☐ Check that you loaded the proper configuration into the emulator.

☐ Check that you loaded the absolute file and started it running before you attempted to call the Software Performance Analyzer.

If symbols are not loaded

☐ Check that the symbol data base does exist. To check the symbol data base:

1 Enter "**!ls**" on the command line and see if the file "<FILENAME>.Ys" exists in your present working directory ("<FILENAME>.Ys" is the symbol data base).

2 If "<FILENAME>.Ys" exists, enter:
load symbols <FILENAME>.

3 If "<FILENAME>.Ys" does not exist, enter
!srubuild <FILENAME>.
Then enter: **load symbols <FILENAME>**.

☐ Recompile your program.

- Enter: **!make runtest**

If the Software Performance Analyzer won't make a measurement

- ☐ Check that events are defined.
- ☐ Check that events are selected.
- ☐ Check that the emulator is still running your program.
- ☐ Check that trig1 has occurred if the measurement is set up to wait for trig1.
- ☐ Check to see if you have an enable specification and the enable has occurred.

If measurement results are incorrect

- ☐ Check if the emulator reported a slow clock.
- ☐ Check if the emulator was running in its monitor program during the measurement. Time spent running in the monitor program is added to the totals measured in some duration measurements. Refer to Chapter 7 for a discussion of the effects of the emulation monitor on activity measurements and duration measurements.
- ☐ Check if prefetching or recursion have caused a problem. Refer to the reference information about prefetch and recursion in Chapter 7.
- ☐ Check if the cache of your emulation processor is turned on. If it is, either turn it off, or refer to Chapter 7 for information on how to use markers to overcome the effects of an enabled cache.

☐ Check if you are making measurements in a real-time operating system. Function-duration and interval-duration measurements require a special setup when they are made in a real-time operating system. You must ensure that the measurement of duration is only made when the task of interest is active. HP recommends you set up your operating system to notify you when it activates and deactivates the task of interest by having it write to a pair of variables (such as: start_task2, and end_task2). Then you can use the enable/disable feature of the Software Performance Analyzer to ensure that durations are only recorded when the task of interest is active (such as: enable on start_task2, and disable on end_task2). Refer to the paragraph titled "To prepare for a measurement of durations within limited regions of execution" in Chapter 2 of this manual.

You can obtain additional information to help you understand your measurement results by using your emulators emulation-bus analyzer. Trace on the addresses of the events that produce confusing results (trace on event-start and event-end addresses). The trace list of the emulation-bus analyzer will show what the Software Performance Analyzer is trying to measure.

For additional information, refer to the online help information available by choosing **Help→General Topic ...**, and selecting **Real-Time OS Considerations** from the Help index. Using the command line, enter **help real_time_OS**.

If the Software Performance Analyzer did not save data when it saved its profile specification

☐ Check to see if you changed any measurement specification after capturing the data. The data will only be saved if the setup used to capture it has not been changed. Changes to any of the following specifications will prevent the Software Performance Analyzer from saving its data:

any **setup_measurement** command

a **define** command

a **delete** command

a **time_range** change command

a **select** command

an **unselect** command

an **expand** command

a **load** command

a **set status qualifier** command

a **profile** command that produced no output

If the Event Rate Overflow message appears on the display

- ☐ Check to see if your program is short, having only a few functions that repeat so often that the Software Performance Analyzer had insufficient time to calculate the time and record the measured data.
- ☐ Check if your program enters a loop that executes a few small functions several thousand times, allowing insufficient time to calculate and record measured data.
- ☐ Unexpand one or more events in your table or histogram, if events are presently expanded.
- ☐ Unselect a few of the functions that are being called most often, or select fewer functions.

The Software Performance Analyzer may be thought of as a microprocessor trying to keep up with your target microprocessor. Normally, the Software Performance Analyzer has an advantage because it is examining fewer functions than the total number of functions that are executing. The Software Performance Analyzer also typically has an advantage because it only looks for entry and exit pairs of addresses (not all of the addresses) of functions or intervals.

Once an entry and exit pair have been found, the Software Performance Analyzer must calculate the time and record the statistics for that pair. In normal test conditions, there is sufficient time to do the calculations and record keeping.

If your program is short and has only a few functions that execute repeatedly, or if your program enters a loop that executes a few small functions several thousand times, the Software Performance Analyzer may not be able to perform its calculations and keep up with the incoming events. In this case, the incoming events are stored in an internal buffer. If this buffer becomes full, the Software Performance Analyzer stops gathering events temporarily while it processes all of the events in the buffer. When the buffer is empty, the Software Performance Analyzer resets and starts gathering events again.

When events are expanded in a table or histogram display, additional processing time is required. This requirement for additional processing time might cause event rate overflow to occur.

If the Event Rate Overflow message appears on the display

When event rate overflow occurs, many event entry and exit points can be lost. The Software Performance Analyzer operates in a sampling-like mode. You can determine the approximate amount of sampling that has occurred by comparing the run time shown at the top of the display with the profiled time shown at the bottom of the Time column. For example, if the run time is 2 minutes and the profiled time is 60 seconds, the sampling is about 50 percent.

If an enable and disable pair are specified and event rate overflow occurs, the measurement will resume after the events buffer has been processed, but the Software Performance Analyzer may be disabled. The Software Performance Analyzer keeps track of whether it is enabled or disabled even during periods of event rate overflow. The amount of time shown as the disable time will be approximately correct after an event rate overflow has occurred.

The time lost while the Software Performance Analyzer is not gathering events is deducted from the total execution time. Some additional time is typically added to the Undefined Address time. The time percentages shown might not be exactly correct. The max, min, and mean times recorded for the events will be correct for the data they have captured.

The message, "Event rate overflow" will be shown on the table or histogram whenever an event rate overflow has occured during a measurement. The same message will appear on the Status line only during the time the overflow condition is occurring.

The search for a trigger event (to generate trig2) is not affected by an event rate overflow condition.

If the Stack Overflow message appears on the display

☐ Add NOP padding between functions in your source file to prevent function start addresses from appearing as unused prefetches.

☐ Unselect events that represent alternate functions in the source file, and rerun your test. You can reselect these alternate events and unselect the others in a later measurement.

The stack overflow status message appears in function-duration measurements when the appropriate exit points are not found for events whose entry points are already in the stack. The internal stack of the Software Performance Analyzer is 1000 states deep and behaves like a function-call stack. If you have defined functions with incorrect exit points, or if extra entry points are prefetched in a unique way, the stack will fill up with entry points. This will happen if your compiler has placed the exit point of one function together with the entry point of another function at the same program address (such as the same long-word address in the program memory of a 68020 microprocessor). Now assume that the Software Performance Analyzer is only measuring the second function. (The Software Performance Analyzer will not allow both functions to be measured because they overlap.) Because the exit point and entry point are both fetched together in the same fetch cycle, the Software Performance Analyzer will record a function entry anytime the long-word address is fetched. This will occur even when the long-word address is being accessed simply to obtain the exit of the previous function.

Normally, the problem of stack overflow can be corrected by separating the functions from one another by inserting some NOP instructions between them. A second method you can use to correct the problem is to isolate the functions that cause the problem. Do this by unselecting groups of functions and trying to make the measurement again. You can reselect the offending functions to include them in later measurements with other functions. A third method you can use to correct this problem is to adjust the definition of the start and end of the function. Refer to the paragraphs that describe steps you can take to correct for unused prefetches in Chapter 7.

When the stack overflows, the time associated with the events in the stack will be added to the event named, "Undefined Addresses".

The search for a trigger event (for trig2) is not affected by a stack overflow condition.

If the Event Rate Underflow message appears on the display

- ☐ The simplest way to fix an event rate underflow condition is to select at least one function that is executed a least once every second.

- ☐ A second solution may be to use the enable and disable pair to disable the Software Performance Analyzer when it is running in the block of functions that you don't care to measure.

- ☐ A third solution may be to create interval events to represent the events you want to measure and use the interval duration measurement mode. Interval duration measurements have no minimum rate requirement, but they also don't correct for common prefetch conditions.

  Function duration measurements need events (entry or exit points) to occur at least once every 1.25 seconds. This minimum event rate will keep the prefetch correction circuitry in sync with the time counter. If the minimum event rate is not met, the event rate underflow message will appear and a few entry and/or exit points will be lost. The effect you will see most often when event rate underflow occurs is the lack of recording functions that last more than 1 second.

  The time that is lost when an the event rate underflow occurs will be added to the time shown beside the event named, "Undefined Addresses".

  The search for a trigger event (to generate trig2) is not affected by an event rate underflow condition.

If only small blocks of the histogram or table are updated during the measurement

☐ Network transfer rates might affect the display-update process of the Software Performance Analyzer. If you are using the Software Performance Analyzer through an RS-232 network, the transfer rate of the network will affect your display. HP recommends you do not use the Software Performance Analyzer through an RS-232 network.

The Software Performance Analyzer updates its display one time each second while a measurement is in progress. The display-update process takes about 0.4 second. The transfer rate of RS-232 is too slow to supply all of the information needed to completely update your screen in one 0.4-second period. Therefore, the display of measurement results are updated in a phased process. You will see a block of your display updated each second. At any given moment, some of the information will be current and the rest of it will be old. The only way to overcome the phased-update problem is to work through a faster network. If you are using the Software Performance Analyzer through a local-area network (LAN), your display will be updated correctly.

If the display often freezes for several seconds during a measurement

☐ Check to see if the recalculation of stability is causing the problem. Each time the Software Performance Analyzer recalculates stability, it freezes its display-update process. If you have included many events in your measurement, the process of recalculating stability may freeze the display-update process for several seconds. You can make recalculations of stability happen less often, or you can turn off the recalculation altogether.

Examples

To change the stability recalculation, try one of these commands:

display stability after_every 120 **seconds**
display stability off

If events are not being defined for some functions, but are for others in a source file

☐ Check the error_log display for additional information on how many events were defined.

☐ Check to see if the problem is listed as overlapping events in the error_log. The problem of how overlapping events can occur is explained below.

Many microprocessors fetch operands on long-word boundaries. This is no problem for the Software Performance Analyzer, except when the compiler places the end instruction of one function and the start instruction of another function together in the same long word. For the Software Performance Analyzer to work properly, the compiler must place the start addresses and end addresses of all symbols (functions) in different long-word addresses. Otherwise, the Software Performance Analyzer reports overlapping addresses for the two events whose start and end addresses are together in the same long word.

With HP 64000 AxLS compilers, separating function-end and function start addresses into different long words is done by adding "debug" options (-OG) to your compiler command. The debug options cause the compiler to insert NOP instructions between the functions when it creates the executable file.

The same problem occurs if your microprocessor fetches byte instructions on word boundaries (the exit of one function and the entry of another function might be placed in the same 16-bit word). Again, the Software Performance Analyzer needs unique addresses for each function entry and function exit.

If you cannot add NOP instructions between two functions, you can still use the Software Performance Analyzer. Use the HPSPAADJUST feature to offset the points where the Software Performance Analyzer recognizes function-start and function-end addresses. HPSPAADJUST offsets the recognition addresses to points within the functions instead of the actual function-start and function-end addresses. The HPSPAADJUST option must be exported at the shell. You will need to redefine your events after using the HPSPAADJUST feature in order to record new event-start and event-end addresses in the events list. Refer to the information that describes steps you can take to correct for unused prefetch in Chapter 7, or in the online help screen of the Software Performance Analyzer.

If the content of the Time and Time% columns do not total 100%

Sometimes the content of the columns total more than 100% and sometimes they total less than 100%. The Time and Time% columns typically total 100% in the following measurements:

- Program Activity
- Memory Activity
- Function Duration excluding all calls
- Function Duration excluding profiled calls

The Time and Time% columns typically total more than (or less than) 100% in the following measurements:

- Function Duration including all calls
- Interval Duration

In activity measurements, the sampling process may cause the percents to fluctuate for a period of time during the measurement. Therefore, the percents will not always total 100%.

In duration measurements, if an event rate overflow or underflow condition occurs, the column contents may not total 100%.

The percents displayed in the Time% column show the percent of execution time used by each individual function compared to the profiled time (not compared to the execution time of all the other functions). In a duration measurement including calls, the time recorded for a function will include the time spent executing functions that it called. If these called functions are also displayed, their times will also be shown. This means the execution time of a function will be added to its event name and also to the event names of any functions that called it. Because of this, the summation of Time and Time% of functions can easily exceed 100 percent.

As a simple example, consider the case of a looping program that calls three identical functions: afunc, bfunc, and cfunc. If each function is called serially, you might expect to see 33% of the total time spent in each of the three functions, and that is what the Software Performance Analyzer will display.

Now consider the case of a looping program that calls only afunc, and afunc (in turn) calls only bfunc, and bfunc (in turn) calls only cfunc. The call sequence might look like this:

```
afunc-start
   bfunc-start
      cfunc-start
      cfunc-end
   bfunc-end
afunc-end
afunc-start
   bfunc-start
      cfunc_start
```

In the above example, afunc including calls could be using 99% of the profiled time, bfunc including calls could be using 66% of the profiled time, and cfunc could be using 33% of the profiled time. This is what the Software Performance Analyzer will display.

If you were to define intervals for each of the three functions above, the interval duration measurement would yield the same results.

If you were to switch to the excluding calls measurement mode, you would again see afunc, bfunc, and cfunc each using only 33% of the profiled time.

In very small programs, a recursive function can show a high Time% because each recursive call is added to the time of the called function as well as the times of all of the calling functions in the recursive chain. The summation of times in the recursive chain can be greater than 100% for a recursive function. This is typically true after you have identified the function as recursive in the Events list.

For example, consider the measurement of a single recursive function (afunc) that recursively calls itself 3 times. If the function is not declared recursive, the recursive nature of the function may be lost due to prefetch correction, and the Time % may indicate 99% (the number of calls will be displayed as 1/3 of the actual number of calls).

If the function is declared recursive, the Time % may indicate 198%, and will be the result of the summation of 99% for the first call of afunc, 66% for the second call of afunc, and 33% for the third call of afunc.

If the trigger (trig2) does not seem to work correctly

The trig2 trigger is generated by the Software Performance Analyzer after the specified time of the trigger event has been exceeded. The trigger timer (for trig2) starts on the first occurrence of the start address of the interval or function to be measured. The timer continues to count until the end address is found.

If the sequence A-start1, A-start2, A-start3, A-end1, A-end2, A-end3 occurs in your executable file, and you have specified: "setup_measurement drive trig2_after 100 msec A" the trigger timer will measure the time from A-start1 to A-end1. If the specified time of the interval is exceeded (100 msec in the command above), the trigger will be generated at the moment the time is exceeded.

The trigger timer does not correct for prefetch or recursion, does not exclude calls to other functions, and does not exclude time spent servicing interrupts. The only time period that the trigger timer will exclude from its count is time that the Software Performance Analyzer is disabled, if any. In other words, the trigger timer will only be frozen between a disable address and the next enable address. The disable/enable feature can be used to exclude interrupts from a trigger event, provided that the enable and disable are properly defined. For example, "setup_measurement disable start_address interrupt", and "setup_measurement enable end_address interrupt".

The interval duration measurement and the trigger timer will record the same time periods, except in the case of an event rate overflow.

A measurement of function duration including calls will typically record approximately the same time period as the trigger timer, except that function duration measurements are corrected for prefetch.

A measurement of function duration excluding calls will always record a different time from the trigger timer because the trigger timer does not exclude calls to other routines.

If the "XSigServe" process continues to run after you exit the Graphical User Interface

☐ This is a normal condition. The "XSigServe" process translates kill characters such as ctrl-c and ctrl-\ to kill signals that are sent to the interface. It continues to run for each emulation session. The "XSigServe" process will die when you log out. It does not use system cycles unless the graphical user interface is running. You can kill the "XSigServe" process manually, if desired.

If the drag-thru menu selection is too slow when using the Graphical User Interface

☐ Use "single click" menu selection, as follows:

1 Move to the menu bar item you want to select and click the mouse button (instead of click-hold-drag). This leaves the menu "tacked open".

2 Now move the mouse pointer to the next menu selection and click again.

Use this method for Sun OpenWindows environment. The low resolution of the Sun optical mouse may make "drag-thru" selections inaccurate.

If the help screens cover the display window when using the Graphical User Interface

This is a normal condition. When called, the help screens first appear on top of the display window where they were called. You can move the help selection box and the help text browser window to unused areas of your display using the normal window move function. Once you have moved these display windows, they will stay in the new locations throughout the session.

Part 3

Measurement Concepts

Part 3

This part of the manual contains the following chapters:

Chapter 7. Software performance measurement techniques and difficulties

Chapter 8. How good are your test results?

Refer to the HP manual titled, "Concepts of Emulation and Analysis" for a greater understanding of general emulation and analysis concepts

7

Software Performance Measurement Techniques and Difficulties

This chapter provides an understanding of what the Software Performance Analyzer is and how it performs its work, along with explanations of problems it is

designed to overcome. The following information is covered in detail in this chapter:

- What the Software Performance Analyzer does.
- Kinds of problems that can be solved using performance analysis.
- How to prepare your program for Software Performance Analysis.
- How the Software Performance Analyzer decides whether to include events from your events list in a measurement.
- How the Software Performance Analyzer determines whether your event is a function or a variable.
- How the Software Performance Analyzer makes activity measurements.
- How to define additional status types for your emulator.
- Effects of the emulation monitor on activity measurements.
- Using delay in activity measurements.
- Effects of reset on activity measurements.
- How the Software Performance Analyzer makes duration measurements.
- How function-duration measurements use an internal stack
- Comparing measurements of time, calls, and cycles.
- Using expanded time ranges.
- Generating triggers during measurements.
- Effects of reset on duration measurements.
- Effects of the emulation monitor on duration measurements.
- Using delay in duration measurements.
- Using disable/enable pairs in duration measurements.
- How a cache can affect performance measurement results.
- How unused prefetches can affect performance measurement results.
- How the Software Performance Analyzer measures recursive functions.

- Using HPSPAADJUST to overcome problems caused by prefetch.
- Using markers to overcome problems caused by an enabled cache and/or prefetch
- Overcomming measurement difficulties that are unique to Intel 80960 Sx.
- Overcoming difficulties measuring processors that manage memory.
- Overcoming the effects of multi-byte return instructions.
- Analyzing software performance in assembly language files.

What does the Software Performance Analyzer do?

The Software Performance Analyzer records information about the execution of events. Events represent addresses in the absolute file. Events can be defined to represent functions or static variables. You can also define events to represent broad ranges of address space, and you can define events to represent intervals between two addresses.

Two kinds of measurements are made by the Software Performance Analyzer: activity measurements, and duration measurements. Both measurement types record time spent executing selected events.

The Software Performance Analyzer can show you a histogram or table that lists all of the events that were included in the measurement, along with all of the recorded data for each event. This way, you can compare the performance of each of the selected events.

The Software Performance Analyzer can give greater details about the execution of each event when making duration measurements. This is done by adding a set of time ranges to an event. Executions of the event are then mapped into these time ranges. These time ranges show whether or not executions of the event are completed within the same range of time, or whether there are great differences between the time required to complete one execution of an event and the time required to complete another execution of the same event.

The process of Software Performance Analysis

The Software Performance Analyzer makes its measurements on an executable file running in emulation. The Software Performance Analyzer works through the emulator. The symbols in the emulator symbol data base are used by the Software Performance Analyzer when it defines events to represent functions and static variables in the executable file. If you define an event to represent a function, static variable, range, or interval, the Software Performance Analyzer will check your definition against the emulator symbol data base to make sure the symbol name and symbol type in your definition agree with the symbols data base. Events that are entered as hexadecimal values will be accepted without checking against the emulator symbol data base, but the Software Performance Analyzer will check to make sure your definitions do not overlap addresses already represented by events in the events list.

What kinds of problems can be solved by using the Software Performance Analyzer?

The Software Performance Analyzer can quickly identify slow-running functions in programs that have many functions. It can also compare relative efficiencies of portions of your program with one another.

The Software Performance Analyzer helps a designer understand the execution of software. Software Performance Analyzer measurements may be taken when a designer needs to answer the question, "Why does it take so long to execute my program?" "Which function or functions are taking extra-long times to execute?" Once the designer identifies the functions that are slowing down the system, the designer can then analyze those functions to correct problems so that they run faster.

The Software Performance Analyzer will often be used by someone who recognizes that there is a problem in the software that causes it to take too long to execute, but the program is huge. The Software Performance Analyzer shows you which routines (or even which libraries) are slowing execution of the program. The Software Performance Analyzer helps identify the functions and libraries that should be optimized.

What kinds of problems can be solved by using the Software Performance Analyzer?

Another user of the Software Performance Analyzer may be someone who has written a block of code with several functions and simply wants to see how efficiently each of the functions runs by comparing each function against all the other functions. The Software Performance Analyzer will show side-by-side comparisons of up to 84 functions, indicating max, min, and mean execution rates.

The table and histogram displays show the functions in your software, along with the amount of time spent executing each one of them. You can sort the list of functions to place those that took the most time at the top of the list. With this information, you can quickly find the most time-consuming functions in your source files.

The following typical measurement quickly identifies a function that has a problem.

1 A measurement of function durations was made with: **Profile→Profile...**, and Function Duration Excluding Profiled Calls was selected in the dialog box.

2 The measurement was stopped when the desired information stability was obtained: **Profile→Stop Profile**

3 The events were sorted by time: **Display→Sort Events→Time**

4 The following histogram was obtained. The "apply_productions" function is the one to investigate to see why it is taking so much system time.

```
Histogram: Function Duration exclude profiled Run Time: 1:11      Stability:100%
_Name_(sort:_time)_____|__Time__|__%___0%_____20%_____40%_____60%_____80%___100%
   2 apply_productions |  36.8 s| 51.49|********************
  31 stack_library     |  10.4 s| 14.59|******
  27 scan_string       |   5.0 s|  7.05|***
  19 math_library      |   3.3 s|  4.64|**
  20 move_byte         |   3.1 s|  4.39|**
  23 report_errors     |   3.1 s|  4.38|**
  21 outputline        |   1.9 s|  2.60|*
   5 clear_buffer      |   1.7 s|  2.41|*
  26 scan_number       |   1.4 s|  1.95|*
  15 input_line        |   1.3 s|  1.81|*
  16 lookup_token      |   1.2 s|  1.66|*
  33 syntax_check      | 681.2ms|  0.95|
  24 report_result     | 581.3ms|  0.81|
  28 semantic_check    | 229.2ms|  0.32|
__12_get_next_token____|_174.7ms|__0.24|_____________________________________
 Profiled Absolute     |  71.5 s| 100% 0%     20%     40%     60%     80%   100%
```

Preparing your program for Software Performance Analysis

There are a variety of compiler issues that must be addressed to ensure that a program is suitable for analysis by the Software Performance Analyzer. The following list is a brief synopsis of the some of those issues. Refer to the Measurement Problems chapter in this manual for additional information about compiler issues that affect performance measurements.

- Symbols must be available to the emulation system. Compile your programs in a manner that allows symbols to be available to the emulation system. In particular, the Software Performance Analyzer will need to use compiler-level function and static variable symbols. Within the symbol database, the filenames that compose your program should be preserved as full path filenames. Some compilers only record the base filename and not the full path to the file unless an option is specified. In those compilers, be sure to specify that option.

- The Software Performance Analyzer assumes that the functions have a single entry point and a single exit point. In addition, the entry is assumed to be the start of the address range of the function and the exit is assumed to be the end of the address range of the function. If you have a compiler option that guarantees that the function will only have one exit point, then use that option. Also, assembly functions that you want to measure should be set up this way. Refer to the section entitled "Analyzing software performance in assembly language files" in this chapter for additional information.

- Overcome multi-byte return instructions. If you have an option in your compiler that allows you to use single-byte instead of multiple-byte return instructions, as is available with some Intel and Intel-like microprocessors, it is best to use the single-byte return. If you elect to use a multiple-byte return instruction, then you can use the HPSPAADJUST shell variable to correct for possible overlap problems that may occur. If you are using both single-byte and multiple-byte return instructions in functions that you want to include within a performance analysis, you will need to use markers to make the desired measurements. Refer to the section titled "Overcoming the effects of multi-byte return instructions" later in this chapter.

- Overcome function address overlap. If possible, direct your compiler to pad a few NOP's between functions. If you can pad a few NOP's (3-NOP's is ideal) between functions, you can ignore the effects of prefetching and function address overlaps. If you can at least long align each function, you can avoid the function address overlap problem. Function overlap occurs when the return instruction of one function is packed in the same long word with the start instruction of the following function. This problem typically occurs when using a Motorola 68020, 68030, or 68040 microprocessor. The HPSPAADJUST shell variable, which allows you to adjust the start and end addresses of functions, will correct for this problem.

- Use markers. If you are using a Motorola 68040 or Intel 80960 microprocessor, you must use markers to make function-duration measurements. The HP marker preprocessor can simplify the task of adding markers in your files if you are using an HP AxLS or MRI compiler. Refer to the man pages cc68040mt(1), mcc68kmt(1), or mcc960mt(1) for complete details. Also, refer to the discussion about markers in this chapter.

- Overcome effects of caches. If you want to make performance measurements with the instruction and data caches turned on, you will have to compile your files using markers. You must also allow the data to write through the cache so that the markers will be available to the Software Performance Analyzer. Refer to the discussion about markers and setting the HPSPAMARKERS shell variable to tell the Software Performance Analyzer that you are using markers.

How the Software Performance Analyzer picks events to include in a measurement

To be included in a measurement, an event must meet two requirements:

- It must be selected (have *, ?, or r beside its name in the events list).
- It must be the type(s) appropriate for the measurement.

Measurement Type	Appropriate Event Type
Program Activity	functions and ranges
Memory and IO Activity	static variables and ranges
Interval Duration	intervals

How the Software Performance Analyzer determines whether your event is a function or a variable

When you enter a definition for an event to represent a function or static variable, the Software Performance Analyzer accesses the symbols data base for the executable file running in emulation and checks the event name and event type against the content of the symbols data base. Your definition is accepted if the symbol name and type match a symbol in the data base. If not, the Software Performance Analyzer will question your definition. This safeguard helps ensure the accuracy of your measurement results by making sure they are made on valid events.

How the Software Performance Analyzer makes activity measurements

Activity measurements are normally the first measurement to use when investigating a problem because they can collect data about a large number of events. Activity measurements are overview-type measurements. They are designed to give you a broad understanding of the time and memory bus cycles required by various segments of your program. They identify the areas of your program that use the most processor bus cycles and take the most time to execute.

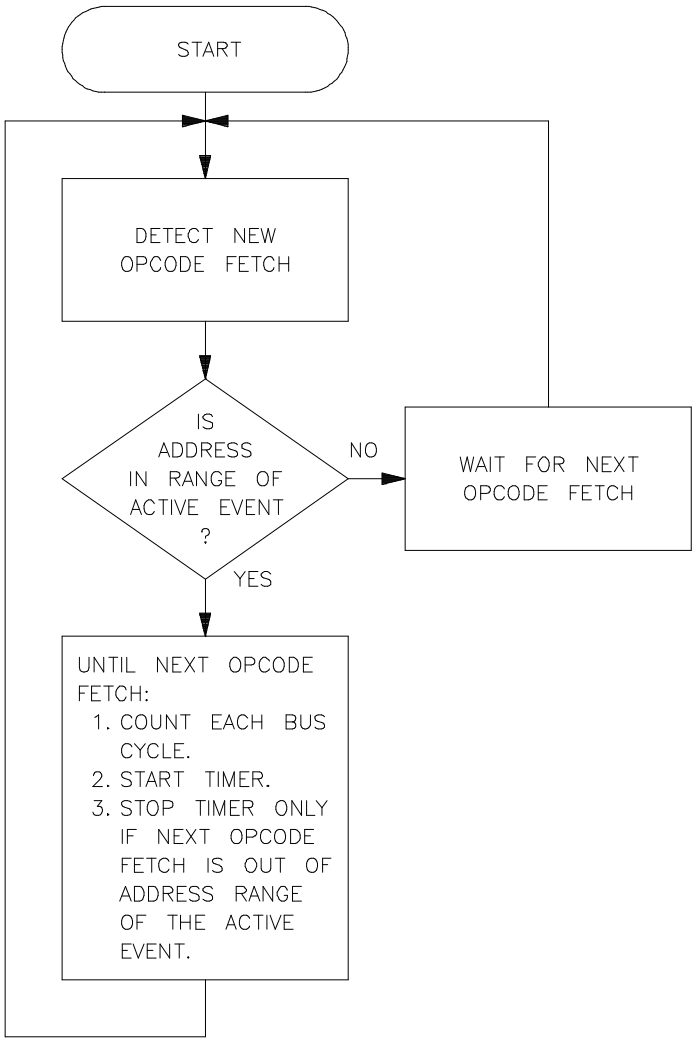
Activity measurements are made by sampling. The sampling technique concentrates the resources of the Software Performance Analyzer on one event at a time, for about 2.5 milliseconds per sample. During one sample period, the Software Performance Analyzer counts each bus cycle that fits within the address range of the active event, and is of the specified type. In addition, a timer is started at the beginning of each bus cycle. The timer records time until the bus cycle exits or an appropriate termination condition is found.

The Software Performance Analyzer makes two types of activity measurements: program_activity, and memory_and_io_activity. The following paragraphs discuss the two types of activity measurements in detail. Additional details and examples of activity measurements are given in the online help screens of the Software Performance Analyzer; choose **Help→General Topic ...**, and from the Help Index, select **Duration and Activity Distinctions.**

Program activity

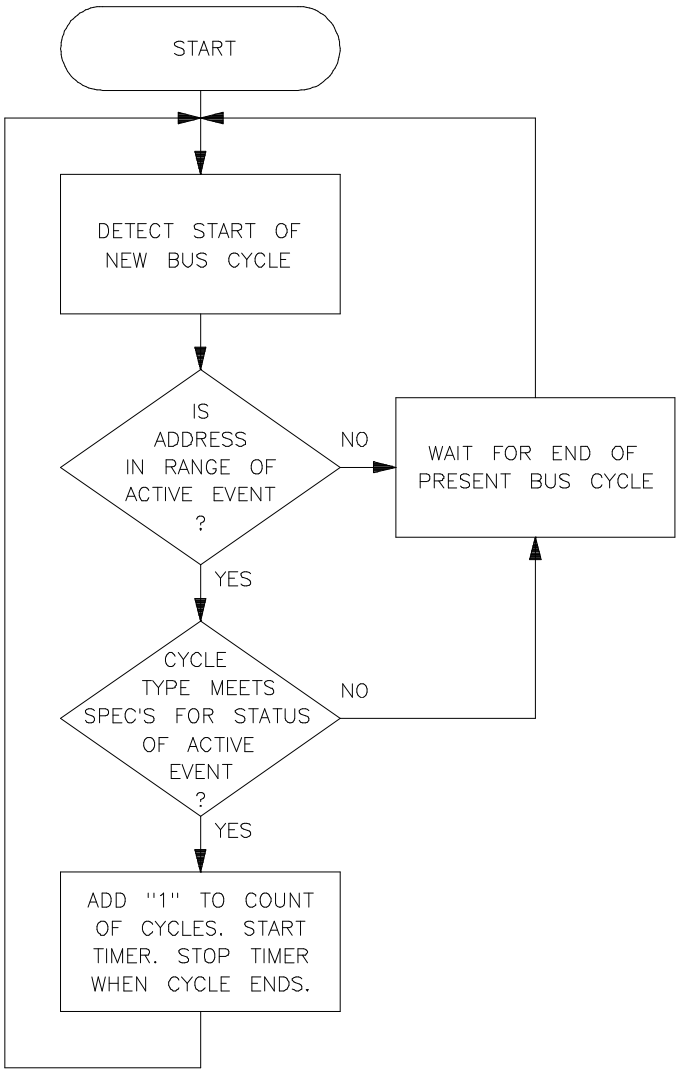
This measurement records instruction execution within ranges of addresses. Events that represent functions and ranges can be included in this measurement. Events that represent variables and I/O addresses, or events that represent intervals, cannot be included in this measurement.

When measuring program activity (see diagram next page), the Software Performance Analyzer checks each opcode fetch to see if it is within the address range of the present active event. If an opcode is fetched from the address range of the active event, its event count is incremented, and a timer is started. All reads, writes, etc., are recorded as part of the active event until the next opcode is fetched. If the next opcode is fetched from the same range of addresses, the count for the active event continues to be incremented. If the next opcode is fetched from a different address range, the timer and counter for the active event are turned off.


START
DETECT NEW OPCODE FETCH
IS ADDRESS IN RANGE OF ACTIVE EVENT ?
NO
WAIT FOR NEXT OPCODE FETCH
YES
UNTIL NEXT OPCODE FETCH:
1. COUNT EACH BUS CYCLE.
2. START TIMER.
3. STOP TIMER ONLY IF NEXT OPCODE FETCH IS OUT OF ADDRESS RANGE OF THE ACTIVE EVENT.

Memory_and_io Activity

When measuring memory and I/O activity (see diagram below), the Software Performance Analyzer records the number of read, write, or other processor-specific bus cycles (as specified by the user), and the amount of time required to execute each of them. Events that represent functions or intervals are not included in this measurement.



Example of an activity measurement

This example assumes the following two events have been defined:

Event1 = the address range of function1.
Event2 = the address range of an array of 80 integers.

Assume your first measurement is a program activity measurement that includes event1. During the first sampling period for event1, function1, is executing. The Software Performance Analyzer might record 2453 cycles and 68.0 usec of time.

The next time the Software Performance Analyzer samples activity for event1, function1 might not be active. In this case, the Software Performance Analyzer will record 0 cycles and 0 time. After enough sample periods have been given to event1, the Software Performance Analyzer activity measurement will record an average of how much time function1 is executing in relation to all other events.

The power of program activity measurements is that they allow you to define events that cover large segments of memory. You can define an event for each of the libraries of a program; the Software Performance Analyzer can quickly determine which library is using the most processor resources. With this information, you can define events to represent each of the functions in the slow library and make a new profile measurement to find out which function is taking too much time.

Assume your next measurement is a memory and I/O activity measurement that includes event2. During the first period that the Software Performance Analyzer samples activity for event2, it records 20 writes. The Software Performance Analyzer would show 20 cycles, and perhaps 10 usec of time (20 x 500 nsec) for event2. Each sampling period for event2 will probably record a different time, but after enough samples have been taken, the Software Performance Analyzer will be able to show how much time, on average, is spent executing event2 during any given second of program execution.

Effects of the emulation monitor on activity measurements

If the microprocessor is running the emulation monitor program, a variety of different results may be obtained, depending on the type of monitor program (foreground or background) and the type of measurement (program or memory activity). To obtain the most accurate activity measurement, avoid using the emulation monitor, or use it sparingly because some monitor activity might be recorded as event activity during a measurement.

Using delay in activity measurements

You can delay the start of an activity measurement by setting up the Software Performance Analyzer to wait for a trigger from the emulation bus analyzer or to wait for its own enable specification to be satisfied. If you specify both a trigger and an enable, your measurement will not start until first the trigger is received and second the enable specification is satisfied, in that order.

Effects of reset on activity measurements

Activity measurements are made with the microprocessor running your target program. If the microprocessor is reset, the activity measurement will be suspended. If you start the microprocessor from reset, the Software Performance Analyzer will not be able to recognize an enable condition (if you set up an enable specification) during the first 5 milliseconds after the microprocessor begins running from emulation reset.

Defining additional status types for your emulator

The most common status conditions that will be seen by the Software Performance Analyzer are defined in a file that is supplied with your analyzer software. If you want to define additional status conditions, you can edit the status64708A file associated with your emulator and add the desired status. The status64708A file contains instructions that show you how to add status entries.

Status files (status64708A) are located in directories named $HP64000/inst/emul/<product_number>/etc.
Where $HP64000 is a shell variable defining the location of the hp64000 directory, and <product_number> is in the form 64742A, 64747A, ...

Be careful when adding status entries to the file. Do not change the existing status entries or the Software Performance Analyzer will not operate properly.

How the Software Performance Analyzer makes duration measurements

Duration measurements are real-time, non-sampled measurements. They continuously capture information about all of the events selected in the measurement (unlike activity measurements which sample information for only one event at a time). The Software Performance Analyzer records the number of calls, execution duration, and maximum and minimum execution times for each event. Two types of duration measurements can be made by the Software Performance Analyzer: interval duration, and function duration.

During a duration measurement, the Software Performance Analyzer waits until it sees the entry address of one of the selected events. It measures elapsed time until it sees the exit address of the same event. At the end of the measurement, the Software Performance Analyzer will show you how much time was spent executing each of the selected events.

The following paragraphs discuss the two types of duration measurements. Additional details and examples of duration measurements are given in the help screens of the Software Performance Analyzer; choose **Help→General Topic ...**, and from the Help Index, select **Duration and Activity Distinctions.**

Interval duration

Interval events have a start address and an end address. When the Software Performance Analyzer makes an interval-duration measurement, it records the time between the interval-start address and the interval-end address. These addresses can be segments of an executable program, such as the start of function1, and the end of function4. Intervals can be defined to have start addresses that are higher or lower than their end addresses in the range of program code.

Intervals can also be defined using memory and I/O addresses. Interval-start and interval-end addresses can be further qualified by including processor status in your profile command. You can even define an interval that has the same address for both the interval-start and interval-end addresses. This is useful if you want to measure durations between writes to a selected variable.

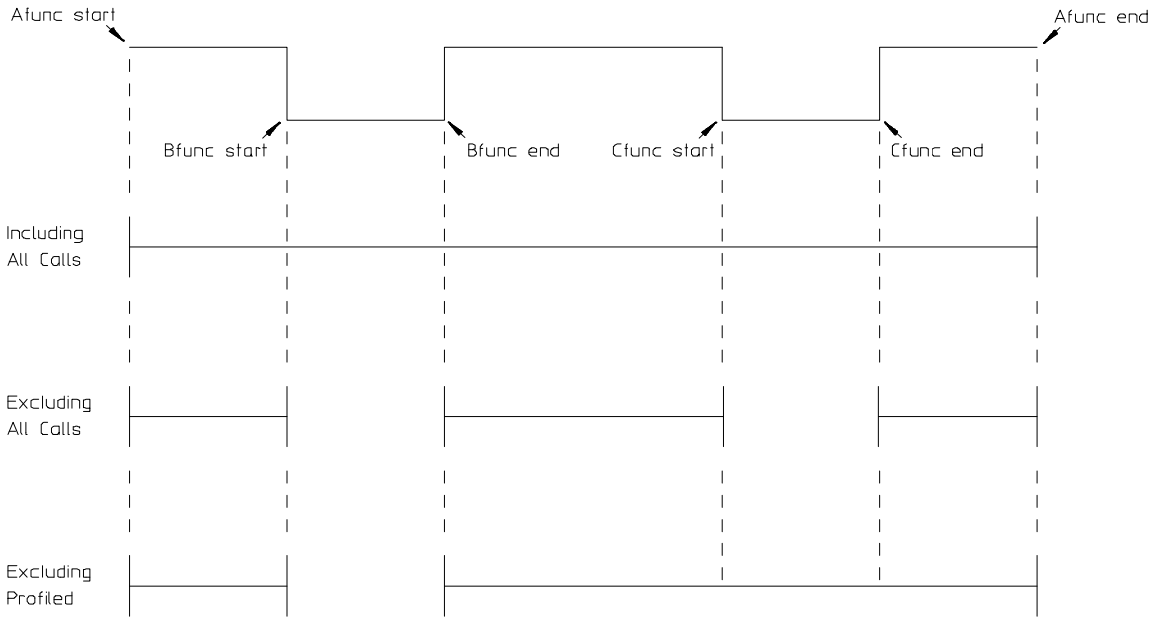
Note that interval_duration measurements are not corrected for unused prefetches. Refer to interval_duration measurements without prefetch correction later in this chapter.

Function duration

Function events represent individual functions in the source file. When the Software Performance Analyzer makes a function-duration measurement, it measures the execution time of selected source-file functions. In function-duration measurements, you can have the Software Performance Analyzer:

- include all of the time spent executing code of other functions called by the present function, and include all time spent servicing interrupts.
- exclude all time spent executing code of other functions called by the present function, and exclude all time spent servicing interrupts.
- exclude all time spent executing code of other functions called by the present function if that code is represented by an event selected in the measurement, but include execution of code of other functions if that code is not represented by any event selected in the measurement.

The illustration below shows a function-duration measurement of Afunc in each of the three function-duration measurement modes. In the diagram, Afunc calls Bfunc. When execution of Bfunc is complete, Afunc calls Cfunc. Afunc and Bfunc are both being profiled in the measurement, but Cfunc is not.



How function-duration measurements use an internal stack

The Software Performance Analyzer uses an internal stack to make function-duration measurements. This stack stores function-entry points to be matched with function-exit points. Proper calculations of function durations depend on the validity of this internal stack.

If multiple entry points occur or if exit points do not occur for entry points already on the stack, the internal stack of the Software Performance Analyzer can yield function durations that are inaccurate or even missing.

Consider the case where Afunc calls Bfunc which in turn calls Cfunc. If execution is in the middle of Cfunc, the stack will contain Afunc.entry, Bfunc.entry, and Cfunc.entry. When Cfunc.exit occurs, it will be matched with Cfunc.entry and the duration of Cfunc will be calculated. Then Cfunc.entry will be removed from the stack. This is the normal operation of the stack. As each exit point is found, the corresponding entry point is found on the stack and the duration of the function is calculated and accumulated.

Now consider the case described above (Afunc.entry, Bfunc.entry, and Cfunc.entry are all on the stack), but Bfunc.exit and Cfunc.exit are not properly set up. The Software Performance Analyzer finds Afunc.exit. It matches Afunc.exit with Afunc.entry, yielding an appropriate time for Afunc, and then it purges Bfunc.entry and Cfunc.entry from the stack.

If none of the exit points are found, the stack will grow until it overflows. The internal stack is set at a size limit of 1000 deep. If two functions, Afunc and Bfunc, are set up such that the exit points are never found, the stack will grow because Afunc.entry and Bfunc.entry will always be added but never removed from the stack.

Finally, consider the case of a recursive function, Dfunc, that has not been declared recursive. Because the function is not treated as a recursive function, the multiple entry points are ignored; only the first Dfunc.entry point is saved. Therefore, the time associated with the total number of recursive calls to function Dfunc may be summed into one call to function Dfunc. The extra Dfunc exits may be purged by the hardware (because they are treated as prefetches), or they may be matched with other Dfunc.entry points much earlier on the stack, creating erroroneous time measurements.

To summarize, if the Software Performance Analyzer seems to yield strange results, there are two approaches you might try to solve the problem:

- Use the emulation-bus analyzer to trace a few of the function-entry and function-exit points to verify that the functions are behaving in a stackable manner. Remember that prefetch can cause strange problems; be sure you are compiling in such a way to avoid prefetch.
- Use marker technology (discussed in this chapter); this will avoid the prefetch and compiling restrictions altogether.

Comparing measurements of time, calls, and cycles

Activity measurements record cycles and time. Cycles is a count of the bus cycles that were executed within the address range of the event.

Duration measurements record calls and time. Calls is a count of calls to the function or interval identified as an event.

Time is the length of time spent executing within the defined event.

The "cycles" information might show you that a particular module is running slowly (a small number of bus cycles executed for the amount of time used). This condition could identify an inefficient routine, for example, one that fetches instructions or data from slow memory when the information is already available in a cache or faster memory.

The "calls" information might show you that a particular routine is called an unusual number of times. For example, you might find that a print routine is being called to perform a print function one thousand times for each print that is needed.

The "time" information might show an event that normally completes in a few milliseconds, but on rare occasions it takes several hundred milliseconds to complete. You can trigger a trace in the emulation-bus analyzer if this event exceeds normal execution time and find out why these rare executions take so much time to execute.

EXPANDED time ranges

Expanded time ranges are available in duration measurements to let you look at the execution of an event in greater detail. If you expand an event in a histogram or table display during a measurement, each new time recorded for the event (whether in an interval-duration or function-duration measurement) will be shown beside the appropriate time range, as well as added to the total amount of time recorded for the event.

Trigger generation

When making duration measurements, the Software Performance Analyzer can generate a trigger. The trigger can be used to cause the emulator to break to its monitor program. It can also be used to start a measurement in an associated state, timing, or emulation bus analyzer. The trigger is supplied on trig2. The example command below sets the analyzer to generate a trigger if the event named "move_byte" runs continuously for at least 2 milliseconds:

setup measurement drive trig2_after 2 msec move_byte <RETURN>

Note that the trigger command always calculates its interval as an include_calls time (even if the present measurement is function_duration exclude_calls). In addition, the trigger command is not corrected for prefetch conditions. Prefetch correction is discussed in detail at the end of this chapter.

Effects of reset on duration measurements

Avoid resetting the target microprocessor when making a duration measurement. Duration measurements should be made with the microprocessor running your target program. If the target microprocessor is reset for more than 1.0 second, the current interval or function being measured will be deleted and the measurement will be suspended. When the target microprocessor restarts, the measurement will continue. All durations of functions or intervals occurring during the reset time will be lost.

Effects of emulation monitor on duration measurements

Avoid using the emulation monitor program, or using emulator features that require the monitor program (such as, **display memory**) when making a function-duration or interval-duration measurement. If the emulator switches from the target program to its monitor program while measuring the duration of an event, the time spent running in the monitor will be added to the time recorded for the duration of the event. If you are making a duration measurement, excluding calls, and the emulator switches to its monitor program, other error conditions can occur. If you specify generation of trig2 after a duration of some event, and if the emulator runs in its monitor program while that event is being measured, trig2 will be generated in error.

Using delay in duration measurements

You can delay the start of a duration measurement by setting up the Software Performance Analyzer to wait for a trigger from the emulation bus analyzer or by specifying an enable condition to be recognized in the Software Performance Analyzer. If you specify both a trigger and an enable condition, the measurement will not start until first the trigger is received and second the enable condition is found, in that order. The initial time (before the Software Performance Analyzer is enabled) is not recorded.

Using disable/enable pairs in duration measurements

If you specify a disable condition and an enable condition, any time spent disabled during a measurement (between finding the disable condition and finding the next enable condition) will be shown in the histogram or table beside "Disable time". The initial disable time (before the first enable is found) is not recorded. Note that

"Disable time" only appears in tables and histograms when an enable/disable pair is specified.

How a cache can affect Software Performance Analyzer measurement results

Many microprocessors use a cache to store recently used instructions. By keeping recently used instructions in a cache, the microprocessor can access these items if they are used again without having to initiate external bus cycles. When the microprocessor fetches an instruction, it checks to see if that instruction is already in its cache, and if it is, it fetches the instruction from the cache and does not perform any external bus cycles.

When the microprocessor is operating with its cache enabled, software performance analysis is limited because the analyzer can only recognize activity occurring on the external buses, and no bus cycles are performed to fetch instructions that reside in the cache. Therefore, transactions involving the cache will either be incomplete or in error. For example, if you are measuring interval duration, the starting address of the interval may be placed in the cache the first time it is called. After that, each time the interval is executed again, its starting address may be obtained from the cache; the software performance analyzer will not be able to measure any further executions of the interval.

Usually, you will want to disable the cache of the microprocessor when performing software analysis of a program. In this way, all of the instructions executed by the microprocessor will be fetched on the processor memory bus where they can be recognized by the software performance analyzer. There is a way to measure function durations and interval durations with the Software Performance Analyzer when the cache is enabled; refer to the discussion on markers, later in this chapter. Using markers is required when measuring function durations and interval durations in an Intel 80960 Sx microprocessor because its cache is always enabled.

How unused prefetches affect measurements of the Software Performance Analyzer

How unused prefetches affect activity measurements

Activity measurements will be affected if an address in the range of the active event appears in an unused prefetch. This will typically add a small amount to the count of functions when they are not actually executing. This has very little effect on the overall accuracy of activity measurements. If you insert NOP instructions in your source file to separate your functions from each other, activity measurements for the functions will be completely accurate (slightly more correct than if you did not insert the padding). If you insert NOP instructions between functions, you may see slightly more activity recorded in the event called Undefined Addresses. This additional activity is the prefetching of the NOP addresses.

How unused prefetches affect duration measurements

Unused prefetches can have a large effect on duration measurements. The Software Performance Analyzer would have an easy time measuring the durations of functions or intervals if it weren't for prefetching (and recursion, discussed later). It would simply start its clock when it saw the first address and stop its clock when it saw the last address of a function or interval. Then it would record the time on the clock.

Software Performance Analyzer measurements are more complicated when code is executed by a processor that prefetches instructions and uses an instruction queue. Consider the case when your processor prefetches through the end of one function into the start address of the function you want to measure. You don't want the clock to start running from the occurrence of this unused prefetch.

Prefetch correction is designed into function-duration measurements (but not interval-duration measurements) to remove this unused prefetch. Some types of prefetches can be corrected by the circuitry and some types of prefetches are removed by processes in the software of the Software Performance Analyzer. If excessive numbers of prefetches occur that must be removed by the software, you may see a message on the status line warning about stack overflow.

No prefetch correction circuitry or algorithm is foolproof. You must examine all measurement results looking for errors that might be caused by prefetching in order to properly interpret the results.

Prefetch correction designed into function-duration measurements

In function-duration measurements, prefetch correction circuitry is turned on. This circuitry removes two of the following three types of unused prefetches. The performance analyzer software removes most of the third type of unused prefetches:

```
   A
   |     Function A loops back into A but prefetches through Aend
 ^ |     and non-measured memory.  Hardware prefetch correction
 | |     will remove the prefetched Aends.
  -Aend
Non-measured memory
```

```
   A
   |     Function A loops back into A but prefetches through Aend
 ^ |     and Bstart.  Hardware prefetch correction will remove
 | |     the prefetched Aends and Bstarts.
  -Aend
   Bstart
```

```
Non-measured function
   |     A non-measured Function loops back into itself but
 ^ |     prefetches through Astart.  Hardware prefetch correction
 | |     will NOT remove the prefetched Astarts.  A software
 | |     process of the Software Performance Analyzer will
 | |     normally remove most occurrences of this prefetch.
  -Non-measured end
   Astart
```

This provides proper prefetch correction for most functions, except recursive functions (discussed later).

Interval-duration measurements without prefetch correction

In interval-duration measurements, the prefetch correction circuitry used for function-duration measurements is turned off. If you are measuring the duration between two address points, an unused prefetch of an address point may start or end the interval measurement. Interval-duration measurements record the time between the first occurrence of the interval-start address and the first occurrence of the interval-end address. For example, suppose your interval-duration measurement records the time between address A and address B. Then consider the following sequence of addresses: A,A,A,A,B,B,B. The interval-duration measurement will measure from the first occurrence of A through the first occurrence of B. All other occurrences of A and B will be ignored. If the first

occurrence of A was the result of an unused prefetch made at the end of some other function, the recorded interval will be from the prefetch of A to the first occurrence of B. Inserting NOP instructions ahead of A and B in the source file should correct most prefetch problems. Another way to work around this problem is to use the HPSPAADJUST feature described later in this chapter. A final way to treat the problem of prefetching may be to qualify the interval duration addresses with a status of write (useful if you are trying to measure the time between writes to two addresses). Memory locations where data is stored are not prefetched.

How the Software Performance Analyzer measures recursive functions

If you do not identify the recursive function correctly

If a function is recursive to itself and you do not designate it as recursive in the events list, prefetch correction will typically remove all calls but the first one, and all exits but the last one. Therefore, time will be measured from the start of the recursive function to the last exit of the recursive chain.

```
A-start<-time.start                A-end<-time.end
    ↓                                 ↑
    A-start                        A-end
        ↓                           ↑
        A-start                 A-end
```

The count of calls will also be incorrect. The Software Performance Analyzer will only record one call to the recursive function instead of the true number of recursive calls.

If you identify the recursive function correctly

If you identify the recursive function as recursive in the events list, the prefetch correction circuitry will be turned off. The Software Performance Analyzer will measure the recursive function correctly, if the entry and exit addresses of the recursive function do not appear as unused prefetches.

The following example shows how the Software Performance Analyzer will measure the times from start to end of each recursive call. Each recursive call will be recorded, individually. (Note: when computing execution duration of the function, the values of time1.start through time1.end, time2.start through time2.end, and time3.start through time3.end will be added to the measurement report.)

```
A-start<-time1.start                A-end<-time1.end
    A-start<-time2.start         A-end<-time2.end
        A-start<-time3.start  A-end<-time3.end
```

If a recursive function is identified as recursive in the events list, and if its end address appears as an unused prefetch, the Software Performance Analyzer may report misleading time measurements.

```
A-start<-time1.start                A-end
    A-start<-time2.start         A-end
        A-start<-time3.start  A-end<-time1.end
                              A-end-prefetch<-time2.end
                              A-end-prefetch<-time3.end
```

This type of prefetch error can occur if you have a loop near the exit of function A. By manually inserting NOP instructions in your source file ahead of the exit addresses that follow branch instructions, you can prevent the above problem.

Steps you can take to correct for unused prefetches

Adding NOP instructions between functions.

One way to limit the effects of unused prefetches is to insert NOP instructions between the functions in your source file (before function entry addresses and after function exit addresses). Hewlett-Packard C compilers will automatically insert NOP instructions between the functions when you use the "debug" (**-OG**) option of the compiler. The NOP adresses are prefetched at the end of a function instead of prefetching the entry address of the next function. This improves the accuracy of your measurement results. You can also change your specification of function-start and function-end addresses by specifying addresses that are further inside the function (see the instructions for using HPSPAADJUST next in this chapter).

A second common prefetch correction is to make sure there is sufficient NOP padding between the exit of a function and any loop construct that precedes the exit within the function. Looping near the exit of a function may prefetch the exit address of the function several times. Again, Hewlett-Packard compilers with the debug option will typically insert enough NOP instructions or inline code before the exit to reduce the probability of the exit being prefetch by looping activity.

Offsetting address recognition with HPSPAADJUST.

If you are not using a Hewlett-Packard C Cross Compiler with the "debug" (**-OG**) option, there is another way to overcome prefetch problems. Instead of inserting NOP instructions between functions and before function-end addresses, apply offset values to the function-start and function-end addresses when the function events are defined. This way, function events are recognized at addresses within the functions instead of at the function-start and function-end addresses. The HPSPAADJUST feature of the Software Performance Analyzer causes all of the addresses of function events to be offset by the needed values when the events are defined.

To use the HPSPAADJUST feature, at the shell (before you enter the Software Performance Analyzer), set the HPSPAADJUST shell variable. Then when the Software Performance Analyzer defines events, it will apply the needed offsets to obtain the proper event-start and event-end addresses. The following example shows the format of the HPSPAADJUST shell variable:

$ HPSPAADJUST="<start_offset> <end_offset>"

where:
<start_offset> is the number of bytes to add to the function-start address.
<end_offset> is the number of bytes to subtract from the function-end address.

Using HPSPAADJUST to overcome prefetch for Motorola 68000, 68010, 68302, 6833x and 68340 microprocessors.

For these microprocessors (if using a non-HP C Cross compiler), HP recommends the following HPSPAADJUST setting:

$ HPSPAADJUST="2 0"

After the offset is applied, the byte addresses of the events for the above microprocessors will be adjusted to word boundaries to match the addressing schemes of the emulated microprocessors.

Example result of using: **$ HPSPAADJUST="2 0"**

		Original		Adjusted		Aligned (word)	
Event Type	**Event Name**	**Start**	**End**	**Start**	**End**	**Start**	**End**
Function	Afunc	102	2ff	104	2ff	104	2fe
Function	Bfunc	300	3fd	302	3fd	302	3fc
Interval	Afunc thru Bfunc start	102	300	104	302	104	302
Interval	Afunc end thru Bfunc end	2ff	3fd	2ff	3fd	2fe	3fc

Note that the adjustment applied to an interval address depends on whether the address is the start or end of a function, not whether the address is the start or end of the interval.

If you use the HPSPAADJUST feature, you must ensure that your newly defined event-start and event-end addresses are outside of all loop constructs in the functions. If you compile your code using the MRI compiler, the -Kt or -Kf options will ensure that your functions will not loop into these newly defined event-start and event-end addresses.

Using HPSPAADJUST to overcome prefetch for Motorola 68360, 68020, and 68030 microprocessors.

For these microprocessors (if using a non-HP C Cross compiler), HP recommends the following HPSPAADJUST setting:

$ HPSPAADJUST="6 2"

After the offset is applied, the byte addresses of the events for the above microprocessors will be adjusted to long-word boundaries to match the addressing schemes of the emulated microprocessors.

Example result of using: **$ HPSPAADJUST="6 2"**

		Original		Adjusted		Aligned (long)	
Event Type	**Event Name**	**Start**	**End**	**Start**	**End**	**Start**	**End**
Function	Afunc	102	2ff	108	2fd	108	2fc
Function	Bfunc	300	3fd	306	3fb	304	3f8
Interval	Afunc thru Bfunc start	102	300	108	306	108	304
Interval	Afunc end thru Bfunc end	2ff	3fd	2fd	3fb	2fc	3f8

Note that the adjustment applied to an interval address depends on whether the address is the start or end of a function, not whether the address is the start or end of the interval.

If you use the HPSPAADJUST feature, you must ensure that your newly defined event-start and event-end addresses are outside of all loop constructs in the functions. With HPSPAADJUST="6 2", your compiler must generate assembly code that has at least 10 bytes of executed code before the first loop label occurs. If you are using the MRI compiler, the -Kt and -Kf options will ensure that your functions will not loop into these newly defined event-start and event-end addresses.

Additional help for using HPSPAADJUST

Use of the HPSPAADJUST feature is also described in the online help topics of the Software Performance Analyzer. Choose **Help→General Topic...**, and from the Help Index, choose **Problems with Prefetch**. Using the command line, enter: **help prefetch**.

Markers and how to use them to overcome the effects of an enabled cache or prefetching

Markers are write statements placed at the start and end of each function. The Software Performance Analyzer measures periods between executions of these statements to make function-duration and interval-duration measurements. When using markers, the duration of a function is measured from the function-start marker to the function-end marker, instead of from the function-start address to the function-end address.

The primary need for markers is to make software performance measurements with Motorola 68040 and Intel 80960 processors. The emulators for these processors do not allow the Software Performance Analyzer to make valid performance measurements without the use of markers. Marker-based software performance measurements can also be helpful when making measurements for other microprocessors.

Advantages of using markers in your functions

- Markers allow you to measure performance with address and data caches turned on, provided that marker write statements can write through the caches and appear on buses external to the microprocessor.
- Markers allow you to compile your programs in a more optimal manner. For example, when using the HP AxLS compiler, markers allow you to leave out options -OG, which pad NOPs between functions.
- Markers eliminate all of the problems of prefetching because write transactions are never prefetched (even if the write instructions are prefetched).

Conditions to meet before you can use markers

- Each function that is to be measured must have a unique start marker and a unique end marker. If the markers are not unique, or if they overlap with other function markers, the functions will not be defined. The best way to do this is to add two static variables (unsigned short or char to save memory space) in each function that you want the Software Performance Analyzer to measure.
- You must tell the Software Performance Analyzer to look for these markers, and you must also tell it how to recognize them (discussed later).

The format of marker names

Each marker name must be unique. To instrument your functions with markers, add two static variables to each function you want the Software Performance Analyzer to measure; use unsigned short or char variables to save memory space. The name of each start variable and the name of each end variable will consist of a prefix followed by the name of the function. You can use any prefix for the marker prefix, but you must use it consistently throughout the program under test. If you choose to use "s_" for start markers and "e_" for end markers, then each instrumented function will have additional variables named s_<function name>, and e_<function name>. If you choose to use "abc_" for start markers and "xyz_" for end markers, then each instrumented function will have variables named abc_<function name>, and xyz_<function name>.

Example measurement using markers

Assume the function main has a start variable labeled s_main and an end variable labeled e_main. When main begins to execute, a value is written to s_main. When main finishes its execution, a value is written to e_main. The value that is written to these variables is not important; the Software Performance Analyzer does not read the values of markers, it only looks for a status of "write" to the marker addresses.

Effects of adding markers to your code

As an example, consider the case of a Motorola 68EC030 and the compiler options that can be used to make software performance measurements.

Measurement type and results	AxLS Compiler Options	Typical Performance Improvement of Function	Typical Size Improvement
Traditional SPA measurement	-OG	0% faster	0% smaller
Marker-based SPA measurement	-O + markers	5% to 20% faster	1% to 5% smaller
Marker-based SPA Measurement with Caches enabled.	-O + markers + caches	40% to 80% faster	1% to 5% smaller

Note that the -OG option must be used with traditional SPA measurements to avoid prefetch problems. You cannot make valid Software Performance Analyzer measurements without it.

Another way to view the effects of adding markers is to see how they slow the execution of your file. The addition of markers slows execution of each function by approximately the equivalent time needed to execute two assembly move instructions. If a typical function executes 100 assembly instructions, the performance penalty using markers may be only 2%. Adding the markers also adds about 20 bytes to each function that is instrumented. Using an HP AxLS compiler, both the performance and size penalty can be compensated by using the -O compiler option to compile in a more optimal manner than when using the normal -OG option (which must be used when making software performance measurements without markers).

To instrument your code with markers

One way to instrument your code with markers is to use the HP Marker Preprocessor, discussed next. Another way is to add "#ifdef" statements at the appropriate places in your program. By using #ifdef statements, you can conditionally compile markers in and out of your executable file. For example:

```
#ifdef MARKERS
   static unsigned short s_testvalue;
   static unsigned short e_testvalue;
   static unsigned short s_main;
   static unsigned short e_main;
#endif

int testvalue(myvalue)
int myvalue;
{
#ifdef MARKERS
   s_testvalue = 0;
#endif
   if (myvalue > 100)
   {
#ifdef MARKERS
       e_testvalue = 0;
#endif
       return(1);
   }
   else
   {
#ifdef MARKERS
       e_testvalue = 0;
#endif
       return(0);
   }
}

main()
{
#ifdef MARKERS
   s_main = 0;
#endif
   testvalue(10);
#ifdef MARKERS
   e_main = 0;
#endif
}
```

The HP Marker Preprocessor

A more convenient method to instrument your code with markers is to use the HP Marker Preprocessor. The HP marker preprocessor instruments markers at the beginning and ending of each function during the compile process. To use the HP Marker Preprocessor, replace your normal HP AxLS or MRI compile command with the appropriate marker compile command (such as cc68040mt, mcc68kmt, or ccc68kmt).

The marker preprocessor is supported on AxLS and MRI compilers. It is especially useful with HP emulators for Motorola 68020, 68030, and 68040 processors, and with HP emulators for Intel 80960SA/SB processors. You can also use the marker preprocessor with HP emulators for Motorola 68000, 68302, 6833x, and 68340 processors. For details of how to use the marker preprocessor, refer to the man pages cc68000mt(1), mcc68kmt(1), mcc960mt(1), and ccc68kmt(1).

If you use the HP Marker Preprocessor, you will also need to define a section in your linker command file to contain the markers. Simply add the section named "markers" to your linker command file. The section that contains the markers should be located in a region of RAM memory similar to the data section. Refer to the marker man pages for complete details.

The marker preprocessor is not supported for use with HP emulators for Intel 8086, 80C186, 80C18X, 80286; Hatachi H8/510, H8/532, H8/536; Mitsubishi 7700; and Nec V53 processors.

To tell the Software Performance Analyzer to use markers

Enter the following shell variable before you start your emulation session:

If using sh(1) or ksh(1), enter:

```
HPSPAMARKERS="[yes/no] <start prefix>[+offset] <end prefix>[+offset]"
export HPSPAMARKERS
```

If using csh(1),

```
setenv HPSPAMARKERS "[yes/no] <start prefix>[+offset]
<end prefix>[+offset]"
```

where [yes/no] = yes markers are used and must be included in definitions when events are defined, or no markers are not used in the executable file.

<start prefix> = the unique prefix preceding function names that identifies start markers.

<end prefix> = the unique prefix preceding function names that identifies end markers.

[+offset] = a byte offset to optionally add to the address of the marker. An offset of +2 tells the Software Performance Analyzer to look for writes to the address of the prefixed symbol +2 bytes.

Note that you can set the shell variable HPSPAMARKERS in your .profile so that your performance analyzer is automatically set up each time you login.

If you export the shell variable HPSPAMARKERS="yes s_ e_" before you start the emulation session, each function event (when defined) will include the full address range of the function and the addresses of both of its markers. The marker addresses will be used to make function-duration and interval-duration measurements. The normal address range of the function will be used to make activity measurements.

```
_Number_Name_________________Address_Range_____Type______Address_Markers_
    1 * apply_controller     00001026-000010AC func      000603C6 000603C8
    2 * apply_productions    00000E42-00000EE6 func      000603B6 000603B8
    3 * calculate_answer     000010B4-0000113E func      000603CA 000603CC
    4 * clear_buffer         00000CB0-00000D02 func      000603A6 000603A8
    5 ? data1                00060056-00060059 var
    6 ? data2                00060376-00060379 var
    7 * main                 00001294-0000131C func      000603DA-000603DC
    8 * move_byte            00000A28-00000A6E func      0006038A 0006038C
    9 ? main_move_byte       00001294-00000A28 interval 000603DA 0006038A
   10 ? data1_data2          00060056-00060376 interval 00060056 00060376
```

Note that markers won't improve the quality of activity measurements (Program Activity or Memory and IO Activity) when the microprocessor cache is turned on. Operation of the cache will obscure some of the activity. To get accurate activity measurements, the cache must be turned off.

To tell the Software Performance Analyzer to NOT use markers

Set up the shell variable HPSPAMARKERS="no" before you start your emulation session. In most cases, you can also simply unset the shell variable to turn the markers off. In the case of the Motorola 68040 and the Intel 80960, you must explicitly set HPSPAMARKERS="no" to turn off the checking for markers.

How an MRI compiler instruments code for markers

The MRI compiler automatically instruments markers into your functions when you use the -Kt compiler option. The Software Performance Analyzer can read these MRI markers and define events using them. The format of the MRI marker-variable write statements are either "_r_<function name>" or "r_<function name>", depending upon which MRI compiler you have. The MRI Motorola compiler will add statements to write to _r_main at the start of the function main, and _r_main+2 at the end of the function main. To use these MRI tags as markers, export the shell variable: HPSPAMARKERS="yes _r_ _r_+2".

The shell variable format must be exactly as described in the paragraph titled, "To tell the Software Performance Analyzer to use markers."

Make sure you do not add spaces between the prefix and the offset; the prefix and offset must be viewed as a single token (e.g. _r_+2, not _r_ + 2).

How the Software Performance Analyzer makes its measurements when it uses markers

When markers are instrumented in your code and the Software Performance Analyzer is told to use them, all functions and intervals will be identified by their marker addresses. Functions that do not have marker addresses will not be defined, unless you define them yourself using one of the "define single-event" methods. Only functions having markers will be accepted in function-duration measurements.

When making interval-duration measurements, the Software Performance Analyzer will look for function markers if they exist, but will use any related addresses that can be found. For example, defining an interval from the start of main to the start of a routine lableled move_byte will use the appropriate markers for the start of each of these functions, if they exist. However, an interval event defined between accesses to variables labeled data1 and data2 will not use markers but will use the addresses of the variables.

The status qualification for interval-duration measurements that use markers must be "any" or "data_write"; a status of "data_read" or "prog" won't work because markers are only recognized as write transactions to marker addresses.

When using the enable/disable feature, the Software Performance Analyzer will enable and disable on the marker addresses of the specified events if you have specified function-duration or interval-duration measurements. However, when making activity measurements, the true start and end addresses of the events will be used as the enable point.

When using markers, the "function-duration excluding all calls" measurement mode is not available; the "function-duration excluding profiled calls", and "function-duration including all calls" modes are available.

Additional help for using HPSPAMARKERS

Use of the HPSPAMARKERS feature is also described in the online help topics of the Software Performance Analyzer. Choose **Help→General Topic...**, and from the Help Index, choose **Using Markers**. Using the command line, enter: **help markers**.

Overcoming measurement difficulties when measuring performance of an Intel 80960 Sx

The Software Performance Analyzer obtains its information from the analysis bus of the emulator. It depends on capturing transactions from the emulation microprocessor that appear on the bus. The instruction cache of the 80960 Sx is always enabled, and therefore, many of its instructions are obtained from the cache and never appear on the bus.

You can make valid function-duration and interval-duration measurements in an Intel 80960 Sx processor using markers. In fact, the Software Performance Analyzer, when used with the Intel 80960 Sx, is preconfigured to use MRI markers as defined by the MRI Mcc960 Compiler and its -Kt compiler option. To invoke another marker format for the Intel 80960 Sx, set the HPSPAMARKERS shell variable. Refer to the paragraph titled "Markers and how to use them to overcome the effects of an enabled cache or prefetching" earlier in this chapter.

Activity measurements in an Intel 80960 Sx processor are not very accurate because of the bus cycles that are obscured by operation of the cache. Still, some information can be gained from activity measurements of this processor:

- The results of program activity measurements will show the average number of noncache memory cycles that occur in a given period. This may help you to decide to restructure your program to reduce the number of noncache hits in a particular routine.
- The memory and IO activity measurement yields accurate results for variables because data reads and writes are not affected by the cache. Variables that appear to be used excessively may indicate a problem in your program.

The Software Performance Analyzer will report a large amount of program activity as undefined addresses during an activity measurement because reads of data memory have the same processor status as program memory reads. HP recommends you make program activity measurements using the relative measurement mode to avoid seeing an excessive count of undefined addresses in your measurement results.

To tell the Software Performance Analyzer to NOT use markers, set up the shell variable HPSPAMARKERS="no" before you start your emulation session.

Overcoming measurement difficulties when measuring performance of a Motorola 68040

The Motorola 68040 microprocessor has a deep instruction prefetch queue. This queue causes the Software Performance Analyzer to yield invalid results when making function-duration and interval-duration measurements if these measurements are based on fetches of the start and end addresses of the functions and intervals. The Software Performance Analyzer must use markers (write statements at the beginning and ending of each function) in order to make valid duration measurements. You can automatically insert the needed markers into your files by using the HP marker preprocessor.

The HP marker preprocessor is invoked when you use special compiler commands. Replace the standard AxLS or MRI compiler commands (cc68040, mcc68k, or ccc68k) with the special compiler commands (cc68040mt, mcc68kmt, or ccc68kmt). These compiler comands work about the same as their standard counterparts, except that they first execute the HP marker preprocessor, which adds markers to the functions it instruments. These markers write to a data section labeled "markers". When you link your program, you must add the marker section to your linker command file. Refer to the man pages **cc68040mt(1)**, **mcc68kmt(1)**, and **ccc68kmt(1)** for details.

If you are not using the AxLS or MRI compiler, you will need to instrument your own markers into the functions you want to measure. Refer to the section titled "To instrument your code with markers" earlier in this chapter for details of how to instrument your program with markers.

When using markers, you can turn on the instruction cache and the data cache (as long as write statements can write through the data cache), and you can compile your programs in an optimized manner. Your performance measurements will be made correctly by the Software Performance Analyzer. With the instruction and data caches on and your programs compiled in an optimized manner, you can make performance measurements of your program as it will typically run in your final product.

The Software Performance Analyzer, when used with the HP MC68040 emulator, is preconfigured to read markers as instrumented by the marker preprocessor and define function events based on the marker addresses. To use any other marker format for the MC68040, set the HPSPAMARKER shell variable.

The HPSPAADJUST shell variable is also automatically set to "2 0" for the MC68040. You will not need to set it yourself. HPSPAADJUST avoids problems with address overlaps in the program you are testing.

To tell the Software Performance Analyzer to NOT use markers, set up the shell variable HPSPAMARKERS="no" before you start your emulation session. If you turn off markers (with the MC68040), remember that only activity measurements will be valid. The function duration measurements will be invalid due to prefetch problems.

Refer to the on-line help screens for detailed information about how to instrument your code and use markers when measuring performance of your program running on an HP MC68040 emulator.

Overcoming difficulties in measurements of processors that manage memory

The only additional requirement for successful performance analysis of a processor that manages memory is that when the MMU of the processor is turned on, the deMMUer of the emulator must also be turned on. The Software Performance Analyzer must receive logical addresses (not physical addresses) from the analysis bus of the emulator when it makes performance measurements.

Overcoming the effects of multi-byte return instructions

Multiple-byte return instructions are typically a problem with Intel and Intel-like microprocessors. With these microprocessors, the Software Performance Analyzer is looking for executed cycles, not simply read cycles from addresses. In addition, the Software Performance Analyzer is assuming that the first byte is the start of a function and the last byte is the end of a function. With functions that use single-byte return instructions, the last byte of the function is the executed return instruction that the Software Performance Analyzer is looking for and the measurement of function duration is made correctly.

With a function that uses multiple-byte return instructions, the last byte of the function is typically an address that is read but not executed. In this case, the Software Performance Analyzer may be looking for a transaction that will never occur. To correct for this problem, you need to tell the Software Performance Analyzer the location of the executed byte of the return instruction. If your compiler consistently uses a three-byte return instruction, where the first of the three bytes is executed, you can adjust the end point of each function by setting the HPSPAADJUST shell variable to "0 2"; this will tell the Software Performance Analyzer to use the end address of the function minus 2 bytes as the true end of the function. In a like manner, you can set HPSPAADJUST to "0 1" if your compiler consistently generates 2-byte return instructions, where the end byte is an address value to be read. If your compiler generates return instructions of different lengths in the same program, you must use markers to make duration measurements.

There is a slight possibility that you can avoid the use of markers when your compiler generates multiple-length return instructions. If you set the HPSPAADJUST variable to the worst instruction "0 2" and then visually inspect the generated assembly code of the other functions, and if you find that the compiler always locates an executed instruction at the requested two bytes before the end of the function, then you can avoid using markers. The location of the executed instruction must be an address that is also part of the exit, not part of a branch instruction. Again, this depends on which compiler you are using, and will require some investigation of the assembly code.

Analyzing software performance in assembly language files

The following paragraphs are for the software developer who is writing an assembly language program and wants to test its performance using the Software Performance Analyzer. The Software Performance Analyzer is designed to obtain its address information from an emulator database that contains symbols developed by a compiler. The paragraphs below show how to write assembly language programs that yield C language style databases.

Be sure to write your program in such a way that each function has only one entry point and one exit point. The entry point must be at the start of the function. The exit point must be the last statement of the function (you cannot exit from the middle of the function block).

A function symbol must be present in the emulator database that represents the range of the assembly language function. One way to make sure that a function symbol will be generated for the emulator database is to add appropriate directives to the assembly code so that the compiler will generate a function symbol. Another method allows you to avoid having to enter these directives; it includes the assembly statements within the body of a C function, as shown in the example below. Note that HP 64000 C Cross compilers enable this method with their ASM and END_ASM pragmas.

```
afunc()
{
#pragma ASM
; Assembly statements
;
#pragma END_ASM
}
```

Be sure to add the appropriate number of NOP's between functions to avoid prefetch from the exit of one function into the entrance of another.

Align the start instruction and end instruction of your function on addresses that are fetch boundaries of your microprocessor. This ensures that the addresses that represent the start and end of the function will be fetched and executed. Some microprocessors have three-byte return instructions, where the microprocessor alignment is on 16-bit boundaries and the last two bytes are only read, but not executed. These three-byte return instructions should not be used.

Avoid use of setjump and longjump assembly instructions.

You can avoid some of the constraints discussed above by adding markers to your assembly code. Simply add write statements to be used as start and end markers when you enter and before you exit the function. Refer to the paragraph titled, "Markers and how to use them to overcome the effects of an enabled cache or prefetching", earlier in this chapter. Even if you are using markers, you will still need to add assembly directives to generate the needed function symbol.

Two example assembly functions are shown below. Each example has the appropriate directives to generate a function suitable for use with the HP 68000 emulator and Software Performance Analyzer. For other compilers and other assemblers, you may need to experiment in order to determine the equivalent directives.

This is the equivalent source of the following functions.

```
/* Function Test Program for SPA */
int a;

afunc()
{
   a += 1;
}

main()
{
   afunc();
}
```

This file is written to be compiled with an HP 68000 C Cross compiler. Note the use of the ?file, ?x_f_d, ?end, and ?endf directives.

Example to be compiled with HP 68000 C Cross compiler

```
   CHIP   68000
   NAME   runtest
*
   ?file     '/hp/hp64000/demo/spa/demotest/runtest.s'
   SECT     prog,2,C,P
   NOP
   ?f_x_d  'afunc'
   XDEF    _afunc
_afunc
   LINK    A6,#-0
   ADDQ.L  #1,(_a+0).L
   NOP
   UNLK    A6
   RTS
   ?end
   NOP
   ?f_x_d  'main'
   XDEF    _main
_main
   LINK    A6,#-0
   JSR     (_afunc+0).L
   NOP
   UNLK    A6
```

```
   RTS
   ?end
   NOP
   SECT    data,2,D,D
   XDEF    _a
   ALIGN   2
_a
   DCB.B   4,0
   ?endf
   END
```

This file is written to be compiled with an MRI mcc68k compiler. Note the use of the ?file, ?x_f_d, ?end, and ?endf directives. Also, the assembler options listed in the OPT line are needed.

```
* Assembler options:
   OPT      D,CASE
   NAME     runtest
   ?file    /usr/Testing/usr_hp64000/demo/bba/demomri2/runtest.s
   XCOM     _a,4
   SECTION  code,,C
   nop
   XDEF     _afunc
   ?f_x_d   'afunc'
_afunc:
   addq.l   #1,_a
   rts
   ?end
   nop
   XDEF     _main
   ?f_x_d   'main'
_main:
   bsr.w    _afunc
   nop
   rts
   ?end
   nop
   ?endf
   END
```

Finally, the following approach allows you to obtain the needed function symbols in the emulator data base without having to use the special directives that were needed in the preceding examples; it places the assembly language program code inside the pragmas of a compilable C file. Note that ASM pragmas were also used to insert a NOP between the functions to avoid problems with prefetch for a 68000 microprocessor.

```
/* Function Test Program for SPA */
int a;

afunc()
{
#pragma ASM
   ADDQ.L   #1,(_a+0).L
   NOP
#pragma END_ASM
}

#pragma ASM
   NOP
#pragma END_ASM

main()
{
#pragma ASM
   JSR     (_afunc+0).L
   NOP
#pragma END_ASM
}
```

8

How Good Are Your Test Results

By knowing how to interpret the information given to you by the Software Performance Analyzer, you can ensure that you get the test results you need in order to meet the specifications and requirements of your design project. The information in this chapter will help you interpret your test results.

Statistics calculated by the Software Performance Analyzer are based upon mean and standard deviations of time. In addition, the Software Performance Analyzer calculates a stability value that is based upon both the mean and standard deviation. The paragraphs in this chapter define these terms:

- Definition of mean, which differs in activity and duration measurements.
- Definition of standard deviation in activity and duration measurements.
- When mean and standard deviation may be misleading.
- What stability means and how it is calculated.

Mean

The way the mean is calculated depends on the measurement type. In duration measurements, the calculated mean is the true mean. Every duration of an event is recorded. The sum of all durations of an event is divided by the number of times the event was called.

In activity measurements, the calculated mean is the average amount of time that the event is active in each second of program execution. The mean is calculated using information obtained during the sampling process. The mean will only be correct after a measurement has run long enough to allow sufficient samples to be obtained for the event. The following is the equation used to calculate the mean in an activity measurement:

$$MEAN_{Activity} = \frac{Sum\_Event\_times}{Number\ of\ samples} * \frac{1.0\ second}{One\_sample\_period}$$

Where:

Sum_Event_times = total time the event was sampled active during the sample periods that were used for the event.
Number of samples = Number of sample periods used to sample information for the event.
One_sample_period = length of time spent in each sample period (approximately 2.5 ms).

The following is an example calculation of the mean in an activity measurement. This example assumes that the measurement ran until 50 sample periods had been used for event1. It was active for a sum of 34.0 ms. The time of one sample period was 2.5 ms. The mean of a 1-second interval for event1 is calculated as follows:

(34.0 ms / 50) * (1.0 sec /2.5 ms) = .272 sec, or 272 ms

The mean indicates that in any given second of program execution, event1 is active for an average of 272 ms.

Standard deviation

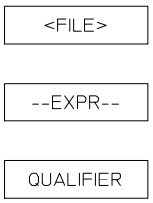
The standard deviation is the calculated deviation about the mean. In a duration measurement, the standard deviation is represented by this equation:

$$StDv_{Duration} = \sqrt{\frac{1}{Num\_calls - 1} * (Sum\_time\_sq - (Num\_calls * Mean\_sq))}$$

where:

Num_calls = Number of calls to this event.
Sum_time_sq = Sum of the squares of the individual time durations of this event.
Mean_sq = Mean * Mean.

In an activity measurement, the standard deviation is calculated by this equation:



where:

Num_samples = number of times this event was sampled.
Sum_time_sq_sample = sum of the squares of the individual event times (times when event was active during any of its sample periods).

```
Mean_sq_sample = Mean_sample * Mean_sample
Mean_sample = (Sum of event times) divided by (Num_samples).
One_sample_period = length of time spent in each
   sample period (approximately 2.5 ms).
```

Note that the standard deviation for an activity measurement is also scaled to reflect the standard deviation of the mean, assuming a one-second sampling interval. A large standard deviation implies that the event was not uniformly active during the sampling intervals that were used for the event.

Referring to the example discussed earlier, which had a mean of 272 ms per 1-second period of program execution, a standard deviation of 40 ms would imply that the above event is typically active between 232 ms and 312 ms for each second of program execution.

When Mean and Standard Deviation May Not Give Best Results

The Mean and Standard Deviation may not give the best statistical results for execution periods that do not have Gaussian distributions. Consider the problem of trying to calculate statistics on the duration of a function that executes at three different time intervals (x, 2x, and 10x). If the function were to execute five times in each of its three intervals, the calculated mean would be 4.33x, and the standard deviation might be 4.17x. In short, the mean and standard deviation you see may not always provide the best statistical information about the period being measured.

Stability

Stability is an indication of how well the Software Performance Analyzer has characterized the measurement. In a pure sense, stability is a measure of the average of standard deviation error tolerances subtracted from 100%.

The stability percent is an indication of how stable the data has become. For example, if the stability is 98% (with confidence level set at 95%), you can be 95% confident that the data is 98% correct.

The longer a measurement runs, the greater its stability will be. Some measurements become stable quickly. Others take a longer period of time to become stable. The following equations are used to calculate stability:

$$\text{Stability} = 100\% - 100 * \frac{\text{Sum\_StDv\_errors}}{\text{Number\_of\_events}}$$

where:

Sum_StDv_errors = Sum of StDv_error_e1 + StDv_error_e2 + ... + StDv_error_eX

StDv_error_e1 = Standard deviation error tolerance for event 1

StDv_error_e2 = Standard deviation error tolerance for event 2

StDv_error_eX = Standard deviation error tolerance for event X

Number_of_events = Number of events that were active in this measurement and had some time recorded during the measurement.

$$StDv\_error\_eX = \frac{Event\_X\_Standard\_Deviation * t}{Event\_X\_Mean * \sqrt{Number\_calls\_or\_samples}}$$

where:

t = table entry in Student's "T" distribution for given level of confidence.

The stability calculation is based upon the Student's "T" distribution. The Student's "T" distribution is used because it provides a more accurate result for small sample sizes. As the sample size increases, the Student's "T" distribution approaches a standard normal distribution.

Part 4

Reference

Part 4

This part of the manual contains the following chapters:

Chapter 9. The user interface

Chapter 10. Syntax of Software Performance Analyzer commands

Chapter 11. Error messages

Chapter 12. The events list

Chapter 13. Interpreting tables, histograms, and measurement specifications

Chapter 14. Using trig1 and trig2 to control measurements with emulators and other analyzers

Chapter 15. Hidden commands of the Software Performance Analyzer

Chapter 16. Specifications of the Software Performance Analyzer

9

The User Interface

The Software Performance Analyzer can operate through one of two user interfaces: (1) the softkey interface, and (2) the graphical user interface. Both interfaces are described in this chapter.

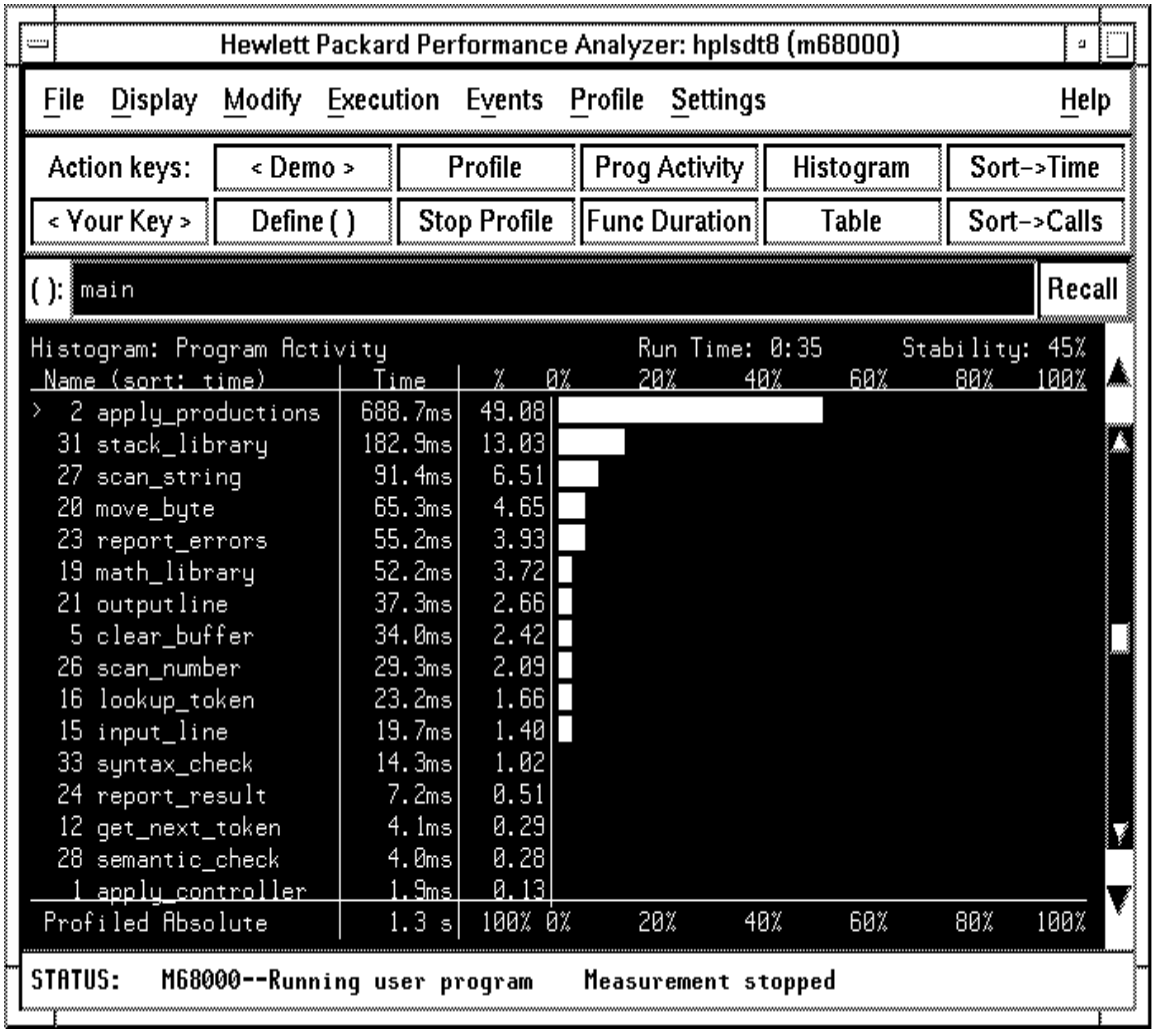
The Softkey User Interface

Whether you are using a display capable of running X windows or you are using a terminal, you can use the softkey interface. The top of the Softkey User Interface is the display area. It shows all the Software Performance Analyzer displays: histograms, tables, events lists, etc. Below the display area is the Status line. It shows all status and error messages. Below the Status line is the command line. To control operation of the analyzer, you must either press softkeys or type commands on the command line.

```
Histogram: Function Duration include calls    Run Time: 0:41      Stability: 99%
_Name_(sort:_time)_____|__Time__|__%___0%_____20%_____40%_____60%_____80%___100%
  22 parse_command     |  37.9 s| 90.68|**********************************
   1 apply_controller  |  31.9 s| 76.22|****************************
   2 apply_productions |  27.2 s| 65.01|************************
  12 get_next_token    |   6.2 s| 14.83|******
  31 stack_library     |   6.1 s| 14.59|******
  16 lookup_token      |   5.3 s| 12.62|*****
  28 semantic_check    |   3.8 s|  9.08|****
  27 scan_string       |   2.9 s|  7.05|***
  25 request_command   |   2.1 s|  4.90|**
  14 initialize        |   2.0 s|  4.80|**
  19 math_library      |   1.9 s|  4.64|**
  20 move_byte         |   1.8 s|  4.38|**
  23 report_errors     |   1.8 s|  4.37|**
  24 report_result     |   1.5 s|  3.52|*
___5_clear_buffer_____|___1.2_s|__2.90|*_____________________________________
 Profiled Absolute     |  41.8 s| 100% 0%     20%     40%     60%     80%   100%

STATUS:   M68000--Running user program    Measurement in process________........
profile function_duration include_calls


profile   define   setup   display          EXPAND   delete    end    ---ETC--
```

The Graphical User Interface

The graphical user interface provides several ways to command actions to be performed by the Software Performance Analyzer. It also provides additional capabilities that are not available through the softkey user interface. In order to use the Graphical User Interface, you must have a display capable of running X windows.



Features provided by the graphical user interface that are not provided by the softkey interface include:

- Menu-driven command entry
- User-definable Action keys
- Pushbutton control of the interface
- Directory and symbol context selection dialog boxes
- File selection dialog boxes
- Command recall dialog box
- Event definition and selection, and profile command dialog boxes
- Dialog boxes for specifying a set of time ranges for duration measurements, measurement enable/disable specifications, and generation of a trigger to start measurements in associated equipment
- Pop-up menus on the histogram, table, events list, symbols screens, status line, and command line entry area.

Features of the graphical user interface

Many entries are made by placing the display cursor on top of a displayed button and clicking (pressing and releasing) a button on the mouse. Unless otherwise specified, clicking on a selection is done using the left mouse button.

The top three lines in the graphical user interface consist of the menu bar, the action keys line, and the entry buffer. These lines can be used to compose commands and control measurements in the Software Performance Analyzer. They are discussed below.

The menu bar

The top line has key words that begin commands for the analyzer. The keywords in the menu bar open pull-down menus that offer additional parameters. Where appropriate, additional windows will open to offer options for your selections. For example:

The "File" pulldown offers several selections. If you select "Load", a sub-menu appears. If you select "Symbols Only ...", a dialog box will open to show the symbol files available in your present working directory. Simply click on a file name and click on OK, and your symbols file will be loaded. This is the same as composing a command line in the softkey interface, but it offers more help when entering filenames. If you have the command line portion of the graphical user interface turned on, you will see the softkey interface form of the command line you have composed appear in the command line section of the interface.

The Action Keys line

The action keys line of the interface has keys that command immediate action. If you click on the "Profile" action key, the profile command will be executed. You can define your own action keys to perform immediate actions that you desire. Your action keys can simplify your test procedures. Refer to Chapter 5 for instructions showing how to define action keys.

The entry buffer line

The entry buffer begins with "():". Use this buffer to create information strings to be used in commands in the interface. You can enter information into the entry buffer from the keyboard, by cutting words or lines from a display, or by clicking on information in the Recall dialog box.

The display area

The display area shows all the displays that are available in the softkey interface (such as, histogram, tables, and events list). Pop-up windows are available in the display area to speed-up operations. When you see a pointing finger, a pop-up window can be called for that portion of the display. Scroll bars appear if the information available to the display is greater than one screen long.

The status line

The status line displays messages the same as it does when you use the Softkey User Interface. When you click on an error or warning message, the message will go away. A pop-up menu is available to show status-line information.

The command line

The command line area can be used to compose commands just like in the Softkey User Interface. The command line area can also be turned off, if desired. If you turn off the command line, all commands must be composed by using the Graphical User Interface features discussed above.

Dialog boxes of the Graphical User Interface

To simplify the task of composing commands for actions to be performed by the Software Performance Analyzer, dialog boxes are provided. These dialog boxes are obtained when you click on pulldown menu items whose names end in "...". The dialog boxes listed below are unique to the Software Performance Analyzer; they are discussed in detail on the pages following the table of mouse button and keyboard bindings:

- File Selection (store or load file selection)
- Define Events
- Define Single Event
- Select Events
- Profile
- Time Ranges
- Setup Enable and Disable
- Setup Trig2

Popup menus of the Graphical User Interface

The popup menus that are unique to the Software Performance Analyzer are discussed following the dialog boxes in this chapter. The popup menus cause immediate action to be taken on the item in the display where the popup menu is called. You can call a popup menu for any item in a display if the mouse cursor in the display is a hand symbol. The popup menus that are unique to the Software Performance Analyzer are listed below:

- Events Display
- Histogram/Table Display

- Global (or Local) Symbols Display

Mouse button and keyboard bindings

Because the Graphical User Interface runs on different kinds of computers, which may have different conventions for mouse buttons and key names, the Graphical User Interface supports different bindings and the customization of bindings.

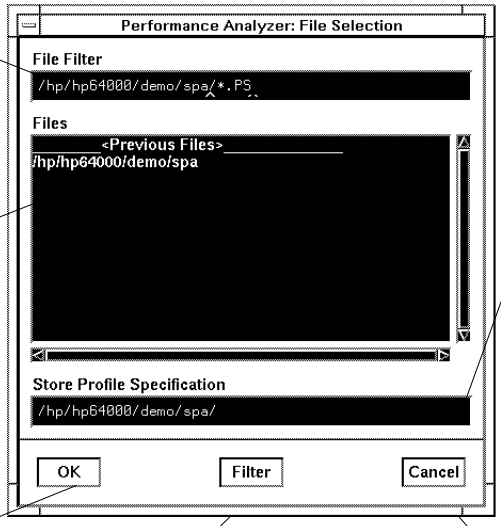
This manual refers to the mouse buttons using general (or "generic") terms. The following table describes the generic mouse button names and shows the default mouse button bindings.

Mouse Button Bindings and Description

Generic Button Name	Bindings: HP 9000	Bindings: Sun SPARCsystem	Description
paste	left	left	Paste from the display area to the entry buffer.
command paste	middle[1]	middle[1]	Paste from the entry buffer to the command line text entry area.
select	right	right	Click selects first item in popup menus. Press and hold displays menus.
command select	left	right	Displays pulldown menus.
pushbutton select	left	left	Actuates pushbuttons outside of the display area.

[1] Middle button on three-button mouse. Both buttons on two-button mouse.

File Selection (for Store or Load Profile) dialog box



Place the mouse pointer in the text entry area and specify the directory whose files will be listed below.

Click on a file in this block to select a predefined or previously specified file name.

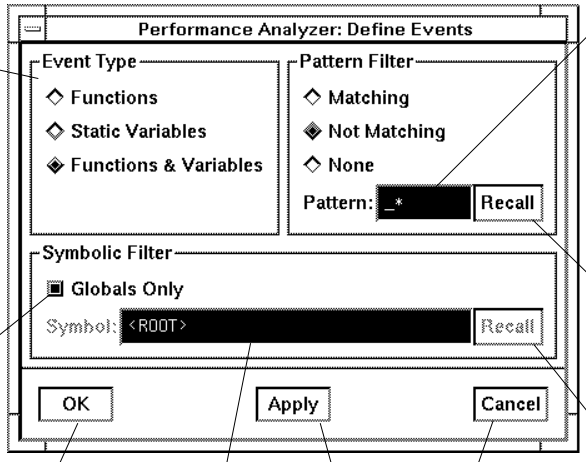
Place the mouse pointer in the text entry area and type the name of the file that will store the present profile specification, or that contains the profile specification to be loaded into the Software Performance Analyzer.

Click OK to store (or load) the profile specification and close the dialog box.

Click on Filter to update the Files list for the directory named in the Filter entry area.

Click this button to cancel storing (or loading) of the profile specification and close the dialog box.

Define Events dialog box



Choose the desired type of events to be created by clicking the associated button.

Toggle this button to make events for all global symbols or for all symbols resident in the modules or files specified below.

Place the mouse pointer in the text entry area and type in the modules or files where functions and/or variables reside for which you want to define events. You can enter multiple modules or files by separating each with a comma.

Click OK to define events and close the dialog box.

Click Apply to define events and leave this dialog box open on screen.

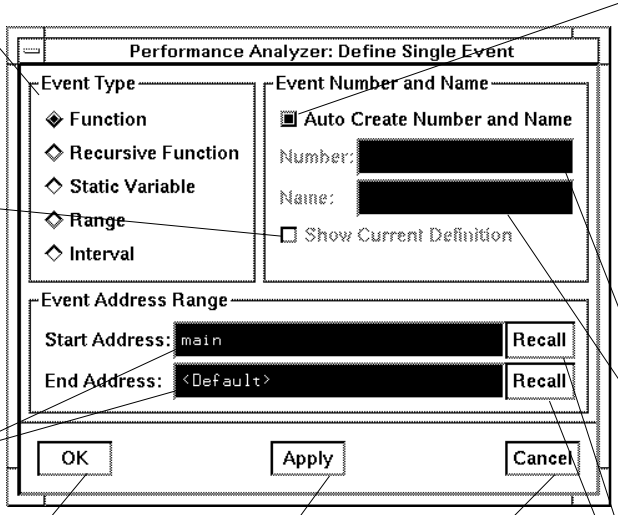
Click this button to cancel the event-defining process and close this dialog box.

Place the mouse pointer in the text entry area and type in the pattern to be matched or not matched by function and variable symbol names in order to have events created for them.

Click this button to select predefined or previously specified patterns.

Click this button to select predefined or previously specified modules or files that identify the scope where events will be defined.

Define Single Event dialog box



Choose the desired type of event to be created by clicking the associated button.

Click this button to show the current definition of the event whose name and/or number appears above.

Place the mouse pointer in the text entry area and type the start address and end address that are to be associated with the new event. You can enter a symbol in Start Address; SPA will assign the end address of that same symbol if you leave End Address blank.

Click OK to define the single event specified and close the dialog box.

Click Apply to define the single event specified and leave the dialog box open.

Click this button to cancel the event definition and close the dialog box.

Toggle this button to choose either automatic creation of a name and number, or use the Number and Name entries when creating an event for the content of the Event Address Range fields.

Place the mouse pointer in the text entry area and type the number and/or name of the event to be defined.

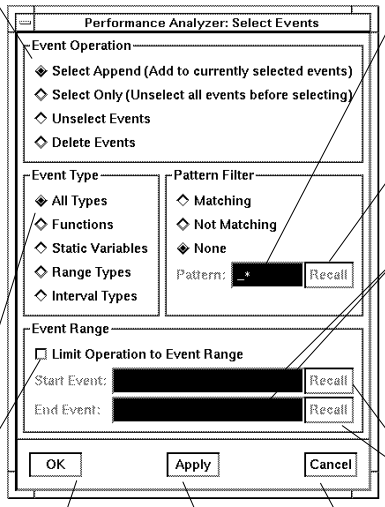
Click these buttons to select predefined or previously specified entries.

Select Events dialog box

Choose the desired operation on the specified events by clicking the associated button. You can:
- Select all events that match specifications below, and append them to the list of selected events.
- Select only events that match specifications below, and unselect all other events.
- Unselect all events that match specifications below.
- Delete from the current display all events that match specifications below.

Choose the desired type of event to be operated on by your "Select Events" specification.

Toggle this button to make your specification apply to all events in the current display, or to only the events between the specified start event and/or specified end event in the events list.



Place the mouse pointer in the text entry area and type in the pattern to be matched/not matched by names in the events list.

Click this button to select predefined or previously specified patterns.

Place the mouse pointer in the text entry area and type the name or number of the first and/or last events in the event list that define the portion of the list to receive the Event Operation specified above.

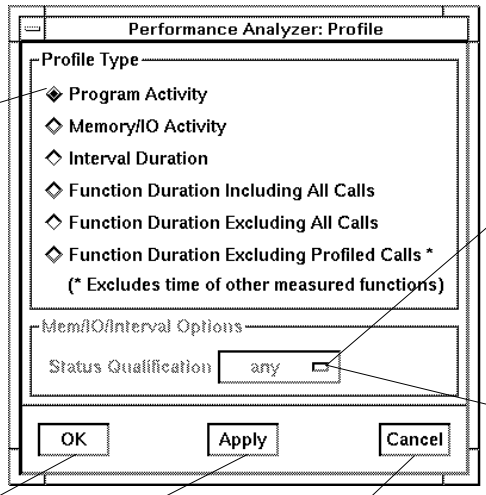
Click these buttons to select predefined or previously specified entries.

Click OK to activate the Event Operation and close the dialog box.

Click Apply to activate the Event Operation and leave the dialog box open.

Click this button to cancel the Event Operation and close the dialog box.

Profile dialog box



Choose the desired type of measurement by clicking the associated button.

In Memory/IO Activity measurements, click this button to choose the type of processor status to qualify the address cycles to be saved in analyzer memory.

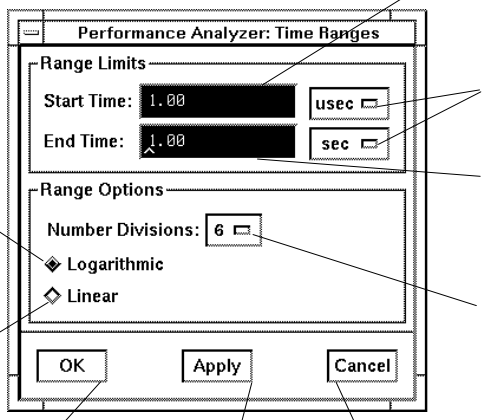
In Interval Duration measurements, click this button to choose the type of processor status to qualify the start and end addresses of the interval measurement.

Click OK to begin the profile measurement selected and close the dialog box.

Click to begin the profile measurement selected and leave the dialog box open.

Click this button to cancel the profile measurement specification and close the dialog box.

Time Ranges dialog box



Place the mouse pointer in the text entry area and enter the lowest value of the time ranges.

Click these buttons to select the desired time unit.

Place the mouse pointer in the text entry area and enter the highest value of the time ranges.

Click this button to select the number of ranges desired.

Click this button to divide the Range Limits fields into a set of logarithmic ranges.

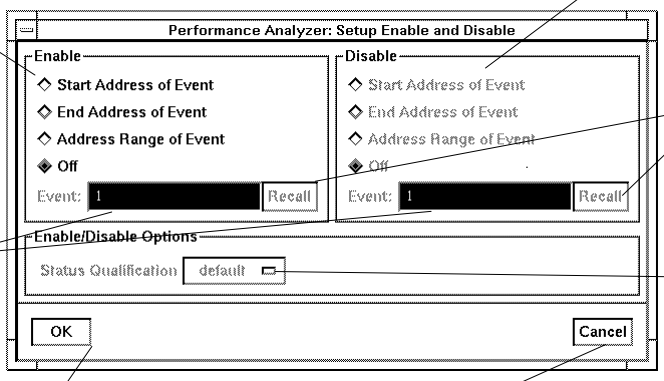
Click this button to divide the Range Limits fields into a set of linear ranges.

Click OK to modify the time ranges of the measurement specification and close the dialog box.

Click Apply to modify the time ranges of the measurement specification and leave the dialog box open.

Click this button to cancel the time range modification and close the dialog box.

Setup Enable and Disable dialog box



Choose the desired address point within this enable event.

Place the mouse pointer in the text entry area and type the name or number of the event to enable and/or disable the measurement.

Click OK to establish the specified enable/disable setup and close the dialog box.

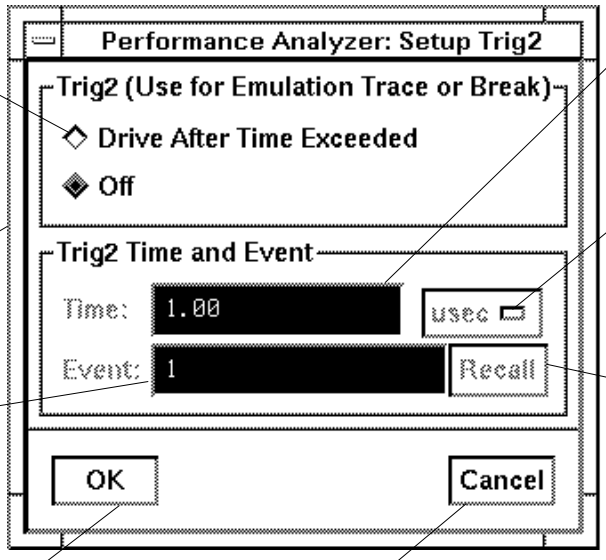
Click this button to cancel the modifications and close the dialog box.

Choose the desired address point within the disable event by clicking the associated button.

Click this button to select predefined or previously specified entries.

Click this button to change the processor status used to qualify the enable and disable addresses. Note: default causes the enable and disable addresses to be qualified on the same processor status used by the profile command.

Setup Trig2 dialog box



Click this button to generate Trig2 when the event below exceeds the time specified.

Click this button to turn off the Trig2 specification.

Place the mouse pointer in the text entry area and type in the name or number of the event that will cause Trig2 when it runs longer than the specified time.

Click OK to activate the Trig2 specification and close the dialog box.

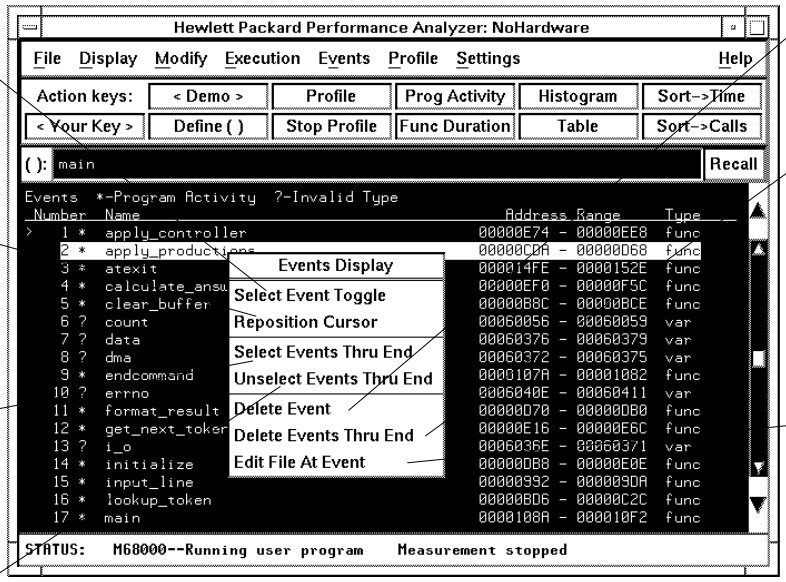
Click this button to cancel the Trig2 modification and close the dialog box.

Place the mouse pointer in the text entry area and type in the time to be exceeded.

Click this button to select the desired time unit.

Click this button to select predefined or previously specified entries.

Events Display popup menu



Click to select or unselect the highlighted event.

Click to move the cursor to the highlighted event.

Click to select this event and all others to the end of the list.

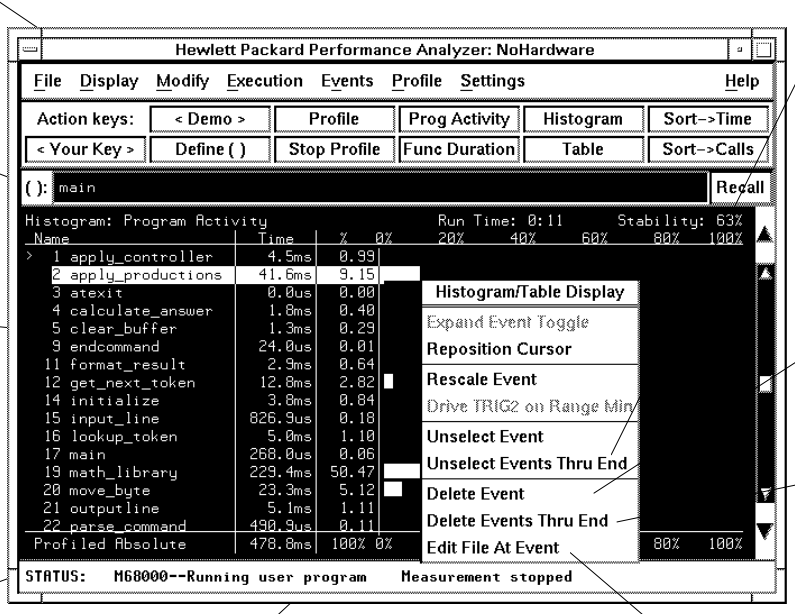
Click to unselect this event and all others to the end of the list.

Click to delete the highlighted event.

Click to delete the highlighted event and all other events to the end of the list.

Click to open an edit window into the source file where this event is defined.

Histogram/Table Display popup menu



Click to place time ranges below the highlighted event.

Click to move the cursor to the highlighted event.

Click to rescale the histogram for best resolution of the highlighted event.

Click to drive trig2 when the expanded event runs long enough to be recorded in the highlighted time range

Click to unselect the highlighted event.

Click to unselect the highlighted event and all other events to the end of the list on display.

Click to delete the highlighted event.

Click to delete the highlighted event and all other events to the end of the list on display.

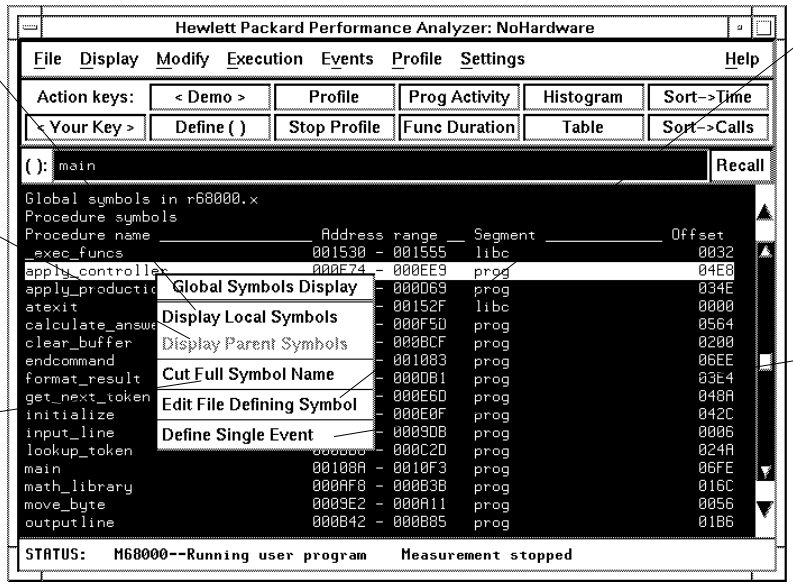
Click to open an edit window into the source file where this event is defined.

Global (or Local) Symbols Display popup menu

Click to display symbols local to the highlighted symbol.

Click to display the parent symbol of the highlighted symbol.

Click to place the full symbol name in the entry buffer.



Click to open an edit window into the source file where the highlighted symbol is defined.

Click to define a single event to represent the highlighted symbol.

10

Syntax of the Software Performance Analyzer Commands

This chapter discusses commands that appear in pulldown menus and on the softkeys. Syntax diagrams are provided to help you understand the commands. Each command token is described. Refer to Chapter 15 for hidden commands.

How Pulldown Menus Map to the Command Line

Pulldown	Command Line
File→Context→Directory	cd
File→Context→Symbols	cws
File→Load→Emulator Config ...	load configuration ...
File→Load→Executable ...	load <abs_file>
File→Load→Program Only ...	load <abs_file> nosymbols
File→Load→Symbols Only ...	load symbols ...
File→Load→Profile Spec ...	load profile_spec ...
File→Store→Profile Spec ...	store profile_spec ...
File→Copy→Display ...	copy display to
File→Copy→Histogram ...	copy histogram to
File→Copy→Table ...	copy table to
File→Copy→Events ...	copy events to
File→Copy→Measurement Spec ...	copy measurement_spec
File→Copy→Global Symbols ...	copy global_symbols to
File→Copy→Local Symbols () ...	copy local_symbols_in --SYMB-- to
File→Copy→Error Log ...	copy error_log to
File→Copy→Event Log ...	copy event_log to
File→Log→Playback ...	<command file>
File→Log→Record ...	log_commands to
File→Log→Stop	log_commands off
File→Emul700→Graphic Windows	N/A
File→Emul700→<other products>	N/A
File→Edit→File ...	! vi <file> ! no_prompt_before_exit
File→Edit→At () Location ...	! vi +<line> <file> ! no_prompt_before_exit
File→Term ...	!
File→Exit→Window (save session)	end
File→Exit→Locked (all windows, save session)	end locked
File→Exit→Released (all windows, release emulator)	end release_system

Pulldown	Command Line
Display→Context ...	pwd, pws
Display→Histogram	display histogram
Display→Table	display table
Display→Events	display events
Display→Measurement Spec	display measurement_spec
Display→Sort Events→Time	display sort_events time
Display→Sort Events→Calls	display sort_events calls
Display→Sort Events→Cycles	display sort_events cycles
Display→Sort Events→Address	display sort_events address
Display→Sort Events→Alphabetic	display sort_events alphabetically
Display→Sort Events→Defined	display sort_events defined
Display→Performance Data→Time	display data time
Display→Performance Data→Calls	display data calls
Display→Performance Data→Cycles	display data cycles
Display→Performance Data→Absolute	display data absolute
Display→Performance Data→Relative	display data relative
Display→Time Ranges ...	display time_ranges ...
Display→Global Symbols	display global_symbols
Display→Local Symbols ()	display local_symbols_in --SYMB--
Display→Error Log	display error_log
Display→Event Log	display event_log
Modify→Emulator Config ...	modify configuration ...
Modify→Setup Enable/Disable ...	setup_measurement enable/disable ...
Modify→Setup Trig2 ...	setup_measurement drive trig2_after ...
Modify→Setup Performance Analyzer Default	setup_measurement default
Execution→Run→from PC	run
Execution→Run→from Transfer Address	run from transfer_address
Execution→Run→from Reset	run from reset
Execution→Break	break
Execution→Reset	reset
Events→Display	display events
Events→Define Events ...	define multiple_events ...
Events→Define Single Event()	define single_event <EVENT>
Events→Define Single Event ...	define single_event ...
Events→Select Events ...	select_events ..., unselect_events ..., delete_events ...
Events→Delete All Events	display events; delete_events
Events→Renumber Events	renumber_events

Pulldown	Command Line
Profile→Profile ...	profile ...
Profile→Profile Again	profile
Profile→Stop Profile	stop_profile
Settings→Status Qualification→any	set status_qualification any
Settings→Status Qualification→<TYPE>	set status_qualification <TYPE>
Settings→Decimal Alignment	set decimal_alignment on/off (toggle)
Settings→Event Numbers	set event_numbers on/off (toggle)
Settings→Command Line	N/A (toggles the command line)

How Popup Menus Map to the Command Line

Histogram/Table Display Popup	Command Line
Expand Event Toggle	expand <event name or number> (toggle)
Reposition Cursor	N/A (use keyboard keys)
Rescale Event	display histogram rescale to_max (for selected event)
Drive TRIG2 on Range Min	setup_measurement drive trig2_after <TIME> <EVENT> unselect_events <EVENT>
Unselect Event	unselect_events <EVENT> thru end
Unselect Events Thru End	delete_events <EVENT>
Delete Event	delete_events <EVENT> thru end
Delete Events Thru End	! vi +<line> <file> ! no_prompt_before_exit
Edit File At Event	

Events Display Popup	Command Line
Select Event Toggle	select <event name or number> (toggle)
Reposition Cursor	N/A (use keyboard keys)
Select Events Thru End	select events append <EVENT> thru end
Unselect Events Thru End	unselect_events <EVENT> thru end
Delete Event	delete_events <EVENT>
Delete Events Thru End	delete_events <EVENT> thru end
Edit File At Event	! vi +<line> <file> ! no_prompt_before_exit

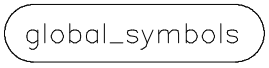
Symbols Display Popup	Command Line
Display Local Symbols	display local_symbols_in <SYMBOL>
Display Parent Symbols	display local_symbols_in <SYMBOL>, display global_symbols
Cut Full Symbol Name	N/A
Edit File Defining Symbol	! vi +<line> <file> ! no_prompt_before_exit
Define Single Event	define single_event <SYMBOL>

Syntax Conventions

Conventions used in the command syntax diagrams are defined below.

Oval-shaped Symbols

Oval-shaped symbols contain command tokens. Command tokens, together with symbols and numeric values, make up complete Softkey Interface commands. Most command tokens appear on softkey labels. Those that do not appear as softkeys must be typed into the command line (for example, **log_commands** and **wait**). An example of an oval-shaped symbol is as follows:



Rectangular-shaped Symbols

Rectangular-shaped symbols contain prompt softkeys, softkey-changers, and references to other syntax diagrams. Prompt softkeys are enclosed in angle brackets (< and >). Softkey-changers are enclosed in dashes (--). References to other diagrams are shown in all capital letters without any enclosing symbols.

Examples of all three kinds of rectangular symbols follow:

Prompt Softkey. Press to get a hint about the kind of information needed.

Softkey Changer. Press to get another set of softkeys. Some softkey changers have their own syntax diagrams in this chapter.

Reference to a diagram showing details in this chapter.

Circles

Circles contain operators and delimiters used in expressions and on the command line. An example of a circle symbol is as follows:

The —NORMAL— Key

The softkey labeled **—NORMAL—** is a special softkey-changer; use it to return to the former set of softkeys. For example, you can press the **—EXPR—** softkey to call up a set of prompt softkeys to help you complete an expression. After you complete the expression, you can return to the set of softkeys containing the **—EXPR—** softkey by pressing the **—NORMAL—** softkey.

Summary of Commands

Softkey Interface commands are summarized in the following table:

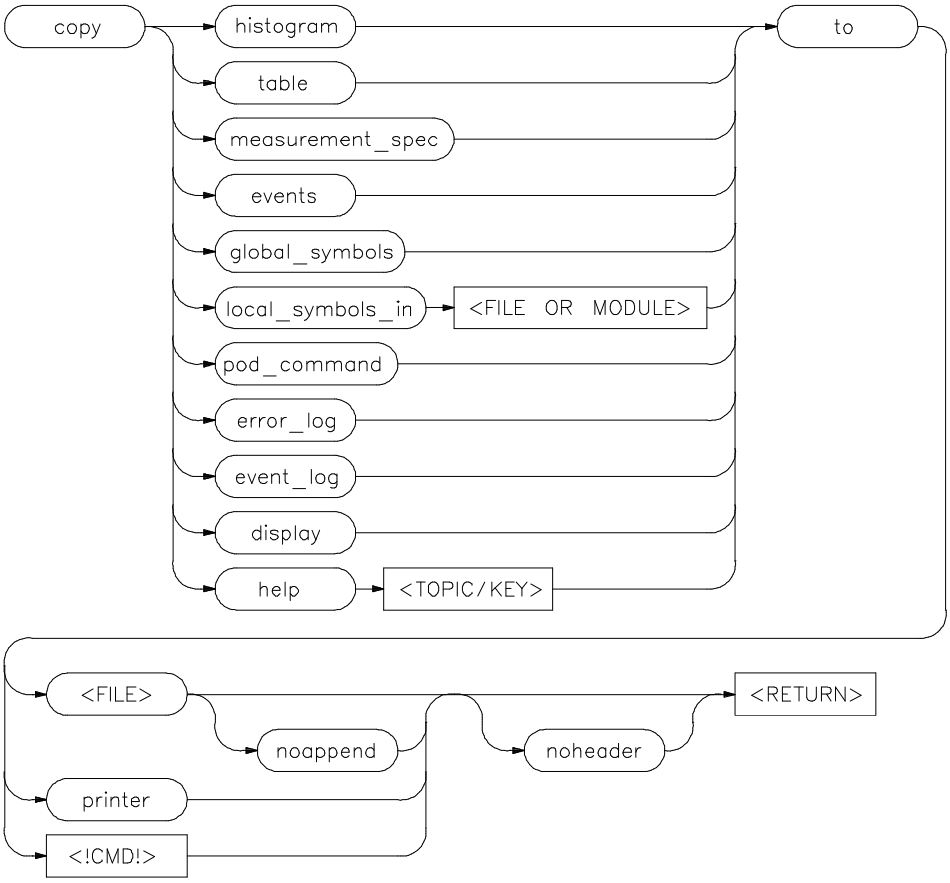
Software Performance Analyzer Commands

UNIX_COMMAND[1]	display measurement_spec	select_events functions
break[2]	display pod_command	select_events interval_types
cd (change directory)[1]	display sort_events	select_events range_types
copy display	display stability	select_events variables_static
copy error_log	display rescale	set <environment variable>
copy events	display table	set byte_alignment
copy event_log	display time_ranges	set confidence
copy global_symbols	end	set decimal_alignment
copy help	forward[1]	set event_numbers
copy histogram	help[1]	set histogram_character
copy local_symbols_in	load <FILE>[2]	set langinfo
copy measurement_spec	load configuration[2]	set status_qualification
copy pod_command	load profile_spec	set symbols
copy table	load symbols	setup_measurement default
cws(change working symbol)[1]	log_commands[1]	setup_measurement disable
define multiple_events	modify configuration[2]	setup_measurement drive
define single_event	pod_command[1]	setup_measurement enable
delete_events functions	profile function_duration	setup_measurement start
delete_events interval_types	profile interval_duration	setup_measurement termination
delete_events range_types	profile memory_and_io_activity	stop
delete_events variables_static	profile modify_command	stop_profile
display data	profile program_activity	store profile_spec
display error_log	pws (print working symbol)[1]	unselect_events functions
display events	reset[2]	unselect_events interval_types
display event_log	renumber_events	unselect_events range_types
display global_symbols	run[2]	unselect_events variables_static
display histogram	run from reset[2]	version[1]
display local_symbols_in	run from transfer_address[2]	wait[1]

[1] Refer to the chapter titled, "Hidden Commands of the Software Performance Analyzer"

[2] These are emulator commands that can be forwarded from the Software Performance Analyzer. Refer to your emulator manual for command details.

copy



The copy command copies selected information to your system printer or to a listing file, or pipes it to a UNIX filter.

The parameters are as follows:

histogram — histogram lets you copy the current histogram to the destination you choose.

table — table lets you copy the current table to the destination you choose.

measurement_spec	measurement_spec lets you copy the current Software Performance Analyzer measurement specification to the destination you choose.
events	events lets you copy all of the events in the events list to the destination you choose.
global_symbols	global_symbols lets you copy a list of all global symbols in memory to the destination you choose.
local_symbols_in	local_symbols_in lets you copy a list of local symbols in a specified source file to the destination you choose.
pod_command	pod_command lets you copy a summary of the most recent commands sent to the HP64700 pod via the command, "pod_command".
error_log	error_log lets you copy a list of the most recent errors that were caused by commands entered on the command line (or supplied by command files).
event_log	event_log lets you copy the most recent events (not "events_list" events) that have occurred in the emulation/analysis system. Events in the log include software breakpoints and other changes in the status of the emulator and/or analyzer.
display	display lets you copy the information currently on the screen to the destination you choose.
help	help lets you copy the contents of the on-line help files to the destination you choose.
to	to lets you specify the destination of the copied information. to must be included in the copy command line.
<FILE>	<FILE> prompts you for the name of the listing file where you want the specified information to be copied.
noappend	noappend causes the copied information to overwrite any file with the same name as <FILE>. If you do not specify "noappend", the copied information will be appended to <FILE>, if a file by that name exists.
noheader	noheader specifies the information be copied without headings.

printer

printer specifies your system printer as the destination for the copy command. Note: Before you can specify printer as the destination device, you must first define PRINTER as a shell variable.

If using sh(1) or ksh(1), enter:

$ HP64PRINTER = lp
$ export HP64PRINTER

If using csh(1),

setenv HP64PRINTER lp

UNIX CMD

UNIX CMD represents a UNIX filter or pipe that is the destination of your copy command. UNIX commands must be preceded by an exclamation point (!). If you place an exclamation point after your UNIX command, your system will return to the emulation environment after command completion, allowing you to continue command-line execution. Emulation will not be affected by a UNIX command that is a shell intrinsic.

!

The exclamation point "!" is the delimiter for UNIX commands. An exclamation point must precede all UNIX commands. An exclamation point following a UNIX command is optional.

If an exclamation point is also part of a UNIX command, escape the exclamation point with a backslash (e.g. \!).

Examples

copy histogram to <FILE>

copy table to printer

copy events to printer

copy display to <FILE>

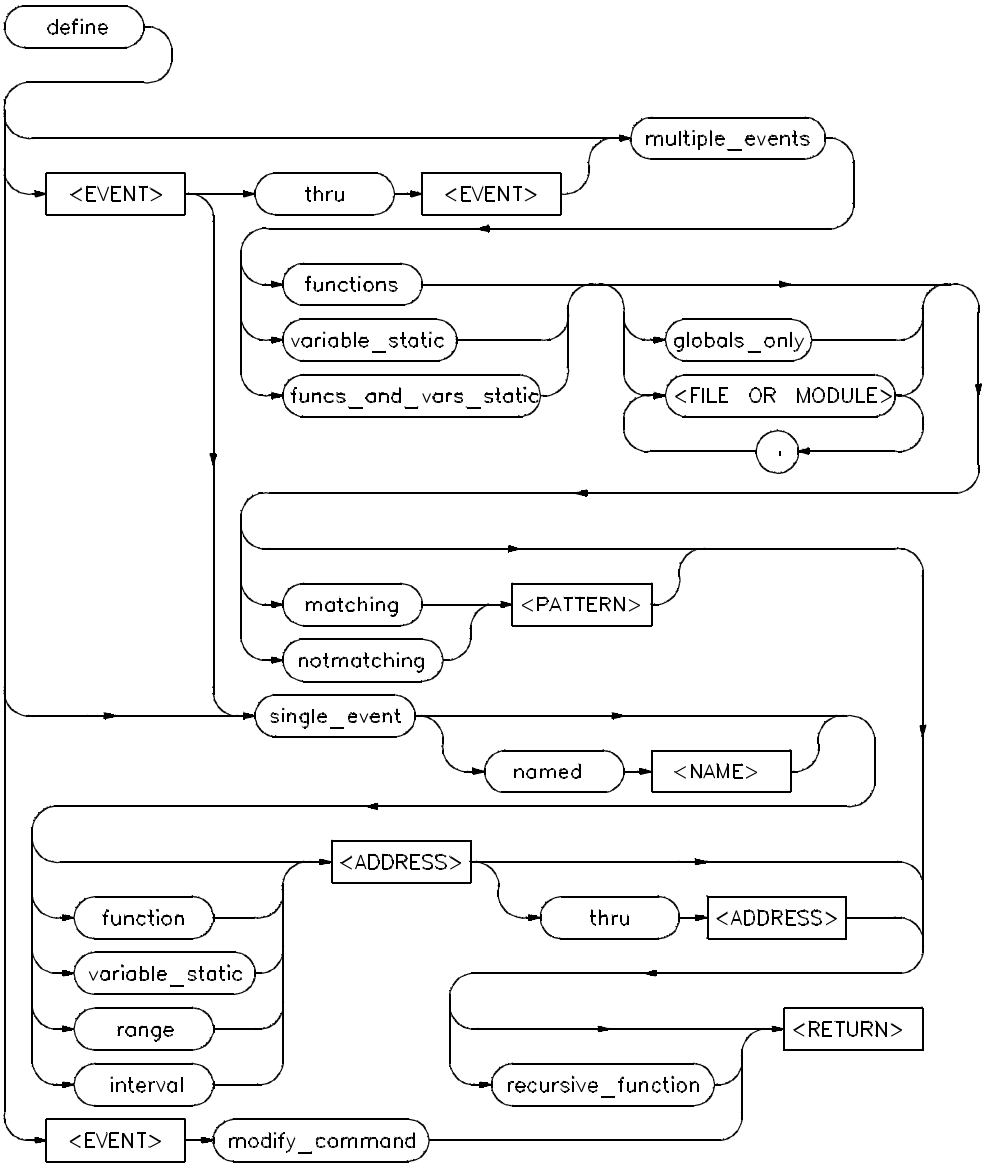
copy local_symbols_in SOURCE.S: **to printer**

copy pod_command to <FILE>

copy error_log to <FILE>

copy event_log to <FILE>

define


define
multiple_events
<EVENT>
thru
<EVENT>
functions
variable_static
funcs_and_vars_static
globals_only
<FILE OR MODULE>
,
matching
notmatching
<PATTERN>
single_event
named
<NAME>
<ADDRESS>
function
variable_static
range
interval
thru
<ADDRESS>
<RETURN>
recursive_function
<EVENT>
modify_command

The define command lets you define events to be used in Software Performance Analyzer measurements. These events can represent functions, static variables, single addresses, or simple address ranges in your absolute file. The Software Performance Analyzer can accept up to 1000 events in its events list. If your absolute file has more than 1000 functions and/or variables, there are ways to qualify your event definitions so that you can measure only the functions and/or variables you want to measure.

Your target program may have functions and variables that were developed by other people, but you may only want to test the performance of the functions and variables you developed. The define command makes it easy for you to define events for just the functions and variables you want to test. For example, you can enter a command to define events for only the functions and/or variables in a particular file:
"**define multiple_events funcs_and_vars_static** myfile.c:"

You may want to examine only the functions that deal with a particular task, such as data-base management. If you gave names to these functions that began with particular characters, such as "db", then you could use a command like: "**define multiple_events functions matching** "db*"

You may know the names of a few files whose functions you want to test. You can define events for only the functions in those files with a command, such as: "**define multiple_events functions** driver.c: , convert.c: , update.c:"

No overlapping events are allowed. If you try to define an event that represents addresses that are already represented by other events, the analyzer will ask if you want to delete all overlapping events. If you answer "yes", all of the other events that overlap the range of your new event will be deleted, and your new event will be created, automatically.

Interval events are the one exception to the rule for overlapping events, described above. Interval events can overlap all other events, except that no interval event can have either a start or end address that is the same as a start or end address in another interval event.

If no events have been defined when the profile command is executed, the Software Performance Analyzer will execute this command before performing the profile measurement:
define multiple_events funcs_and_vars_static globals_only notmatching "_*"

This causes the Software Performance Analyzer to search your symbol data base and create up to 1000 events to represent functions and static variables that are

listed in the symbol data base. The "_*" specification prevents defining events for the assembler-generated and compiler-generated symbols.

The parameters are as follows:

<EVENT> — <EVENT> is the number of the event you want to define or modify.

multiple_events — multiple_events lets the Software Performance Analyzer create a series of events with a single command. For example, "**define multiple_events functions**" tells the Software Performance Analyzer to find all of the functions in your symbol data base and create events to represent them.

If you specify an event type (functions, variables_static, etc.), events of the specified type will be created to represent corresponding symbols in your symbol data base.

If you include a range of event numbers with this command (**define** 1 **thru** 12 **multiple_events functions**), events that already have those numbers in your present events list will be deleted. Then the Software Performance Analyzer will define new events for those numbers.

If you do not include a range of event numbers with your command, the Software Performance Analyzer will create new events until the limit of 1000 events is reached, or until there are no more events of the specified type and/or name that can be created from your symbol data base.

single_event — single_event lets you create or modify a single event as specified by the event number.

"**define** 12 **single_event function** Function_1" causes the Software Performance Analyzer to define event number 12, identify it as a function, and assign it the same name and address range presently occupied by Function_1 in your absolute file.

"**define** 4 **modify_command**" causes the Software Performance Analyzer to bring the present definition of event 4 to the command line where you can edit it.

functions — functions specifies that events are to be defined to represent functions in the symbol table.

funcs_and_vars_static — funcs_and_vars_static specifies that events are to be defined to represent both functions and static variables in the symbol table.

variables_static — variables_static specifies that events are to be defined to represent static variables in the symbol table.

range	range causes the Software Performance Analyzer to treat the address range you enter as a simple address range, not a function or static variable. Range events can be included in measurements of memory and program activity. Range events are not included in measurements of function or interval duration.
interval	interval causes the Software Performance Analyzer to treat the address values you enter as an interval for an interval duration measurement. Intervals are only valid in interval duration measurements.
globals_only	globals_only specifies that events are to be defined to represent only global functions and/or global static variables in the symbol table.
matching <PATTERN>	matching <PATTERN> specifies a character sequence that the Software Performance Analyzer will try to match with the names of functions and/or static variables in the symbol data base. Events will be defined for functions and/or static variables whose names match <PATTERN>. The Software Performance Analyzer accepts shell-like pattern-matching symbols for expressing patterns (example: "f*"). Put <PATTERN> in quotes.
notmatching <PATTERN>	notmatching <PATTERN> specifies a character sequence that the Software Performance Analyzer will try to match with the names of functions and/or static variables in the symbol data base. Events will be defined for all functions and/or static variables, except those whose names match <PATTERN>. The Software Performance Analyzer accepts shell-like pattern-matching symbols for expressing patterns (example: "f*"). Put <PATTERN> in quotes.
named <NAME>	named <NAME> lets you specify a name for an event. The name you specify will override the name generated by the Software Performance Analyzer.
recursive_function	recursive_function further describes a function as a recursive function. The Software Performance Analyzer needs this information to make accurate function-duration measurements on recursive functions.
modify_command	modify_command recalls the definition of the specified event to the command line.

Examples

define multiple_events functions

define multiple_events functions globals_only

define multiple_events variables_static

define multiple_events funcs_and_vars_static

define 1 **thru** 12 **multiple_events functions** file1.c:

define multiple_events functions matching "ab*"

define multiple_events functions notmatching "ab*"

define single_event range Symbol_1 **thru** Symbol_2

define 12 **single_event function** Function_1

define 14 **single_event function** Function_2 **recursive_function**

define single_event named data1 **variable_static** 01234h **thru** + 010h

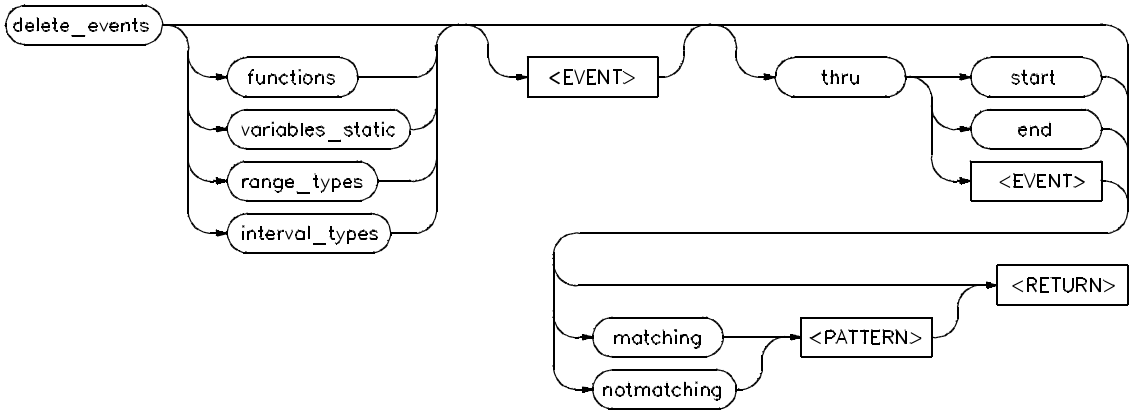
define single_event named data2 **range** 0400fh **thru** 04bcfh

define single_event function 0800fh **thru** 08bcfh

define single_event interval Function_1

define 4 **modify_command**

delete_events



The delete_events command lets you delete events you do not need. You can identify events to be deleted by specifying the type of events (functions), the range of event numbers (6 thru 15), or the event names. Deleted events are removed from all displays and all measurement specifications.

If you delete events "thru end" on a histogram or table, all events from the cursor position through the end of the current display will be deleted. For example, if the current display is a histogram from a function_duration measurement, it will show events that represent source-file functions. It will not show other kinds of events. If you place the cursor on an event in the histogram, and delete "thru end", all events from the cursor position through the end of the histogram will be deleted. All events ahead of the cursor position, and all events that were not part of the histogram display will still remain in the events list.

The command "**delete_events** <RETURN>" will delete all events. If in the histogram or table, only the events in the display will be deleted.

The parameters are as follows:

functions — functions specifies deletion of function-type events.

variables_static — variables_static specifies deletion of variable-type events.

range_types — range_types specifies deletion of range-type events.

interval_types	interval_types specifies deletion of interval-type events.
thru	thru specifies deletion of all events within the range of event numbers indicated by the thru option. The "thru end" option deletes all events from the cursor position thru the end of the current display.
<EVENT>	<EVENT> is the number of the event to delete.
matching <PATTERN>	matching <PATTERN> specifies deletion of all events whose names match the <PATTERN>.
notmatching <PATTERN>	notmatching <PATTERN> specifies deletion of all events, except those whose names match the <PATTERN>.

Examples

delete_events

delete_events functions

delete_events variables_static

delete_events 10 **thru** 105

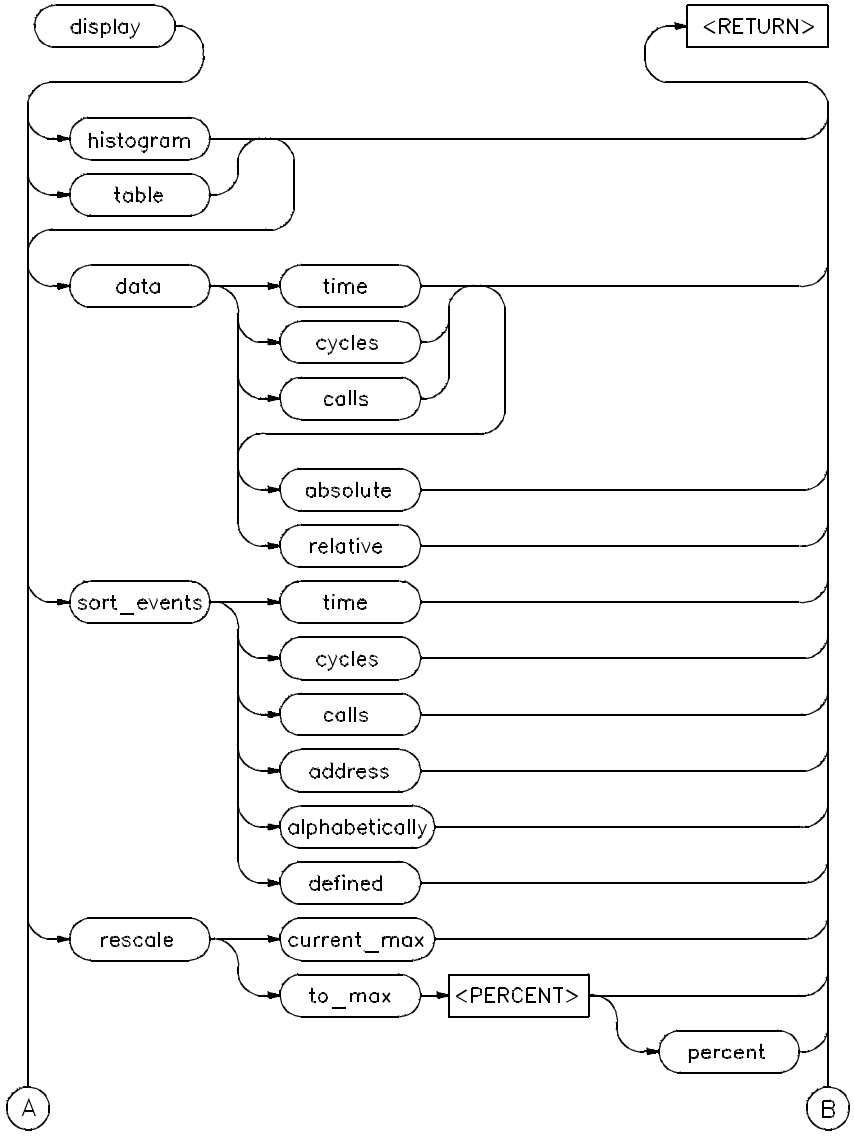
delete_events thru start

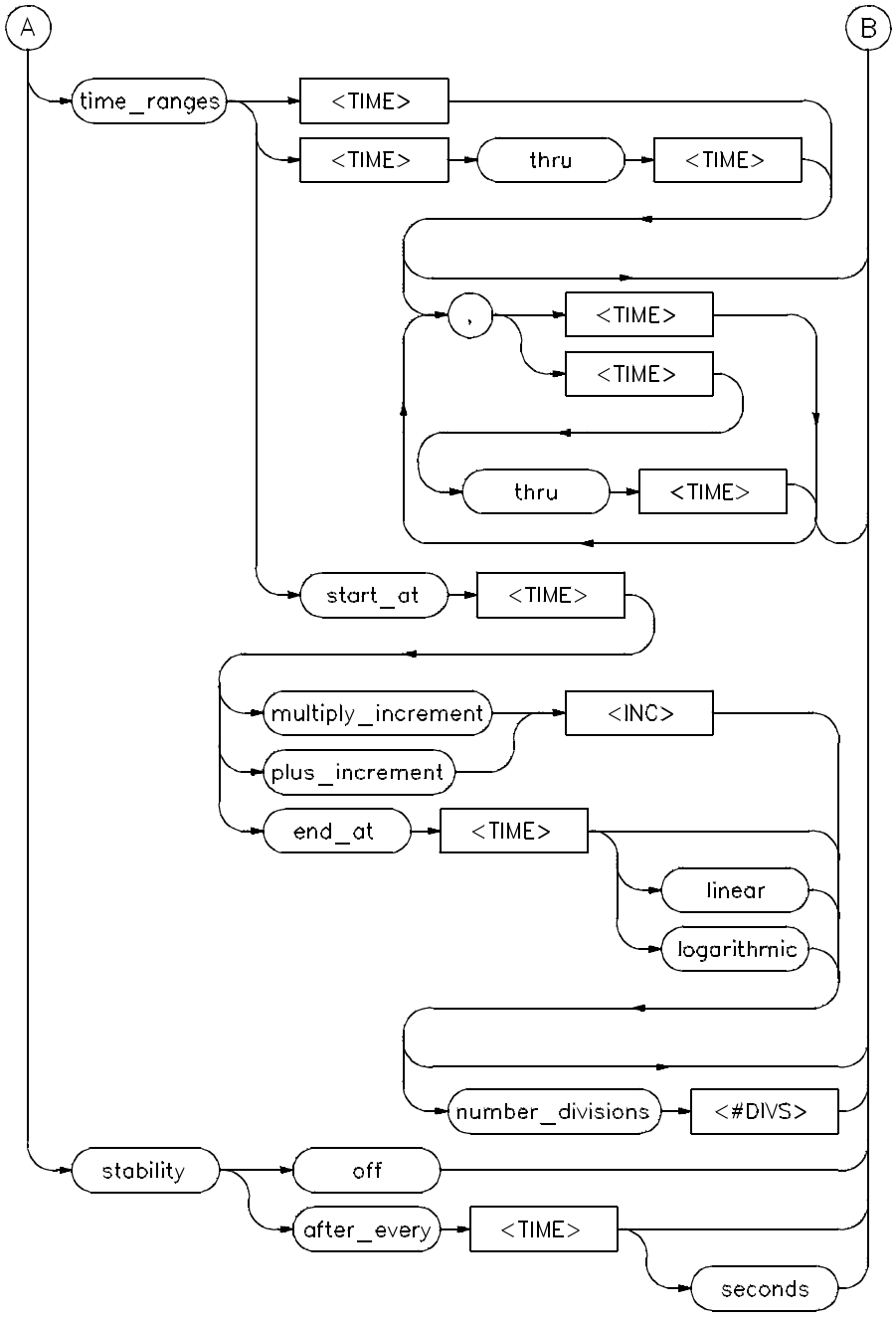
delete_events thru end

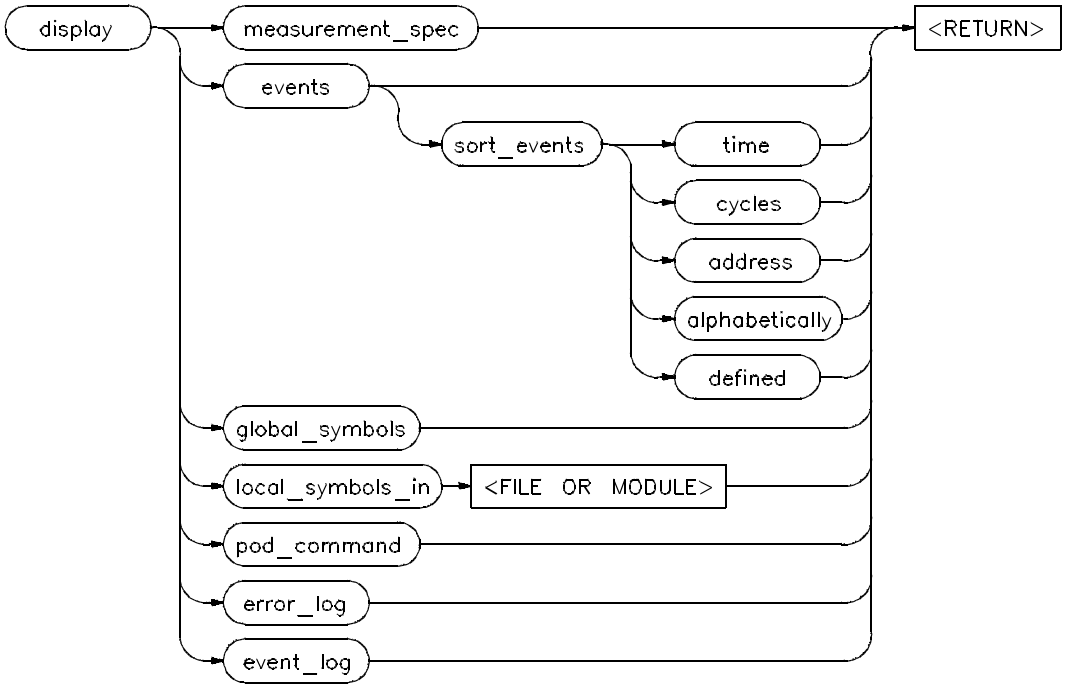
delete_events thru 205

delete_events functions thru 205 **notmatching** "sru_*"

display


display
<RETURN>
histogram
table
data
time
cycles
calls
absolute
relative
sort_events
time
cycles
calls
address
alphabetically
defined
rescale
current_max
to_max
<PERCENT>
percent
A
B


A
B
time_ranges
<TIME>
<TIME>
thru
<TIME>
,
<TIME>
<TIME>
thru
<TIME>
start_at
<TIME>
multiply_increment
<INC>
plus_increment
end_at
<TIME>
linear
logarithmic
number_divisions
<#DIVS>
stability
off
after_every
<TIME>
seconds



The display command places the information you select on your screen. You can use the roll_up, roll_down, home_up, home_down, next_page, previous_page, cursor_up, and cursor_down keys to view displayed information.

Depending on the information selected, defaults may be the options selected for the last display command.

The parameters are as follows:

histogram — histogram lets you display the current histogram.

table — table lets you display the current table.

measurement_spec — measurement_spec lets you display the current measurement specification.

events — events lets you display the list of defined events.

global_symbols — global_symbols lets you display a list of all global symbols in memory.

local_symbols_in — local_symbols_in lets you display a list of the local symbols in a specified source file.

<FILE>	<FILE> is the name of the local file (e.g. myfile.S:).
data	data lets you change the histogram to show time or cycles for activity measurements, or to show time or calls for duration measurements. In addition, you can choose to display your data in absolute or relative values.
sort_events	sort_events causes the Software Performance Analyzer to rearrange the events in any order you choose. You can have the events sorted by times, cycles, calls, addresses in numerical order, names of events in alphabetical order, or by how the events were defined or numbered.
rescale	rescale lets you change the scale of the histogram by specifying a maximum display percent.
stability	stability lets you turn on or turn off the stability calculation. It also lets you specify how often stability information is recalculated during a measurement.
time_ranges	time_ranges lets you specify the time ranges used in measurements of EXPANDED events. To see the time ranges, make a duration measurement and press the EXPAND key beside an event of interest.
error_log	error_log lets you display the most recent errors that resulted from commands entered on the command line (or supplied from command files).
event_log	event_log lets you display the most recent events (not "events_list" events) that have occurred in the emulation/analysis system. These events include software breakpoints and other changes in the status of the emulator and/or analyzer.
pod_command	pod_command lets you display a summary of the most recent commands sent to the HP64700 pod via the command, "pod_command".

Examples

display histogram data relative

d**isplay histogram sort_events time**

display table sort_events address

display events sort_events alphabetically

display measurement_spec

display global_symbols

display local_symbols_in myfile.S:

display pod_command

display error_log

display event_log

display data calls

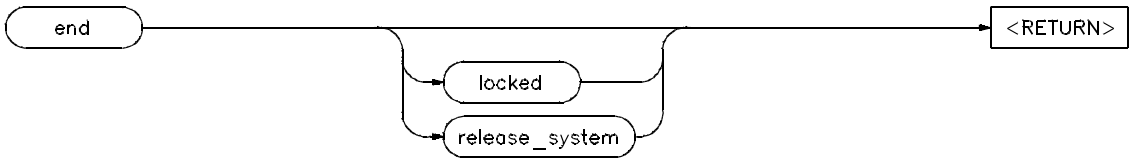
display sort_events cycles

display rescale to_max 50 **percent**

display stability after_every 60 **seconds**

display time_ranges start_at 10 **usec end_at** 20 **msec logarithmic**

end



The end command terminates the current session of Software Performance Analysis. Only a single window into the user interface is allowed. The '**end**' command ends that window.

Unless you choose "**end release_system**", the current Software Performance Analysis setup is stored so that you can resume the session later when you reenter the Software Performance Analyzer.

Note that typing "<control> D" is the same as "**end** <RETURN>".
Typing "<control> \" or "<control> |" (i.e. sending SIGQUIT to the user interface process) is the same as "**end release_system** <RETURN>".

The parameters are as follows:

<default> Return to the environment (UNIX shell, PMON, or emulation interface) where you were when you entered the '**emul700 -u skperf** <logical name>', or '**emul700 -u xperf** <logical name>' command. Save the current measurement specification, and lock the Software Performance Analyzer to the current user so that the session may be continued later.

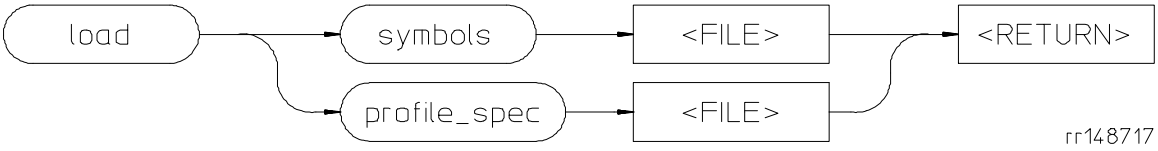
locked Close all active interfaces of the Software Performance Analyzer and emulator (in one or more windows and/or terminals). Each interface will return to the environment where its '**emul700**' command was entered. Thus, "**end locked**" is the same as entering "**end**" in each one of the windows.

release_system Close all active interfaces of the Software Performance Analyzer and emulator (in one or more windows and/or terminals). Each interface will return to the environment where its '**emul700**' command was entered. In addition, the Software Performance Analyzer and emulator will be unlocked so they can be used by someone else on your UNIX system. The information needed to continue your session will be lost. (If you do not release the system, no other people can use it.)

Example

```
end
end locked
end release_system
```

load



The load symbols command lets you load a symbol data base into the user interface.

The load profile_spec command lets you load a previous profile specification along with its captured data into the Software Performance Analyzer. After loading a profile_spec, you can use the display command to view the data that was stored, or you can perform new measurements using the setup that was stored in the profile_spec file.

In addition, you can load a configuration and an absolute file if your emulator and/or debugger has software version number 5.00, or greater. Refer to your emulator manual for full syntax and command description.

The parameters are as follows:

profile_spec — profile_spec lets you load a profile specification file that you previously created using the store profile_spec command. To start a measurement after a new profile_spec is loaded, simply enter the "profile" command.

symbols — symbols lets you load a symbol data base associated with an absolute file without loading the absolute file.

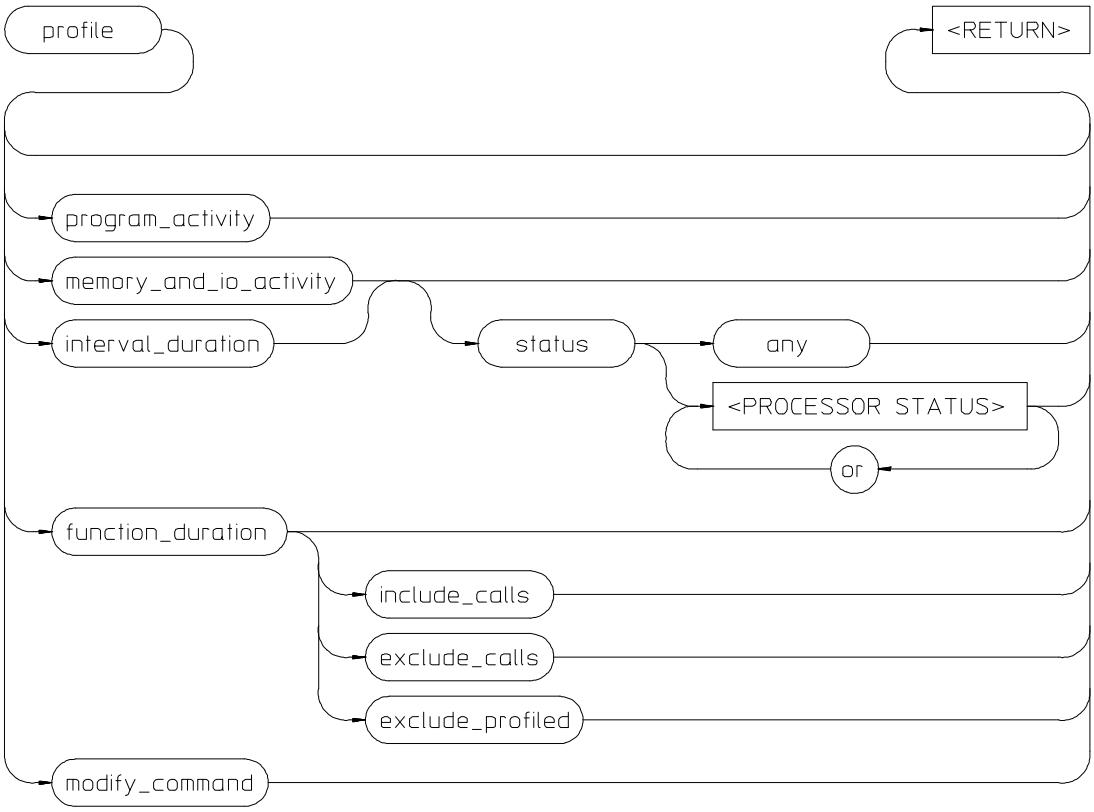
<FILE> — <FILE> is the pathname of the file (symbols or profile_spec) to be loaded from the system disk. You do not need to include the file name suffix (".Ys" for a symbol data base, and ".PS" for a profile specification). The file name suffix will be automatically appended to the file name you specify.

Examples

load symbols myexecutable

load profile_spec profspec

profile



The profile command causes the Software Performance Analyzer to begin the specified measurement. If you enter "**profile**" alone, the Software Performance Analyzer will start the same measurement it made after the last profile command. If this is the first profile measurement, the Software Performance Analyzer will start a default measurement. If you enter "**profile modify_command**", the Software Performance Analyzer will recall the last profile command to the command line. If no events have been defined when you enter "**profile**", the Software Performance Analyzer will create a list of events by searching the symbol data base for functions and static variables.

The default profile command is: **profile program_activity**

The parameters are as follows:

program_activity	program_activity specifies that the Software Performance Analyzer record program execution for selected events that represent functions and address ranges.
memory_and_io_ activity	memory_and_io_activity specifies that the Software Performance Analyzer record use of the selected events that represent static variables and ranges. The Software Performance Analyzer will record memory cycles and cycle times for each of the selected events. You can further qualify the types of cycles that the Software Performance Analyzer will record by specifying status conditions.
interval_duration	interval_duration specifies that the Software Performance Analyzer record the amount of time between cycles at the beginning and at the ending of the specified intervals. You can qualify the types of cycles that the Software Performance Analyzer must look for at the interval start and interval end addresses by specifying status conditions.
function_duration	function_duration specifies that the Software Performance Analyzer record the amount of time it takes to execute selected events that represent functions. A function_duration measurement can be further qualified by "including calls" or "excluding calls" to other functions. The time recorded for a function in an inverval_duration measurement is typically the same as the time recorded in a function_duration including_calls measurement.
exclude_calls	exclude_calls specifies that a function_duration time does not include any time spent executing code in other functions that were called by this function. Note also that all time spent servicing interrupts will be excluded from the function duration.
exclude_profiled	exclude_profiled specifies that a function_duration time does not include any time spent executing code in other functions that were called by this function, if those other functions are also included in this measurement. However, time spent executing code in other functions that are not part of this measurement will be added to the time recorded for this function.
include_calls	include_calls specifies that a function_duration time must include all of the time spent executing code in other functions that were called by this function.
status	status further qualifies a measurement. In memory_and_io_activity measurements, status specifies the type of cycles and time to be recorded. If you specify "status any", all cycles that match the event being sampled will be recorded. If you specify a single type of status, only cycles that also match the type you specify will be recorded. In an interval_duration measurement, status specifies the type of cycles to begin and end a duration measurement. Using status, you can measure time between writes or reads or program execution.

Status specifications depend on the processor you are testing. Use the "set status_qualification" command to set global status qualifications for your processor, such as "supervisor" cycles, or "user" cycles.

Examples

profile

profile program_activity

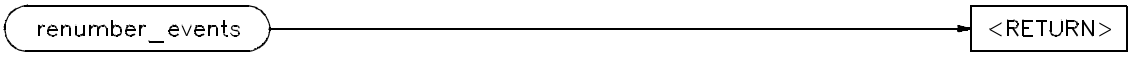
profile memory_and_io_activity status memread or memwrite

profile interval_duration status prog

profile function_duration exclude_calls

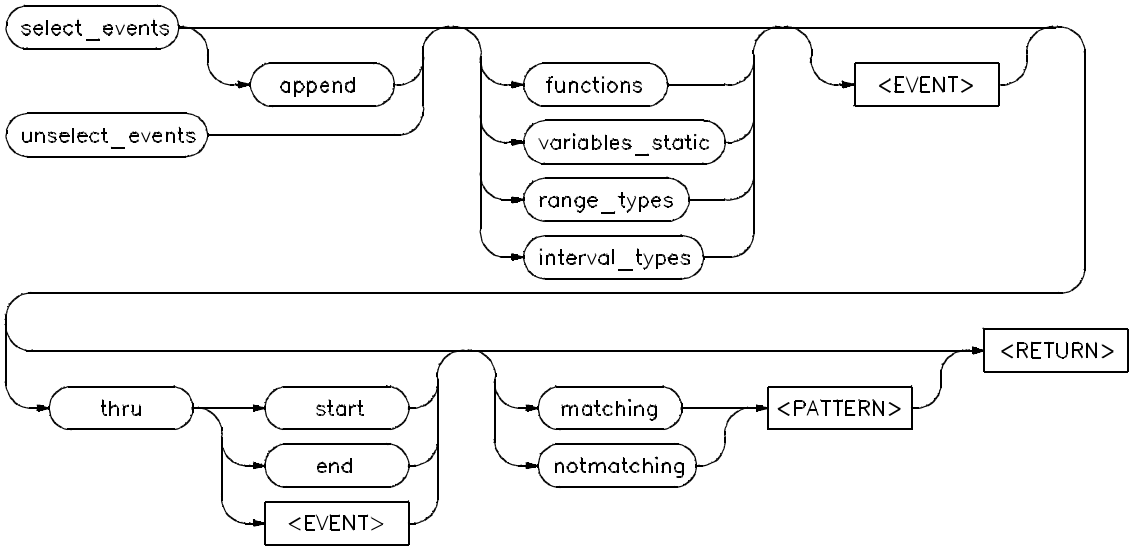
profile modify_command

renumber_events



The renumber_events command lets you renumber the events in the present display of the Software Performance Analyzer. If you renumber the events and then sort them by some other criteria (such as time, calls, cycles, or addresses), the event numbers may be out of order.

select_events



The select_events command lets you select groups of events to be included in a Software Performance Analyzer measurement. You can qualify the events to be selected by specifying a type of event, a range of event numbers, or the event names.

Events that are selected will be included in the next measurement (if they are appropriate for the measurement type, and if the number of events does not exceed the hardware resource limit of the Software Performance Analyzer).

The command "**select_events** <RETURN>" will select all events.

The parameters are as follows:

append — append specifies that the events selected by this command will be added to the events that are already selected. If you do not use the append option, all events will first be unselected, and then only the events specified in your command will be selected.

functions — functions specifies only function-type events are to be selected.

variables_static — variables_static specifies that only variable-type events are to be selected.

range_types	range_types specifies that only range-type events are to be selected.
interval_types	interval_types specifies that only interval-type events are to be selected.
thru	thru is used to make a number range. All events with numbers in the number range will be selected. The "thru end" command will select all events from the cursor position thru the end of the current display.
<EVENT>	<EVENT> is the event number to select.
matching <PATTERN>	matching <PATTERN> specifies selection of all events whose names match the <PATTERN>.
notmatching <PATTERN>	notmatching <PATTERN> specifies selection of all events, except those whose names match the <PATTERN>.

Examples

select_events

select_events functions

select_events append functions matching "mem*"

select_events variables_static

select_events 10 **thru** 105

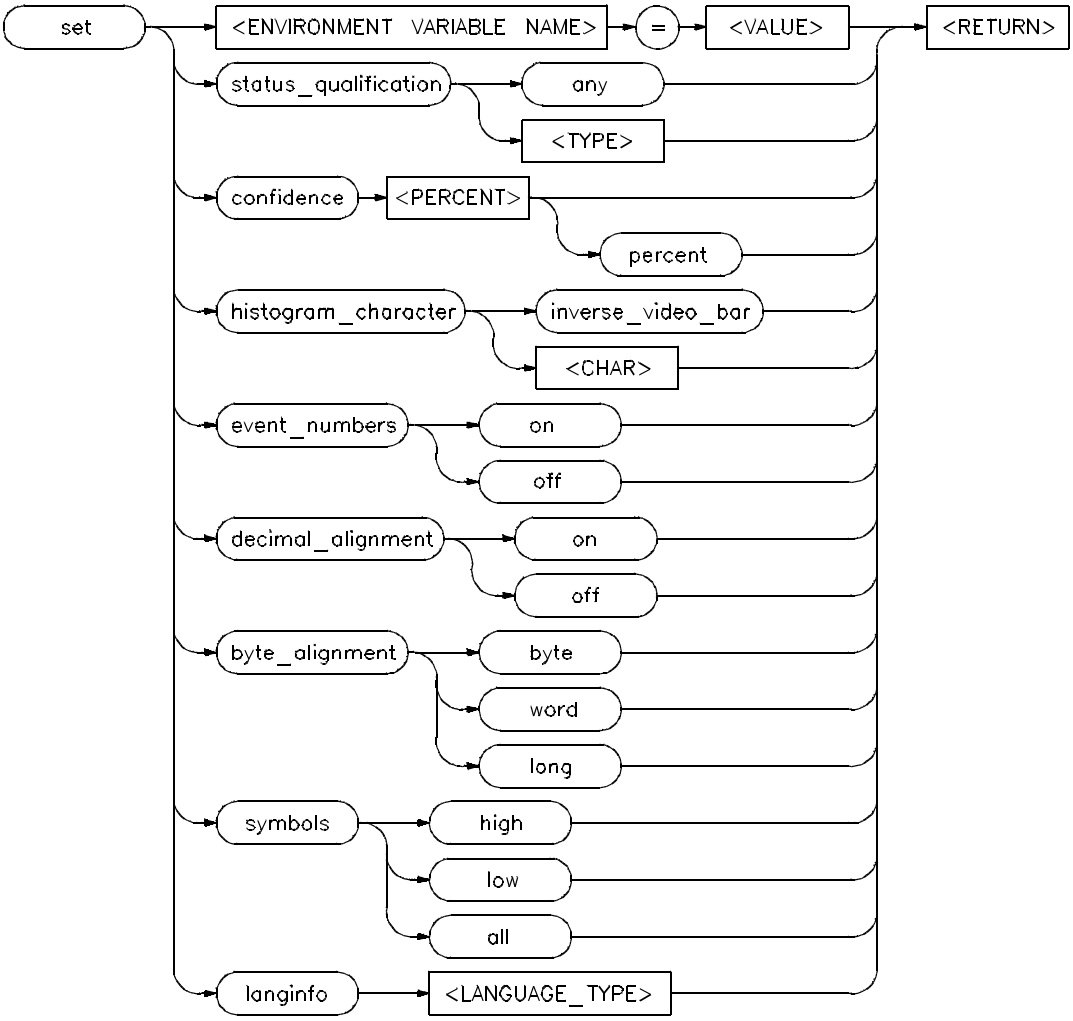
select_events thru start

select_events thru end

select_events append thru 205

select_events functions thru 205 **notmatching** "sru_*"

set



NOTES FOR <ENVIRONMENT VARIABLE NAME>:
1. <VALUE> = <NUMERICAL VALUE> OR <STRING>.
2. PUT <STRING> IN QUOTES IF IT HAS INTERNAL SPACES.

set

The set command modifies the measurement and display options for the Software Performance Analyzer. In addition, you can define shell environment variables with the set command (explained on the next sheet, titled, "set <Environment variable name>").

Initial default values are as follows:

symbols are set high
confidence is set at 95 percent
event_numbers are on
decimal_alignment is on
histogram_character = inverse_video_bar
byte_alignment is preset to match the normal operating mode of your emulator
status_qualification any
langinfo C

The parameters are as follows:

<VAR> — Refer to the next page for a discussion of "set <Environment variable name>" and examples of its use.

symbols — symbols lets you specify the type of symbols that you wish to use as high (user defined in the source file), low (assembly level), or both.

confidence — confidence specifies the level of confidence required in the value of stability that is generated. Setting the confidence to a higher percent will make the analyzer run longer to reach a given level of stability.

event_numbers — event_numbers determines if event_numbers are to be displayed along with event_names. If event_numbers are turned off, the additional spaces can be used to display more characters of the event names.

decimal_alignment — decimal_alignment forces the decimal points to be aligned in the columns of values. With this selection, the values might not be displayed to three significant digits.

histogram_character — histogram_character lets you define the character to be used to make the bars on a histogram display. Either select "inverse_video_bar", enter a decimal value between 33 and 127, or quote the character to use. With some fonts, the character 127 makes a good histogram bar. If you select "inverse_video_bar", the inverse-video bars will be used to make the histogram.

byte_alignment — byte_alignment lets you specify the boundaries of address ranges to be aligned by byte, word (even bytes), or long (every 4th byte). The Intel 80960 Sx emulator can align to byte, short (even bytes), or word (every 4th byte).

status_qualification — status_qualification defines the global status qualification to be used with the Software Performance Analyzer. This command is only available if the emulator you are using has multiple operating modes such as, "supervisor" or "user".

langinfo — langinfo defines the language type being used. Setting this to match the language allows the symbols to be used properly by the Software Performance Analyzer.

Examples

set symbols low

set confidence 99 **percent**

set event_numbers off

set decimal_alignment on

set histogram_character '*'

set histogram_character 127

set byte_alignment byte

set status_qualification user

set langinfo ADA

set <Environment variable name>

You can use the set <Environment variable name> command to define system environment variables for use within the analysis session. For example, if you enter the command:

set x = /usr/hp64000/demo/spa/demo1

then, "$x" (the string represented by "x") may be used in place of "/usr/hp64000/demo/spa/demo1" in your commands. For example:

cd $x will perform the same function as **cd /usr/hp64000/demo/spa/demo1**

If you set <Environment variable name> equal to a string that contains internal spaces, put the string in quotation marks.

(If you defined and exported HP-UX environment variables before the emulation session, you can use those environment variables in the Software Performance Analyzer. You won't have to execute new "set <Environment variable name>" commands from within this session.)

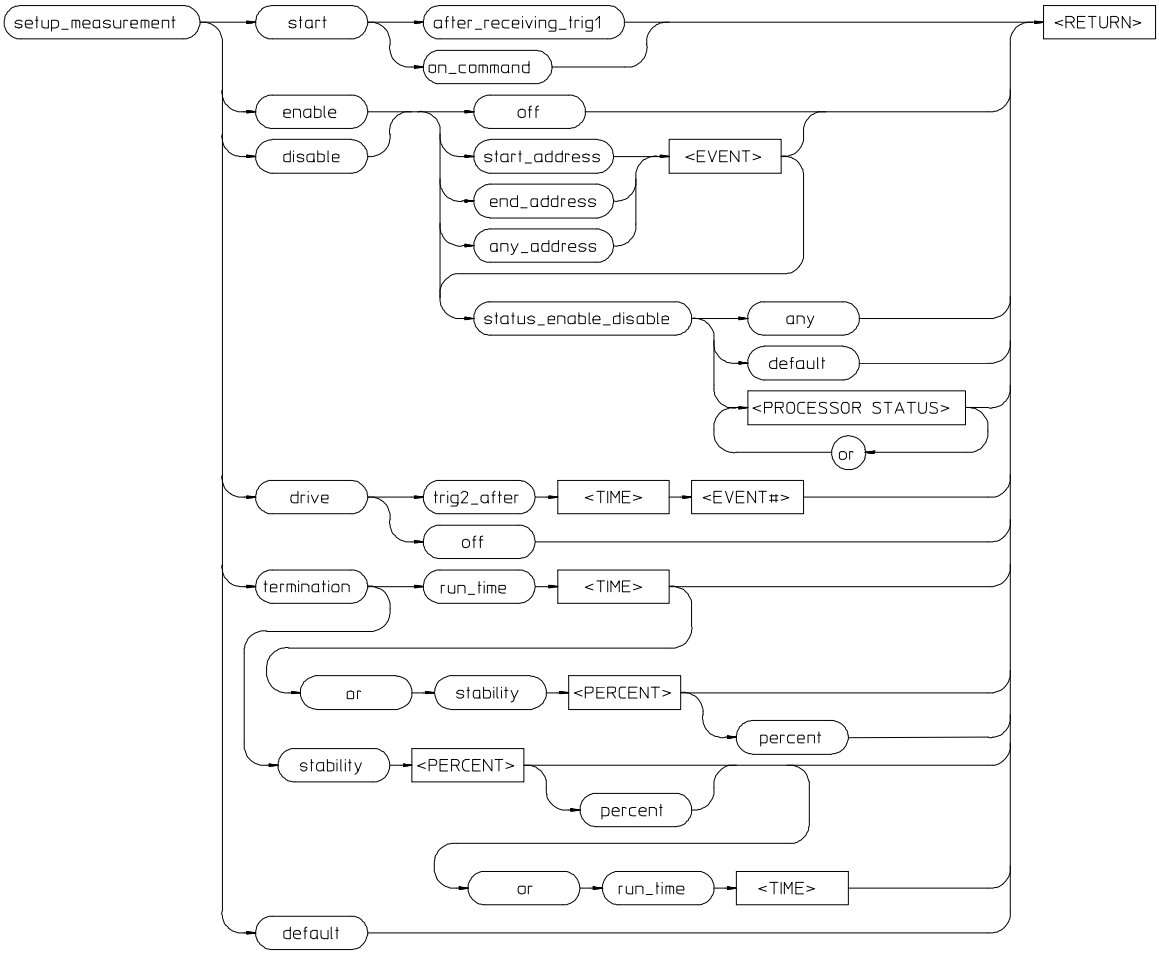
Examples

set emuldir = /users/<yourlogin>/emul68000
set loadrt = "load symbols runtest"
set disph = "display histogram sort_events time"

The above commands allow you to use:

cd $emuldir
$loadrt
profile
$disph

setup_measurement



setup_measurement

The setup_measurement command lets you define several qualifications for your Software Performance Analyzer measurement. These qualifications include when to start the measurement, conditions on which to enable or temporarily disable the measurement, trigger conditions, and when to terminate the measurement.

By default, the measurement starts when you enter the profile command.

The parameters are as follows:

start — start lets you specify a condition on which to start a profile measurement.

enable — enable lets you specify an address condition on which the Software Performance Analyzer can begin (or continue) to gather data.

disable — disable lets you specify an address condition on which to temporarily suspend the gathering of software performance data. Disable is only available during function_duration and interval_duration measurements.

status_enable_disable — status_enable_disable lets you select a status condition to qualify the enable and disable addresses. If you specify a status_enable_disable, it will qualify both your enable and disable addresses. One of the following three conditions can be selected:

- A status condition selected from those appearing on the softkeys.
- "default" which causes both the enable and disable conditions to be qualified on the status used by the profile command.
- "any" turns off the status qualification for both the enable and disable addresses.

drive — drive lets you set up the Software Performance Analyzer to drive trig2 after a particular event has executed continuously for a specified period of time. The emulator can use the signal from trig2 to trigger its emulation bus analyzer or to cause an emulation break. Drive is only available during function_duration or interval_duration measurements.

termination — termination lets you specify conditions that terminate the measurement. The Software Performance Analyzer can be set to terminate its measurement after a specified time or after a stability level is reached. The interface must be running in order to terminate a measurement on a stability condition.

default — default sets the entire measurement specification of the Software Performance Analyzer to initial conditions.

Examples

setup_measurement start after_receiving_trig1

setup_measurement enable start_address main

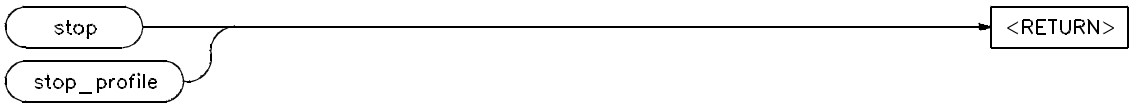
setup_measurement disable any_address wait_for_io

setup_measurement termination run_time 5 **minutes or stability** 98 **percent**

setup_measurement drive trig2_after 100 **msec** int_loop

setup_measurement default

stop_profile



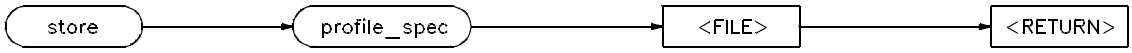
The stop_profile command causes the Software Performance Analyzer to terminate its present profile measurement.

Examples

stop_profile

stop

store



The store command lets you store the present measurement setup and the present profile data in a profile specification file (filename.PS file).

<FILE> is the name of the file that stores the measurement setup and profile data. The store command creates a new file of the specified name. It will overwrite an existing file by that name if entered from a command file. A prompt will ask permission to overwrite the old file if the store command is entered from the keyboard.

Examples

store profile_spec myprofile

symbol_offset

A symbol and offset expression is a combination of symbols, operators, numerical values, and parentheses specifying an address to be used in Software Performance Analyzer commands. The expressions can be any combination of "symbol + offset" or "symbol - offset" where the offset is a combination of symbols, operators, and numerical values.

The parameters are as follows:

--SYMB-- --SYMB-- is a symbolic reference to an address, file, or other value. Symbols may be UNIX paths, referenced line numbers in a file, file segments (prog, data, common), or global and local symbols.

\+ Algebraic addition (plus).

\- Algebraic negation (minus).

( ) Parentheses may be used in offset expressions. For every opening parenthesis, a closing parenthesis must exist.

<NUMBER> <NUMBER> is a numeric value in any base (decimal, hex, octal, or binary).

<OP> <OP> is an algebraic or logical operand. <OP> may be:

+	(plus)
-	(minus)
/	(divide)
*	(multiply)
|	(logical OR)
&	(logical AND)
mod	(modulo)
~	(logical NOT)

Examples

```
DISP_BUF + 5
SYMB_TBL + (OFFSET / 2)
START
myfile.S: line 30
file1.c:PROCEDURE.LOCAL_VAR - (2 * 100h + 20h)
```

--SYMB--

--SYMB-- is a symbolic reference to an address, file, or other value. Symbols may be UNIX paths, referenced line numbers in a file, file segments (prog, data, common), or global and local symbols.

The parameters are as follows:

file	file is a UNIX path specifying a source file. If no file is specified, and the identifier is not a global symbol in the executable file that was loaded, then the default file is assumed (the last absolute file specified by a display local_symbols_in command).
identifier	identifier is the name of an identifier as declared in the file.
line	line specifies that the following value is a line number.
<LINE>	<LINE> prompts you to enter a line number.
scope	scope is the name of the portion of the program where the specified identifier is defined or active.
segment	segment indicates that the following string specifies a program segment (prog, data, common) in the source file.
<SEGMNT>	<SEGMNT> prompts you to enter a program segment. The softkeys appear as prog, data, and common. The actual text that appears when the softkeys are pressed is "PROG", "DATA", and "COMMON". A user-defined segment name can be used, as well.
<SYMBOL>	<SYMBOL> prompts you to enter a symbol name in one of the following forms:

symb [procedure {entry | text}]
[file] segment <SEGMNT>
[file] line <LINE>

where: file is: [.|:] [{filename:|scope.} ...] filename:
symb is: [.|:] [{filename:|scope.} ...] scope
scope is: identifier[(type)]

A <SYMBOL> may also be followed by "start" or "end" when an address range is returned, so that the desired address may be chosen.

<SYMBOL> start
<SYMBOL> end

(type) type differentiates between identifiers with the same name but of different type (filename, fsegment, module, procedure, procspecial, static, or task).

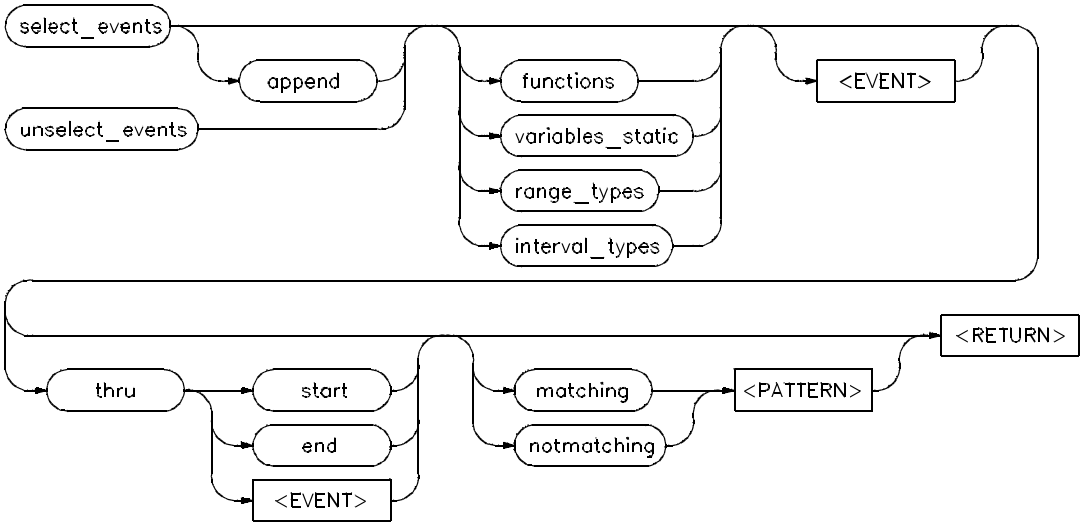
: A colon (:) separates the UNIX path specifier from the line, segment, or symbol specifier. In the case of a line or segment selection, there must be a space after the colon. For a symbol specifier, there must NOT be a space after the colon if the path specifier is present, otherwise there may or may not be a space after the colon.

Note that if a path specifier precedes the :, there should NOT be a blank between them.

Examples

```
module.S: line 5

keybd.S:scankeys.LOOP1

generic.C: segment data

something_global

:main(procedure) start

line 151

sample.C: segment PROG

main.c:index

file1.c:file2.c:        # file2.c as included in file1.c

:package1."file.c":     # the file "file.c" which is a child of package1

"a:b.c":alpha           # the symbol alpha in the file named a:b.c
```

unselect_events



The unselect_events command lets you unselect events from a Software Performance Analyzer measurement. You can identify the events to be unselected by specifying an event type, a range of event numbers, or the event names. When events are unselected, they are removed from the current measurement results, and will not be part of a future measurement. Unselected events are still in the "events list" and can be selected again, if desired.

The command "**unselect_events** <RETURN>" will unselect all events.

The parameters are as follows:

functions — functions specifies that only function-type events will be unselected.

variables_static — variables_static specifies that only variable-type events will be unselected.

range_types — range_types specifies that only range-type events will be unselected.

interval_types — interval_types specifies that only interval-type events will be unselected.

thru — thru is used to identify a range of event numbers. All events in the event number range will be unselected. The "thru end" command will unselect all events from the present cursor position thru the end of the display.

<EVENT>	<EVENT> is the number of the event to be unselected.
matching <PATTERN>	matching <PATTERN> specifies unselecting all events whose names match <PATTERN>.
notmatching <PATTERN>	notmatching <PATTERN> specifies unselecting all events, except those whose names match <PATTERN>.

Examples

unselect_events

unselect_events functions

unselect_events functions matching "mem*"

unselect_events variables_static

unselect_events 10 **thru** 105

unselect_events thru start

unselect_events thru end

unselect_events thru 205

unselect_events functions thru 205 **notmatching** "sru_*"

11

Error Messages

Error Messages

This chapter contains descriptions of error mesages that can occur while using the Software Performance Analyzer. The error messages are listed in alphabetical order. Each description includes the cause of the error message and the action you should take to correct the condition.

Note that most error messages are recorded into the error log display. You can view them there, if necessary. Also, the Multiple Event Definition Summary shown in the error log is discussed at the end of this chapter.

Software Performance Analyzer Messages

Address not found

Cause: This message occurs in emulators that have segment:offset addressing modes. The Software Performance Analyzer could not create a valid address based upon the segment address that it was passed.

Action: Enter a correct segment address.

Address range overlaps event XX; delete overlapped events?

Cause: You have tried to define a new event whose address overlaps at least the event whose number appears in the message. Other events may also be overlapped by your new event. Do you wish to delete all of the overlapping events?

Action: Answer either yes or no. If you answer yes, the Software Performance Analyzer will delete all of the overlapping events and define your new event. If you answer no, the Software Performance Analyzer will make no changes and will delete your new event definition.

Any_address is not valid with type 'interval'

Cause: You tried to set up an enable or disable condition using an interval-type event and selected "any_address". Interval-type events do not have a range associated with them. The resulting enable specification has been converted to enable on the starting_address of the interval_type event.

Action: Select either the starting_address or ending_address of interval-type events when using them in enable and disable specifications.

Cannot find file: /usr/hp64000/inst/emul/64742A/etc/info64708A

Cause: The software for the Software Performance Analyzer was not properly installed or has been removed.

Action: Reinstall the User Interface Software for the Software Performance Analyzer. Installation is discussed in Chapter 17.

Cannot find file: /usr/hp64000/inst/emul/64708A/tables/confidence

Cause: The software for the Software Performance Analyzer was not properly installed or has been removed.

Action: Reinstall the User Interface Software for the Software Performance Analyzer. Installation is discussed in Chapter 17.

Cannot initialize performance analyzer

Cause: The Software Performance Analyzer cannot be initialized properly.

Action: Try repowering the HP64700 emulation system.

Cannot start multiple performance analyzer windows

Cause: You have tried to start the Software Performance Analyzer when it is already running in another window on your screen. The Software Performance Analyzer can only have one active window to an emulator at a time.

Action: Find the original window and close it, or execute "end release_system" to restart the emulation system.

Command not allowed while a measurement is in process

Cause: You have tried to enter a command that cannot be executed while a measurement is in process.

Action: Stop the Software Performance Analyzer measurement. Then the command can be executed.

Defined XX (of XXX) events; display error_log for details

Cause: This is an information message that appears when defining multiple events. It indicates the number of events that were defined from the total number of possible events (functions and static variables) found in the source file.

Deleted XX events incompatible with current symbol database

Cause: This is an informational message about the number of events that had to be deleted after the profile command was entered. These events had to be deleted because the loading of a new symbol database made them invalid. They are invalid because either:

- There are no symbols in the database for these events.
- They now have addresses that overlap other events.
- They now are trivial functions (functions whose starting and ending addresses are aligned to the same identical address).

Disable event not valid, address overlaps enable

Cause: Your disable and enable events have overlapping addresses. Disable and enable events cannot have overlapping addresses. Both events must have unique addresses or address ranges.

Action: Redefine either your enable event or your disable event so that both events have unique addresses or address ranges.

Disable event not valid with activity measurements

Cause: You tried to run an activity measurement that had a disable event as part of its specification. Disable events cannot be used in activity measurements. Disable events can only be used in function-duration and interval-duration measurements.

Action: Remove the disable event from your activity measurement specification.

Disable not available without a valid enable

Cause: The enable event is not valid. Therefore, the disable event is not available.

Action: Correct your definition of the enable event. This will make the disable feature available.

Disable shares addresses with event XX

Cause: The disable event and the selected event named in this message have identical or overlapping addresses. The disable event must have addresses that are different from all of the events that are to be measured.

Action: Redefine the disable event or unselect the event whose addresses overlap the disable event. Then make the measurement.

Enable and Disable events cannot be profiled

Cause: You tried to include your enable and/or disable event in the profile measurement. Enable and disable events cannot be selected to be included in the current profile measurement.

Action: Unselect the enable and disable events. Then make your measurement.

Enable shares addresses with event XX

Cause: The enable event and the selected event named in this message have identical or overlapping addresses. The enable event must have addresses that are different from all of the events that are to be measured.

Action: Redefine the enable event or unselect the event whose addresses overlap the enable event. Then make the measurement.

Event does not exist

Cause: You have not correctly spelled the event name, or have entered an event number that does not exist.

Action: Check the name and number of your event on the events list display. An event name can be up to 40 characters long and may start with an underscore "_". It might be easier to use the event number instead of the event name.

Event name is invalid

Cause: You are trying to define an invalid name for an event.

Action: Enter a valid event name in your definition. Valid event names consist of alphanumeric characters and begin with an alpha character or underscore "_".

Event number does not exist

Cause: You entered an event number that does not exist. No events have been defined for the event number you entered.

Action: Check the number of your event on the events list display. Then enter the correct number in your command.

Event number is invalid (1 thru 1000)

Cause: You are trying to enter an event number that is not between 1 and 1000.

Action: Make sure you define events whose numbers are between 1 and 1000.

Event rate overflow

Cause: The defined events are coming in too fast or have come in too fast at some point during the measurement. This has not allowed time for the Software Performance Analyzer to capture them and perform its calculations. Some function durations are lost.

Action: Select fewer functions to measure in the present measurement. Refer to Chapter 6 for a detailed discussion about the event rate overflow condition.

Event rate underflow

Cause: The defined events are coming in too slowly for the Function Duration measurement. Some function durations are lost. Function entry or exit points must occur at a rate of at least 1 every 1.25 second.

Action: Select a function that is executed at least once every second and include it in the measured functions. A second solution is to define interval events to represent the functions you want to measure and use the interval duration measurement mode. Event rate underflow will not occur in interval duration measurements. Refer to Chapter 6 for a detailed discussion about the event rate underflow condition.

XX events incompatible with new symbol database

Cause: This is an informational message about the number of events that will have to be deleted when the profile command is entered. These events will have to be deleted because the loading of a new symbol database makes them invalid. They are invalid because either:

- There are no symbols in the database for these events.
- They now have addresses that overlap other events.
- They now are trivial functions (functions whose starting and ending addresses are aligned to the same identical address).

File could not be opened

Cause: File could not be opened for read or write.

Action: Check your permissions on the file.

File not found

Cause: File or path to file does not exist.

Action: Check the spelling in your command.

Function is trivial and cannot be measured

Cause: You are trying to measure a function that is too short for the Software Performance Analyzer to measure. In many cases, the start and end addresses of the function are adjusted and aligned to the same address. This often occurs when a function consists of only a return command.

Action: Make sure the function you are trying to measure is long enough to have unique word or long-word addresses for its start and end addresses.

Function marker start and end addresses are identical

Cause: The addresses of the function-start and function-end markers were found to be identical. This is an invalid condition.

Action: Check your definition for the symbols that the function should use to define marker addresses. For further information on defining and using markers, refer to Chapter 7.

Function start and end addresses are identical

Cause: You are trying to measure a function that is too short for the Software Performance Analyzer to measure. In many cases, the start and end addresses of the function are adjusted and aligned to the same address. This often occurs when a function consists of only a return command.

Action: Make sure the function you are trying to measure is long enough to have unique word or long-word addresses for its start and end addresses.

HP64700 I/O error; communications timed out

Cause: The HP 64700 cardcage has lost communication with your host.

Action: Reconnect the communication channel and/or repower the HP 64700 emulation system.

Interval overlaps event XX: delete overlapped events?

Cause: You have tried to define a new interval event whose address overlaps at least the event whose number appears in the message. Other events may also be overlapped by your new interval event. Do you wish to delete all of the overlapping events?

Action: Answer either yes or no. If you answer yes, the Software Performance Analyzer will delete all of the overlapping events and define your new event. If you answer no, the Software Performance Analyzer will make no changes and will delete your new event definition.

Invalid syntax in HPSPAADJUST Environment Variable

Cause: You used incorrect syntax when you tried to export your HPSPAADJUST environment variable.

Action: Export your HPSPAADJUST environment variable using correct syntax. Correct syntax is: HPSPAADJUST="<NO.1> <NO.2>".

Where:
<NO.1> is the number of bytes to adjust forward from the start address of the function.
<NO.2> is the number of bytes to adjust backward from the end of the function.

Typical HPSPAADJUST values are "2 0" for 68000, 68010, 68302, 6833x, and 68340; and "6 2" for 68020 and 68030. Refer to Chapter 7 for complete information about using HPSPAADJUST.

Invalid syntax in HPSPAMARKERS Environment Variable

Cause: You tried to export a specification for HPSPAMARKERS using incorrect syntax.

Action: Export your HPSPAMARKERS environment variable using correct syntax. The correct syntax for an HPSPAMARKERS specification is either:
HPSPAMARKERS="no", or
HPSPAMARKERS="yes start_prefix[+offset] end_prefix[+offset]".

Where:
start_prefix is the prefix to be added to a function name to create the name of the variable that is written when the function is entered.
end_prefix is the prefix to be added to a function name to create the name of the variable that is written when the function exits.
[+offset] is an optional number of bytes to adjust from the variable address.

A typical HPSPAMARKERS value for some MRI compilers is:
"yes _r_ _r_+2"

Limit of 10 expanded events exceeded, only 10 are now expanded

Cause: A new measurement found that more than 10 events were expanded.

Action: There is nothing you need to do. The display will automatically unexpand all events after the first 10 expanded events.

Limit of 10 expanded events reached, unexpand other events

Cause: You tried to expand an event when you already have 10 events expanded. Only 10 events can be expanded at one time.

Action: Unexpand one or more events before trying to expand this new event.

Loaded symbol data base

Cause: This information message tells you that a symbol database has been loaded.

Measurement complete - stable

Cause: Software Performance Analyzer measurement is complete due to a stability-termination condition.

Measurement complete - time

Cause: Software Performance Analyzer measurement is complete due to a time-termination condition.

Measurement in process

Cause: Software Performance Analyzer measurement is running.

Measurement spec has changed, unable to store data with spec

Cause: This is a warning message to indicate that only the specification has been saved and not the data from the last measurement. The measurement specification has been changed and the data associated with the last measurement cannot be saved with the current measurement specification.

Action: Make another measurement and save it before you change any aspect of the measurement specification.

Measurement stopped

Cause: Software Performance Analyzer measurement has been stopped by the user.

No available event locations

Cause: You are trying to define more than 1000 events. No more than 1000 events can be defined at one time.

Action: Delete some of the existing events. This will allow resources to define additional events.

No events are valid for this measurement

Cause: Either you have not defined valid events for this measurement or you have not selected the appropriate type of events for this measurement.

Action: Make sure valid events are defined and selected for the measurement you wish to make. The following list shows measurement types and the events that can be included in the measurements:

- Program Activity - Events that represent either functions or ranges.
- Memory and I/O Activity - Events that represent either static variables or ranges.
- Function Duration - Events that represent functions.
- Interval Duration - Events that represent intervals.

No such symbol: <symbol>

Cause: The symbol <symbol> you entered in your command does not exist.

Action: Check the spelling of the symbol name.

No valid event on selected line

Cause: You are trying to select a line from the Graphical User Interface that does not have an event.

Action: Select only the event lines when using the popup windows on the histogram, table, and events list displays.

No valid symbol on selected line

Cause: You are trying to select a line from the global and local symbols display that does not have a valid symbol.

Action: Select only symbol lines when using the popup menu of the global and local symbols display.

Not allowed while a measurement is in process

Cause: You tried to enter a command for an action that cannot be taken when a measurement is in process.

Action: Stop the measurement. Then you can enter your command.

Not a compatible profile specification file - load aborted

Cause: The profile specification file is not compatible with the current software version and cannot be loaded. This happens when a more recent profile specification file is loaded into an older version of the Software Performance Analyzer Software.

Action: Reinstall the User Interface Software for the Software Performance Analyzer. Installation is discussed in Chapter 17.

Not a valid profile specification file - load aborted

Cause: The profile specification file <file.PS> has been corrupted and cannot be loaded.

Action: Delete the file.

Possible 64708A hardware problem, run performance verification

Action: Try the following sequence:

1 Exit the user interface.

2 Cycle power on the emulation card cage.

3 Restart the emulation system.

4 Enter the Software Performance Analyzer.

If the above steps do not solve the problem, try reseating the HP 64708A analyzer card.

If reseating the analyzer card does not solve the problem, and you are sure that all software is properly installed, run the performance verification procedures described in Chapter 17 of this manual.

Profile command sets status_enable_disable to default

Cause: This error message appears when the current selected profile measurement requires that the enable and disable status be changed. This error only occurs with the Intel 80286 emulator when you have requested a status that cannot be measured with the current profile command.

Action: Review your status specification and select one that is appropriate for the profile measurement. The enable and disable status must be of type "exec" for program_activity and function_duration measurements. In interval_duration and memory_activity measurements, if you have selected "exec" as the measurement status, the enable and disable status must be of type "exec". Otherwise, if you have selected any other measurement status, the enable and disable status can be of any type except "exec".

For Intel 80286 Only:

Measurement	Measurement Status	Enable and Disable Status
program_activity	exec	default or exec
function_duration	exec	default or exec
memory_activity	exec	default or exec
memory_activity	any status except exec	any status except exec
interval_duration	exec	default or exec
interval_duration	any status except exec	any status except exec

Slow clock

Cause: The emulator is not sending analysis clocks to the Software Performance Analyzer.

Stack overflow

Cause: The internal stack of the Software Performance Analyzer is not finding appropriate exit points for the function entry points it has found. This can occur when your emulator is prefetching the starting addresses of measured functions in a manner that is not recognized by the prefetch-correction circuitry of the Software Performance Analyzer.

Action: Make the following checks to correct this problem:

- Make sure your symbol data base is up to date.
- Make sure your functions are defined properly.
- Compile your executable file with the debug (-OG) options; these insert NOP padding between function exits and function entries.
- Select fewer functions for the present measurement.

Refer to Chapter 6 for a detailed discussion about the stack overflow condition.

Symbols not accessible; Symbol database not loaded

Cause: The symbol database for the executable file is not loaded.

Action: Load the symbol database in order to use the symbols.

Termination condition specified cannot turn off stability

Cause: You tried to turn off the stability calculation, but the stability calculation is being used as part of the termination condition.

Action: Either remove the termination condition from your specification, or leave the stability calculation turned on.

Time range of 1.0us - 1.0us defaulted to 1.0us - 2.0us

Cause: You entered a 1.0 usec through 1.0 usec time range. There are several ways you might have done this. This range has been defaulted to 1.0 usec to 2.0 usec.

Action: Remove the defaulted time range and enter the desired time range.

Time range values are identical; defaulted to 1.0us - value

Cause: You entered identical values for the start and end of the time range. Your specification was defaulted to a range of from 1.0 usec to the value you entered.

Action: Reenter your specification to obtain the desired time range.

Trigger event cannot be a static variable or range

Cause: You selected a variable or range type event to be the trigger event. A trigger event must be either a function or interval event, depending upon the type of measurement being made.

Action: If making a function duration measurement, select a function event as the trigger event. If making an interval duration measurement, select an interval event as the trigger event.

Trigger event cannot have identical start and end addresses

Cause: You selected a trigger event that has the same address for both its start address and its end address.

Action: Select a trigger event in an interval duration measurement that has different start and end addresses.

Trigger event cannot match enable or disable events

Cause: You selected the same event to be a trigger event and either the enable or disable event. The events selected for enable or disable cannot match the trigger event.

Action: Either select another event for an enable or disable event, or select a different trigger event.

Trigger event must be type 'func' for Function Duration

Cause: You did not select a function type event to be the trigger event in a function duration measurement.

Action: Select a function type event to be the trigger event, or change to an interval duration measurement and select an interval type event to be the trigger event.

Trigger event must be type 'interval' for Interval Duration

Cause: You did not select an interval type event to be the trigger event in your interval duration measurement.

Action: Select an interval type event to be the trigger event, or change to a function duration measurement and select a function type event to be the trigger event.

Trigger event only valid with Duration measurements

Cause: You tried to include a trigger event within your specification for an activity measurement. Trigger events cannot be found during activity measurements.

Action: Change to a function-duration or interval-duration measurement to use your trigger event.

Unable to find file for selected event

Cause: The source file cannot be found for the event you selected in the Graphical User Interface.

Unable to obtain the current task information

Cause: The Software Performance Analyzer cannot create a valid address based on the current task.

Action: Redefine the event and enter the full address for this event or enter a symbol for this event.

Unrecognized file type - <file>

Cause: The symbol file you specified is not a symbol file.

Action: Check the spelling of the file name.

User interface "debug" blocked on its own forwarded command

Cause: The user interface will not accept the command you just entered because it is responding to a previously forwarded command.

Action: Wait until the forwarded command is completed. Then enter your command again.

User interface "debug" or "emul" not running

Cause: The command you entered cannot be forwarded to either of these interfaces because they are not running.

Action: Start up either a debugger or an emulation interface and enter your command again.

User interface "emul" blocked on its own forwarded command

Cause: The user interface will not accept the command you just entered because it is responding to a previously forwarded command.

Action: Wait until the forwarded command is completed. Then enter your command again.

Validating defined events (XX of XXX)

Cause: This is an informational message about how many of the defined events have been validated correctly after a new symbol data base was loaded.

Waiting for enable

Cause: Software Performance Analyzer measurement is waiting for the occurrence of the enable condition defined in the setup_measurement command.

Waiting for trigger

Cause: Software Performance Analyzer measurement is waiting for the trig1 signal to be delivered from the emulation system. Trig1 will be supplied to the Software Performance Analyzer when the emulator finds its trace point (trigger plus delay specification).

Warning: at least one integer truncated to 32 bits

Cause: This warning occurs when the address value you entered is greater than 32 bits.

Action: Correct the address value to 32 bits.

Warning: emulator is not configured to receive TRIG2

Cause: This warning appears when the Software Performance Analyzer is set up to drive trig2 and the emulator is not configured to receive trig2.

Action: Modify the configuration of the emulator to receive trig2, either for trace arming, or to cause a break to the emulation monitor.

Warning: trigger timer does not exclude calls

Cause: This occurs when you are defining a trigger event in a function duration "excluding calls" measurement. It is a reminder that the trigger time will not match the time shown on the excluding_calls measurement display.

64708A firmware incompatible, update 64708 and 64708S

Cause: The firmware on the HP 64708A analyzer card has not been properly updated. Run the progflash utility and update both the HP 64708 and HP 64708S. Refer to Chapter 18 for details.

64708A software problem, reload all B1487 software + firmware

Cause: The software is not communicating properly. Perhaps some part of the software is out of date.

Action: Update to the latest software by reinstalling the HP B1487 product. Make sure that all of the HP B1487 filesets are loaded, including the HP B1487PERF fileset. When the software has been installed, progflash the HP 64708 and HP 64708S firmware onto the HP 64708A analyzer card. Refer to Chapter 18 for details.

Error log displays

The Multiple Event Definition is placed in the error log. This is an informational message.

```
Multiple Event Definition Summary:
  Total   Functions :  10
  Address Overlaps  : ( 0)
  Pattern Mismatches: ( 0)
  Trivial Functions : ( 0)
  New Defined Events:  10
```

The multiple event definition summary reports the results of using the multiple_events command. The contents are defined as follows:

Total Functions:	Total functions scanned.
Address Overlaps:	Number of functions that had address overlaps. Use the AxLS compiler debug option (-OG) to avoid address overlaps of entry and exit events.
Pattern Mismatches:	Number of functions that did not meet the pattern qualification that was entered.
Trivial Functions:	Number of functions that were found to have identical start and end addresses after the appropriate byte-alignment was applied.
New Defined Events:	Number of new defined events.

12

The Events List

This chapter discusses the events list. It shows you how to interpret the content of the events list. It also shows you how to create events for the events list, and explains details you should understand when creating events.

This chapter discusses the following:

- How to interpret an Events List.
- Detailed definition of the term, "event".
- How events are used by the Software Performance Analyzer.

Interpreting the Events List

This paragraph discusses the Events List (shown below) used by the Software Performance Analyzer. The events list tells you if an event was (or was not) included in a measurement, and the reason the event was left out of the measurement. This chapter shows how you can make sure a desired event will be included in your measurement.

The event number shows the order in which the associated event was defined.

The symbol beside the event is defined in the table on the next page.

In the case of functions and static variables, the name of the event is the same as the name of the source-file symbol it represents. In the case of ranges and intervals, the name is either the name you specified when you defined the event, or a combination of the symbols or addresses used in defining the event.

```
Events  *-Function Duration include calls  ?-Invalid Type
_Number__Name_____________________________Address_Range_____Type____
    1 *  apply_controller                  00000C82 - 00000CCE  func
    2 *  apply_productions                 00000B98 - 00000BE6  func
    3 *  atexit                            00001216 - 00001246  func
    4 *  _01_calculate_answer              00000CD0 - 00000D1C  rec func
    5 ?  _02_calculate_answer              00000CD0 - 00000D1C  interval
    6 ?  count                             00060056 - 00060059  var
    7 ?  data                              00060376 - 00060379  var
    8 *  clear_buffer                      00000AD4 - 00000AFC  rec func
    9 *  endcommand                        00000DB2 - 00000DB2  func
   11 *  format_result                     00000BE8 - 00000C10  func
   12 ?  request_command__syntax_check     00000D1E - 00000B36  interval
   13 ?  i_o                               0006036E - 00060371  var
   15 ?  exec_cmd__exec_cmd                00062000 - 00062000  interval
   16 ?  utility_routines                  00080000 - 000865FF  range
   17 *  main                             00000DB4 - 00000E0A  func
___19_?__math_library____________________00063000_-_0006744B__range___
```

Interpreting Symbols In The Events List

Symbol	Interpretation
*	Selected to be in the next profile measurement. Was included in the last profile measurement.
?	Selected to be in the next profile measurement. Was not included in the last profile measurement because it was the wrong type for that measurement.
r	Selected to be in the next profile measurement. Was not included in the last profile measurement because or resource limits. The measurement already had all of the events it could include before it saw this event.
(no symbol)	Not selected to be in any measurement.

Event Types

Type	Interpretation
func	Source-file function. Represents the address range occupied by the corresponding function in the executable file.
rec func	Recursive source-file function. Represents the address range occupied by the corresponding recursive function in the executable file.
var	Static variable. Represents a static variable or array variable in the executable file, or the address of an IO port.
interval	Interval. Defined by a start address and an end address. The start address may be higher, lower, or the same as the end address. Addresses inside the interval boundaries are not recognized.
range	Address range. Must begin on an address lower than the ending address. All addresses are recognized as part of the range event.

The Address Range is the addresses in the executable file that are represented by the event.

The event types are listed and described in the "Event Types" table.

What is an event?

An event is a name that represents an absolute address or a range of absolute addresses in your executable file. If an event represents the address of a variable, its name will be the same as the name of the variable. If an event represents a function, its name will be the same as the function name.

Through the emulator, the Software Performance Analyzer can access the symbols data base for your absolute file under test and create a list of events whose names and addresses are the same as those in the data base. If you would like to make event definitions of your own, you can define a single event that represents any address or range of addresses (such as an event that represents many source-file functions). You can assign any name you choose to your event. Duplicate names will be made unique by preceding them with a number. The number will appear like: _01_NAME, _02_NAME, etc.

Note that extremely short functions that have no code associated with them may become aligned so that they have the same starting address and ending address. Such functions cannot be defined. For functions to be defined, they must have different, byte_aligned, entry and exit addresses.

How events are used

The Software Performance Analyzer uses events in its measurements. It can recognize events that are appropriate for measurements and events that are not. You can select all events in your present events list and then start a measurement. The Software Performance Analyzer will only include appropriate events in the measurement.

If the measurement you start is a "memory_and_io_activity" measurement, the Software Performance Analyzer will read through the events that are selected in the events list and only accept events that represent variables or ranges of code. If you then start a program activity measurement, the Software Performance Analyzer will read through the events list and only accept selected events that represent functions and ranges. If you finally start a measurement of function duration, the Software Performance Analyzer will only accept selected events that represent functions defined in the absolute file under test. If you start an interval_duration measurement, only events that represent intervals will be accepted.

Profile Measurement Type	**Event Type**				
	Function	**Recursive Function**	**Variable**	**Interval**	**Range**
Program Activity	X	X			X
Memory and I/O Activity			X		X
Interval Duration				X	
Function Duration	X	X			

The events list when markers are used

The display below shows the format of an events list whenmarkers are used. The events list contains all of the information that is in an events list without markers, and additionally contains the marker addresses (for example, apply_controller shows the start and end addresses of the function, and the addresses of its associated start marker and end marker. Note that the apply_productions_calculat interval (number 6) was defined as an interval beginning with the start address of apply_productions, and ending with the start address of calculate_address. Therefore, the start markers of those two functions are used as the start and end markers of the interval.

For complete details of how to use markers, refer to Chapter 7.

```
Events  *-Function Duration include calls  ?-Invalid Type
_Number_Name_____________________________Address Range___Type______Address Markers
    1 * apply_controller             00000FDE-00001062 func     000600B6 000600B8
    2 * apply_productions            00000E02-00000EA4 func     0006009E 000600A0
    3 * calculate_answer             0000106A-000010F2 func     000600BC 000600BE
    4 ? count__count                 000600E0-000600E0 interval 000600E0 000600E0
    5 ? count                        000600E0-000600E3 var
    6 ? apply_productions__calculat 00000E02-0000106A interval 0006009E 000600BC
    7 ? dma                          000603FC-000603FF var
    8 * endcommand                   00001242-00001260 func     000600D4 000600D6
    9 ? errno                        00060496-00060499 var
   10 * format_result                00000EAC-00000EFA range
   11 * get_next_token               00000F70-00000FD6 func     000600B0 000600B2
   12 ? i_o                          000603F8-000603FB var
   13 * initialize                   00000F02-00000F68 func     000600AA 000600AC
   14 * input_line                   00000992-000009F6 func     00060056 00060058
   15 * lookup_token                 00000CD0-00000D36 func     0006008C 0006008E
   16 * main                         00001268-000012EC func     000600DA 000600DC
___17_?_mask_value__________________000600E4-000600E7_var_____________________
```

13

Interpreting Tables, Histograms, and Measurement Specifications

This chapter discusses the displays of measurement results presented by the Software Performance Analyzer. It shows how to interpret each of the displays.

The contents of this chapter discuss detailed information for:

- Interpreting a Table display.
- Interpreting a Histogram display.
- Interpreting a Measurement Specification display.

Interpreting a Table

This paragraph discusses the table displays of the Software Performance Analyzer. Each column on the table display is explained. Two table displays are shown for measurements made by the Software Performance Analyzer: one to show the results of an activity measurement and the other to show the results of a duration measurement. Their contents are different.

A table of activity measurement results is shown and described below:

- Run Time shows how long your measurement has run.
- Stability is an indication of how well the Software Performance Analyzer has characterized the measurement. In a pure sense, it is a measure of the average of standard deviation error tolerances subtracted from 100%.

```
Table: Program Activity                        Run Time: 1:13      Stability: 57%
_Name_(sort:_time)_____|_Cycles_|__Time__|_Time_%_|_Mean(1s)|_StDv(1s)|_Time/cyc
   2 apply_productions | 2.64E06|   1.4 s|  49.88 |  498.8ms|  446.5ms|  546.9ns
  31 stack_library     |  782034| 419.8ms|  14.49 |  144.9ms|  248.5ms|  536.8ns
  27 scan_string       |  404891| 214.7ms|   7.41 |   74.1ms|  208.5ms|  530.2ns
  20 move_byte         |  251109| 134.8ms|   4.65 |   46.5ms|  141.5ms|  536.9ns
  23 report_errors     |  250261| 134.3ms|   4.64 |   46.4ms|  140.8ms|  536.8ns
  19 math_library      |  180832|  98.5ms|   3.40 |   34.0ms|  101.3ms|  544.8ns
  21 outputline        |  156125|  81.4ms|   2.81 |   28.1ms|  158.9ms|  521.2ns
   5 clear_buffer      |  134565|  73.4ms|   2.53 |   25.3ms|  142.0ms|  545.6ns
  16 lookup_token      |  102740|  56.6ms|   1.96 |   19.6ms|   62.7ms|  551.1ns
  15 input_line        |  106926|  55.5ms|   1.92 |   19.2ms|  129.2ms|  519.2ns
  26 scan_number       |   57989|  31.1ms|   1.07 |   10.7ms|   90.0ms|  536.8ns
  24 report_result     |   52309|  28.5ms|   0.99 |    9.9ms|   83.9ms|  545.7ns
  33 syntax_check      |   39934|  21.9ms|   0.76 |    7.6ms|   47.5ms|  548.5ns
  28 semantic_check    |   16318|   9.4ms|   0.33 |    3.3ms|    9.8ms|  577.0ns
__12_get_next_token____|__11475|___6.4ms|___0.22 |____2.2ms|___18.4ms|__561.8ns
 Profiled Absolute     | 5.22E06|   2.8 s|  100%  |
```

- Cycles is the number of cycles recorded for the associated event.
- Time is the total time of all cycles recorded for the event.
- Time % is an expression of the time recorded for this event as a percent of the profiled time.
- Mean(1s) shows the average amount of execution time that is used by the associated event during any given second of program execution.
- StDv(1s) is the variation between 2.5 ms samples of event activity scaled to 1 second.
- Time/cyc is the average time required to complete one bus cycle for the associated event.
- Profiled time is the amount of time that has been completely profiled in making the activity measurement. If you were to have 1 second of profiled time, this would indicate that an equivalent of execution time has been completely dissected into all of the specified address ranges, functions, or variables included in the measurement.

A table of duration measurement results is shown and described below:

- Run Time shows how long your measurement has run.
- Stability is an indication of how well the Software Performance Analyzer characterized the measurement. In a pure sense, it is a measure of the average of standard deviation error tolerances subtracted from 100%.

```
Table: Function Duration include calls        Run Time: 0:49      Stability: 99%
_Name_(sort:_calls)____|_Calls_|__Time__|_Time_%|__Max__|__Min__|_Mean__|Std_Dev
  19 math_library      |  27244|    2.3 s|   4.64|188.0us| 19.0us| 85.2us| 44.6us
      1.00us - 10.0us  |      0|   0.0us|   0.00|  0.0us|  0.0us|  0.0us|  0.0us
      10.0+us- 100us   |  17919|   1.1 s|   2.11| 97.0us| 19.0us| 58.8us| 27.0us
      100+us - 1.00ms  |   9325|   1.3 s|   2.54|188.0us|110.0us|136.0us| 22.5us
      1.00+ms- 10.0ms  |      0|   0.0us|   0.00|  0.0us|  0.0us|  0.0us|  0.0us
      10.0+ms- 100ms   |      0|   0.0us|   0.00|  0.0us|  0.0us|  0.0us|  0.0us
      100+ms - 1.00s   |      0|   0.0us|   0.00|  0.0us|  0.0us|  0.0us|  0.0us
      non_range        |      0|   0.0us|   0.00|  0.0us|  0.0us|  0.0us|  0.0us
 ______________________|_______|________|_______|_______|_______|_______|_______
  31 stack_library     |   9992|   7.3 s|  14.59|730.3us|730.2us|730.2us|  0.0us
  27 scan_string       |   3996|   3.5 s|   7.04|880.3us|880.2us|880.2us|  0.0us
   2 apply_productions |   2997|  32.5 s|  65.00| 11.7ms| 10.0ms| 10.8ms|525.4us
  20 move_byte         |   2997|   2.2 s|   4.38|730.3us|730.2us|730.2us|  0.0us
  23 report_errors     |   2997|   2.2 s|   4.38|730.3us|730.2us|730.2us|  0.0us
__28_semantic_check____|__1998|___4.5_s|___9.09|__2.3ms|__2.3ms|__2.3ms|__0.0us
 Profiled Absolute     |  58299|  50.0 s|       |
```

- Calls is the number of times the associated event was called during the measurement.
- Time is the total time of all executions of the associated event.
- Time % shows what percent of profiled time was recorded for the associated event.
- Max shows the longest single duration measured during any execution of the event.
- Min shows the shortest single duration measured during any execution of the event.
- Mean shows the average time for one execution of the event.
- Std Dev is the variation between executions of the event (the value of one standard deviation of the event).
- Expanded events show greater detail. Each time range under an expanded event shows information about the executions of the event that were completed within the time range.
- Profiled time is the amount of time that has been completely profiled in making the measurement. In duration measurements, this is typically equivalent to the run time.

Interpreting a Histogram

The following paragraphs discuss the content of the histogram display (shown below), which shows the results of a Software Performance Analyzer measurement. The example histogram shows values of time recorded during the measurement. You can also obtain histograms that show values of cycles (for activity measurements), and calls (for duration measurements).

- Run Time shows how long your measurement has run.
- Stability is an indication of how well the Software Performance Analyzer characterized the measurement. In a pure sense, it is a measure of the average of standard deviation error tolerances subtracted from 100%.
- Time shows the total amount of time recorded for the associated event.
- % is an expression of the time recorded for this event as a percent of the profiled time.
- 0% thru 100% shows graphic bars to represent the amount of time used by the associated event, compared with the amounts of time used by the other events.

```
Histogram: Program Activity                    Run Time: 0:40      Stability: 69%
_Name_(sort:_time)_____|__Time__|__%___0%_____10%_____20%_____30%_____40%____50%
   2 apply_productions | 779.7ms| 49.30|**************************************
  31 stack_library     | 211.5ms| 13.37|***********
  27 scan_string       | 110.9ms|  7.01|******
  19 math_library      |  86.9ms|  5.50|****
  20 move_byte         |  74.6ms|  4.72|****
  23 report_errors     |  63.1ms|  3.99|***
   5 clear_buffer      |  37.5ms|  2.37|**
  16 lookup_token      |  30.4ms|  1.92|**
  26 scan_number       |  30.0ms|  1.90|**
  21 outputline        |  26.7ms|  1.69|*
  15 input_line        |  20.8ms|  1.31|*
  33 syntax_check      |  12.7ms|  0.81|*
  28 semantic_check    |   5.5ms|  0.35|
  12 get_next_token    |   5.1ms|  0.32|
__24_report_result_____|___4.6ms|__0.29|_____________________________________
 Profiled Absolute     |   1.5 s| 100% 0%     10%     20%     30%     40%     50%
```

Interpreting a Measurement Specification

The following paragraph discusses the content of the measurement setup (shown below) of the Software Performance Analyzer. The measurement setup below contains more information than the default measurement setup. Each element of information in a measurement setup is discussed below.

The measurement specification display shows the following:

- The name of the symbol file for the present program under test.
- The default measurement type (measurement performed by pressing "profile" alone.
- Condition that starts the measurement in the Software Performance Analyzer.
- Event that enables the Software Performance Analyzer to collect data.
- Event that temporarily disables the collection of data.
- Trigger generation that occurs after continuous execution of specified event.
- Condition, when achieved, causes measurement to end automatically.
- Whether the default type of output display is histogram or table.

```
Measurement Specification   Symbol File: /usr/hp64000/demo/spa/demo1/runtest.x
_____________________________________________________________________________
PROFILE   |  function_duration include_calls
          |
SETUP     |  start after_receiving_trig1
          |  enable start_address scan_string (27)
          |  disable end_address semantic_check (28)
          |   status_enable_disable default (prog)
          |   (Enable and disable events cannot be selected/measured)
          |  drive trig2_after 150 msec parse_command (22)
          |  termination stability 98 percent
          |   (termination by stability only occurs if this interface is active)
          |
DISPLAY   |  table data absolute
          |  sort_events time
          |  rescale to_max 100 percent
          |  stability after_every 30 seconds
          |  time_ranges 1.00us - 10.0us
          |              10.0+us- 100us
__________|______________100+us_-_1.00ms_____________________________________
```

- Whether data is expressed in absolute or relative values.
- The present way events are sorted in tables and histograms.
- The present scale of information on histogram displays.
- An indication of how often the data stability is recalculated.
- The present set of time_ranges defined. You can use the roll or arrow keys to see all of the time ranges if they are not presently on screen.

14

Using trig1 and trig2 to Control Measurements with Emulators and Other Analyzers

This chapter discusses the high-speed communication that can be used to enable measurements, trigger measurements, and cause emulation breaks when using the Software Performance Analyzer. The contents of this chapter include:

- Trigger lines used by the Software Performance Analyzer.
- Trigger events reduce the events that can be in a measurement by one.
- Trigger events, how they are handled during activity measurements.
- Restrictions on the event used to generate trig2.
- How trigger operates.

Trigger lines used by the Software Performance Analyzer

The Software Performance Analyzer is designed to use two trigger signals when coordinating measurements with other instruments: trig1, and trig2. These trigger signals are described below:

trig1 is an input to the Software Performance Analyzer. It is a measurement-enable level. You can set the Software Performance Analyzer to hold off its measurement until after it receives trig1. In this way, you can hold off starting your measurement until the instrument generating trig1 (normally the emulation bus analyzer) finds a desired location in state flow, such as the completion of an initialization process.

trig2 is an output from the Software Performance Analyzer. It commands immediate action. You can set the Software Performance Analyzer to generate trig2 during a duration measurement if an event you select executes continuously for too long a period of time. You can set up the emulator to break to its monitor program when it receives trig2, or you can set up the emulation bus analyzer to take a trace of program execution when it receives trig2.

Trigger events must be selected

If you specify trigger generation after an event has executed for some period of time, then the event named in your trigger specification must be selected in the measurement. Therefore, the number of other events that can also be selected in the measurement is reduced by one.

Trigger events in activity measurements

If you have a trigger event specified, and you select an activity measurement (triggering is not possible in activity measurements), your specification will continue to be listed in the measurement setup, but it will not be active.

Restrictions on the event used to generate trig2

- A trigger event for a function_duration measurement must be a function or recursive function.
- A trigger event for an interval_duration measurement must be an interval.
- If you are making a measurement that uses an enable and/or disable, the enable and/or disable specifications must use a different event from the one used for the trigger event. You cannot use the same event for both a trigger event and an enable or disable event.
- A trigger event cannot be defined to have the same address for both its start address and its end address.

How trigger operates

Trigger operation is independent of the duration-measurement mode. The trigger generator operates very much like an interval-duration measurement. The first address of the trigger event will start the trigger counter. If the trigger counter exceeds the time specified for trigger generation before the trigger-event stop-address is found, the trigger will occur. A delay of about 500 ns will occur after the specified time has been reached before the trigger actually appears.

Qualifying the trigger

The trigger-event start address and stop address will be qualified by the global status specification you use (such as "set status_qualification user" or "supervisor", as applicable to your target microprocessor). The trigger-event addresses will also be qualified by your interval-duration status specification (such as "profile interval_duration status prog", or "status data_read", as applicable to your target microprocessor).

If you trigger on functions that are prefetched

The trigger event is not corrected for unused prefetches. If the starting address of your trigger event appears in an unused prefetch, the Software Performance Analyzer might begin counting time to generate trig2. The result might be trigger generation before the event has been active. In the same way, if the end address of your trigger event appears in an unused prefetch, the "trig2_after" specification may never be exceeded because the prefetch of the end address terminates the event time too early.

If prefetching causes problems for the execution of your trigger specification, you can modify your source file to put additional NOP instructions before the entry and exit addresses of the function used by the trigger specification. The NOP instructions will be prefetched instead of the entry and exit addresses.

Another way to solve the above problem takes more work to set up, but avoids the prefetch problem, entirely. Edit your source file to add an instruction that writes to a variable just after the function start, and another instruction that writes to a different variable just before the function end. Then define an interval event that starts at the first variable and ends at the last variable. Perform an interval-duration measurement that is qualified on write transactions, and specify trigger to occur when the duration of your interval event is exceeded.

Refer to the discussion on using markers at the end of Chapter 7 for information that may help you with this last method of overcoming prefetch problems when generating triggers.

If you trigger on the duration of a recursive function

The trigger counter will record the time during the recursive call string from the first occurrence of the trigger event start address to the first recursive end address (not the last recursive end address). The result will be trigger generation if the period of the recursive call string is long enough to meet your specification, measured from the unused prefetches of the function (if any) to the first recursive end address.

If calls are excluded

The trigger counter is not corrected for the excluding-calls mode. If your measurement mode is "function_duration exclude_calls", the trigger counter will continue to run during execution of calls to other events.

If an enable/disable window is used

If your measurement uses an enable/disable specification, the trigger counter will not run during periods when the Software Performance Analyzer is disabled.

15

Hidden Commands of the Software Performance Analyzer

This chapter lists and describes commands you can use with the Software Performance Analyzer that do not appear on the softkeys. You must type these commands directly on the command line.

The following commands do not appear on the softkeys. In cases where it is appropriate, parameters will be offered on the softkeys once you have typed the command on the command line. Commands described in this chapter include:

- cd (change directory)
- pws and cws (print working symbol and change working symbol).
- <!CMD!> (UNIX COMMAND).
- version (to see the software version number on the STATUS line).
- help.
- log_commands.
- pod_command.
- wait.
- forward.

The following hidden commands are emulator commands. They are added as a convenience for the user of the Software Performance Analyzer. The commands are only available if you are running a Graphical User Interface or Debugger Graphical User Interface with software version number 5.00, or higher. Refer to your emulator manual for a complete description:

- run.
- run from reset.
- run from transfer_address.
- reset.
- break.
- modify configuration.

Change directory

You can change your working directory while in a measurement session by typing a **cd** command on the command line. To see your present working directory, type **pwd** on the command line.

Working symbol (pws and cws)

The Symbolic Retrieval Utilities (SRU) handle symbol access within emulation. SRU maintains trees representing the symbol structure and scoping within your program code. You can specify a path in the tree by typing the **cws** (current working symbol) command. After you specify a symbol in this way, other symbol accesses are assumed to be relative to this symbol unless you specify complete paths. You can display the current working symbol on the status line by typing the **pws** (print working symbol) command.

More information about SRU is given in the chapter on SRU in the *Softkey Interface User's Guide/Reference* manual for your emulator/analyzer.

UNIX COMMAND (<!CMD!>)

UNIX commands can be entered from the command line by preceding them with an exclamation point (!). If you want command line execution to continue in the softkey interface after the UNIX command is finished, follow your command with an exclamation point.

Examples

To enter a UNIX command from the command line of the Software Performance Analyzer:

```
!ls
!more myfile1.c
```

To enter a command that has command line execution continuing in the softkey interface after the UNIX command is finished:

copy table to !lp! **noheader**

Software version

You can type **version** on the command line to see the software version number and release date of the Software Performance Analyzer. This information will be shown on the STATUS line.

help

When typed on the command line, help displays information about the features of the Software Performance Analyzer. Typing **help** or **?** displays softkey labels that list the options on which you can receive help. When you select an option, the system will list the information to the screen.

You can either press a softkey representing the help file, or type in the name of the help file. If you type in the help file name, make sure you use the complete name. Not all softkey labels have the complete help file names.

The following is a summary of the help files available when you type **help** or **?**:

system_commands
profile
define
setup
display
SELECT
delete
load
store
renumber
copy
select
unselect
set
absolute
relative
sort
rescale
stability
time_ranges
EXPAND
end
stop
enable
trigger
symbols
symbol_offset

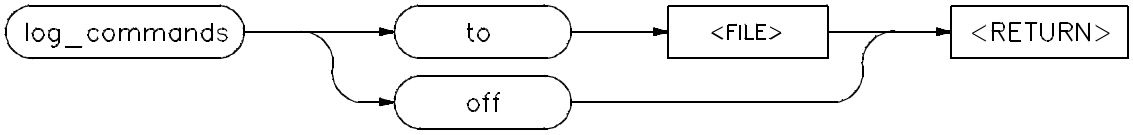
pod_command
wait
run
break
reset
modify
dur_act_distinctions
real_time_OS
prefetch
markers
I80960

Examples

help SELECT

? profile

log_commands



This feature allows you to keep a record of commands that are executed during a performance measurement session. During a measurement session, all commands that are executed are also stored in a file. After the session, you can edit the file to create a command file.

To execute the saved commands after the log file is closed, type the name of your log file on the command line.

Default value

Commands are not logged (stored) in a file.

Parameters

<FILE>	This prompts you to enter the name of the file where you want to store commands that are executed while running the Software Performance Analyzer.
off	This turns off the process that logs commands to a file.
to	This allows you to specify a file for the logging of commands.

Examples

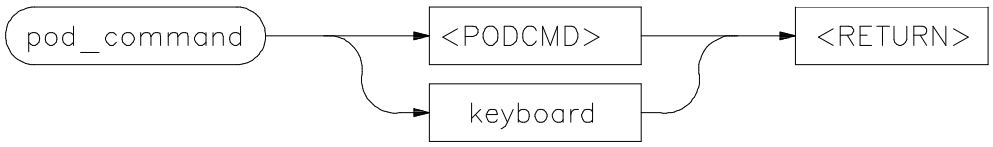
log_commands to myfile

log_commands off

See also

help system_commands

pod_command



This command lets you control the emulator directly through the Terminal Interface. You can access this interface and enter Terminal Interface commands using **pod_command**. The options to **pod_command** allow you to supply one command at a time, or you can select a keyboard mode which gives you interactive access to the Terminal Interface.

Avoid using the following commands while using **pod_command**:

stty, po, xp	Do not use. These commands will change the operation of the communications channel, and are likely to hang the Softkey Interface and the channel.
echo, mac	Using these may confuse the communications protocols in use on the channel.
wait	Do not use. The pod will enter a wait state, blocking access by the softkey Interface.
init	This will reset the emulator pod and force an end release_system command.
t	Do not use. The trace status polling and unload will become confused.
lanpv	Do not execute this command while the emulator is connected to the LAN (local area network).

To see the results of a particular **pod_command** (the information returned by the emulator pod), use **display pod_command**.

There is no default, but you must specify either a particular Terminal Interface command as a quoted string or enter the **Keyboard** mode.

If you suspect problems in your system hardware, you can use the **pod_command** to run performance verfication (PV) on the emulator and Software Performance Analyzer from the Graphical or Softkey User Interface. Although this will reset the emulator and force you to end and release the system, after you know that your system components are operating correctly, just start up a new emulation session.

You can execute the **lan** command in pod_command to view LAN configuration settings. Do not execute the **lanpv** command in pod_command while the emulator is connected to the LAN. Instead, you must set up an RS-232 connection between the emulator and the host computer to run **lanpv**. The **lan** and **lanpv** commands in the terminal interface User's Guide describe this.

Parameters

keyboard	Enters an interactive mode where you can simply type Terminal Interface commands (unquoted) on the command line. Use **display pod_command** to see the results returned from the emulator.
<POD_CMD>	Prompts you for a Terminal Interface command as a quoted string. Enter the command in quotes and press **RETURN**.
suspend	This command is displayed once you have entered keyboard mode. Select it to stop interactive access and return to the Graphic or Softkey User Interface.

Examples

This example shows a simple interactive session with the Terminal Interface.

```
display pod_command
pod_command keyboard
cf
tsq
tcq
```

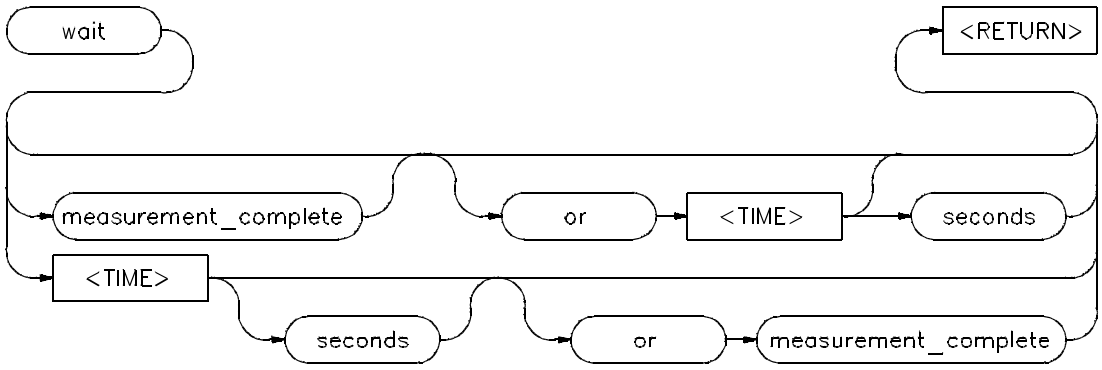
See also

display pod_command

help pod_command

Also see the *Terminal Interface User's Guide/Reference* manual.

wait



The wait command is a delay command. Delay commands are normally used within command files where they give added flexibility (although delay commands can also be used outside command files). By using delay commands in command files, you can give an emulation system and target processor time to reach some condition or state before bringing in the next command.

The wait command is not shown on the softkeys of the Software Performance Analyzer. Type the command from the keyboard. After you type "wait", the wait command parameters will appear on the softkeys.

Default Value

Wait for <control> C (SIGINT).

The parameters are as follows:

measurement_complete

measurement_complete causes the system to wait for a measurement in progress to finish before the next command is executed.

Note that the Software Performance Analyzer measurement will be complete when the termination condition is found. You can set up the termination condition by using the **setup_measurement termination** command.

<TIME>

<TIME> is the number of seconds of delay before accepting the next command.

Examples

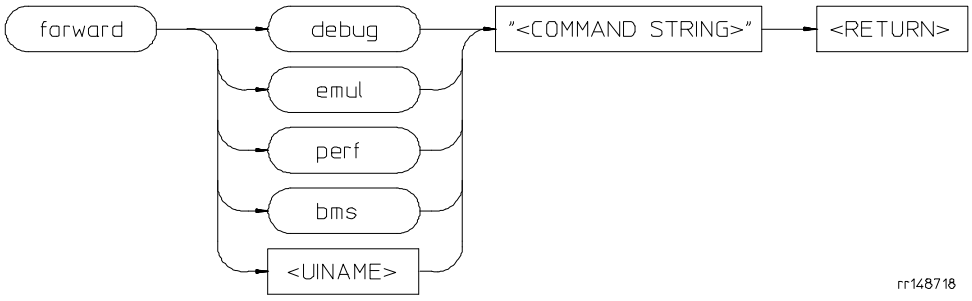
wait wait for <control> C before accepting the next command.

wait 6 wait for <control> C or 6 seconds before accepting the next command.

wait measurement_complete wait for <control> C, or wait for the present measurement to complete. If no measurement is in progress, the wait will be satisfied immediately.

wait measurement_complete wait for <control> C, for the present measurement to complete, or for 20 seconds (whichever occurs first) before accepting the next command.

forward



A background process allows commands to be forwarded from one HP 64700 interface to another. All interfaces having software versions above 5.00 can forward commands; only Graphical User Interfaces can receive commands that have been forwarded.

The parameters are as follows:

bms — sends the quoted command string to the broadcast message server.

debug — sends the quoted command string to the high level debugger.

emul — sends the quoted command string to the emulator.

perf — sends the quoted command string to the Software Performance Analyzer.

<UINAME> — prompts for the name of the user interface to receive the quoted command string.

Examples

To send the command, "run from transfer_address", to an emulator, enter:

forward emul "run from transfer_address"

To send the Program Run command to a high-level debugger, enter:

```
forward debug "Program Run"
```

Or, because only the capitalized key is required for a debugger, enter:

```
forward debug "P R"
```

16

Software Performance Analyzer Specifications

Software Performance Analyzer Specifications

Emulators Supported

HP Product No.	Description	HP Product No.	Description
64146	MELPS7700 Emul	64751	68340 Emulator
64732	H8/510 Emulator	64757	NEC V53 Emul
64737	H8/532 Emulator	64760	80960 Sx Emulator
64739	H8/536 Emulator	64762	8086 Emulator
64742	68000 Emulator	64763	8088 Emulator
64744	68000EC Emulator	64764	80C186 Emulator
64745	68010 Emulator	64765	80C188 Emulator
64746	68302 Emulator	64766	80286 Emulator
64747	68030 Emulator	64767	80C18X Emulator
64748	68020 Emulator	64780	68360 Emulator
64749A/B/C	6833X Emulator	64783	68040 Emulator

Measurements Available

- Program Activity (sampled)
- Memory And I/O Activity (sampled)
- Interval Duration (non-sampled)
- Function Duration Including All Calls (non-sampled)
- Function Duration Excluding All Calls (non-sampled)
- Function Duration Excluding Profiled Calls (non-sampled)

Hardware Timing Specifications

- Maximum Bus Rate: 25 MHz
- Timing Clock Rate: 50 MHz (20 ns)
- Timing resolution: +/- 40 ns
- Maximum Continuous Measurement Time: 24 days

Trigger Specifications

- Maximum Measurable Trigger Duration: 2.5 seconds
- Minimum Measurable Trigger Duration: 1 usec
- Trigger Duration Resolution: 500 ns +/- 160 ns
- Emulator Trigger Recognition Delay: 500 ns approx.

Event Specifications

- Maximum Number of Defined Events: 1000
- Maximum Number of Expanded Events: 10
- Maximum Number of Expanded Time Ranges: 10

Enable/Disable Feature:

- Enable/Disable Pairs: One
- Enable: Available in all Measurements
- Disable: Available in Interval/Function Duration Measurements

Activity Measurements (sampled):

- Maximum Number of Measured Events: 254
- Maximum Continuous Event Rate: 25 MHz
- Sampling Period: 2.5 ms per active event
- Scanning Period: 5.0 ms + (2.5 ms * number_of_events)
- Random restart to avoid synchronization with user program under test
- Cannot respond to an enable during first 5 msec after microprocessor transition from reset to running

Interval Duration Measurements (real-time data collection; non-sampled):

- Maximum Number of Measured Events: 84
- Maximum Continuous Event Rate: 80 usec for entry/exit pair [Note 1]
- Minimum Continuous Event Rate: None
- Maximum Number of Events at Burst Rate of 25 MHz: 500 approx.

Function Duration Measurements (real-time data collection; non-sampled):

- Maximum Number of Measured Events: 84
- Maximum Continuous Event Rate: 100 usec for entry/exit pair [Note 1]
- Minimum Continuous Event Rate: 1.25 sec for entry or exit event
- Maximum Number of Events at Burst Rate of 25 MHz: 500 approx.

Histogram Types Available: Time, Calls, Cycles

Table Data Available:

- Activity: Cumulative Cycles, Cumulative Time, Time %, Mean Time, Stdv Time, Time/Cycle
- Duration: Cumulative Calls, Cumulative Time, Time %, Max Time, Min Time, Mean Time, Stdv Time

Sorting Orders Available:

Time, Calls, Cycles, Address, Alphabetical, Defined order

Notes:

[1] Expanded events, or prefetched event entry/exit addresses will reduce the Maximum Continuous Event Rate.

Part 5

Installation and Service

Part 5

This part of the manual contains the following chapters:

Chapter 17. Installation and service

Chapter 18. Installing/updating Software Performance Analyzer firmware

17

Installation and Service

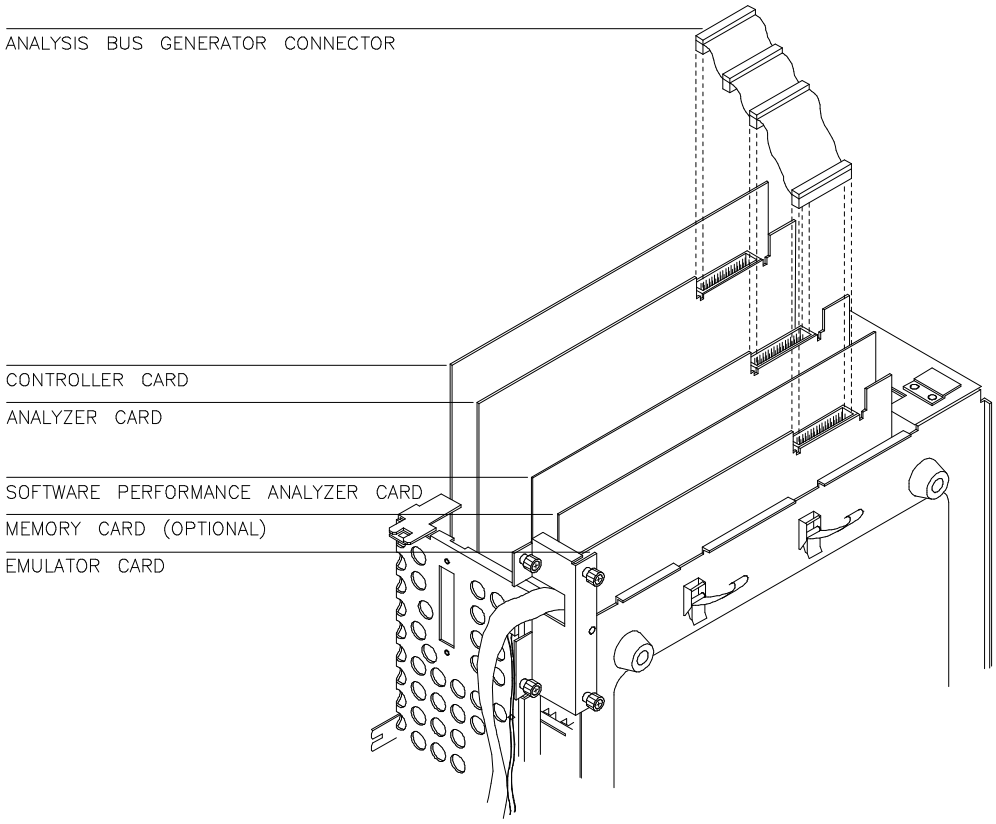
Equipment supplied

The Software Performance Analyzer (SPA) package contains

- *Software Performance Analyzer User's Guide* and *Support Services.*
- Software Performance Analyzer media.
- Software Performance Analyzer hosted user interface license.
- Analysis Bus cable.
- Software Performance Analyzer board.

Tools needed

- Torx T-10 screw driver



Antistatic precautions

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

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

Installation Considerations

The Software Performance Analyzer is designed to be used with HP 64700 Series Card Cages that are connected to host computers by way of LAN or RS422. Connecting HP 64700 Series Card Cages to host computers through RS232 is not recommended when running the Software Performance Analyzer. If using an RS232 connection, you may experience long delays in the operation of the user interface. This occurs when the Software Performance Analyzer transfers large amounts of data over the connection.

Installation Overview

When you order a complete system (an emulator and Software Performance Analyzer in an HP 64700A Card Cage), the Software Performance Analyzer card is preinstalled in the card cage at the factory. The installation procedure in this chapter is provided for users that already have an HP 64700A Card Cage and want to install the Software Performance Analyzer in the card cage.

Caution

If you already have a modular HP 64700A Card Cage and want to add the Software Performance Analyzer, the HP 64700 Series generic firmware and analyzer firmware may NOT be compatible and the software will indicate incompatibility. In this event, you must purchase a Flash EPROM board to update the firmware. Instructions for installing this board and programming it from a PC or HP 9000 are provided in the *HP 64700A Card Cage Installation/Service* manual.

After hardware and software installation, run a performance test to verify that the emulator and Software Performance Analyzer are working properly. After you verify performance of the system, you are ready to use the Software Performance Analyzer.

Use this chapter to accomplish:

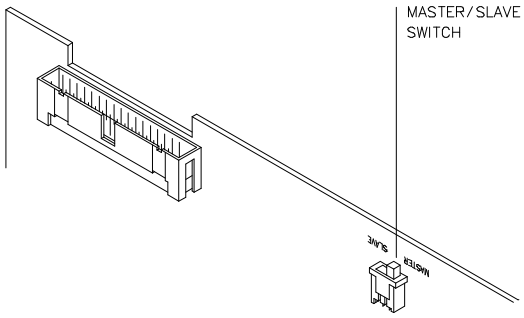
- hardware/software removal and installation procedures,
- running performance verification,
- ordering parts.

Refer to the *HP 64700 Series Card Cage Installation/Service* manual for:

- information on system configurations,
- installing product software,
- software updates, and ordering parts related to the card cage.

Before installing the circuit card of the Software Performance Analyzer

Before installing the Software Performance Analyzer card, set the master/slave switch to **master.** The slave position is not used. The figure below shows the master/slave switch on the Software Performance Analyzer card.



To install the circuit card of the Software Performance Analyzer

1 Turn power off.

2 Remove the HP 64700A Card Cage top cover.

3 Remove the HP 64700A Card Cage side panel.

4 Install the Software Performance Analyzer card.

5 Connect the analysis bus generator cable as shown in the first figure in this chapter. Note that this step will not be necessary in every system installation. Some installations will have a different version of the cable from the cable shown in the installation figure. Certain other installations will not have this cable at all.

6 Replace the HP 64700A Card Cage side panel and top cover.

The table on the following page shows the different configurations when using a Software Performance Analyzer card. Instructions for removing and installing cards into and out of the HP 64700A Card Cage are provided in the *HP 64700A Installation/Service* manual and on the HP 64700A Card Cage label on the bottom of the HP 64700A Card Cage.

Possible Card Slot Configuration

- If a Flash EPROM card is NOT required and the emulation subassembly requires only one card, you will have two empty slots.

Slot	Card
1	Host Controller Card
2	Analyzer Card
3	**SPA Card**
4	empty
5	empty
6	Emulator Controller Card

- If the emulator sub-assembly uses only one card, and requires a Flash EEPROM card, you will have one empty slot.

Slot	Card
1	Host Controller Card
2	Analyzer Card
3	**SPA Card**
4	empty
5	Flash EEPROM Card
6	Emulator Controller Card

- If the emulator requires two cards (Emulator memory and controller), and requires a Flash EPROM card, all available card slots are used.

Slot	Card
1	Host Controller Card
2	Analyzer Card
3	**SPA Card**
4	Flash EEPROM Card
5	Emulator Memory
6	Emulator Controller Card

To install the software

The installation of the Software Performance Analyzer should occur after you complete all of the software installation and verification of your emulator. The table below lists most of the emulators that are supported by the Software Performance Analyzer. A complete list of emulators supported by the Software Performance Analyzer is in the Specifications Chapter of this manual.

HP Product No.	Description
64742	68000 Emulator
64745	68010 Emulator
64746	68302 Emulator
64747	68030 Emulator
64748	68020 Emulator
64749A/B/C	6833X Emulator
64751	68340 Emulator
64760	80960 Sx Emulator
64762	8086 Emulator
64763	8088 Emulator
64764	80C186 Emulator
64765	80C188 Emulator
64766	80286 Emulator
64767	80186EA/EB/EC
64780	68360 Emulator
64783	68040 Emulator

If you intend to use the Software Performance Analyzer Graphical User Interface, you should first install and test the emulation Graphical User Interface. Refer to the "Installing the Software" chapter in the Graphical User Interface User's Guide for complete details.

If you intend to use the Software Performance Analyzer with the Softkey Interface, you should first install and test the emulator's Softkey Interface. Refer to the Softkey Interface Installation Notice supplied with your emulator Softkey Interface documentation. This notice describes what you should do to install and/or update the appropriate Softkey Interface software for your emulation system.

To install software on an HP 9000 hosted system

During the install process, you have some choices about how much you load from the product media. As a general rule, you should load everything from the media because this ensures that you will not miss filesets and therefore have problems with the operation of the software. However, you may not need or want to install certain ***partitions*** or *filesets* from the product media when installing the Software Performance Analyzer. There are at least two reasons why that is so.

- You may not have the system performance necessary or choose, for some other reason, not to use the Graphical User Interface and instead use the conventional Softkey Interface. If that is the case, then you should exclude the filesets that contain the Graphical User Interface because:
  - You will save about 3.5 megabytes of disk space.
  - The Graphical User Interface is the default interface if you are using X Windows. If you load the Graphical Interface but do not use it, you will have to manually override it each time you begin an emulation session.
- You may not need to install the SPA versions of the AxLS C Cross compiler libraries. Or you may want to include only the SPA version of the AxLS C libraries for the AxLS C Cross compiler you are using. If you are using MRI compiler tools or other language tools, do not include these libraries. These libraries are modified to work more correctly with the Software Performance Analyzer. In particular, they have been compiled with the -OG option to add NOP's between functions to avoid problems with prefetch. At this writing, SPA versions of AxLS C Cross compiler libraries were available for the following four compilers:
  - cc68000
  - cc68020
  - cc68030
  - cc68332

To exclude partitions or filesets

The following sub-steps assume that you may want to exclude partitions or filesets. Perform the following sub-steps to load the software on your system:

1 Become the root user on the system you want to update.

2 Make sure the tape's write-protect screw points to SAFE.

3 Put the product media into the tape drive that will be the *source device* for the update process.

4 Confirm that the tape drive BUSY and PROTECT lights are on. If the PROTECT light is not on, remove the tape and confirm the position of the write-protect screw. If the BUSY light is not on, check that the tape is installed correctly in the drive and that the drive is operating correctly.

5 When the BUSY light goes off and stays off, start the update program by entering

```
/etc/update
```

at the HP-UX prompt.

6 When the HP-UX update utility Main Menu screen appears, confirm that the source and destination devices are correct for your system. HP recommends that you install the software under the same path where your other HP 64000 software is installed (typically "/"). The Help screen will tell you how to change the source and/or destination devices. Refer to your HP-UX System Administrator documentation if you need more details about how to modify these values.

7 Select the choice on the update menu that allows you to view the product partitions.

8 Mark "y" beside 64700 SERIES FIRMWARE partition to load the firmware of the Software Performance Analyzer.

9 If you plan to install and use the Software Performance Analyzer Graphical User Interface, do the following:

- Mark "y" beside the SOFTWARE ANALYSIS TOOLS partition.
- Skip to sub-step 11 of these instructions.

10 *If you do not want to install the Graphical User Interface, do the following:*

- Enter the command that lets you View the Filesets that make up the SOFTWARE ANALYSIS TOOLS partition.
- Mark the fileset B1487XUI with "n" to exclude it from installation.
- Mark all other filesets in the partition with "y" to confirm installation.
- Return to the partition screen.

11 To prevent installation of the SPA versions of the AxLS C Cross compiler libraries:

- Enter the command that lets you view the filesets that make up the SPA COMPILER LIBRARIES partition.
- Mark each fileset with "y" to include it or "n" to exclude it.
- Return to the partition screen.

12 From the partition screen, choose the update utility softkey (named Start Loading) that starts the installation process.

To install software on a Sun SPARCsystem

The tape that contains the Graphical User Interface software to be installed on Sun SPARCsystem workstations may contain several products. Usually, you will want to install all of the products on the tape. However, to save disk space, or for other reasons, you can choose to install selected filesets.

If you intend to use the Softkey Interface instead of the Graphical User Interface, do not install the filesets with ".XUI" suffixes. (If you do not to install the Graphical User Interface, you will not have to use a special command line option to start the Softkey Interface.)

Refer to the *Software Installation Notice* for software installation instructions. After you are done installing the software, look in the manual for your emulator to see how to start the X server and OpenWindows, and how to set the necessary environment variables.

To verify installation of the Software Performance Analyzer User Interfaces

To verify that the Software Performance Analyzer is properly installed, you should be able to execute the sequence of steps in the quick start guide in Chapter 1 of this manual.

To verify performance of the Software Performance Analyzer

1 Turn on power.

2 Establish communication with the emulator from your host or ASCII terminal.

3 Enter: pv n <return>

where **"n"** is the desired number of cycles for running performance verification (pv).

There are different hardware system configurations for the HP 64700 Series system. For information on hardware configurations, refer to the HP 64700A Installation/Service manual. The interface you are using depends on the type of host computer used. The **pv** command is a Terminal Interface command.

- If you are using the Terminal Interface, simply enter the command

```
pv 1 <return>
```

If you are using another interface, you can still run performance verification. You might be using one of two other interfaces, Softkey Interface or PC Interface. You can access Terminal Interface commands (known as pod commands) through the Softkey or PC interface.

- If you are using the Softkey Interface, enter

  pod_command "pv 1" <return>

- If you are using the PC Interface, enter

  System Terminal pv 1 <return>

Examples

Invoke the Self-test routines by typing the command:

pv n

(where n=1,2,3,.... specifies the number of times to run the set of performance verification tests.)

A message similar to the following should appear:

```
R>pv 1

  Testing: HP64753A Z80 Emulator
    PASSED
    Number of tests: 1           Number of failures: 0
  Testing: HP64740 Emulation Analyzer
    PASSED
    Number of tests: 1           Number of failures: 0
  Testing: HP64708A Software Performance Analyzer
    testing: MASTER SPA  (Test Time is ~3 Minutes) .....
             MASTER SPA --> passed
    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.03.00 13Dec90

    HP64753A Z80 Emulator
    HP64740 Emulation Analyzer
    HP64708A Software Performance Analyzer
    HP64701A LAN Interface
R>
```

If you have a Software Performance Analyzer failure, you can replace the card through your local Hewlett-Packard representative, and through the Support Materials Organization (SMO)

To ensure software compatibility

There are various sets of firmware resident in the assemblies contained in the HP 64700A Card Cage. The versions of these code modules could change occasionally and it is important to insure that all the versions are compatible between the group of products that you have installed.

There are five assemblies that have firmware in the HP 64700A Card Cage. These assemblies are the:

- Host Controller card
- Emulator card
- Analyzer card
- Software Performance Analyzer card
- Local Area Network card (if present)

By entering the **ver** command, you can determine if you have the correct versions for your system. HP 64700 Series Emulation System software must be version 3.00 or later. A typical display when you enter the **ver** command is shown below.

```
R>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

     HP64753A Z80 Emulator
       Version:     A.00.01 25Jan88
       Speed:       10 MHz
       Memory:      64 KBytes

     HP64740 Emulation Analyzer
       Version:     A.02.00 29Jun89

     HP64708A Software Performance Analyzer
       Version:     A.05.21 22Nov93

     HP64701A LAN Interface
       Version:     A.00.03 09Apr91
```

To ensure software compatibility

If you purchased a complete Emulation/Analysis System from HP, all the products contained in the HP 64700A Card Cage contain compatible firmware at the time of sale. Software compatibility problems can occur when you swap the host controller card, emulator card, analyzer card, or local area network (LAN) card from one HP 64700A Card Cage to another.

For example, you might purchase an emulator subassembly and replace the original emulator subassembly with the one you just purchased. In this case, the host controller may contain a version of firmware that is older than required to operate the new emulator; hence, compatibility problems are caused by a newer emulator needing newer generics. All emulators will work with the latest generics, but you may not have all the features available using old generic firmware. The emulator software will warn you of incompatible software.

The Software Performance Analyzer card, LAN card (if present), and some emulator cards have Flash EPROMs that can be updated with current versions of firmware. Other Products (assemblies) that do not use the Flash EPROM technology can also be updated with the latest firmware by adding a Flash EPROM card that is available from Support Materials Organization (SMO). A Flash EPROM card can be inserted in an available slot in the card cage to override old versions of firmware in products with conventional EPROMs. The HP 64700A Card Cage host controller is already programmed to look for a Flash EPROM card if one is inserted into the card cage.

The latest versions of firmware for the host controller card, analyzer card, and LAN card along with a program called *progflash* are part of the B1471 software for the HP 9000 Workstation, HP 64700 Option 006 software for PCs, and B1471 software for Sun SPARCsystems. Current versions of emulation firmware come with the Softkey Interface software, and on a separate diskette for PC Interface software.

The latest versions of firmware for the Software Performance Analyzer circuit card (HP 64708A) are included within the HP B1487 filesets. A new Software Performance Analyzer card will be programmed with the latest software before it is sent to you. If you need to load updated software on a Software Performance Analyzer card, you will need to progflash both the HP Product Number 64708 and HP Product Number 64708S firmware.

When you load all your new versions of software onto your host computer, you are ready to load the new version of firmware from your host computer to the assemblies that are in the HP 64700A Card Cage.

To load the new firmware, use the *progflash* command. The progflash command displays a list of card cages, and subassemblies in each card cage on your system. From these lists, you can select which product to update. For more information on using the *progflash* command, and updating your HP 64700 Series firmware, refer to the HP 64700 Series Card Cage Installation/Service manual.

Parts List

HP B1487A #AAX	Software Performance Analyzer media and manual. The Manual HP Part Number is shown on the Printing History page at the front of this manual. Order extra copies using an HP number like B1487-99003, which is the B1487-97003 manual installed in a 3-ring binder. (included as part of HP B1487A #AAX, above.)
HP B1487 #UBX	Software Performance Analyzer hosted user interface license.
HP 64708A	Analysis Bus Cable and Software Performance Analyzer. Software Performance Analyzer circuit card, HP Part Number 64708-66505. Analysis Bus Cable, HP Part Number 64708-61601. (both included as part of HP 64708A, above.)

What is an Exchange Part?

Defective parts can be returned to HP for repair in exchange for a rebuilt part. The HP part number for a rebuilt Software Performance Analyzer card is:

64708-69505	Software Performance Analyzer Card - Rebuilt

18

Installing/Updating Software Performance Analyzer Firmware

Installing/Updating Software Performance Analyzer Firmware

The HP 64708 Software Performance Analyzer firmware is included with the interface software, and the program that downloads the Software Performance Analyzer firmware is included with the HP B1471 64700 Operating Environment product.

The Software Performance Analyzer firmware is supplied under two HP Part Numbers: 64708, and 64708S. You must progflash both the 64708 and 64708S firmware at the same time to properly update the software of the Software Performance Analyzer.

Before you can update the Software Performance Analyzer firmware, you must have already installed the HP 64708A Analyzer card into the HP 64700A, connected the HP 64700A to a host computer or LAN, and installed the Software Performance Analyzer interface and HP B1471 software as described in the "Installation and Service" chapter.

This chapter shows you how to:

- Update firmware with the "progflash" command.
- Display current firmware version information.

To update Software Performance Analyzer firmware with "progflash"

- Enter the **progflash -v <emul_name> <product>** command.

  The **progflash** command downloads code from files on the host computer into Flash EPROM memory on the HP 64708A Analyzer card.

  The **-v** option (verbose) causes progress status messages to be displayed during operation.

The **<emul_name>** option is the logical emulator name as specified in the /usr/hp64000/etc/64700tab.net file, such as m68000.

The **<product>** option names the product whose firmware is to be updated, such as 64708 and 64708S for the Software Performance Analyzer.

If you enter the **progflash** command without options, the downloading process becomes interactive. If you do not include **<emul_name>**, your computer displays the logical names in the /usr/hp64000/etc/64700tab.net file and asks you to choose one. If you do not include **<product>**, your computer displays the products that have firmware update files on the system and asks you to choose one. You can abort the interactive **progflash** by pressing <CTRL>c.

progflash will return 0 if it is successful; otherwise, it will return a nonzero (error) and a message will be written on the standard error output. You can verify the update by displaying the firmware version information.

Examples

To update the Software Performance Analyzer firmware in the HP 64700A that contains the HP 64708A Software Performance Analyzer card:

```
$ progflash <emul_name> 64708 <RETURN>
```

Wait 15 seconds after the above command completes. Then enter:

```
$ progflash <emul_name> 64708S <RETURN>
```

To display current firmware version information

- Use the Terminal Interface **ver** command to view the version information for firmware currently in the HP 64700A.

  When using the Graphical User Interface or Softkey Interface, you can enter Terminal Interface commands with the **pod_command** command. For example:

```
display pod_command <RETURN>
pod_command "ver" <RETURN>
```

Examples The Terminal Interface **ver** command displays information similar to:

```
                  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.03.01 13Mar91

  HP64742 Motorola 68000 Emulator
    Version:   A.00.05 17Feb88
    Speed:     12.5 MHz
    Memory:    126 Kbytes

  HP64740 Emulation Analyzer
    Version:   A.02.02 13Mar91

  HP64708A Software Performance Analyzer
    Version:   A.05.21 22Nov93

  HP64701A LAN Interface
    Version:   A.00.04 21Oct91
```

If there is a power failure during a firmware update

If there is a power glitch during a firmware update, some bits may be lost during the download process, possibly resulting in an HP 64700A that will not boot up.

- ☐ Repeat the firmware update process.
- ☐ If the HP 64700A is connected to the LAN in this situation and you are unable to connect to the HP 64700A after the power glitch, try repeating the firmware update with the HP 64700A connected to an RS-232 or RS-422 interface.

Glossary

absolute Absolute information accounts for all time spent making the measurement, or all cycles executed during the measurement, whether or not the time or cycles met the specifications of any of the events selected for the measurement.

If you select "absolute" in a measurement, you'll get an event named "Address Undefined" that represents all ranges that aren't presently assigned to labels in your data base.

The percentages used to identify each of the selected events will be calculated by including the "Address Undefined" event. To exclude the "Address Undefined" event, calculating percentages only on the selected events, select "Relative" instead of "Absolute".

In addition, if you have defined a disable-enable pair for a duration measurement, you will see "Disable Time". This is the amount of time recorded when the Software Performance Analyzer was in the disable state.

activity Activity measurements are designed to give you a broad understanding of the time and memory bus cycles required by various segments of your program. They identify the areas of your program that use the most processor bus cycles and take the most time to execute. A single activity measurement can capture information on a large number of events. The Software Performance Analyzer makes two types of activity measurements: memory_and_io_activity, and program_activity. These measurements are discussed in detail in Chapter 7 in this manual.

calls The "Calls" column shows how many times the associated event (function or address range) was called to execute, not the number of calls it made to other events. The "Calls" column might show that a particular routine is called an inordinate number of times, such as finding that a malloc routine is called one million times.

confidence Refer to Stability and Confidence.

cycles Cycles is a count of bus cycles that fell within the address range of the defined event. The "Cycles" column might show that a particular module is running slowly (a small number of bus cycles executed over a long period of time). This could indicate inefficiencies, such as a routine that goes out to slow memory for information that it could also find in faster memory, such as in a cache.

defining events You can define multiple events or single events from the command line. The Software Performance Analyzer processes commands differently, depending on whether you are defining multiple events or single events. These differences are described below.

defining multiple events: A command that defines multiple events causes the Software Performance Analyzer to look in the symbols data base for the file running in emulation, and define events to represent symbols it finds there. A "define multiple_events" command can create a set of events to represent functions, static variables, or both. You cannot define multiple events to represent ranges or intervals.

defining single events: You can define single events to represent functions and static variables. If you do this, the Software Performance Analyzer will obtain the address information from the symbols data base for the file running in emulation, just as it does when defining multiple events. When defining single events, you can also define events to represent ranges and intervals.

Refer to Chapter 12 in this manual for a description of events, and to Chapter 2 for complete details of how to define events.

disable when the measurement process is temporarily suspended. Refer to "Enable" for further information.

disable time Disable Time is the time during the measurement when the Software Performance Analyzer was not gathering information (between a disable and a new enable) in a duration measurement. Disable Time is only shown if the data is displayed in "Absolute" format.

duration Duration measurements measure durations of all selected events during the measurement (unlike activity measurements which sample information for only one event at a time). The Software Performance Analyzer records the number of calls, average execution time, maximum execution time, and the minimum execution time for each event. Two types of duration measurements can be made by the Software Performance Analyzer: interval duration, and function duration. These measurements are discussed in detail in Chapter 7 of this manual.

enable When the Software Performance Analyzer is allowed to make a measurement according to its present measurement specification. This occurs after start up or after a disable event was found. You cannot enable (or disable) on an event that is also specified as the trigger event.

events Events in the events list represent absolute addresses. An event may represent a single address (such as the address of a variable), or a range of addresses (such as the range occupied by a function). Through the emulator, the Software Performance Analyzer can access the symbols data base for your absolute file under test and create a list of events whose names and addresses are the same as those in the data base. The Software Performance Analyzer uses the events from its events list in its measurements. The Software Performance Analyzer can recognize events that are appropriate for measurements and events that are not appropriate. Refer to Chapter 12 for complete information about events, and to Chapter 2 for instructions on how to define them.

excluding calls You can use the "exclude_calls" command token when making function-duration measurements. Test results will show the execution times and number of calls to the functions in the measurement. The times recorded for each function will exclude time spent executing other functions that were called by the functions. Time spent servicing interrupts will also be excluded from the recorded times for the functions.

EXPANDED time ranges If you specify that you want to expand an event on a histogram and show time ranges of 10 usec, 20 usec, and 40 usec, then the Software Performance Analyzer will show you three time ranges, as follows:

1 - 10 usec

10+ - 20 usec

20+ - 40 usec

The + sign shown in two of the three time ranges above indicates that the time range begins with times that are slightly longer than the associated number. For example, 10 appears in 1 - 10 usec and in 10+ - 20 usec, above. A time measurement of exactly 10 usec will be recorded in the 1 - 10 usec time range. If the measurement of time is just slightly longer than 10 usec, it will be shown in the 10+ - 20 usec time range.

functions Events defined to represent functions will have the same name as the function they represent, except that underscores "_" will be used to substitute for special characters in the function name. Events that represent functions can be included in measurements of program_activity and function_duration.

function duration When the Software Performance Analyzer makes a function-duration measurement, it measures the execution time of selected source-file functions. A time measurement will start at the first address of function1 and end at the last address of function1. In function-duration measurements, you can have the Software Performance Analyzer either exclude all time spent in calls to other functions (just measure the time spent executing the code of function1), or include all of the time spent in calls to other functions. If including calls, the time measured in function1 is the time from the start of function1 to the end of function1, regardless of where execution occurs between the start and end points. For further details, refer to Chapter 7 in this manual.

including calls You can use "including calls" when making function-duration measurements. Your results will show the total execution time from the function-start address to the function-end address, regardless of where execution may have occurred between the start and end addresses.

interval An interval is any address space defined by a starting address and an ending address. The starting address may have a lower value than the ending address, or it may have a higher value than the ending address. The starting and ending addresses may have the same value. An interval whose starting and ending addresses are the same would be useful if you wanted to measure how often a static variable was accessed for a write transaction. For a detailed discussion of intervals, refer to the paragraph titled, "To define a single event from the keyboard" in Chapter 2.

interval duration When the Software Performance Analyzer makes an interval-duration measurement, it measures the time between two points in a program. The measurement starts when the Software Performance Analyzer detects execution of the first address in the interval, and stops when the Software Performance Analyzer detects execution of the last address in the interval. For further details, refer to Chapter 7 in this manual.

label Refer to "name" in this glossary.

markers Markers are write statements that are instrumented into your code at the start and end of your functions. The Software Performance Analyzer can use markers to make function-duration and interval-duration measurements. When using markers, the duration of a function is measured from the function-start marker to the function-end marker, instead of from the function-start address to the function-end address. Refer to Chapter 7 for a detailed discussion on markers and how to use them.

max The Max column in the table display shows the longest duration measured during the present, or most recent, duration measurement. This was the slowest execution of this event.

mean The Mean column shows the average execution time of the associated event. The details of how this calculation is made are discussed in Chapter 8.

In activity measurements, the mean is the average execution time of the event during a typical 1-second period of program execution. In other words, in any given 1-second period of execution, the associated event will be executing for the indicated amount of time.

In duration measurements, the mean is the average time spent from entry address to exit address. The duration mean is not obtained by sampling. Therefore, it reflects the sum of all execution times for the function or interval, divided by the number of executions of the function or interval.

memory_and_io activity This measurement records reads and/or writes to the addresses of variables and I/O ports, or to any addresses within ranges of addresses. Events that represent functions or intervals are not included in this measurement. For complete details, refer to Chapter 7 in this manual.

min The Min column shows the shortest duration measured during any execution of the associated event. This was the fastest execution of the associated event.

name The name of an event can be 40 or more characters long. Unnamed events will be automatically named by the Software Performance Analyzer.

Most names will be identical to the name of the function or variable that is represented by the event, except that all non-alphanumeric characters will be replaced with underscores '_'. If an event is a numeric address range, the Software Performance Analyzer will create a name that includes the address range of the event (example: _1000h_1010h). A leading underscore will be added to all numeric events. If you want to assign a new name to one of the numeric-named events, or if you want to delete a numeric-named event from your data base, be sure to include the leading underscore in your command, or use the event number.

non-sampled measurement mode In the non-sampled mode, (which is used in "duration" measurements), the Software Performance Analyzer compares incoming information with all of the events included in the measurement at the same time.

pattern filter This dialog box selection area allows you to specify a UNIX shell-type string to limit the list of events created. For example, to omit most assembler-generated symbols, which usually begin with underscores, set the pattern filter to "Not Matching" and "_*". Wild cards can be used in the string (*=any string; ?=any character; [Ab]=either A or b in this character space.

plus (+) sign This is used in time ranges when the Software Performance Analyzer shows that a value is slightly greater than the value indicated. For example, if you specify that you want the Software Performance Analyzer to display a histogram that shows the following time ranges: 10 usec, 20 usec, and 40 usec, then the Software Performance Analyzer will show three time ranges, as follows:

1 - 10 usec
10+ - 20 usec
20+ - 40 usec

The + sign in your display indicates that the time range begins with a time that is just slightly above the associated number. For example, 10 appears in 1 - 10 and in 10+ - 20 above. A time measurement of exactly 10 usec will be recorded in the 1 - 10 usec time range. If the time is just long enough to exceed the 1 - 10 time range, it will be recorded in the 10+ - 20 usec time range.

prefetch correction Prefetch correction occurs when your microprocessor prefetches but does not execute the start or end address of a function or interval the Software Performance Analyzer is trying to measure. Prefetch correction circuitry will attempt to remove prefetched entry and/or exit points when making function duration measurements. Prefetch correction is not used on interval duration measurements.

profile The "profile" command causes the Software Performance Analyzer to perform a measurement of the same type as was performed last. If your last profile measurement was "profile function_duration", the Software Performance Analyzer will do a new "profile function_duration" measurement, by default. If you just entered the Software Performance Analyzer, there is no previous measurement. In this case, a "profile" command will execute the following commands:

```
define multiple_events funcs_and_vars_static global
notmatching '_*'

profile program_activity
```

The following four types of profiles can be performed by the Software Performance Analyzer:

profile program_activity: A measurement that looks for instruction execution. Events that represent functions and ranges can be included in this measurement. Events that represent variables or I/O addresses, or events that represent intervals cannot be included in this measurement.

profile memory_and_io_activity [status: read/write/etc.]: A measurement that records reads and writes to variables and I/O ports. Functions and intervals are not included in this measurement.

Profile interval_duration [status: read/write/etc.]: All events that represent intervals can be included in this measurement. It records time periods between the start and end addresses of the interval. Variables, IO ports, functions, and ranges are not included.

Profile function_duration [exclude_calls/include_calls/exclude_profiled]: Only events that represent source-file functions can be included in this measurement. No variables, intervals, or user-defined address ranges can be included. If you "exclude_calls", the Software Performance Analyzer will exclude all time spent in calls to other functions. If you "include_calls", all time spent in calls to other functions will be recorded as part of the event. If you "exclude_profiled", only other functions being measured will be excluded.

profiled calls This mode of function duration measurement is a compromise between the including-calls and excluding calls measurements. The profiled-calls mode does not record time spent executing code that is already represented by an event that is also being measured. Time spent executing interrupts, and time spent executing code that is not represented by any other event in the measurement is included in the time of the calling event.

profiled time Profiled time is the amount of time that has been completely profiled in making the measurement. If you were to have 1 second of profiled time in an activity measurement, this would indicate that an equivalent of execution time has been completely dissected into all of the specified address ranges, functions, or variables included in the measurement. In duration measurements, profiled time is typically equivalent to the run time.

program activity This measurement records instruction execution within ranges of addresses. Functions and ranges can be included in this measurement. Events that represent variables or I/O addresses, or events that represent intervals cannot be included in this measurement. For complete details about this measurement, refer to Chapter 7 in this manual.

range A range is a range of addresses. The beginning address has a value that is lower than the ending address. During an activity measurement, the range increases its time count when any address within its boundaries is accessed, as long as sampling is in progress for the range event.

recursive functions If a function is recursive, be sure you define it as "recursive" when you create an event for it. Otherwise the results you get for this event may only include a portion of its activity. The Software Performance Analyzer has prefetch correction that allows it to ignore unused prefetches of addresses when it is collecting information about a function. All of the recursive calls could be treated as prefetched entries of a non-recursive function, and the recursive exits could be treated as prefetched exits. This will cause the measurement to be made from the first occurrence of the function entry address to the last occurrence of the function exit address. When a function is identified as recursive, the prefetch correction is turned off. This allows all entries and exits to be recorded. Although, if some of these entries and exits are prefetched, the results of the time duration of a recursive function could be inaccurate. Refer to Chapter 7 for a detailed discussion.

relative Relative measurement results show all time spent executing the selected events, or all cycles executed within the selected events. Relative does not account for time spent or cycles found that were outside the specifications of the events that were included in the measurement.

The percentages shown in a histogram or table display will be calculated as though all execution during the measurement met the specifications of one of the events included in the measurement. To obtain percentages that include executions in addresses that were outside the specifications of the events included in the measurement, use "Absolute" instead of "Relative".

sampled measurement mode Sampling techniques (which are used for activity measurements) allow the Software Performance Analyzer to include up to 254 events in a single measurement. Sampling measurements are done by looking for incoming activity that matches one event at a time. For example, the Software Performance Analyzer may look for activity that meets the specification of event1 for one sample period (approximately 2.5 ms). Any activity that meets the specification of event1 will be recorded. Activity that does not match event1 will be ignored (even if it meets the specification of some other event). After the sample period, the Software Performance Analyzer stops looking for activity that matches event1 and starts looking for activity that matches event2. The Software Performance Analyzer continues to cycle through the selected events in the events list, increasing the total sample time for each event, until the measurement ends.

sorting Events can be sorted by time, calls, cycles, address ranges, event names, or event numbers. The display will reflect the results of the last sorting command.

If the display was sorted by time, calls, or cycles, and you start a new measurement, the display will indicate that the new sort may be invalid. This is indicated by a question mark '?' beside the term "sort" in a column heading, such as, "(sort? time)".

When the measurement is halted, the measurement results will automatically be resorted according to the present sort specification.

If you exit the Software Performance Analyzer and then reenter later, the Software Performance Analyzer will show the content of its memory sorted according to the specification in force at the time you exited.

software performance analysis The Software Performance Analyzer helps a designer understand the execution of the software modules in the absolute file. Software Performance Analyzer measurements may be taken when a designer needs to answer questions, such as:
"Why does it take so long to execute my program?"
"Which module or modules are taking extra-long times to execute?"

Once the modules are identified that are slowing down the performance of the system, the designer can analyze the source files of those offending modules and correct their problems so they run faster.

stability and confidence Stability is an indication of how well the Software Performance Analyzer has characterized the measurement. Stability in a pure sense is a measure of the average of standard deviation error tolerances subtracted from 100%. The longer you let a measurement run, the greater will be the stability of its measurement results. Some measurements will become stable quickly. Others may take a longer period of time to become stable.

The Software Performance Analyzer lets you specify that its measurement must run until its captured data reaches a desired stability. The Software Performance Analyzer will stop the measurement when it has run long enough to give you the stability you specify. Use a command such as:

setup_measurement termination stability 95 percent.

The "display stability ..." command lets you to specify how often the Software Performance Analyzer recalculates its stability during a measurement. You can also turn off the stability calculation (as long as you do not have your termination condition based on stability). If you specify a termination condition based on stability and the stability calculation is turned off, the Software Performance Analyzer will turn it on.

Note that it may take the Software Performance Analyzer up to 2.0 seconds to calculate stability because of the length of time required to unload all of the data.

In addition to the data, your selection of the confidence factor also affects stability. The confidence factor defines the level of confidence that the Software Performance Analyzer must attain in the stability of the measurement. The default confidence for a Software Performance Analyzer measurement is 95%. If you specify that the Software Performance Analyzer terminate its measurement when its data reaches 85% stability, then the measurement will continue until the Software Performance Analyzer is 95% confident that its data has reached at least 85% stability.

Setting the confidence to a higher percentage will require the Software Performance Analyzer to gather data for a longer period of time. Setting the confidence to a lower percentage will allow the Software Performance Analyzer to reach its stability specification more quickly.

You can set any desired confidence to be met by the data captured by the Software Performance Analyzer. Your measurement will continue to run until the Software Performance Analyzer has captured information that meets the confidence and stability you set. Setting the following values of confidence may yield the associated values of stability for equal periods of measurement time:

confidence	stability
51%	96%
95%	85%
99%	75%

You can interpret the above relationship as: the Software Performance Analyzer is only 51% confident that its present measurement data has 96% stability, but it is 99% confident that its data has 75% stability.

standard deviation The Standard Deviation column shows a measure of the variation of the time measurements from the average time measured. It is equal to the root mean square of the individual deviations from the average.

If the display shows Mean=23.3ms, and StdDev=2.89 ms, this means the average measurement falls between 26.19 ms and 20.49 ms (23.3+/-2.89 ms).

The method of calculating standard deviation is discussed in Chapter 8.

static variables A static variable is a fixed location statically allocated variable within your source file. An event that represents a static variable may show a single address or a range of addresses (the range occupied by an array variable, for example). Measurements of memory_and_io_activity are made to record information about static variables.

symbolic filter This dialog box selection area allows you to specify a limit to the scope of the symbols used to create new events. You can type the name of any valid symbol in the text entry field. You can type two or more symbols at a time by using commas (e.g. module1, module2). The default value "<ROOT>" means search the entire symbol database.

tags See markers.

time The time column shows the length of time spent executing code within a defined event. You might find "time" useful if an event normally completes in a few milliseconds, but on rare occasions, it takes several hundred milliseconds to complete. You could trigger a trace if execution of this event should ever exceed some normal execution time. Then your trace might show why these rare executions are taking such a long time. Your trace could answer the question, "What is the routine doing when it takes so long to execute?"

The Time column shows the total time spent executing this event (all executions added together took the amount of time shown). The Time% column shows how much of the total execution time met the specification of the associated event. For additional information, refer to EXPANDED time ranges in this glossary.

trigger Triggers can be generated by the Software Performance Analyzer when making an interval_duration or function_duration measurement. Trigger generation does not affect the measurement being made by the Software Performance Analyzer. It is simply an added output that can be used by the emulator or other analyzers during a measurement. The emulator can be set up to break to its monitor when the trigger occurs. An analyzer can be set up to perform a measurement when the trigger occurs.

There are three rules that apply to triggers generated by the Software Performance Analyzer:

RULE 1: The event that is used as the trigger event must be appropriate for the measurement. If making a function_duration measurement, the event must be a function event. If making an interval_duration measurement, the event must be an inverval event.

RULE 2: The trigger event must be selected for the measurement. If you unselect the event that is specified as the trigger event, the Software Performance Analyzer will automatically reselect it.

RULE 3: Trigger events cannot also be used in enable or disable specifications.

In interval_duration measurements, trigger events have the same status as the inverval_duration measurement. For example, if your command was:
profile inverval_duration status data_write
then your trigger event will only be recognized if it is a write transaction.

Note that the trigger command always calculates its interval as an include_calls time (even if the present measurement is function_duration exclude_calls). In addition, the trigger command is not corrected for prefetch conditions. Prefetch correction is discussed in detail in Chapter 7.

undefined addresses If you select "absolute" for your display of measurement results, the Software Performance Analyzer will add an event named "Undefined Addresses". This event represents all addresses that aren’t presently represented by any events in the measurement. The Software Performance Analyzer always records information for the "Undefined Addresses" event so that it can be displayed if you request measurement results in absolute values.

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.

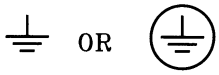
Safety Symbols Used In Manuals

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.

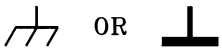
Hot Surface. This symbol means the part or surface is hot and should not be touched.

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

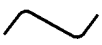


Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.

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 equipment hich 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.