Help Volume

Emulation: Pentium(R) Pro and Pentium(R) II Processor family.

Using the Pentium® Pro/II Emulation Control Interface

The Emulation Control Interface works with an emulation module or emulation probe to give you an emulation interface for target systems that use Intel® Pentium® Pro Processors, Pentium® II Processors, Pentium® II Mobile Processors, and Mobile Modules. With an analysis probe and logic analyzer, you can also make coordinated trace measurements.

See Also

Analysis System help volume)

Glossary of Terms (see page 91)

Related Help

Setting Up and Starting the Emulation Control Interface (see the *Emulation: Setting Up* help volume)

Contents

Contents

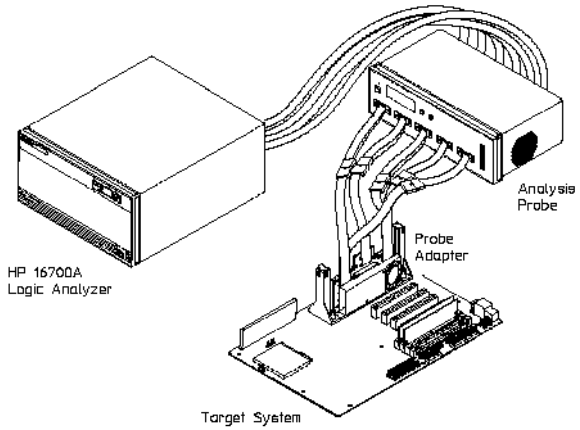
Contents

Contents

1

Using the Pentium® Pro/II Emulation Control Interface

At a Glance



- Logic Analysis System Contains measurement modules such as the 16550A State/Timing Logic Analyzer Module. Contains the Emulation Control Interface that controls the emulation module or emulation probe.
- Analysis Probe Provides convenient connections for state analysis and inverse assembled trace listings for the corresponding Intel processor.

E2466B	Intel Pentium Pro Processor.
E2466C	Intel Pentium II Processor to 300 MHz.
E2487A with E2492A Test Head	Intel Pentium II Processor at 333 MHz and above.
E2487A with E2494A Test Head	Intel Pentium II Mobile Processor and Mobile Module.

- Emulation module or emulation probe Connects to a debug port designed into your target system or to the appropriate analysis probe interface—it accesses the debugging facilities built into the Intel processor to give you control over processor execution and easy access to processor registers, target system memory, and I/O.
- Target System Your system using the Intel Pentium Pro Processor,

Pentium II Processor, Pentium II Mobile Processor, and Mobile Module.

Connecting the Emulator to the Target System

The E3493B Emulator can be connected to a target system directly through a debug port connector on the target board. It can control operation of an Intel Pentium Pro Processor, Intel Pentium II Processor, Intel Pentium II Mobile Processor, or Intel Mobile Module.

Designing a target system test access port

For information about incorporating a debug port into your target design, refer to the *Pentium Pro Family Developer's Manual*, Volume 1, chapter 16, "Tools" (Intel order #2426900-001). Other important information can be found in Chapter 11, "Electrical Specifications."

In addition, POWERON from the target system (pin 9 of the debug port) should be sourced from the GTL and Vtt, not from +5 volts.

To test the E3493A Emulator

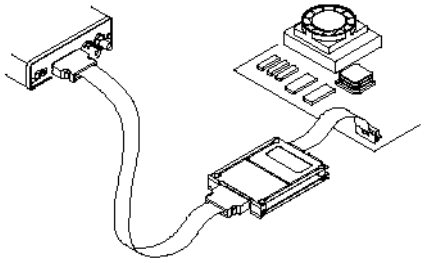
If this is the first time that you have used the emulator, run the built-in performance verification test (see the *Emulation: Setting Up* help volume) before you connect to a target system. Refer to the "Problems" chapter in the *E3493A Pentium Pro Emulator Installation/Service Guide* for detailed information on performance verification.

To connect to a target system via the debug port

Connect the E3493A Emulator to the target system using the 30-conductor cable assembly provided.

1. Disconnect power from the target system and the emulator.
2. Plug one end of the 50-pin cable into the E3493A Emulator.
3. Plug the other end of the 50-pin cable into the interface board.
4. Plug one end of the 30-pin connector into the connector on the interface board labeled "TARGET".
5. Plug the other end of the 30-pin cable into the debug port in the target system.

6. Turn on power to the emulator.
7. Turn on power to the target system.



See Also *E3493A Pentium Pro Emulator Installation/Service Guide*

Configuring the Emulator

Use the *Setup* window to configure the emulator for your target system.

1. Open the Setup window.
2. Make the appropriate selections for the options you wish to change.

   The emulator configuration options are:

   - “JTAG Scan Chain” on page 14
   - “JTAG Frequency” on page 16
   - “"Break In" Port” on page 17
   - “"Trigger Out" Port” on page 17
   - “When a Processor Resets” on page 18
   - “When a Processor Inits” on page 19
   - “Break on Read/Write to Debug Registers?” on page 19
   - “Turn Cache On/Off” on page 20
   - “Branch Trace Messaging” on page 20
   - “Break on SMM Mode” on page 20
3. When you have finished making changes, select *File* and then select *Close*.

See Also

- “To save configuration settings” on page 21
- “To restore saved configuration settings” on page 21

JTAG Scan Chain

You must define the order of the processors and other devices on the JTAG chain at the debug port.

CAUTION: Be sure to set the JTAG chain correctly. An incorrect JTAG chain can damage your target system.

To define the JTAG chain, use the following syntax:

```
{p|<ilength>}[,{p|<ilength>}...]
        or
{p[0-3]|<ilength>}[,{p[0-3]|<ilength>}...]
```

- *p* defines a Pentium Pro processor; the first is p0, the second is p1, and so on.
- *p[0-3]* defines a Pentium™Pro processor which uses the specified preq# line
- *<ilength>* defines the JTAG instruction length of a non-Pentium Pro device

The processor number (p0, p1, ...) corresponds to the line on the debug port (PREQx#) that controls the processor.

After you have entered the JTAG chain definition, press the *Enter* key to make the new definition take effect.

Tips

- In some systems, it may be easier to enter the JTAG scan chain while the system is powered up and running. After the JTAG scan chain is entered, it is verified. The scan chain can only be verified when the processors are not reset. In some systems the act of identifying the scan chain causes an unacceptable delay and the processors enter monitor and stay there. To avoid this, let the processors run as you enter the scan chain; then you may reset and run normally.
- If you do not know your scan chain, you may enter the command line window and enter the `gchain` command. Read the warnings carefully and decide whether you should actually have the emulator run the tests to discover the scan chain.

Examples

- p

  This defines a single processor. In the Run Control Tool windows, this processor will be identified as processor number "0".

- p,p

  This defines a chain with two processors. The processors are assumed to be connected to the PREQ0# and PREQ1# lines, respectively. In the Run

Control Tool windows, the processors will be identified as processor "0" and "1".

- p,10,7,7,p,10,p

  This defines a chain with 3 processors and 4 other devices. This assumes that the first processor uses PREQ0#, the second uses PREQ1#, and so on.

- p1,10,7,7,p3,10,p0

  This specifies that the first processor uses PREQ1#, and so on.

NOTE: If you move the emulator from one target system to another, be sure to turn the emulator off and then on again, or reconfigure the JTAG chain.

JTAG Frequency

This configuration option specifies the clock speed (JTAG TCK frequency) at which the emulator communicates with the debug port on the target system.

The JTAG frequency of the Pentium Pro/II Series Processor must be less than 1/6 the bus frequency.

The speeds you can specify are:

```
10 MHz   (10.4 MHz; use for bus frequency > 66 MHz)
5 MHz    (5.2 MHz;  use for bus frequency > 33 MHz)
2.5 MHz  (2.6 MHz;  use for bus frequency > 15 MHz)
1.5 MHz  (1.3 MHz;  use for bus frequency > 8 MHz)
0.6 MHz  (650 kHz;  use for bus frequency > 4 MHz)
.3 MHz   (325 kHz;  use for bus frequency > 2 MHz)
.1 MHz   (163 kHz;  use for bus frequency > 1 MHz)
.05 MHz  (81 kHz;   use for bus frequency > 0.5 MHz)
```

If your target system has additional loads on the JTAG lines, or if it does not meet the requirements described in the *Emulator for Pentium Pro Installation and Service Guide*, using a slower JTAG clock speed may enable the emulator to work.

"Trigger Out" Port

The Trigger Out BNC port can be used to tell devices external to the emulation probe when processor execution is in the *monitor*.

You can use this port to qualify a logic analyzer clock, so that monitor cycles are not captured.

You can also use this port to signal the target system, for example, so that watchdog timers don't time-out when processor execution is in the monitor.

The options are:

High On Any Processor Break

Output is high when any processor breaks to the monitor.

Low On Any Processor Break

Output is low when any processor breaks to the monitor.

Always High

Output is always high.

Always Low

Output is always low.

NOTE: If you are using the emulation module, this same functionality can be set up in the Group Run Arming Tree of the Intermodule window.

"Break In" Port

The Break In port allows devices external to the emulation probe (for example, a logic analyzer's trigger output) to stop user program execution.

Break All Processors On Rising Edge

Rising edges on this port will cause processor execution on all processors

to break into the *monitor*. This is the proper selection when the port is connected to the logic analyzer trigger output.

Break All Processors On Falling Edge

Falling edges on this port will cause processor execution on all processors to break into the *monitor*.

Disabled

Rising or falling edges on this port have no effect on the emulation probe.

NOTE: If you are using the emulation module, this same functionality can be set up in the Group Run Arming Tree of the Intermodule window.

When a Processor Resets

This configuration option specifies how the emulator behaves when any of the target Pentium Pro processors is reset.

Restore Debug Registers, Break

A RESET will break into the monitor. Debug registers will be restored, and the monitor will continue running until another command causes it to exit.

Restore Debug Registers, Run

A RESET will break into the monitor. The debug registers will be restored, and then the user program will begin running.

Restore Debug Registers if Used, Run

A RESET will break into the monitor. Debug registers will be restored, and the user program will begin running.

Do Not Restore Debug Registers, Run

A RESET will immediately run the user program. A RESET will not break into the monitor. The debug registers will not be restored.

Notes:

1. These actions occur only when RESET is deasserted.

2. Selecting *Reset* in the Run Control window or typing "rst" in the Command Line Interface window asserts RESET continuously until another button or command causes RESET to be deasserted.

When a Processor Inits

This configuration option specifies how the emulator behaves when the Pentium Pro/II Series Processor's INIT# input signal is asserted (without the processor's RESET input signal being asserted).

Restore Debug Registers, Break

An INIT without a RESET will break into the monitor. Debug registers will be restored.

Restore Debug Registers, Run

An INIT without a RESET will break into the monitor. Debug registers will be restored, and the user program will begin running.

Do Not Restore Debug Registers, Run

An INIT without a RESET will not break into the monitor. Debug registers will not be restored. The user program will begin running immediately.

Break on Read/Write to Debug Registers?

Yes	Any program reads or writes to the debug registers will cause a break. The actual read or write will not be performed.
No	Any program reads or writes to the debug registers do not cause a break. The read or write completes successfully.

NOTE: Selecting *No* makes it is possible for your program to turn off or modify breakpoints that you have set, so use this with caution.

Turn Cache On/Off

This option enables or disables the cache for all processors.

The cache is enabled or disabled immediately when you select the *On* or *Off* button.

CAUTION: Be careful when data in cache may not be the same as data in corresponding memory. Refer to the processor user's manual, if necessary.

NOTE: The software running in your system under test can alter the on/off state of the cache. The selection you make here does not prevent the software from changing the state of the cache afterward. For example, if you enter Windows, windows will set the cache to *On*.

Branch Trace Messaging

This option can be controlled for each processor in a multiprocessor system.

None No Branch Trace Message special cycles are generated.

Bus When a branch is taken, the processor will emit a special cycle. The analyzer can use this to do accurate instruction disassembly.

Register When a branch is taken, the processor will store the linear address of the source of the branch in the register named 'lastbranchfromip' and the destination in '000024bc'. If the branch was caused by an interrupt, the registers 'lastintfromip' and 'lastinttoip' are used. The processor defaults to *Register*.

Break on SMM Mode

This configuration option allows you to control what happens when a processor enters System Management Mode (SMM).

None

The processor will not break when it enters or exits SMM.

Break on Entry

The processor will break into the monitor when it enters SMM.

Break on Exit

The processor will break into the monitor when it exits SMM.

Break on Both Entry and Exit

The processor will break into the monitor when it enters SMM, and then it will break again when it exits SMM.

To save configuration settings

You can save emulation configuration settings as part of a logic analysis system configuration file. This saves all workspace configurations, including the emulation module and emulation probe configuration settings.

See Also

Saving a New Configuration File (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume)

To restore saved configuration settings

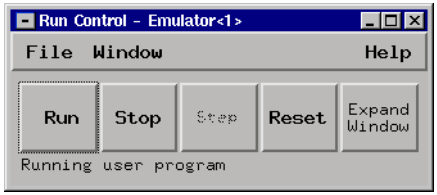
If you saved emulation configuration settings as part of a logic analysis system configuration file, you can restore them by loading the configuration file. This restores all workspace configurations, including the emulation module and emulation probe configuration settings.

See Also

Loading Configuration Files (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume)

Controlling Processor Execution

Processor execution is controlled using the Run Control window.



- To run the program, select *Run*.
- To stop program execution, select *Break*.
- To reset the processor, select *Reset*.
- To step a single instruction, select *Step*.
- To control specific processors in a multiprocessor system, select *Expand Window*.

The *status line*, which appears under the run control buttons, shows the current status of the emulator. The status codes are:

R - emulator forcing reset

r - target forcing reset (the target system reset line is active)

U - running user program

M - running monitor program (in other words, executing the debug exception routine that provides the emulator's capability)

p - no target power

s - waiting to break into monitor

e - exiting monitor

? - JTAG scan chain unknown

See Also

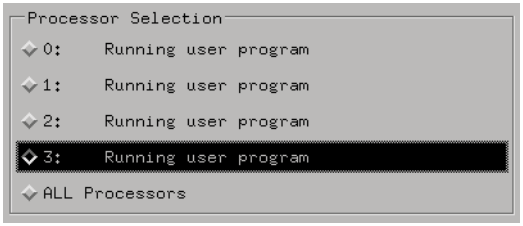
- "Controlling Processor Execution in a Multiprocessor System" on page 23

Controlling Processor Execution in a Multiprocessor System

To control the execution of individual processors, select *Expand Window* in the *Run Control* window.

To select a processor

- Select the button next to the processor you want to control.



The processor numbers are defined by the target system's “JTAG Scan Chain” on page 14.

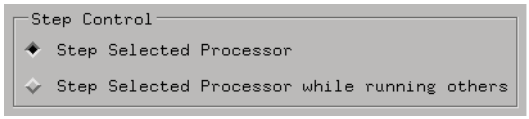
To run a single processor

- Select the processor.
- Select *Run*.

  The selected processor will run. The other processors will run in the monitor.

To step a single processor

- Select the processor.
- Select what the other processors should do:
  - *Step Selected Processor* will leave the other processors unaffected when you select *Step*.
  - *Step Selected Processor while running others* will run the other processors when you select *Step*.

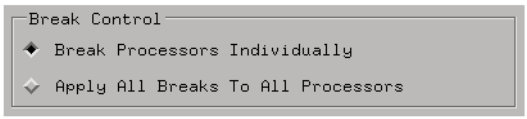


- Select *Step*.

  Reset will be released and the selected processor will execute one instruction. Then the selected processor will run in the monitor.

To break a single processor

- Select the processor.
- Select *Break Processors Individually*.



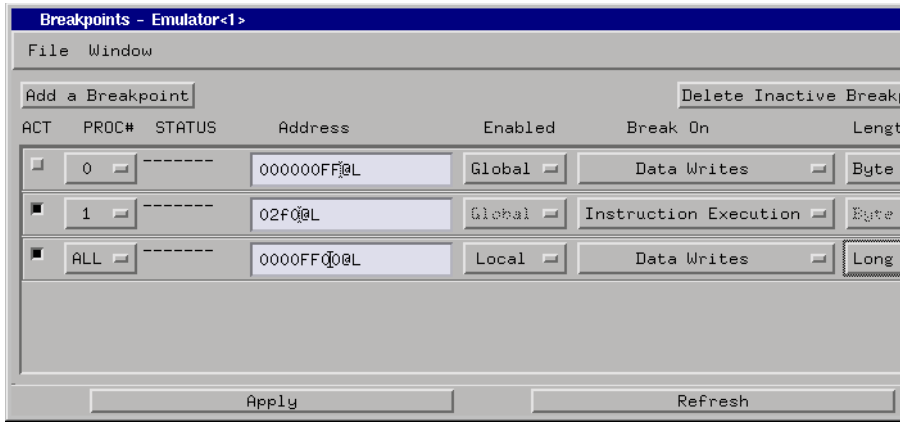
- Select *Break*.

  The selected processor will break and then run in the monitor. The other processors will be unaffected.

Note that breaking a single processor may cause problems with software timers on other processors. In some target systems, you may always need to use *Apply All Breaks to All Processors*.

Using Breakpoints

The Breakpoints window allows you to set breakpoints to stop processor execution when an instruction at a particular address is executed, or when a memory location is written or read.



To set a breakpoint

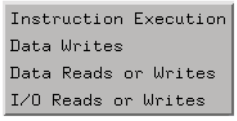
1. Select *Add a Breakpoint*.
2. Select the enable/disable toggle button to enable the breakpoint.

   A breakpoint is set when the button is in: 
3. Select the processor number to which the breakpoint applies.
4. Enter the address of the breakpoint.

NOTE: Breakpoint addresses may be linear (see page 36) or real mode (see page 36). addresses.

The addresses you see in the Memory Disassembly window or in logic analyzer traces are physical addresses. When processor paging is disabled, linear addresses are equivalent to physical addresses, so you can enter a physical address as if it were a linear address (for example, "000000FF@L"). When paging is enabled, use the logic analyzer to break on physical addresses (see “To break execution on a trigger” on page 46).

5. Select what to break on:



6. If you selected a non-execution (see page 28) breakpoint, select whether it should be enabled globally or locally.

   Typically, breakpoints are *Global*. *Local* breakpoints apply only to the task that is active when the breakpoint is set. A task switch will clear *Local* breakpoints.

7. If you selected a non-execution breakpoint, select the data access size:

   If an address in the range *Address .. Address + Access Size* is accessed, execution will break.

8. Select *Apply* to set the breakpoint.

NOTE: Breakpoints are not set by the emulator until you select the *Apply* button.

After you select the *Apply* button, the STATUS of breakpoints which were successfully set will be labeled "ENABLED".

If you set a breakpoint on *ALL* processors, the breakpoint will "split" into individual breakpoints for each of the processors; if the breakpoint

cannot be set on one of the processors, it will still be set on the other processors.

If you set more than one breakpoint at the same address on the same processor, all but one of the breakpoints will be deleted.

Breakpoints are saved when you save a workspace configuration (see page 21).

All breakpoints are hardware breakpoints (see page 28).

To clear a breakpoint

- Select the enable/disable toggle button to disable the breakpoint.

  A breakpoint is cleared when the button is out: 

- Select *Apply* to clear the breakpoint.
- If you do not plan to enable the breakpoint again, select *Delete Inactive Breakpoints* to remove all of the cleared breakpoints. Selecting *Delete Inactive Breakpoints* will also set the enabled breakpoints (as if you had selected *Apply*).

To see what breakpoints are set

- Select *Refresh*.

Any breakpoints you have entered but not applied will be deleted.

This is useful if you have a debugger or other tool which may have set breakpoints, or if you want to confirm that the breakpoints you have set are still in effect.

See Also

"Addresses" on page 36

"Hardware breakpoints" on page 28

"Software breakpoints" on page 28

"Breakpoint status" on page 28

Hardware breakpoints

When you set a hardware breakpoint, the emulator sets up a breakpoint using the Pentium Pro or Pentium II debug registers.

You may set up to four hardware breakpoints per processor.

For more information on the Pentium Pro/II debug registers, see Volume 3 of the *Pentium Pro Family Developer's Manual*.

Software breakpoints

When you set a software breakpoint, the emulator replaces the instruction at the specified address with a breakpoint instruction (INT3).

The emulator stores the original instruction and replaces it when the breakpoint is disabled.

If you *Step* or *Run* from the breakpoint, the original instruction will be executed.

You can only set software breakpoints in the command line interface.

Non-execution breakpoints

Breakpoints other than *Instruction Execution* breakpoints.

For example, breakpoints on data reads or data writes are non-execution breakpoints.

Breakpoint status

The STATUS field in the Breakpoints window can have any of the following values:

Enabled	The breakpoint is set.
Disabled	The breakpoint is not set, but the address and other information is stored in the emulator for future use.
Unknown	The emulator was unable to get information about the breakpoint from the target processor, so it can't determine if the breakpoint is Enabled or Disabled.
------	The breakpoint has never been applied. The breakpoint can be treated as "Disabled", except that the emulator does not know that the breakpoint exists.

Displaying and Modifying Registers

The Register window lets you display and modify the contents of processor registers.

- "To display registers" on page 30
- "To modify register contents" on page 31

The Emulation Control Interface displays groups (also called "classes") of related registers.

To display registers

- To display registers for a particular Pentium Pro or Pentium II processor in the JTAG chain, select the button next to the number of that processor.
- To add a group of registers to the display, select the group from the *Groups* menu.
- To remove a group of registers from the display, select the group from the *Groups* menu.
- To move a group to the bottom of the display, remove the group and then

add it again.

- To read the register values from the processor, select *Read Registers*.

The Registers window will be updated as the processor runs. If you want to disable this automatic updating, change *Freeze Window (No)* to *Freeze Window (Yes)* in the Update menu.

See Also "Accessing segment register hidden fields" on page 31

To modify register contents

1. Open a Registers window and select a processor.
2. Display the group of registers that contains the register whose contents you wish to modify.
3. Select the value field you wish to modify. The underline cursor indicates the field that has been selected.
4. Type the new register value.

   The new value is actually written as soon as:

   - you press the Enter (or Return) key, or
   - the mouse pointer leaves the entry field.

See Also "Accessing segment register hidden fields" on page 31

Accessing segment register hidden fields

Pentium Pro/II segment registers have a "hidden" part which can be viewed and set by the Run Control Tool.

In the Registers window, the hidden part is shown as three fields:

- `.b` is the base address.
- `.l` is the segment limit.
- `.a` is the access control information.

The segment limit field (.f) takes the G bit of the associated access control (.a) register into account. This means that the value in the limit field does not need to be muliplied by 4K if the G bit is set; the multiplication has already been done.

If you enter a value larger than 1M (fffff) into the limit field, it will automatically set the G bit when written to the processor, and the 12 least significant bits will be lost. For example, entering the value f12345 will result in the limit being set to f10000.

See Also

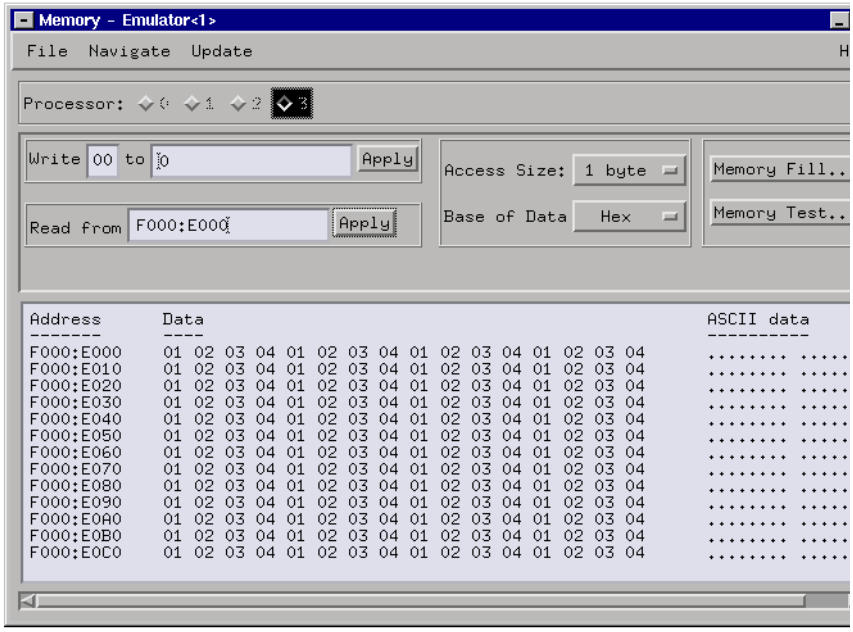
For more information on the segment registers, see Chapter 3, "Protected-Mode Memory Management," in Volume 3 of the *Pentium Pro Family Developer's Manual*.

For more information on segment limits, see the LSL instruction in Volume 2 of the *Pentium Pro Family Developer's Manual*.

Displaying and Modifying Memory

To display memory

1. Open the Memory window.
2. Specify the display format by selecting the appropriate *Access Size* and *Base of Data* options.
3. Enter the address (see page 36) you wish to read in the *Read from* field, and select *Apply*.

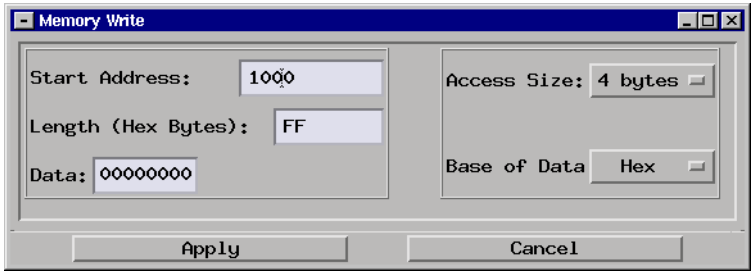


To modify a memory location

1. Open the Memory window.
2. Select *Memory Write...*.
3. Enter the address (see page 36) of the location in the *Start Address* field.
4. Enter a length of 1. (Entering a larger value will fill multiple memory locations with the value.)
5. Enter the value that you want to write in the *Data* field.
6. Specify the size of the memory location and the format of the value you wish to enter by selecting the appropriate *Access Size* and *Base of Data* options.
7. Select *Apply*.

To fill a range of memory locations

1. Open the Memory window.
2. Select *Memory Write...*.
3. Specify the size of the memory locations and the format of the value you wish to enter by selecting the appropriate *Access Size* and *Base of Data* options.
4. Enter the address (see page 36) of the first location in the *Start Address* field, enter the number of addresses to be written with the value in the *Length* field.
5. Enter the value that you want to write in the *Data* field.
6. Select *Apply*.



To display memory in mnemonic format

- Open the Memory Disassembly window.

  Memory is disassembled beginning at the current instruction pointer value.

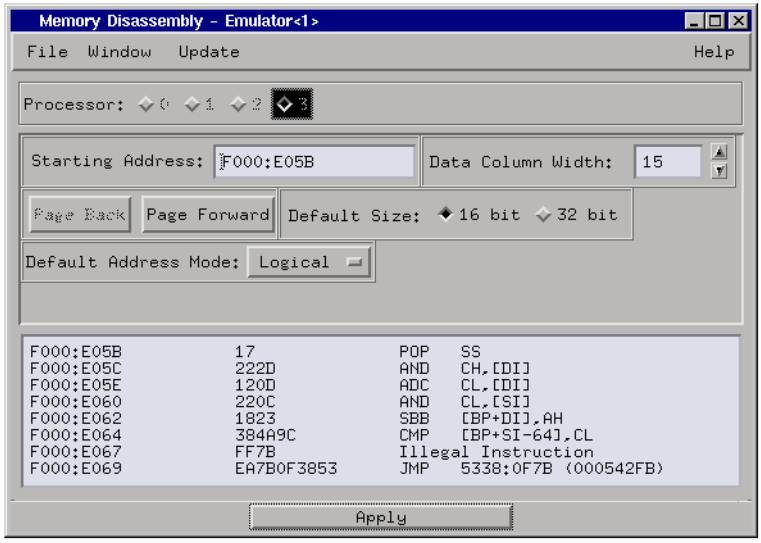
  The line corresponding to the instruction pointer will be highlighted.

To display memory beginning at another address

1. Enter the first address (see page 36) to disassemble in the *Starting Address* field.
2. Select the appropriate *Data Column Width* and *Default Size* options.

3. Select *Apply*.

   The address format displayed will match the format of the address you entered in the *Starting Address* field.



Scrolling

Page Forward and *Page Back* let you scroll the displayed memory locations. You cannot page back beyond the starting address.

Instruction pointer tracking

If you step the processor or run and then stop, the highlighted line will track the current instruction pointer. The address format displayed will match the *Default Address Mode* setting.

If you do not want the highlighted line to track the instruction pointer, select *Freeze Window (No)* in the Update menu.

Addresses

Enter addresses in any of the following formats:

- Physical mode

  Example: `0123ABCD`

- Linear mode

Example: `0123ABCD@L`

- Real mode (segment:offset)

  Example: `0123:ABCD`

- Selector::offset

  Example: `0123::ABCDEF89`

- GDT:LDT:offset

  Example: `0123:4567:ABCDEF89`

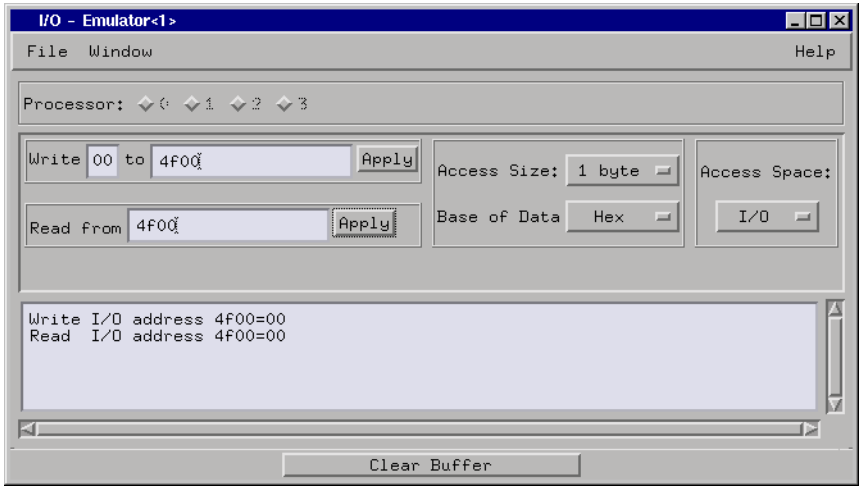
See Also

The Intel *Pentium Family Developer's Manual*.

Displaying and Modifying I/O

1. Open the I/O window.
2. Select the processor.
3. Specify the display format by selecting the appropriate *Access Size* and *Base of Data* options.
4. Use the *Access Space* option to select whether to use memory-mapped I/O or actual I/O space.
5. To display an I/O location, enter the address you wish to read in the *Read from* field, and select *Apply*.
6. To modify an I/O location, enter the value that you want to write in the *Write* field, enter the address of the location in the *to* field, and select *Apply*.

   If the *Access Space* option is set to *I/O* (actual I/O space), the address must by a physical address less than 64K. If the *Access Space* option is set to *Memory* (memory-mapped I/O space), the address mode can be physical, linear, or logical (see page 36).



Clear Buffer empties the contents of the I/O window.

If the *Access Space* option is set to *I/O* (actual I/O space), the address

must by a physical address less than 64K. If the *Access Space* option is set to *Memory* (memory-mapped I/O space), the address mode can be physical, linear, or logical (see page 36).

Downloading an Executable to the Target System

Use the Load Executable window to load executables (or other data) into your target system.

1. Save the executable where the logic analysis system can read it (see page 40).
2. Open (see page 48) the Load Executable window.
3. Select the processor.
4. Select the format (see page 41) of the file.
5. Change the *Access Size* option, if necessary.
6. Enter the name of the file to load.
7. Select *Apply*.

   When the file has been successfully loaded, "Load completed" message will be displayed.

NOTE: To improve downloading speed, choose the largest Access Size under Load Options in the Load Executable window. When this help file was written, 4 bytes was the largest Access Size.

See Also

- "To access a file from the logic analysis system" on page 40
- "To choose a file format" on page 41
- "Error messages while downloading files" on page 42

To access a file from the logic analysis system

You need to save the executable file where the logic analysis system can read it.

If you will be downloading many files, you should NFS-mount the file

system (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume) of the computer where you are compiling your program. If you do this, you can directly access your executable, regardless of how big it is, as soon as it is compiled or assembled. Mounting the file system may require the help of a system administrator.

You can also copy the file (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume) from a floppy disk to the hard disk of the logic analysis system. If the file is too big to fit on the floppy disk, use PKZIP to compress the file and then uncompress (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume) the file using the File Manager.

To choose a file format

You can choose one of the following file formats:

- Motorola S-records Most compilers for Motorola microprocessors can generate Motorola S-records. (see page 43) This file format represents the binary data as hexadecimal numbers in a ASCII file. Each record in the file includes a load address for the data in the record.
- Intel Extended Hex The Intel extended hex (see page 44) format represents binary data as hexadecimal numbers in a ASCII file. Each record in the file includes a load address for the data in the record.
- Plain binary If you choose *Plain binary*, the data in the file will be copied directly to the target system. Because plain binary files contain no information about where to load the file in memory, you must enter a start address.
- ELF Object Many compilers generate ELF Object files. This option will allow statically linked ELF images to be loaded into the target memory. Dynamically linked ELF shared libraries cannot be loaded with this option.

If your executable is not in one of these formats, recompile, reassemble, or relink your program with the appropriate options to generate one of these formats.

See Also

- “Details of the Motorola S-record format” on page 43

- “Details of the Intel extended hex format” on page 44

Error messages while downloading files

- Bad Load Address The start address which you entered for the plain binary file is invalid.
- Checksum errors found in non-data records. Checksum errors were found in one or more non-data records, but no errors were found in the data records. The data records were loaded.
- ELF file has no object image to load. Load not completed. This means we found a valid ELF Object file, but the target image was not included with the file. With no target image, we had nothing to load.
- File could not be opened. The executable file could not be opened. Check that the file exists, that the file permissions allow reading, and that the file is a data file and not a directory.
- File not in Intel Extended Hex file format. Load not completed, check format selection. Check that you have selected the correct file format button. If you still get this message, compare the file to the description in “Details of the Intel extended hex format” on page 44.
- File not in Motorola S-Record file format. Load not completed, check format selection. Check that you have selected the correct file format button. If you still get this message, compare the file to the description in “Details of the Motorola S-record format” on page 43.
- Load Failed: binary data load exceeds memory boundary. If the plain binary file were to be loaded at the the start address which you entered, some of the data would be written past the end of memory. Check the start address, then check that the file has a size that will fit in your target system's memory.
- Load failed: Checksum Errors found in data record(s). Checksum errors were found in one or more data records. The load was aborted: target memory is unchanged. Generate a new executable file.
- Load Failed: ELF file has a format problem. ELF file has been altered in some way that we can no longer recognize it as an ELF file.
- Load Failed: Refer to Status/Error Log window. The Status/Error Log

window will contain additional information about the errors that occurred.

Details of the Motorola S-record format

An S-Record file has the following format:

```
[$$[module_record]
symbol records
$$[module_record]
symbol records
$$]
header_record
data_records
record_count_record
terminator_record
```

Each record in the file consists of

- Record type (2 characters)
  - S0 - Header record
  - S1,S2,S3 - Data record
  - S5 - Record count
  - S7,S8,S9 - Terminator
- Record length, in bytes, not including the record type field (2 characters)
- Load address (2-8 characters)
- The data, represented in hexadecimal. (length determined by the record length field)
- Checksum, calculated as the 1's complement of all bytes in the record, not including the record type field (2 characters)

When downloading a file, anything before the header (such as any module or symbol records) is ignored. All optional records are ignored.

Example

Here is an example of an S-Record file:

```
S00600004844521B
```

```
S20700665400B24C40
S20500665F45F0
S21400600033FC000A0000B24C202F00046C0622006D
...
S2060066424E758E
S50300728A
S80400624455
```

Details of the Intel extended hex format

Each record in the file consists of

- A colon (:) (1 character)
- Number of bytes in the data field (2 characters)
- Load address (4 characters)
- Record type (2 characters)
  - 00 Data
  - 01 End of file
  - 02 Extended segment address (bits 4-19 of a segment base address)
  - 03 Start segment address
  - 04 Extended linear address (the upper 2 bytes of a data load address)
  - 05 Start linear address
- The data, represented in hexadecimal. (length determined by the record length field) For the extended address record types, the data consists of 4 characters representing the appropriate bits of the address.
- Checksum, calculated as the 2's complement of the sum of all of the bytes in the record after the colon (2 characters)

When downloading a file, only record types 00, 01, 02, and 04 are parsed. Any additional records (such as any module or symbol records) before the data records are ignored.

Example

Here is an example of a data record followed by an end of file record:

```
:20000000F00100230B340C4510561B6720782B892CAB30BC40CD4CD
E50EF60F06C01701290
:20002000AC2301011123214531674189 51AB61CDF1EF7121D001C01
F08F9090F18F7190660
:2000400028F5290438F3390248F1490058FC590D68FE690F78FA790
988109812A834B865A7
:20006000C867D889E89A8180A1E0E0ABE1464B00AB00AB21AB32AB4
3AB54AB65AB76AB874B
:20008000AB98AB299C309C819CF29CE39CC49CB59CD69CA79C18BBF
F9B38EC0CEC0DEC0EC1
:2000A000EC0FEC86EC8AEC8BEC8CEC8DEC87EC08EC04EC00EC01EC0
2EC03EC06EC07EC8F88
:0A01A0000000D7200000D780030004
:00000001FF
```

Coordinating Trace Measurements

If you are using a logic analyzer (and perhaps an analysis probe) along with an emulation probe to debug your target system, you can coordinate the logic analyzer trace measurements with the execution of your target processor.

To break execution on a trigger

With an emulation module

1. Create a logic analyzer trigger.
2. In the Intermodule Window (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume), select the emulation module icon and then select the analyzer that is intended to trigger it.
3. Select *Run* in the logic analyzer window to start the trace measurement.
4. Start processor execution, if necessary.

   When the trigger occurs, processor execution should break into the monitor.

With an emulation probe

1. Connect the trigger output to the emulation probe's "Break In" port.
2. Configure the "Break In" port to either break on rising or falling edge signals (see "Configuring the Emulator" on page 14). When using the logic analyzer trigger output, configure the emulation probe to break on a rising edge signal.
3. Start the trace measurement. When the trigger occurs, processor execution should break into the monitor.

4. Start processor execution.

This type of coordination lets you break processor execution on more specific conditions than are provided by the debug register breakpoints. For example, you can trigger and break on the write of a particular value to a particular address.

See Also

- To enable or disable emulator break on trigger (see the *Listing Display Tool* help volume).
- To trigger after, about, or before a source line (see the *Listing Display Tool* help volume).

To trigger an analyzer on a break

If you want to trace execution that occurs before a break:

With an emulation module

1. Set the logic analyzer to trigger on *anystate*.
2. Set the trigger point to *center* or *end*.
3. In the Intermodule Window (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume), select on the logic analyzer you want to trigger, then select the emulation module.

   The logic analyzer is now set to trigger on a processor halt.
4. Wait for the Run Control window to show that the processor has stopped.

NOTE: The logic analyzer will store states up until the processor stops, but will continue running. When the processor stops, you may see a "slow clock" message, or the logic analyzer display may not change at all.

5. In the logic analyzer window, select *Stop* to complete the measurement.

With an emulation probe

1. Connect the emulation probe's "Trigger Out" port to the analyzer's input port.
2. Configure the "Trigger Out" port to either be high or low when processor

execution is in the monitor (see “Configuring the Emulator” on page 14).

3. Configure the logic analyzer's input port. For example, if "Trigger Out" is low when in the monitor, configure the analyzer to look for a falling edge.
4. Set up the analyzer to trigger on the input signal (storing states that occur before the trigger).
5. Start the trace measurement.
6. Start processor execution. When the break occurs, the analyzer should trigger.

To omit monitor cycles from the trace

If you have an emulation probe, you can omit monitor cycles from a logic analyzer trace by using the "Trigger Out" port signal as an analyzer clock qualifier. That is, the analyzer is only clocked when processor execution is outside the monitor.

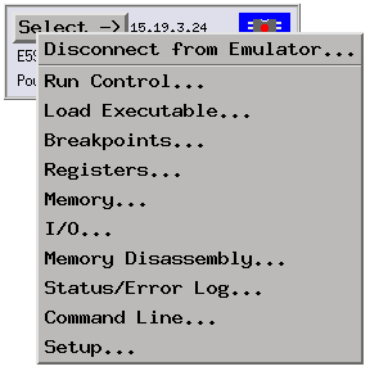
1. Connect the emulation probe's "Trigger Out" port to the analyzer's clock qualifier input.
2. Configure the "Trigger Out" port to either be high or low when processor execution is in the monitor (see “Configuring the Emulator” on page 14).
3. Configure the logic analyzer's clock qualifier input. For example, if "Trigger Out" is low when in the monitor, configure the analyzer to clock only when the qualifier input is high.
4. Set up and start analyzer trace measurements normally. The processor must be executing user code (not executing in the monitor) in order for data to be clocked into the analyzer.

To open windows

The Emulation Control Interface gives you several ways to open new windows:

Opening windows from the Emulation Control Interface icon

1. Move the mouse cursor over the Emulation Control Interface icon in the system window or in the workspace window.
2. Press and hold the right mouse button.
3. Move the mouse cursor over the menu selection for the window you wish to open.
4. Release the right mouse button.



Opening windows from the Window menu

- In any emulation window menu bar, select *Window*.
- Select the emulator.
- Select the window you want to open.

Creating and naming new windows

You can open several copies of certain windows, such as the Memory window.

1. In the menu bar, select *File* then select *New Window*. The new window will be given a number, such as `Memory <<3>>`.
2. (Optional) In the new window, select *File* then select *Rename*. Enter a new name for the window and select *OK*.

The new window can be opened from the Emulation Control Interface icon or from the Window menu.

To close windows

- In the menu bar, select *File* then select *Close*.

Auto-close

You can turn on Auto-close so that when you select a new window from the *Window* menu, the current window will be closed. To turn on Auto-close, select *Window* in the menu bar, then select *Auto-Close [ON]*.

Deleting windows

When you create a new window using *File->New*, the window will still be listed in the *Window* menu after you have closed it. To delete the window from the list, open the window then select *File->Delete*.

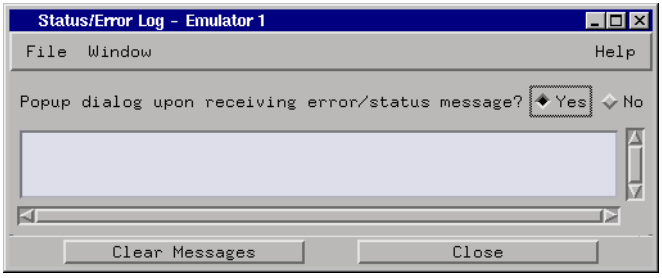
Managing Run Control Tool Windows

To use the Status/Error Log window

The Status/Error Log window is the central repository for error and status messages.

To specify when the Status/Error Log window appears:

1. Open the Status/Error Log window.



2. If you want the Status/Error Log window to automatically appear every time an error or status message occurs, select *Yes* for *Popup dialog upon receiving error/status message?*. If you only want the Status/Error Log window to appear when you open it, select *No*.
3. Close the Status/Error Log window.

Clear Messages empties the contents of the Status/Error Log window.

See Also

"Error/Status Messages" on page 59

To Use the Command Line Interface

1. Open the Command Line window.
2. Enter commands in the *Command Input* field.

   The `help` command provides command syntax and other information. For example, for help on the `m` command, enter the `help m` command.

   Enter comments by beginning a line with the `#` character.
3. Each line begins with a status prompt, such as M> or U>.
   - M> indicates your target system is in the background debug mode, not running user code.
   - U> indicates your target system is running user program code.

   For definitions of all of the status prompts you might see in the Command Line window, enter, *help proc*.

The command line interface is useful, for example, for creating scripts that initialize the target system to a known state. For example, scripts can set initial register or memory values or write an I/O sequence.

A few commands, such as desc (see page 56) and gchain (see page 56), are only available through the command line interface.

Playback Script will execute commands that have been saved in a script file.

Edit Script opens a simple text editor that lets you enter and save a sequence of commands in a file.

Clear Buffer clears the Command Output buffer.

Memory Test... displays the Memory Test window where you can evaluate your target system memory hardware.

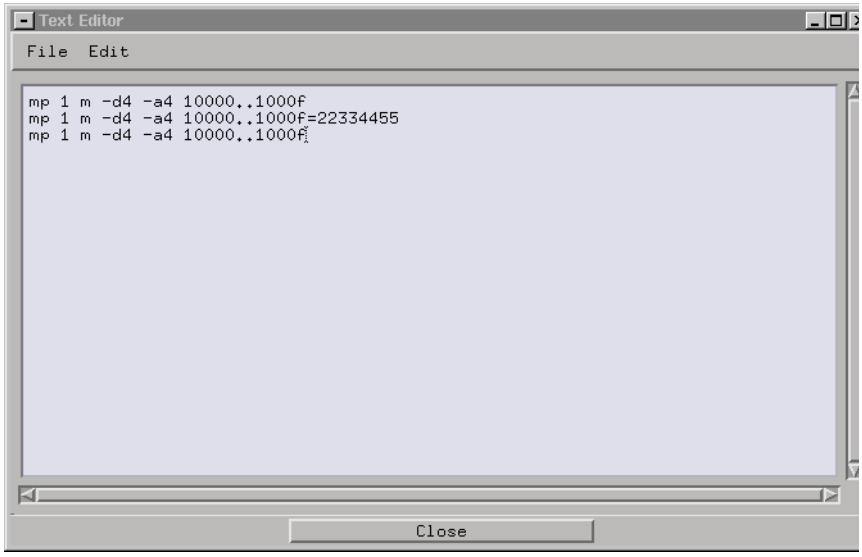
Use the up and down arrow keys to scroll through a list of the last 20 commands which you have entered.

Use the log (see page 54) command to save command line output to a file.

Example

To create a script that reads a range of memory addresses in processor 1, (see page 58) writes 22334455 to those addresses, and then reads the memory address range again:

1. Select *Edit Script*.
2. In the Text Editor window, enter the commands:

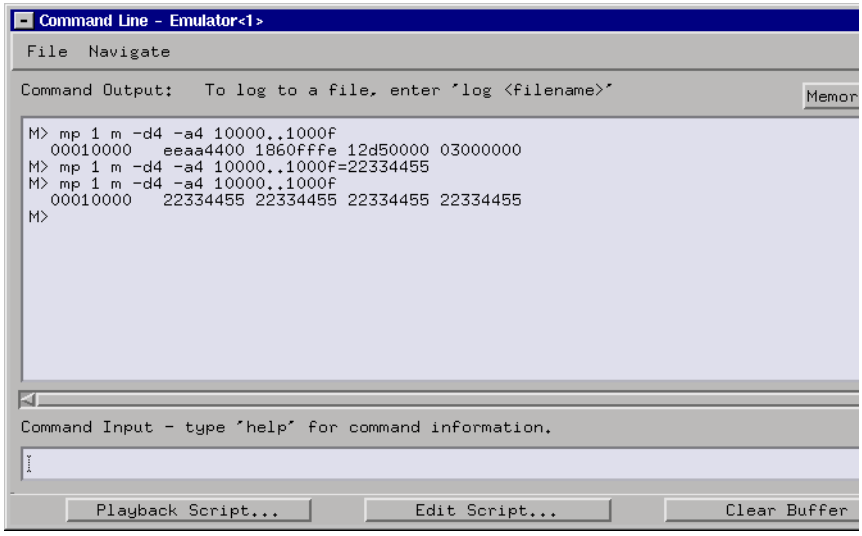


3. Select the *File->Save* command from the menu bar, enter the name of your file in the File Selection dialog, and select *OK*.
4. Close the Text Editor window.

To play back the script just created:

1. Select *Playback Script*.
2. Select the name of the emulator configuration settings script file in the File Selection dialog, and select *OK*.

   The output will look similar to the following:



Limitations

- Some of the commands listed when you enter `help` cannot be used from this window. In particular, the `pv` and `init` commands cannot be used.
- In *demo mode*, commands entered in this window (including the help command) have no effect.

See Also

- "To save command line output in a log file" on page 54
- "To create a script from a log file" on page 55
- "To cancel a command in the Command Line window" on page 55
- "Timouts in the Command Line window" on page 56
- "Commands not available in the Command Line window" on page 57
- "Addressing processors in a multiprocessor system" on page 58
- "Testing Target System Memory" on page 71

To save command line output in a log file

1. In the Command Line window, enter the `log filename` command.

   Enter the full path for the log file.

2. Enter commands or playback a script. The commands and their output will be saved to the file.
3. To stop logging to the file, enter the `log off` command.

Example

To save the values of all of the registers to the file "reglog01.log" enter:

```
log /logic/mylogs/reglog01.log
reg
log off
```

To create a script from a log file

1. Select *Edit Script*.
2. Select *File->Load Script*, then choose the log file.
3. Edit the text to remove:
   - the ">" prompt
   - the output of the command
   - the "Logging to" line
   - the "Logging turned off" line
4. Save the script.

To cancel a command in the Command Line window

Select the *Cancel* button in the Busy dialog to cancel the currently executing command.

Commands that create no output cannot be cancelled and will cause the connection to the emulator to timeout (see page 56).

If you use the **`rep`** command to write a loop, make sure that there is a command in the loop which will generate some output.

For example, **`rep 0 {m 0..100=0}`** will repetitively write 0 to the entire memory range 0..100. Because the **`m`** *address=value* command produces no output, the command cannot be cancelled. By changing the command to **`rep 0 {m 0..100=0;echo .}`** a dot is written

after every complete write of the range 0..100 and you will be able to cancel the command.

Timouts in the Command Line window

Timeout is one minute.

Every command must generate output that occurs at a rate of at least once a minute. Faster rates of output are desirable. For example, the command **`w 70`** (wait 70 seconds) will cause the system to timeout and should not be used. In addition, commands like **`m 0..10000=0`** which fill large segments of memory may cause the system to timeout. Use *Memory Fill* in the Memory Window to fill memory.

Derive JTAG scan chain by experimentation

The *gchain* command can be run using the command line interface. This command runs and releases the system under test, multiple times if necessary, in an attempt to derive the JTAG scan chain.

CAUTION: The *gchain* command sends multiple JTAG commands to the system under test. If your system responds to the pattern "0*10" badly (for example, interpreting this as *assert data on pins*), the gchain command can damage your system. Agilent does not accept any responsibility for use of the gchain command.

Display current descriptors

The *desc* command can be run using the command line interface.

The following is the structure of the *desc* command:

```
desc [-noinvalid] {-idt|-ldt|-gdt} {<selector>|-all}
```

Where:

noinvalid = from this point on, do not display invalid entries.

idt = Display the IDT.

ldt = Display the LDT.

gdt = Display the GDT.

all = Display all entries.

<selector> = Display this selector

Commands not available in the Command Line window

Some of the commands listed when you enter `help` in the Command Line window are not available.

The unavailable commands are:

- cf_var
- cfsave
- cl
- dump
- end
- init
- lan
- load
- mo
- po
- pv
- sio
- sioget
- sioput
- slist
- start
- stty
- ureg

Addressing processors in a multiprocessor system

Processors are addressed in the command-line interface using the "mp" (multiprocessor) command, followed by a number. The number identifies the processor(s) addressed by the command string that follows. The number is a simple hexadecimal representation. For example, in a multiprocessor system having four processors, address the processors using the following matrix:

Processor				Use Command
3	2	1	0	mp
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	a
1	0	1	1	b
1	1	0	0	c
1	1	0	1	d
1	1	1	0	e
1	1	1	1	f

Examples:

mp 1 r causes processor 0 to run.

mp 5 b causes processors 0 and 2 to break to the monitor.

mp 8 s causes processor 3 to step one instruction in the user program.

mp d r causes processors 3, 2, and 0 to run the user program.

Error/Status Messages

Select the messages below for additional information on some of the error/status messages that can occur when using the run control tool.

Some of these messages can appear in either the Error/Status Log window or in the Command Line Interface window; others will appear only in the Command Line Interface window as the result of a command entered there.

Read of register occurred (see page 65)

RPL of CS selector <DPL of segment descriptor (see page 65)

Segment limit violation (see page 65)

Segment not present (see page 66)

Segment selector exceeds GDT (or LDT) limit (see page 66)

Selector does not refer to GDT entry (see page 66)

Selector does not refer to a LDT (see page 66)

Selector points to invalid descriptor (see page 66)

Unable to break (see page 67)

See Also

"To use the Status/Error Log window" on page 51

Address must be for protected mode

This error occurs when running in protected mode if you specify an address, as part of a run or step command, that is not a protected-mode address.

Address must be for real mode

This error occurs when running in real mode if you specify an address, as part of a run or step command, that is not a real-mode address.

Break has been processed - running user program

If you have set the "When a Processor Resets" configuration option to "Break," this message appears after you enter the run command to override the break setting (usually after you enter the reset

command).

Break has been processed - running user program

If you have set the "When a Processor Resets" configuration option to "Break," this message appears after you enter the run command to override the break setting (usually after you enter the reset command).

Breakpoint dr.:.linear address …

This message shows which hardware breakpoint was hit.

Cannot translate protected-mode addresses in real mode

Protected mode addresses (selector::offset or GDT:LDT:offset) cannot be specified when the processor is running in real mode.

Cannot use NULL segment selector

A selector in a selector::offset address or a GDT:LDT:offset address cannot be 0.

INIT caused break - debug registers restored

This message appears when you have set the "When a Processor Inits" configuration option to "Break" and an INIT leading edge causes a break.

INIT caused break - debug registers restored

This message appears when you have set the "When a Processor Inits" configuration option to "Break" and an INIT leading edge causes a break.

IO address must be physical

This message appears if you enter an I/O address that is not in physical format. I/O addresses are limited to 64K physical.

IO address out of range

IO addresses are limited to a physical address between 0 and 64K.

I/O port access size not supported

This message occurs if -a8 (or something larger) is used as part of an io command in the Command Line Interface window.

Invalid option or operand: 8 (4, 2, or 1)

This error occurs if a command is sent to a processor that doesn't exist. Correct the JTAG chain, disconnect the Emulation Control Interface session, and then restart the Emulation Control Interface again.

In the command line window, the error could be caused by a "mp *processor_number* command" with an invalid processor number.

Jtag failed

This error occurs if a JTAG command was not executed correctly.

This probably indicates that there is not a good connection between the processor and the emulator, or that the processor needs to be reset.

Must be in monitor to access descriptor information

This error appears when a request (such as displaying memory at a selector::offset address) requires accessing information from a descriptor table while the target program is running. Break into the monitor and perform the action again.

No Target Power

This message occurs if the target power is disengaged during an attempt to submit a JTAG instruction.

PREQ# fell

This error occurs if the processor's PREQ# pin fell, indicating that it entered monitor, but the emulator was unable to find out why it entered monitor.

This could indicate that:

- A JTAG scan chain which has the right number of processors but the wrong PREQ# assignments was entered. For example, "p0,p1" is entered when the first processor is actually attached to preq1#, and the second is preq0#.

When this happens, the emulator can still program and read the processor, but the preq#/prdy# lines do not have the expected states. Therefore, when a processor enters the monitor, the processor detects that a different processor has entered monitor -- but when it is queried about why it entered monitor, returns the error message.

Solution: Ignore the message and use the 'gchain' command to find the real JTAG scan chain.

- The emultor was initialized, or the emulator firmware was programmed, while the emulator was attached to a live target system.

  Solution: Ignore the message and reconfigure the scan chain.

Page fault

This error occurs if a memory request generates an access attempt to a page of memory that is no longer available (that is, the page present bit of the page table entry is clear).

Processor is not ready

This message occurs if the PRDY signal is not asserted (according to the emulator) when it would ordinarily have to be.

Processor was in the halt state

This message indicates that the processor was halted, and that an operation pulled it out of the halted state.

Some actions that can end a halted state include:

- Setting a breakpoint after the halt instruction, then running.
- Changing a configuration option.
- Reading certain registers.

RESET deassertion caused break - debug registers restored

This message appears when you have set the "When a Processor Resets" configuration option to "Break" and a RESET deassertion causes a break.

RESET deassertion caused break - debug registers restored

This message appears when you have set the "When a Processor Resets" configuration option to "Break" and a RESET deassertion causes a break.

RPL of CS selector < DPL of segment descriptor

The RPL (Requestor Privilege Level) bits in the selector are greater than (of a lesser privilege than) the DPL bits specified in the segment descriptor.

Read of register occurred

This message indicates that something other than the Run Control tool (such as a debugger connected to the emulator) read a register.

Segment limit violation

The offset in either a segment:offset or GDT:LDT:offset address is out of range of the limit for the specified selector.

Segment not present

This error occurs if the specified segment descriptor is marked as not present, and is therefore not available.

Segment selector exceeds GDT (or LDT) limit

This error occurs if the specified selector is out of range for either the GDT or the specified LDT.

Selector does not refer to GDT entry

This error occurs if the entry in the GDT pointed to by the specified selector (in a selector::offset address or a GDT:LDT:offset address) could not be interpreted.

Selector does not refer to a LDT

This error occurs if the GDT:LDT:offset address specified was incorrect. The entry in the GDT pointed to by the specified selector could not be interpreted as a reference to an LDT.

Selector points to invalid descriptor

This error occurs if the entry in the GDT pointed to by the specified selector (in a selector::offset address or a GDT:LDT:offset address) could not be interpreted.

Unable to break

This message appears when the emulator does not detect a "running in monitor" condition when that condition is called for. This probably indicates that the processor needs to be reset.

To update firmware

Update the firmware if:

- The emulation module or emulation probe is being connected to a new analysis probe or *TIM*, or
- The emulation module was not shipped already installed in the logic analysis system, or
- You have an updated version of the firmware.

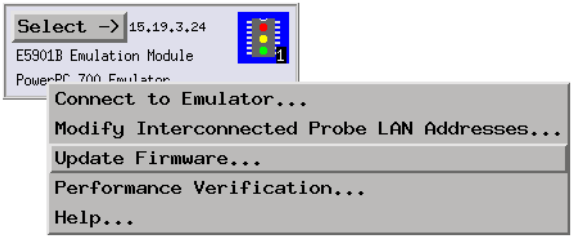
To install firmware from a CD-ROM to the hard disk

- Follow the instructions printed on the CD-ROM jacket.

  This will install the firmware onto the logic analysis system hard disk. Continue with either the "To update emulation module firmware" or the "To update emulation probe firmware" instructions.

To update emulation module firmware

1. End any emulation sessions that may be running. Remove any emulation module icons from the workspace in the Workspace window.
2. Install the firmware onto the logic analysis system's hard disk, if necessary.
3. In the system window, select the emulation module and select *Update Firmware*.



4. In the Update Firmware window, select the firmware version to load.

   Select the *Additional Information* button to display additional information about the firmware you selected.

5. Select *Update Firmware*.

To update emulation probe firmware

1. End any emulation sessions that may be running.
2. Install the firmware onto the logic analysis system's hard disk, if necessary.
3. In the workspace window, drag the emulation probe icon onto the workspace.
4. Select the emulation probe icon and select *Update Firmware...*.
5. In the Update Firmware window, enter the LAN name or address of the emulation probe.
6. In the Update Firmware window, select the firmware version to load.

   Select the *Additional Information* button to display additional information about the firmware you selected.
7. Select *Update Firmware*.
8. Cycle power on the emulation probe.

To display the current firmware version

- Select *Display Current Version* to see what firmware version is already installed in your emulation module or emulation probe.

To install firmware from another source

If you obtained firmware from another source, such as from an ftp server or a World Wide Web site: //www.agilent.com/find/sw-updates

- Follow the instructions provided with the firmware, OR
- Copy the firmware files into /logic/run_cntrl/firmware on the logic analysis system hard disk. Be sure to copy *all* of the files which begin with the product number of the firmware for your microprocessor.

Disconnecting from the Emulator

1. Select the Emulation Module icon or Emulation Control Interface icon and then select *Disconnect from Emulator*.
2. In the Connect - Emulator window, select *Disconnect from Emulator*.

Testing Target System Memory

Many times when a system under test fails to operate as expected, you will need to determine whether the failure is in the hardware or the software. These tests verify operation of the memory hardware in the system under test.

To Test Target System Memory

1. Open the Memory Test window (see page 86).
2. Specify the Memory Test (see page 73) to be performed.
3. Specify the Memory Range (see page 85) to be tested.
4. Specify the Data Value (see page 85) to be used when performing the test.
5. Specify the Options (see page 85), number of times the test is to be repeated and type of error message information to be presented during the test.
6. Open the Command Line window if it's not already open.
7. Select the *Execute Test* button.
8. View test results in the Command Line window.

The Command Input line in the Command Line window will show the equivalent terminal interface command during each test.

The Busy dialog box appears while the test executes. It also shows the equivalent terminal interface command during each test. A *Cancel* button in the Busy dialog box can be used to stop a test in progress.

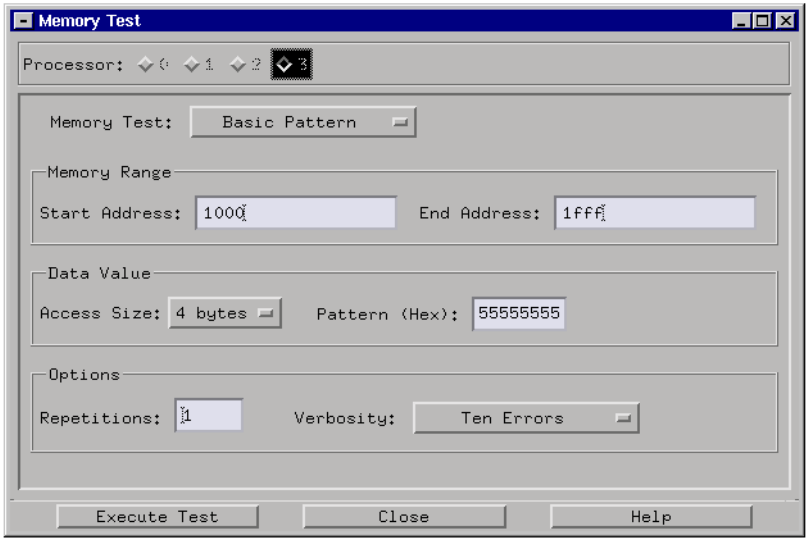
To check untested memory hardware, follow the “Recommended Test Procedure” on page 86.

Example

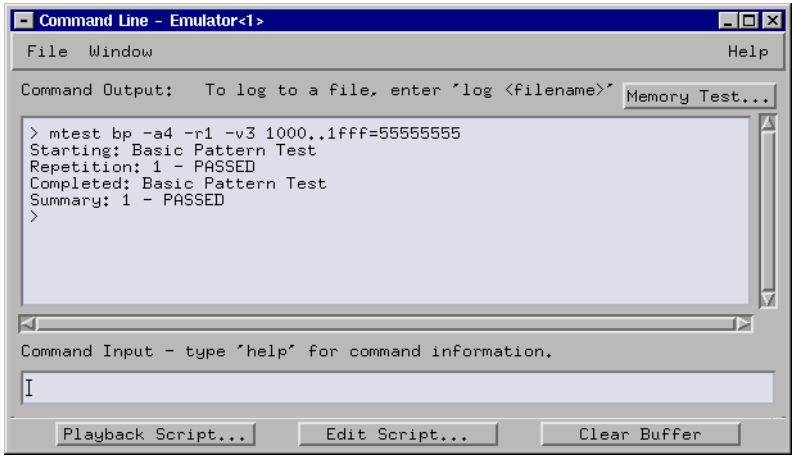
The following example writes the value "55555555" to addresses 1000 through 1fff (hexadecimal). Then it compares the values in memory with the values that were written. Next, the test writes the compliment "aaaaaaaa" to the same range of addresses and compares the new

values in memory with the values that were written.

1. Open the Command Line window and select the *Memory Test* button.
2. In the Memory Test window, specify the memory test shown below.



3. Select the *Execute Test* button.
4. View the test results in the Command Line window.



See Also

- "Recommended Test Procedure" on page 86

Memory Test:

The Memory Test feature of the Emulation Control Interface can perform seven different types of tests. Use these tests to find problems in address lines, data lines, and data storage. Use these tests in combination because no single test can perform a complete evaluation of the target system memory.

Basic Pattern

- Use this test to validate data read-write lines.
- "To perform the Basic Pattern test" on page 74
- "How the Basic Pattern test works" on page 75

Address Pattern

- Use this test to validate address read-write lines.
- "To perform the Address Pattern test" on page 76
- "How the Address Pattern test works" on page 77

Rotate Pattern

- Use this test to validate data read-write lines, and test voltage and ground bounce.
- "To perform the Rotate Pattern test" on page 77
- "How the Rotate Pattern test works" on page 79

Walking Ones

- Use this test to validate individual storage bits in memory.
- "To perform the Walking Ones test" on page 80
- "How the Walking Ones test works" on page 80

Walking Zeros

- Use this test to validate individual storage bits in memory.

Oscilloscope Read

- Use this test to generate the signals associated with reading from memory so they can be viewed on an oscilloscope.

Oscilloscope Write

- Use this test to generate the signals associated with writing to memory so they can be viewed on an oscilloscope.

To perform the Basic Pattern test

1. Open the *Memory Test* dialog box from either the Memory window or the Command Line window.
2. In the *Memory Test* dialog box, select the *Basic Pattern* Memory Test.
3. Type in the *Memory Range* to be tested.
4. Select the *Access Size* to be written, and type in the *Pattern* to be written.
5. Set *Repetitions* to 1. This causes the test to be performed one time.
6. Set *Verbosity* to *Ten Errors*. This is the recommended verbosity for the first test in a series of tests.
7. Select the *Execute Test* button.
8. When the test is complete, the Command Line window will show the test results. Error messages will be shown if any errors were found during the test.

This test verifies that the data lines of the selected memory range are without error. This test also checks the individual memory locations in

the memory range specified.

Consistent errors such as a particular bit incorrect every four bytes typically indicate a problem with the data lines. Random or sparse errors may indicate hardware data memory errors—check individual locations with the Walking Ones and Walking Zeros tests.

This test will halt and generate an error message if your Memory Range specification causes this test to be performed outside the range of valid memory in your target system.

This test will not halt but it will generate an error message if it is run on ROM or on locations with data line or location errors.

You can open a memory window on your logic analyzer to view the memory content. Expect to see the pattern and the compliment of the pattern that was specified.

NOTE: This test will not always detect errors in the address lines. For example, if a bit in the address lines is stuck high or low, the Pattern Test will write to a different location in memory. Then the read from memory for comparison will also be made from that different location so the data will be correct. Use the Address Pattern test with this test to completely evaluate the memory range.

See Also:

- "How the Basic Pattern test works" on page 75
- "If problems were found by the Basic Pattern test" on page 87

How the Basic Pattern test works

This test writes the *Pattern* and the compliment of the *Pattern* to the *Memory Range*, and then compares the values in memory with what was written. The compliment of the *Pattern* and then the *Pattern* are then written, read, and compared.

Example:

	First Write/Read	Second Write/Read
00000000	5555	AAAA
00000002	AAAA	5555
00000004	5555	AAAA

```
00000008        AAAA                5555
```

The Basic Pattern test finds data bits in memory that are stuck high or low. It also detects data lines that may be tied to power ground, or not connected at all.

To perform the Address Pattern test

1. Open the *Memory Test* dialog box from either the Memory window or the Command Line window.
2. In the *Memory Test* dialog box, select the *Address Pattern* Memory Test.
3. Type in the *Memory Range* to be tested.
4. Select the *Access Size* to be tested.
5. Set *Repetitions* to 1. This causes the test to be performed one time.
6. Set *Verbosity* to *Ten Errors*. This is the recommended verbosity for the first test in a series of tests.
7. Select the *Execute Test* button.
8. When the test is complete, the Command Line window will show the test results. Error messages will be shown if any errors were found during the test.

This test verifies that the address lines of the selected memory range are without error.

This test does not ensure that the data lines or individual data locations are without error. If a bit is stuck in a memory location, but is stuck in the written value, the stuck bit will not be detected.

You can view the memory in an analyzer memory window. You should see direct correlation between each address and the data stored at that address.

Consistent errors typically indicate problems in the address lines. This is especially likely if the results of the Basic Pattern test were without errors.

Errors in specific memory locations may indicate errors in the memory

hardware instead of the address lines.

See Also:

- "How the Address Pattern test works" on page 77
- "If problems were found by the Address Pattern test" on page 88

How the Address Pattern test works

This test writes the address of each memory location as data to each location. The data is then read back to see if it matches the addresses.

The pattern written to the memory is generated at the start of the test and is dependent upon the start address, access size, and the number of bytes in the memory range.

Depending on the last *Access Size* selected, subsets of the addresses may be written to memory. For example, if the last access size was 1 byte, address 00000001 will have 01 written to it, and address 00000002 will have 02 written to it.

For example, the data written in address 00001000 will look like this, depending on the last *Access Size*.

```
1 Byte = 00001000  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
2 Byte = 00001000  1000 1002 1004 1006 1008 100a 100c 100e
4 Byte = 00001000  00001000 00001004 00001008 0000100c
8 Byte = 00001000  0000000000001000 0000000000001008
```

The upper four bytes of an 8 Byte access size are not tested for a 4 Byte address. The upper four bytes will always be zeros. Use a smaller access size to test these locations with the Address Pattern test.

Unless the access size is 1 Byte, the odd bits of the memory locations will not be tested. Use the Basic Pattern test to check the odd bits.

To perform the Rotate Pattern test

1. Open the *Memory Test* dialog box from either the Memory window or the Command Line window.

2. In the *Memory Test* dialog box, select the *Rotate Pattern* Memory Test.
3. Type in the *Memory Range* to be tested.
4. Select the *Access Size* to be written, and type in the *Pattern* to be written. A good pattern to use is 01, 0001, or 00000001.
5. Set *Repetitions* to 1. This causes the test to be performed one time.
6. Set *Verbosity* to *Ten Errors*. This is the recommended verbosity for the first test in a series of tests.
7. Select the *Execute Test* button.
8. When the test is complete, the Command Line window will show the test results. Error messages will be shown if any errors were found during the test.

This test can be used to test voltage and ground bounce problems associated with the selected memory range. This test verifies that the data lines of the selected memory range are without error. This test also checks the individual memory locations in the memory range specified.

Consistent errors such as a particular bit incorrect every four bytes typically indicate a problem with the data lines. Random or sparse errors may indicate hardware data memory errors—check individual locations with the Walking Ones and Walking Zeros tests.

This test will halt and generate an error message if your Memory Range specification causes this test to be performed outside the range of valid memory in your target system.

This test will not halt but it will generate an error message if it is run on ROM or on locations with data line or location errors.

You can open a memory window on your logic analyzer to view the memory content. Expect to see the pattern and the compliment of the pattern that was specified.

See Also:

- "How the Rotate Pattern test works" on page 79

How the Rotate Pattern test works

This test writes the *Pattern* and the compliment of the *Pattern* to the *Memory Range*, and then compares the values in memory with what was written. Next, the rotated *Pattern* and the rotated compliment of the *Pattern* are written, read, and compared. Now the *Pattern* is rotated again, and again it is written, read, and compared. This continues until the rotations of the pattern return it to its original arrangement. That constitutes one *Repetition* of the Rotate Pattern test.

Example:

Address	First Write/Read	Second Write/Read	Third Write/Read
00000000	01	FE	02
00000001	FE	02	FD
00000002	02	FD	04
00000003	FD	04	FB
00000004	04	FB	08
00000005	FB	08	F7
00000006	08	F7	10
00000007	F7	10	EF
00000008	10	EF	20

Larger *Access Size* selections take more time because they require more patterns to be written to all locations (2-byte *Access Size* requires writing 32 patterns, and 4-byte *Access Size* requires writing 64 patterns).

The *Access Size* you select will affect the appearance of memory when you view memory after a test. When a test is complete, memory contains the last set of patterns that was written to it. For example, consider the following listing from a Rotate Pattern test that was performed one time with an *Access Size* of 2 bytes, and an initial pattern of 0001.

What you see below is the 32nd set of patterns written to memory during the test.

```
00000000  7fff 0001 fffe 0002 fffd 0004 fffb 0008
00000010  fff7 0010 ffef 0020 ffdf 0040 ffbf 0080
00000020  ff7f 0100 feff 0200 fdff 0400 fbff 0800
00000030  f7ff 1000 efff 2000 dfff 4000 bfff 8000
00000040  7fff 0001 fffe 0002 fffd 0004 fffb 0008
00000050  fff7 0010 ffef 0020 ffdf 0040 ffbf 0080
00000060  ff7f 0100 feff 0200 fdff 0400 fbff 0800
```

```
00000070  f7ff 1000 efff 2000 dfff 4000 bfff 8000
```

The Rotate Pattern test finds data bits in memory that are stuck high or low. It also detects data lines that may be tied to power ground, or not connected at all. This test more than any other in this set of tests will detect problems with voltage and ground bounce associated with the selected memory range.

To perform the Walking Ones test

1. Open the *Memory Test* dialog box from either the Memory window or the Command Line window.
2. In the *Memory Test* dialog box, select the *Walking Ones* Memory Test.
3. Type in the *Memory Range* to be tested.
4. Select the *Access Size* to be tested.
5. Set *Repetitions* to 1. This causes the test to be performed one time.
6. Set *Verbosity* to *Ten Errors*. This is the recommended verbosity for the first test in a series of tests.
7. Select the *Execute Test* button.
8. When the test is complete, the Command Line window will show the test results. Error messages will be shown if any errors were found during the test. You can also select the Memory window to see the content of memory.

The Walking Ones test finds data bits stuck in logical "0".

See Also:

- “How the Walking Ones test works” on page 80

How the Walking Ones test works

This test cycles "1" through each bit position in memory, and checks results. It does this by writing and then reading a pattern sequence of ones and zeros from all memory locations in the range.

For example, the hexadecimal values 01, 02, 04, ... are written to each

location in the *Memory Range*.

Address	1st	2nd	3rd	4th	5th	6th	7th	8th
00000000	01	02	04	08	10	20	40	80
00000001	02	04	08	10	20	40	80	01
00000002	04	08	10	20	40	80	01	02
00000003	08	10	20	40	80	01	02	04
00000004	10	20	40	80	01	02	04	08

NOTE: 1st, 2nd, 3rd, etc. are the first, second, third, etc. complete passes through the memory.

Larger *Access Size* selections take more time because they require more patterns to be written to all locations (2-byte *Access Size* requires writing 16 patterns, and 4-byte *Access Size* requires writing 32 patterns).

Example of 2-byte Access Size writing 16 patterns:

Address	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	...	16th
00000000	0001	0002	0004	0008	0010	0020	0040	0080	0100	...	8000
00000001	0002	0004	0008	0010	0020	0040	0080	0100	0200	...	0001
00000002	0004	0008	0010	0020	0040	0080	0100	0200	0400	...	0002
00000003	0008	0010	0020	0040	0080	0100	0200	0400	0800	...	0004
00000004	0010	0020	0040	0080	0100	0200	0400	0800	1000	...	0008

This test finds data bits stuck in logical "0".

To perform the Walking Zeros test

1. Open the *Memory Test* dialog box from either the Memory window or the Command Line window.
2. In the *Memory Test* dialog box, select the *Walking Zeros* Memory Test.
3. Type in the *Memory Range* to be tested.
4. Select the *Access Size* to be tested.
5. Set *Repetitions* to 1. This causes the test to be performed one time.
6. Set *Verbosity* to *Ten Errors*. This is the recommended verbosity for the first test in a series of tests.
7. Select the *Execute Test* button.
8. When the test is complete, the Command Line window will show the test

results. Error messages will be shown if any errors were found during the test. You can also select the Memory window to see the content of memory.

The Walking Zeros test finds data bits stuck in logical "1".

See Also:

- "How the Walking Zeros test works" on page 82

How the Walking Zeros test works

This test cycles "0" through each bit position in memory, and checks results.

For example, the hex values FE, FD, FB, ... are written to each location in the *Memory Range*.

Address	1st	2nd	3rd	4th	5th	6th	7th	8th
00000000	FE	FD	FB	F7	EF	DF	BF	7F
00000001	FD	FB	F7	EF	DF	BF	7F	FE
00000002	FB	F7	EF	DF	BF	7F	FE	FD
00000003	F7	EF	DF	BF	7F	FE	FD	FB
00000004	EF	DF	BF	7F	FE	FD	FB	F7

NOTE: 1st, 2nd, 3rd, etc. are the first, second, third complete pass through the memory.

Larger *Access Size* selections take more time because they require more patterns to be written to all locations (2-byte *Access Size* requires writing 16 patterns, and 4-byte *Access Size* requires writing 32 patterns).

Example of 2-byte Access Size writing 16 patterns:

Address	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	...	16th
00000000	FFFE	FFFD	FFFB	FFF7	FFEF	FFDF	FFBF	FF7F	FEFF	...	7FFF
00000001	FFFD	FFFB	FFF7	FFEF	FFDF	FFBF	FF7F	FEFF	FDFF	...	FFFE
00000002	FFFB	FFF7	FFEF	FFDF	FFBF	FF7F	FEFF	FDFF	FBFF	...	FFFD
00000003	FFF7	FFEF	FFDF	FFBF	FF7F	FEFF	FDFF	FBFF	F7FF	...	FFFB
00000004	FFEF	FFDF	FFBF	FF7F	FEFF	FDFF	FBFF	F7FF	EFFF	...	FFF7

This test finds data bits stuck in logical "1".

To perform the Oscilloscope Read test

1. Connect your oscilloscope to view signals on the lines to be tested. These

will be the signals generated to perform read transactions from the memory in your target system.

2. Open the *Memory Test* dialog box from either the Memory window or the Command Line window.
3. In the *Memory Test* dialog box, select the *Oscilloscope Read* Memory Test.
4. Type in the *Memory Range* to be read.
5. Select the *Access Size* to be read.
6. Set *Repetitions* to 0. This repeats the test continuously until you cancel the test.
7. Set *Verbosity* to *Summary Only*.
8. Select the *Execute Test* button.
9. When you have finished using your oscilloscope to view the read-from-memory signals, select the *Cancel* button in the *Busy* dialog box that appears on screen.

You will see an error message if your test attempts to read memory addresses outside the range of available memory.

See Also:

- "How the Oscilloscope Read test works" on page 83

How the Oscilloscope Read test works

This test repetitively reads the present content from the *Memory Range* for the number of *Repetitions* specified, typically reads continuously until cancelled.

The Oscilloscope Read test does not print or store the data it has read.

To perform the Oscilloscope Write test

1. Connect your oscilloscope to view signals on the lines to be tested. These will be the signals generated to perform write transactions to the memory

in your target system.

2. Open the *Memory Test* dialog box from either the Memory window or the Command Line window.
3. In the *Memory Test* dialog box, select the *Oscilloscope Write* Memory Test.
4. Type in the *Memory Range* to be written.
5. Select the *Access Size* to be written, and type in the *Pattern* to be written.
6. Set *Repetitions* to 0. This repeats the test continuously until you cancel the test.
7. Set *Verbosity* to *Summary Only*.
8. Select the *Execute Test* button.
9. When you have finished using your oscilloscope to view the write-to-memory signals, select the *Cancel* button in the *Busy* dialog box that appears on screen.

You will see an error message if your test attempts to write to memory addresses outside the range of available memory.

See Also:

- "How the Oscilloscope Write test works" on page 84

How the Oscilloscope Write test works

- This test repetitively writes your selected *Pattern* to the *Memory Range* for the number of *Repetitions* specified, typically continuously until cancelled.
- If your pattern is larger than the access size, it will be truncated to fit. If your pattern is smaller than the access size, it will be zero-padded to fit.
- This test does not generate error messages for unsuccessful write transactions, such as writes to ROM.
- If desired, you can open a memory window in the logic analyzer and view the memory where the pattern was written. If the memory is ROM or if it contains errors, it may not contain the pattern that was written.

Memory Range

A memory test can be performed in one range of memory addresses in your system under test.

1. Type the lowest address value of the range to be tested in the *Starting Address* text box.
2. Type the highest address value of the range in the *Ending Address* text box.

NOTE: If you wish to test only one location, enter the address in both the *Starting Address* and *Ending Address* text boxes.

Data Value

- *Access Size:* Specify the desired memory access size. If you are performing the Address Pattern test, the *Access Size* you choose will affect the value that is written to each memory location.

  If you are performing the Walking Ones, Walking Zeros, or Rotate Pattern test, larger access sizes will take more time because more patterns will be written to the memory locations.

- *Pattern:* Specify the pattern (in hexadecimal) to be used when performing the Oscilloscope Write, Basic Pattern, or Rotate Pattern test.

Options

- *Repetitions:* Enter the number of times you want the memory test to be repeated in the Repetitions text box. The maximum number of Repetitions is 10,000.

  If you enter *Repetitions: 0*, the test activity will run continuously until you select the *Cancel* button in the *Busy* dialog box. This is the normal selection when performing the Oscilloscope Read or Oscilloscope Write test.

- *Verbosity:* Select the type of error message presentation desired in the Verbosity selection box. The available options are:

  Summary Only

  Show no error messages during the tests. At the end of the last repetition, show a summary of the errors that were found during the tests.

  Repetition Summary

  Show no error messages during the tests. At the end of each repetition, show a summary of the errors that were found.

  Ten Errors

  Show error messages for the first ten errors found during each repetition. Then complete the repetition, but show no more error messages. At the end of the tests, show a summary of the errors that were found.

  All Errors

  Show a complete error message each time an error condition is found. At the end of the tests, show a summary of the errors that were found.

To open the Memory Test window

There are two ways to open the Memory Test window.

- Open the Command Line window and select the *Memory Test* button.
- Open the Memory window and select the *Memory Test* button.

Recommended Test Procedure

Two types of tests are offered for testing target memory: oscilloscope tests, and memory functionality tests.

Oscilloscope Tests

1. Connect the oscilloscope to view activity on the bits of interest.
2. Start an Oscilloscope Read (see page 82) or Oscilloscope Write (see page 83) test, as desired.

The test activity will be written onto the bits you specified continuously until you cancel the test.

Use both the Oscilloscope Read test and the Oscilloscope Write test to thoroughly check the connections of interest.

Memory Functionality Tests

1. Run the Basic Pattern (see page 74) test on the entire Memory Range.

   Result:

   - No Problems. Perform the Address Pattern (see page 76) test next.
   - Problems found. Refer to "If problems were found by the Basic Pattern test" on page 87.

2. Run the Address Pattern (see page 76) test on the entire Memory Range.

   Result:

   - No Problems.
   - Problems found. Refer to "If problems were found by the Address Pattern test" on page 88.

If no problems were found by the Basic Pattern test and the Address Pattern test above, you can ignore the rest of the tests. The memory in your system has been tested thoroughly and it is good.

If problems were found by the Basic Pattern test

Below are two examples of problems found by the Basic Pattern test:

- If there is a consistent error, such as:

```
Starting: Basic Pattern Test
Error: 1 at address 00000200:
  Read      5557  (0101 0101 0101 0111)
  Expected  5555  (0101 0101 0101 0101)
```

```
Error: 2 at address 00000204:
  Read      5557  (0101 0101 0101 0111)
  Expected  5555  (0101 0101 0101 0101)
Error: 3 at address 00000208:
  Read      5557  (0101 0101 0101 0111)
  Expected  5555  (0101 0101 0101 0101)
Error: ...
  Read      ...
  Expected  ...
```

Assume the data line bit associated with the error is stuck high. This could happen if the suspected data line bit were soldered to power.

NOTE: For an additional test of suspected memory, perform the Walking Ones (see page 80) and Walking Zeros (see page 81) tests on the problem memory range.

- If there are random errors, such as indicated by the following output:

```
Starting: Basic Pattern Test
Error: 1 at address 00000200:
  Read      8000  (1000 0000 0000 0000)
  Expected  0000  (0000 0000 0000 0000)
Error: 2 at address 000004a2:
  Read      efff  (1110 1111 1111 1111)
  Expected  ffff  (1111 1111 1111 1111)
Repetition: 1 - FAILED found 2 errors
Completed: Basic Pattern Test
Summary: 1 of 1 - FAILED (2 errors total)
```

From the above listing, we assume there are two location errors in memory. At location 200, there is a bit stuck high. At location 4a0, there is bit stuck low. Use the Walking Ones and Walking Zeros tests to verify the errors.

There is one bit stuck high at location 200 so the Walking Zeros test will print one error message when it tests this location. Use the Walking Ones test to isolate the bit that is stuck low at location 4a0. Again, this will print only one error message.

See Also:

- “If problems were found by the Address Pattern test” on page 88

If problems were found by the Address Pattern test

You may see no errors in the Basic Pattern test, but errors in the Address Pattern test. For example, you might see the following result in the Address Pattern test:

```
Error: 1 at address 00000000:
  Read      0020  (0000 0000 0010 0000)
  Expected  0000  (0000 0000 0000 0000)
Error: 2 at address 00000002:
  Read      0022  (0000 0000 0010 0010)
  Expected  0002  (0000 0000 0000 0010)
```

You would see that the data stored at locations 00 through 0f is the data that should be at locations 20 through 2f. This indicates an address line problem. Address bit 5 must be stuck low because the addresses that should have been written to the range 20 through 2f were written instead to 00 through 0f.

NOTE: Random errors typically do not indicate address line errors. Use the Walking Ones (see page 80) and Walking Zeros (see page 81) tests to check the locations of random errors.

See Also:

- "If problems were found by the Basic Pattern test" on page 87

absolute Denotes the time period or count of states between a captured state and the trigger state. An absolute count of -10 indicates the state was captured ten states before the trigger state was captured.

acquisition Denotes one complete cycle of data gathering by a measurement module. For example, if you are using an analyzer with 128K memory depth, one complete acquisition will capture and store 128K states in acquisition memory.

analysis probe A probe connected to a microprocessor or standard bus in the device under test. An analysis probe provides an interface between the signals of the microprocessor or standard bus and the inputs of the logic analyzer. Also called a *preprocessor*.

analyzer 1 In a logic analyzer with two *machines*, refers to the machine that is on by default. The default name is *Analyzer<N>*, where N is the slot letter.

analyzer 2 In a logic analyzer with two *machines*, refers to the machine that is off by default. The default name is *Analyzer<N2>*, where N is the slot letter.

arming An instrument tool must be armed before it can search for its trigger condition. Typically, instruments are armed immediately when *Run* or *Group Run* is selected. You can set up one instrument to arm another using the *Intermodule Window*. In these setups, the second instrument cannot search for its trigger condition until it receives the arming signal from the first instrument. In some analyzer instruments, you can set up one analyzer *machine* to arm the other analyzer machine in the *Trigger Window*.

asterisk (*) See *edge terms*, *glitch*, and *labels*.

bits Bits represent the physical logic analyzer channels. A bit is a *channel* that has or can be assigned to a *label*. A bit is also a position in a label.

card This refers to a single instrument intended for use in the Agilent Technologies 16700A/B-series mainframes. One card fills one slot in the mainframe. A module may comprise a single card or multiple cards cabled together.

channel The entire signal path from the probe tip, through the cable and module, up to the label grouping.

click When using a mouse as the

pointing device, to click an item, position the cursor over the item. Then quickly press and release the *left mouse button*.

clock channel A logic analyzer *channel* that can be used to carry the clock signal. When it is not needed for clock signals, it can be used as a *data channel*, except in the Agilent Technologies 16517A.

context record A context record is a small segment of analyzer memory that stores an event of interest along with the states that immediately preceded it and the states that immediately followed it.

context store If your analyzer can perform context store measurements, you will see a button labeled *Context Store* under the Trigger tab. Typical context store measurements are used to capture writes to a variable or calls to a subroutine, along with the activity preceding and following the events. A context store measurement divides analyzer memory into a series of context records. If you have a 64K analyzer memory and select a 16-state context, the analyzer memory is divided into 4K 16-state context records. If you have a 64K analyzer memory and select a 64-state context, the analyzer memory will be divided into 1K 64-state records.

count The count function records periods of time or numbers of state transactions between states stored in memory. You can set up the analyzer count function to count occurrences of a selected event during the trace, such as counting how many times a variable is read between each of the writes to the variable. The analyzer can also be set up to count elapsed time, such as counting the time spent executing within a particular function during a run of your target program.

cross triggering Using intermodule capabilities to have measurement modules trigger each other. For example, you can have an external instrument arm a logic analyzer, which subsequently triggers an oscilloscope when it finds the trigger state.

data channel A *channel* that carries data. Data channels cannot be used to clock logic analyzers.

data field A data field in the pattern generator is the data value associated with a single label within a particular data vector.

data set A data set is made up of all labels and data stored in memory of any single analyzer machine or

instrument tool. Multiple data sets can be displayed together when sourced into a single display tool. The Filter tool is used to pass on partial data sets to analysis or display tools.

debug mode See *monitor*.

delay The delay function sets the horizontal position of the waveform on the screen for the oscilloscope and timing analyzer. Delay time is measured from the trigger point in seconds or states.

demo mode An emulation control session which is not connected to a real target system. All windows can be viewed, but the data displayed is simulated. To start demo mode, select *Start User Session* from the Emulation Control Interface and enter the demo name in the *Processor Probe LAN Name* field. Select the *Help* button in the *Start User Session* window for details.

deskewing To cancel or nullify the effects of differences between two different internal delay paths for a signal. Deskewing is normally done by routing a single test signal to the inputs of two different modules, then adjusting the Intermodule Skew so that both modules recognize the signal at the same time.

device under test The system under test, which contains the circuitry you are probing. Also known as a *target system*.

don't care For *terms*, a "don't care" means that the state of the signal (high or low) is not relevant to the measurement. The analyzer ignores the state of this signal when determining whether a match occurs on an input label. "Don't care" signals are still sampled and their values can be displayed with the rest of the data. Don't cares are represented by the *X* character in numeric values and the dot (.) in timing edge specifications.

dot (.) See *edge terms*, *glitch*, *labels*, and *don't care*.

double-click When using a mouse as the pointing device, to double-click an item, position the cursor over the item, and then quickly press and release the *left mouse button* twice.

drag and drop Using a Mouse: Position the cursor over the item, and then press and hold the *left mouse button*. While holding the left mouse button down, move the mouse to drag the item to a new location. When the item is positioned where you want it, release the mouse button.

Using the Touchscreen:
Position your finger over the item, then press and hold finger to the screen. While holding the finger down, slide the finger along the screen dragging the item to a new location. When the item is positioned where you want it, release your finger.

edge mode In an oscilloscope, this is the trigger mode that causes a trigger based on a single channel edge, either rising or falling.

edge terms Logic analyzer trigger resources that allow detection of transitions on a signal. An edge term can be set to detect a rising edge, falling edge, or either edge. Some logic analyzers can also detect no edge or a *glitch* on an input signal. Edges are specified by selecting arrows. The dot (.) ignores the bit. The asterisk (*) specifies a glitch on the bit.

emulation module A module within the logic analysis system mainframe that provides an emulation connection to the debug port of a microprocessor. An E5901A emulation module is used with a target interface module (TIM) or an analysis probe. An E5901B emulation module is used with an E5900A emulation probe.

emulation probe The stand-alone equivalent of an *emulation module*. Most of the tasks which can be performed using an emulation module can also be performed using an emulation probe connected to your logic analysis system via a LAN.

emulator An *emulation module* or an *emulation probe*.

Ethernet address See *link-level address*.

events Events are the things you are looking for in your target system. In the logic analyzer interface, they take a single line. Examples of events are *Label1 = XX* and *Timer 1 > 400 ns*.

filter expression The filter expression is the logical *OR* combination of all of the filter terms. States in your data that match the filter expression can be filtered out or passed through the Pattern Filter.

filter term A variable that you define in order to specify which states to filter out or pass through. Filter terms are logically OR'ed together to create the filter expression.

Format The selections under the logic analyzer *Format* tab tell the

logic analyzer what data you want to collect, such as which channels represent buses (labels) and what logic threshold your signals use.

frame The Agilent Technologies or 16700A/B-series logic analysis system mainframe. See also *logic analysis system*.

gateway address An IP address entered in integer dot notation. The default gateway address is 0.0.0.0, which allows all connections on the local network or subnet. If connections are to be made across networks or subnets, this address must be set to the address of the gateway machine.

glitch A glitch occurs when two or more transitions cross the logic threshold between consecutive timing analyzer samples. You can specify glitch detection by choosing the asterisk (*) for *edge terms* under the timing analyzer Trigger tab.

grouped event A grouped event is a list of *events* that you have grouped, and optionally named. It can be reused in other trigger sequence levels. Only available in Agilent Technologies 16715A or higher logic analyzers.

held value A value that is held until the next sample. A held value can exist in multiple data sets.

immediate mode In an oscilloscope, the trigger mode that does not require a specific trigger condition such as an edge or a pattern. Use immediate mode when the oscilloscope is armed by another instrument.

interconnect cable Short name for *module/probe interconnect cable*.

intermodule bus The intermodule bus (IMB) is a bus in the frame that allows the measurement modules to communicate with each other. Using the IMB, you can set up one instrument to *arm* another. Data acquired by instruments using the IMB is time-correlated.

intermodule Intermodule is a term used when multiple instrument tools are connected together for the purpose of one instrument arming another. In such a configuration, an arming tree is developed and the group run function is designated to start all instrument tools. Multiple instrument configurations are done in the Intermodule window.

internet address Also called Internet Protocol address or IP address. A 32-bit network address. It

is usually represented as decimal numbers separated by periods; for example, 192.35.12.6. Ask your LAN administrator if you need an internet address.

labels Labels are used to group and identify logic analyzer channels. A label consists of a name and an associated bit or group of bits. Labels are created in the Format tab.

line numbers A line number (Line #s) is a special use of *symbols*. Line numbers represent lines in your source file, typically lines that have no unique symbols defined to represent them.

link-level address Also referred to as the Ethernet address, this is the unique address of the LAN interface. This value is set at the factory and cannot be changed. The link-level address of a particular piece of equipment is often printed on a label above the LAN connector. An example of a link-level address in hexadecimal: 0800090012AB.

local session A local session is when you run the logic analysis system using the local display connected to the product hardware.

logic analysis system The Agilent Technologies 16700A/B-series mainframes, and all tools designed to work with it. Usually used to mean the specific system and tools you are working with right now.

machine Some logic analyzers allow you to set up two measurements at the same time. Each measurement is handled by a different machine. This is represented in the Workspace window by two icons, differentiated by a *1* and a *2* in the upper right-hand corner of the icon. Logic analyzer resources such as pods and trigger terms cannot be shared by the machines.

markers Markers are the green and yellow lines in the display that are labeled *x*, *o*, *G1*, and *G2*. Use them to measure time intervals or sample intervals. Markers are assigned to patterns in order to find patterns or track sequences of states in the data. The x and o markers are local to the immediate display, while G1 and G2 are global between time correlated displays.

master card In a module, the master card controls the data acquisition or output. The logic analysis system references the module by the slot in which the master card is plugged. For example, a 5-card Agilent Technologies 16555D would be referred to as *Slot C:*

machine because the master card is in slot C of the mainframe. The other cards of the module are called *expansion cards*.

menu bar The menu bar is located at the top of all windows. Use it to select *File* operations, tool or system *Options*, and tool or system level *Help*.

message bar The message bar displays mouse button functions for the window area or field directly beneath the mouse cursor. Use the mouse and message bar together to prompt yourself to functions and shortcuts.

module/probe interconnect cable

The module/probe interconnect cable connects an E5901B emulation module to an E5900B emulation probe. It provides power and a serial connection. A LAN connection is also required to use the emulation probe.

module An instrument that uses a single timebase in its operation. Modules can have from one to five cards functioning as a single instrument. When a module has more than one card, system window will show the instrument icon in the slot of the *master card*.

monitor When using the Emulation Control Interface, running the monitor means the processor is in debug mode (that is, executing the debug exception) instead of executing the user program.

panning The action of moving the waveform along the timebase by varying the delay value in the Delay field. This action allows you to control the portion of acquisition memory that will be displayed on the screen.

pattern mode In an oscilloscope, the trigger mode that allows you to set the oscilloscope to trigger on a specified combination of input signal levels.

pattern terms Logic analyzer resources that represent single states to be found on labeled sets of bits; for example, an address on the address bus or a status on the status lines.

period (.) See *edge terms*, *glitch*, *labels*, and *don't care*.

pod pair A group of two pods containing 16 channels each, used to physically connect data and clock signals from the unit under test to the analyzer. Pods are assigned by pairs in the analyzer interface. The number of pod pairs available is determined

by the channel width of the instrument.

pod See *pod pair*

point To point to an item, move the mouse cursor over the item, or position your finger over the item.

preprocessor See *analysis probe*.

primary branch The primary branch is indicated in the *Trigger sequence step* dialog box as either the *Then find* or *Trigger on* selection. The destination of the primary branch is always the next state in the sequence, except for the Agilent Technologies 16517A. The primary branch has an optional occurrence count field that can be used to count a number of occurrences of the branch condition. See also *secondary branch*.

probe A device to connect the various instruments of the logic analysis system to the target system. There are many types of probes and the one you should use depends on the instrument and your data requirements. As a verb, "to probe" means to attach a probe to the target system.

processor probe See *emulation probe*.

range terms Logic analyzer resources that represent ranges of values to be found on labeled sets of bits. For example, range terms could identify a range of addresses to be found on the address bus or a range of data values to be found on the data bus. In the trigger sequence, range terms are considered to be true when any value within the range occurs.

relative Denotes time period or count of states between the current state and the previous state.

remote display A remote display is a display other than the one connected to the product hardware. Remote displays must be identified to the network through an address location.

remote session A remote session is when you run the logic analyzer using a display that is located away from the product hardware.

right-click When using a mouse for a pointing device, to right-click an item, position the cursor over the item, and then quickly press and release the *right mouse button*.

sample A data sample is a portion of a *data set*, sometimes just one point. When an instrument samples the target system, it is taking a single

measurement as part of its data acquisition cycle.

Sampling Use the selections under the logic analyzer Sampling tab to tell the logic analyzer how you want to make measurements, such as State vs. Timing.

secondary branch The secondary branch is indicated in the *Trigger sequence step* dialog box as the *Else on* selection. The destination of the secondary branch can be specified as any other active sequence state. See also *primary branch*.

session A session begins when you start a *local session* or *remote session* from the session manager, and ends when you select *Exit* from the main window. Exiting a session returns all tools to their initial configurations.

skew Skew is the difference in channel delays between measurement channels. Typically, skew between modules is caused by differences in designs of measurement channels, and differences in characteristics of the electronic components within those channels. You should adjust measurement modules to eliminate as much skew as possible so that it does not affect the accuracy of your measurements.

state measurement In a state measurement, the logic analyzer is clocked by a signal from the system under test. Each time the clock signal becomes valid, the analyzer samples data from the system under test. Since the analyzer is clocked by the system, state measurements are *synchronous* with the test system.

store qualification Store qualification is only available in a *state measurement*, not *timing measurements*. Store qualification allows you to specify the type of information (all samples, no samples, or selected states) to be stored in memory. Use store qualification to prevent memory from being filled with unwanted activity such as no-ops or wait-loops. To set up store qualification, use the *While storing* field in a logic analyzer trigger sequence dialog.

subnet mask A subnet mask blocks out part of an IP address so that the networking software can determine whether the destination host is on a local or remote network. It is usually represented as decimal numbers separated by periods; for example, 255.255.255.0. Ask your LAN administrator if you need a the subnet mask for your network.

symbols Symbols represent patterns and ranges of values found on labeled sets of bits. Two kinds of symbols are available:

- Object file symbols - Symbols from your source code, and symbols generated by your compiler. Object file symbols may represent global variables, functions, labels, and source line numbers.
- User-defined symbols - Symbols you create.

Symbols can be used as *pattern* and *range* terms for:

- Searches in the listing display.
- Triggering in logic analyzers and in the source correlation trigger setup.
- Qualifying data in the filter tool and system performance analysis tool set.

system administrator The system administrator is a person who manages your system, taking care of such tasks as adding peripheral devices, adding new users, and doing system backup. In general, the system administrator is the person you go to with questions about implementing your software.

target system The system under test, which contains the microprocessor you are probing.

terms Terms are variables that can be used in trigger sequences. A term can be a single value on a label or set of labels, any value within a range of values on a label or set of labels, or a glitch or edge transition on bits within a label or set of labels.

TIM A TIM (Target Interface Module) makes connections between the cable from the emulation module or emulation probe and the cable to the debug port on the system under test.

time-correlated Time correlated measurements are measurements involving more than one instrument in which all instruments have a common time or trigger reference.

timer terms Logic analyzer resources that are used to measure the time the trigger sequence remains within one sequence step, or a set of sequence steps. Timers can be used to detect when a condition lasts too long or not long enough. They can be used to measure pulse duration, or duration of a wait loop. A single timer term can be used to delay trigger until a period of time after detection of a significant event.

timing measurement In a timing measurement, the logic analyzer samples data at regular intervals according to a clock signal internal to the timing analyzer. Since the analyzer is clocked by a signal that is not related to the system under test, timing measurements capture traces of electrical activity over time. These measurements are *asynchronous* with the test system.

tool icon Tool icons that appear in the workspace are representations of the hardware and software tools selected from the toolbox. If they are placed directly over a current measurement, the tools automatically connect to that measurement. If they are placed on an open area of the main window, you must connect them to a measurement using the mouse.

toolbox The Toolbox is located on the left side of the main window. It is used to display the available hardware and software tools. As you add new tools to your system, their icons will appear in the Toolbox.

tools A tool is a stand-alone piece of functionality. A tool can be an instrument that acquires data, a display for viewing data, or a post-processing analysis helper. Tools are represented as icons in the main window of the interface.

trace See *acquisition*.

trigger sequence A trigger sequence is a sequence of events that you specify. The logic analyzer compares this sequence with the samples it is collecting to determine when to *trigger*.

trigger specification A trigger specification is a set of conditions that must be true before the instrument triggers.

trigger Trigger is an event that occurs immediately after the instrument recognizes a match between the incoming data and the trigger specification. Once trigger occurs, the instrument completes its *acquisition*, including any store qualification that may be specified.

workspace The workspace is the large area under the message bar and to the right of the toolbox. The workspace is where you place the different instrument, display, and analysis tools. Once in the workspace, the tool icons graphically represent a complete picture of the measurements.

zooming In the oscilloscope or timing analyzer, to expand and contract the waveform along the time base by varying the value in the s/Div

field. This action allows you to select specific portions of a particular waveform in acquisition memory that will be displayed on the screen. You can view any portion of the waveform record in acquisition memory.

Index

Index

Index

Publication Number: 5988-9079EN
January 1, 2003

Agilent Technologies