Help Volume

Remote Programming Interface (RPI) for the Agilent Technologies 16700-Series Logic Analysis System

Remote Programming Interface Programmer's Guide

This is a programmer's guide for the Remote Programming Interface (RPI) for the Agilent Technologies 16700-series logic analysis system. Its purpose is to give you the neccessary information to remotely control the logic analysis system through the execution of remote programs.

In addition to this programmer's guide, you should have a general knowledge of programming in the Basic or C programming language. You should also have a basic understanding of making measurements with a logic analyzer.

The RPI is available in two forms. While only the ASCII RPI is explained in this programmer's guide, information is available on the ActiveX/COM RPI when you upload the Agilent IntuiLink 16700 software components and access the Excel Add-in Toolbar help system.

See Also

Connectivity Help (see the *PC Connectivity* help volume)

Main System Help (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume)

Glossary (see page 77)

Contents

1

Setup and Configuration

In this chapter you will find information on setting up your remote computer and using the RPI procedural commands to remotely control the logic analysis system.

Remote Programming Interface RPI Overview

Agilent Technologies Remote Programming Interface (RPI) allows you to create custom programs to control your Agilentís 16600A and 16700A/B series logic analyzers. RPI is optimized for use in conjunction with Microsoft Win95/98/NT platforms or Unix platforms.

On the PC/Windows platform RPI takes advantage of Microsoft's Component Object Model and ActiveX automation technologies to allow you to write custom programs using Visual Basic, Visual C++, VBA or other COM compatible programming language.

Under Unix, RPI provides a procedural (or ASCII) based programming model. How you use RPI is dependent upon the development platform you have chosen to do your coding on.

NOTE: It is important that you reference the appropriate documentation describing either the PC/Windows/COM RPI or Unix ASCII RPI (depending on what development environment you have chosen to use).

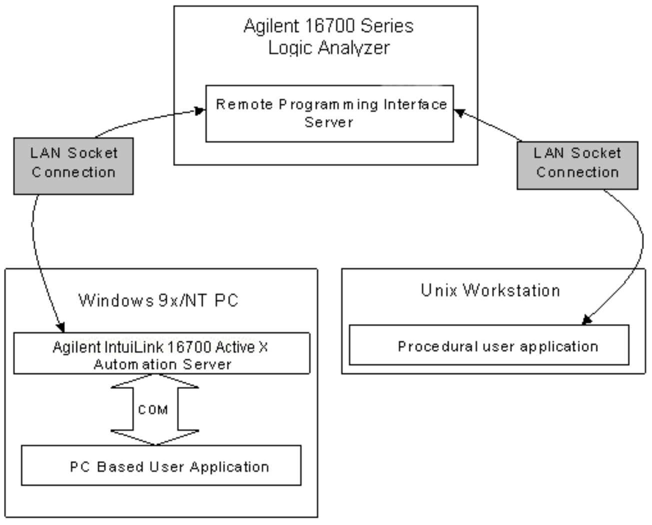
Agilent 16700 Series
Logic Analyzer
Remote Programming Interface Server
LAN Socket Connection
LAN Socket Connection
Windows 9x/NT PC
Agilent IntuiLink 16700 Active X Automation Server
COM
PC Based User Application
Unix Workstation
Procedural user application

Figure 1. Remote Programming Interface Architecture

RPI Architecture

Under Windows, the ActiveX Automation Server provides PC applications with a COM interface to the logic analyzer and uses RPI socket commands to communicate with the logic analyzer itself (see Figure 1). This allows you to write programs that communicate with the logic analyzer using a COM model definition thus taking advantage of the ease of programming offered by the Visual Studio Environment (that is, Visual Basic or Visual C++).

From Unix environments, RPI uses simple, ASCII text commands to communicate to the logic analyzer. This makes it easy to write shell scripts or HLL programs without the need to install any other third party software on your workstation.

RPI for UNIX

The Procedural RPI is a simple mechanism that allows a user on a remote host to open a TCP socket connection to an Agilent 16700A/B-series logic analyzer instrument. Through this connection, simple ASCII string commands are sent, ASCII responses from the instrument are received, and binary or ASCII trace data is transferred to the host running the RPI program.

Use Model

In order to create an easy to use, yet powerful remote control mechanism, the design of the RPI adheres to the basic use model of "load-run-store".

This means that when you want to create a remote control application or a program that runs repetitive tests, you simply go through each test once saving the logic analyzer configuration for each test you wish to repeat later. Then, from your program, you recall the appropriate logic analyzer configuration, run it, and store or act on the results as appropriate.

- “Create a Configuration File” on page 14
- “Load-Run-Store” on page 14

Create a Configuration File

Set up an instrument configuration for the desired measurement while sitting in front of the logic analyzer. Save this configuration to a file. This process allows you to use all the power of the instrument to setup your desired measurement.

Load-Run-Store

Once a configuration file is saved, write an RPI program that remotely loads this pre-saved file. Modify a few critical measurement parameters, run the analyzer until the measurement is complete, then store the results, or the raw trace data for post-processing on the host.

System Setup

The RPI language is easily used on any host platform. However, before you can run your RPI program, the logic analyzer must be set up on the LAN. This is done by getting the appropriate network information from your system administrator, and entering this information into the logic analyzer.

1. Click the System Administration icon on the system screen, then select the Networking tab.
2. Click Network Setup and enter the appropriate informantion.
3. Now select the Security tab and make sure the Remote Programming Interface is enabled.

NOTE: To increase security when the RPI is not being used, disable the RPI interface from this screen.

Learning and Debugging RPI Programs

Once setup on the LAN, you are ready to connect and start writing and debugging RPI programs. A simple way to learn how to program using RPI is to experiment with the RPI command language by opening a telnet connection from your remote computer to the logic analysis system specifying the special port address of "6500".

For example, type: "telnet my_logic_analyzer 6500" where "my_logic_analyzer" is its IP address or machine alias name, then press the Enter key.

This process opens a direct socket connection to the RPI in the logic analysis system. You know you have a connection when the "->" prompt appears on a blank command line. This command line prompt indicates that the RPI is ready to accept a command.

Exercise:

At the prompt type: "modules"

The RPI polls the instrument cardcage and reports a list of all HW modules currently in the frame.

At the prompt type: "lock".

The RPI puts a full screen message box on the instrument console to warn people that the instrument is currently in use via the RPI.

This telnet mechanism is also useful in helping to debug RPI programs under development. You can have a debug telnet connection open at the same time an RPI program is running.

Data Transfers

To provide both a fast and easy to use process for data transfer, an uncompressed binary format is used. One of the benefits of this format is that it's easy to decode and the software required to decode the binary is very simple.

It should be noted that since all data values are transferred in byte-aligned columns, there will be some generation of white space, especially when transferring large numbers of single-bit values.

Although data transfers from logic analyzers and scopes are in binary form, data transfers from the Listing tool will come in ASCII form. For the Listing tool, the ASCII form allows GUI control over the numeric formats used, as well as the use of powerful SW Analysis tools such as the Serial Analysis tool or the Filter tool.

Sample Programs

Source code for some sample RPI programs and an RPI utility library is shipped with your Agilent logic analyzer. They can be found in the directory "/logic/demo/rpi/".

You can transfer all files in this directory, including the makefile, onto your remote host using the various connectivity methods available from the logic analysis system. These include ftp, NFS, PC file sharing, or simply using the built-in floppy drive.

After the files are transferred, you can compile and run the programs to get familiar with the basic capabilities of the RPI.

RPI General Characteristics

The Remote Programming Interface (RPI) is available in two forms. While only information for the procedural (ASCII) user application is documented in this programmer's guide, the following general characteristics apply. For additional information on the Agilent IntuiLink ActiveX Automation Server, refer to the help system included with the Excel Add-in Toolbar.

- “Agilent IntuiLink ActiveX Automation Server” on page 19
- “Procedural (ASCII) User Application” on page 19

Agilent IntuiLink ActiveX Automation Server

The Agilent IntuiLink ActiveX Automation Server is based on the Microsoft Component Object Model (COM). The Agilent IntuiLink package needs to be installed on the PC host. This package can be downloaded from the instrument's web page.

The Agilent 16600A or 16700A/B series logic analysis system must be fully powered up before attempting to connect to the analyzer. A single user is allowed to connect to the logic analysis system server. If another user tries to connect when a user is already connected or before the logic analysis system is fully powered up, he or she will receive an error indicating the connection is refused.

The logic analysis system will continue to run after a remote programming session disconnects.

Procedural (ASCII) User Application

The Agilent 16600A or 16700A/B series logic analysis system must be fully powered up before attempting to connect to the analyzer.

The logic analysis system will continue to run after a remote programming session disconnects.

The procedural user application operates within the main thread of the logic analysis system application.

Programming Conventions

Each command is followed by its command options, all separated by a space. If any command option has argument types, they follow their option, all separated by a space. In the following example, the scope command has three options, "-n name", "-c", and "-meas". The -meas option includes two argument types in "period" and "risetime". The program code would look like the following:

```
scope -n Oscilloscope<B> -c 1 -meas period risetime
```

where:

`scope`

Is the base command.

`-n Oscilloscope<B>`

Is an option that names a scope module as the focus.

`-c 1`

Is an option to specify channel 1.

`-meas`

Is an option that initiates an automatic measurement query.

`period risetime`

Two automatic measurement argument types to return.

Return results (for Oscilloscope<B>, channel 1):

```
period: 9.9E37
risetime: 0.000000420800
```

Other Considerations

Commands, options, and argument types can be full lowercase, full uppercase, or capitalized first letter. All query returns are in lowercase.

System Commands

In this chapter you will find a description of remote control commands that act on the system components such as file operations, module identification, frame configuration, network connectivity and system run function control.

clear

This command clears the workspace of all modules and tools.

This command does NOT affect any system administration functions such as LAN settings, printer settings, etc.

Syntax:

```
clear
```

Options:

No options

Example:

```
clear
```

Clears the workspace of all modules and tools.

config

Use this command to load a previously saved instrument configuration file. This operation will restore the instrument to the same setup that was stored in the configuration file. It also allows the currently configured instrument to save itís current state to a new configuration file.

NOTE: Configuration files can be located on the local hard drive of the instrument OR, through the use of NFS mounting and PC sharing, can be located on any mountable UNIX or sharable PC disk drive.

When saving a configuration, if the file exists, an error message will result. However, using the -f argument will force an overwrite even if file(s) exist.

Syntax:

```
config [-l | -s [-f]] config_file
```

Options:

```
-l config_file
```

Loads a configuration file named "config_file".

```
-s [-f] config_file
```

Saves the current configuration and data to a file named "config_file".

Examples:

```
config -l pentium._E
```

Loads a configuration file named "pentium._E".

```
config -s myconfig
```

Saves the current workspace configuration with data to a file named "myconfig".

ctl_port

This command provides access to the instrument target control port. It will read and return the value present on the pins of the control port or set the port to a specific value. Values for the target control port can be set using the same syntax as analyzer -trig commands:

#hfx	Hex where upper 4 bits are high and lower 4 bits stay the same (don't care).
#b11110000	Binary where upper 4 bits are high and lower 4 bits are low.
#q377	Octal where all 8 bits are high.
#bxxxx1xxx	Set bit 4 high, leave all others the same.

Syntax:

```
ctl_port [? | value]
```

Options:

```
?
```

Reads the target control port and returns an 8-bit value.

```
value
```

Sets the target control port to an 8-bit value.

Returns:

<8 bit value>

Examples:

```
ctl_port ?
```

Reads the 8-bit value from the target control port.

Returns:
```
#he
```

```
ctl_port #hfx
```

Sets the target control port output pins: upper 4bits go to High, lower 4 bits stay what they were.

lock, unlock

This command coordinates access of the instrument with other users. When locked, a full screen message is displayed indicating that the instrument is currently in use by an RPI program. If desired, a custom message can be shown on the local display instead of a default message. As an example, a custom message might give information as to who has the unit locked. The instrument can then be unlocked when desired.

Syntax:

```
lock ["message text"], unlock
```

Options:

```
lock "message text"
```

Locks all users out of the instrument. If a custom message is sent, it must be contained in quotes.

```
unlock
```

Unlocks the instrument to allow other users.

Examples:

```
lock
```

Locks the instrument and displays a system default message.

```
unlock
```

Unlocks a currently locked instrument.

```
lock "Currently in use by Tom"
```

Locks the instrument and displays your custom message "Currently in use by Tom".

modules

Use this command to poll the system to identify the HW modules in the system, and return information on Type, Slot, and State. There are two states that modules can be in, "active" or "available". Available means that the HW module is plugged into a slot in the frame and is available to be included in a measurement. The second state is "active". In this state, the HW module is "activated" by being included in a measurement setup. When included in a measurement setup, the HW module is both visible in the instrument workspace and from the "Navigate" pulldown menu in the instrument GUI. Active modules have either the default or user-defined ASCII names associated with them.

Syntax:

```
modules [-a | -slot slot_id | -expanders]
```

Options:

`with no option`

Returns a list of both Active and Available modules. Type, Slot, and State information for each listed module is returned.

`-slot slot_id`

Returns information on a module in a specified "slot_id". The slot identifier is A-J for measurement modules and 1-4 for emulation modules.

`-a`

Returns a list of Active modules only. Type, Slot, and State information for each listed module is returned.

`-expanders`

Lists how many (and which) expander/slave cards each slot has.

Returns:

For each module listed, the following information is returned:

Type, Slot, State, "Name", "Model", and "Description"

The "Type", is a 2-character string representing a logic analyzer (LA), oscilloscope (SC), pattern generator (PG), and emulation (EM).

The "Slot", is the letter or number identifier of the slot (A-J for measurement modules, 1-4 for emulation modules). Most analyzers have 2 logical machines. The second machine is displayed as B2 for slot B, machine 2. The "State", is shown as either a "1" if the module is active, or "0" if inactive and available.

Also returned is the following HW module information: "Name", "Model", and "Description"

Example: LA B 1 "Analyzer<B>" "16550A" "100MHz State/500MHz Timing" Where: LA=logic analyzer, B=slot B, 1=active state, Name=Analyzer<B>, Model=16550A, and Description=100MHz State/500MHz Timing

Examples:

`modules`

In this case, the Logic Analyzer in slot B is active, as well as the Scope in slot E.

Returns:

```
LA B 1 "Analyzer<B>" "16550A" "100MHz State/500MHz
Timing"
LA B 0 "Analyzer<B2>" "16550A" "100MHz State/500MHz
Timing"
LA D 0 "Analyzer<D>" "16556A" "1M Sample 100 MHz
State/400 MHz Timing"
LA D 0 "Analyzer<D2>" "16556A" "1M Sample 100 MHz
State/400 MHz Timing"
SC E 1 "Scope<E>" "16534A" "2GSa/s Oscilloscope"
EM 1 0 "Emulator<1>" "Emulation Module" "Not
Configured"
```

`modules -a`

Query only the active modules. Note how only the two active modules from above are listed.

Returns:

```
LA B 1 "Analyzer<B>" "16550A" "100MHz State/500MHz
Timing"
SC E 1 "Scope<E>" "16534A" "2GSa/s Oscilloscope"
```

`modules -expanders`

Slot D is a master card, with 1 expander card in slot C: `Slot A: 0 expanders`

```
Slot B: 0 expanders
Slot D: 1 expanders
C
Slot E: 0 expanders
Slot 1: 0 expanders
```

session_mgr

This command accesses the logic analyzer session manager. A query (session_mgr ?) returns the current status of the logic analyzer as either "Running" if a measurement session is currently running, or "Stopped" if no measurement session is running.

If no session is running, one can be started with "session_mgr -start". If you try to start a session when one is currently running, an error is returned. You can stop a currently running session with "session_mgr -stop". If you try to stop a session when one is not running, an error is returned.

When a new session is started, it automatically is started in the mode last saved which will either be "exclusive" or "shared".

When either a new session is started, or a currently running session is stopped, your connection is automatically closed and you will have to re-connect to the logic analyzer to initiate subsequent commands.

Syntax:

```
session_mgr [ ? | -start | -stop ]
```

Options:

`?`

Returns current session status of either ìRunningî or ìStoppedî.

`-start`

Starts a new session.

`-stop`

Stops a currently running session.

Example:

```
session_mgr ?
Stopped
```

Queries for a system status, which returns "Stopped".

start

This command starts HW modules running. The definition of running is dependent on the HW module selected. For analyzer modules, "running" is when their trace analyzers begin looking for a trigger, when oscilloscopes begin looking for a trigger, when pattern generators begin generating vectors, and emulation probes start the processor running.

All active modules may be started at once by using no option, individual modules started with -n name or -slot slot_id, and all modules in a "group run" list can be started with the -g option.

The -rep option applies to analyzers, oscilloscopes, and pattern generator modules but does not apply to emulation probes. When used, it sets these modules to repetitive run mode.

Syntax:

```
start [-n name | -slot slot_id] [-g] [-rep]
```

Options:

`no option`

Starts all active modules running.

`-n name`

Starts the active module named "name" running.

`-slot slot_id`

Starts a specific module named "slot_id". The slot identifier is A-J for measurement modules and 1-4 for emulation modules.

`-g`

Starts all modules configured in the group run list running.

`-rep`

Starts LA, SC, and PG modules running in repetitive mode.

Examples:

`start`

Starts all active modules running.

`start -n Emulator<2>`

Starts the processor in the emulation probe module named "Emulator<2>" running.

```
start -g -rep
```

Starts all modules in the group run list running repetitively.

status

This command queries active modules and returns their measurement status. Status information returned depends on the module being queried. Analyzers and oscilliscopes can be stopped or running. Pattern generators can be stopped or running. Emulators can be running, reset, or in a break state. All active modules may be queried at once by using no option, individual modules with -n name or -slot slot_id, and all modules grouped in the "group run" list are queried with the -g option.

Remember an emulator is not a measurement module, so the state of the target processor on an emulator has no impact on the result of this command unless it is explicitly selected via the -n name.

Syntax:

```
status [-n name | -slot slot_id] [-g] [-v] [-text] [-clear]
```

Options:

`with no option`

Returns status of the frame. Returns either "running" or "stopped".

`-n name`

Returns the status of the active module named "name".

`-slot slot_id`

Returns the status of a specific module named "slot_id". The slot identifier is A-J for measurement modules and 1-4 for emulation modules.

`-g`

Returns the status of all modules in the group run list.

`-v`

Returns verbose status information instead of running/stopped.

`-text`

Retrieve the text messages from the Run Status display.

`-clear`

Clear the text messages in the Run Status display.

Examples:

```
status
```

Query if the frame is running anything.

Returns:
```
stopped
```

`status -v`

Query status for all active modules in the system.

Returns:
```
Analyzer<A>: stopped
Emulator<3>: MPC860 In Background
```

`status -n PatternGen<J>`

Query status for current module named "PatternGen<J>".

Returns:
```
running
```

`status -g -v`

Query status for all active modules in the group run list.

Returns:
```
Pentium: waiting for trigger
Analyzer<F>: waiting in sequence level 3
Emulator<3>: running
```

`status -text`

Show the text in the Run Status messages area.

Returns:
```
Analyzer<E>: Calibration Error
```

`status -clear`

Clear the messages area in the Run Status display

stop

This command stops HW modules that are actively running. The definition of running is dependent on the HW module selected. For analyzer modules, "running" is when their trace analyzers begin looking for a trigger, when oscilliscopes begin looking for a trigger, when pattern generators begin generating vectors, and emulation probes start the processor running.

All running HW modules may be stopped at once by using no option, individual modules may be stopped with -n name or -slot slot_id, and a selected list of modules grouped together in the "group run" list are stopped with the -g option.

The Stop command, using no option, will NOT stop the target processor connected to an emulation module. To do this you must select the emulation module with the -n name or -slot slot_id option.

Syntax:

```
stop [-n name | -slot slot_id] [-g]
```

Options:

`with no option`

Stops all actively running modules.

`-n name`

Stops the actively running HW module named "name".

`-slot slot_id`

Stops a specified module in the slot "slot_id". The slot identifier is A-J for measurement modules and 1-4 for emulation modules.

`-g`

Stops all running modules in the group run list.

Examples:

```
stop
```

Stops all actively running modules.

```
stop -n PatternGen<B>
```

Stops the actively running pattern generator named "PatternGen<B>".

```
stop -g
```

Stops all actively running modules in the group run list.

tools

This command queries the system and identifies the active SW tools. Tools that are "active" are currently included in a measurement setup and appear in the instrument workspace and from the "Navigate" pulldown menu in the instrument GUI.

Syntax:

```
tools
```

Options: No options.

Returns: Name: type (lister, compare, fileout)

Examples:

```
tools
```

Returns:

```
Filter<1>: Filter
Listing<1>: Listing
Compare<1>: Compare
Listing<2>: Listing
Waveform<1>: Waveform
Waveform<2>: Waveform
```

version

This command returns the version number for the product named by the option. If no option is used, the version number of the system software is returned.

Syntax: `version [product]`

Options:

`with no option`

Returns the SW version of the system.

`product`

Returns the SW version of the named product.

Returns: Version number for system or named SW package.

Examples:

`version`

Query version numbers of installed system SW packages.

Returns:
```
A.01.30.00
```

`version MCORE`

Query the SW version of the MCORE processor support package.

Returns:
```
A.01.31.00
```

`version PROC-SUPPORT`

Query the SW version of the PROC-SUPPORT bundle.

Returns:
```
A.01.30.00
```

wait

This command causes the remote programming interface to pause for a number of seconds, or until the current measurement completes. You can wait n seconds or until the measurement completes by using both a delay and the -complete option.

Without specifying a specific module, slot, or group to wait for, "wait -complete" will wait until the entire instrument is stopped. By specifying a specific slot, module, or tool name, or -g, you can wait until a single measurement completes.

CAUTION: With out a timeout value, if a measurement never completes, remote programs will hang.

Syntax:

```
wait [n] [-complete] [-n name | -slot slot_id] [-g]
```

Options:

`n`

Waits "n" seconds.

`-complete`

Waits until measurement is complete.

`-n name`

Waits until the named module stops.

`-slot slot_id`

Waits until module in the indicated "slot_id" completes.

`-g`

Waits until the group run group completes.

Examples

`wait 10`

Waits 10 seconds.

`wait -complete`

Waits until measurement is complete.

`wait 30 -complete`

Wait until the measurement is complete, but not longer than 30 seconds.

```
wait 120 -slot D -complete
```

Wait until slot D completes, but not longer than 2 minutes.

```
wait -n Analyzer<B> -complete
```

Wait until Analyzer<B> completes.

```
wait -g -complete
```

Wait until group run completes.

3

Hardware Module Commands

In this chapter you will find a description of remote control commands that act on the installed hardware modules.

analyzer

The analyzer command has three series of options available. Each series is defined as follows:

- "Options for Module Setup" on page 45 - This series of command options accesses the setup information for the active analyzer module.
- "Options for Data Query" on page 49 - This series of command options accesses the data captured by the active analyzer module.
- "Options for the Trigger Subsystem" on page 52 - This series of command options control the active analyzer's trigger subsystem.

Options for Module Setup

The following command options access the setup information of an active analyzer. The analyzer is made active by specifying its logical name, or by its slot ID. See the note below. This command series sets or returns information on various setup parameters for the specified analyzer module.

Syntax:

```
analyzer [-n name | -slot slot_id]
analyzer -mode [stnorm | stfast | tmfull | tmhalf | tmtrans | ? ]
analyzer -depth [min | max |depth in k-samples | ? ]
analyzer -assign [none | pod#, pod#, ... | ? ]
analyzer -label ?
analyzer -label name [{pos | neg} channels | ? ]
analyzer -label -d [name1, name2, ... | all ]
analyzer -label -f [label_file]
analyzer -threshold pod [? | value]
analyzer -threshall value
analyzer -sethold label [bitpos] [? | value]
```

NOTE: The -n name option is used to specify a specific analyzer module. If there is only one active module, the -n name option is not required. However, if there are multiple analyzer modules active, you must use the -n name at least once to specify a module focus, then again each time you want to change the focus to another analyzer module.

Options:

`-n name`

Sets the focus to the analyzer named "name".

-slot slot_id

Selects a specific analyzer located in "slot_id". The slot identifier is A-J for measurement modules and 1-4 for emulation modules.

-acqmode [stnorm | stfast | tmfull | tmhalf | tmtrans | ?]

Sets the acquisition mode. Option arguments are stnorn=state normal, stfast=turbostate, tmfull=timing full channel, tmhalf=timing half channel, and tmtrans=transitional timing.

-depth [min | max | depth | ?]

Sets acquisition depth. Option arguments are min=minimum, max=maximum, and depth=number of samples in thousands of states (example 8=8000 samples).

-assign [none | pod#, pod#, ... | ?]

Assigns pods. Pods are identified by a slot letter (A-J), and a pod number (1 or 2). Example, A1 or G2.

Note: All pods are assigned in pairs, so A1 will assign A1 AND A2 to the active analyzer. G2 will assign G2 AND G1. Pod letters are not case sensitive.

-label ?

Queries the label structure.

-label name [pos | neg {channels} | ?]

Assigns a label name, defines it as positive or negative, then assigns channels. This is a combination of a pod# and a bit assignment as in pod#[bits]. Bitsî is a comma separated list of channel numbers between 0 and 15, or a range like 15:0.

-label -d [name1, name2, ... | all]

Deletes specified label names (separated by commas), or "all" label names.

-label -f [label_file]

Loads a file of label assignments where label_file is the name of the file.

-threshold pod [? | value]

This command sets the logic threshold for a specific pod. The value is specified in floating-point volts. The pod index is an integer from 1 to the number of pods in the module. For example, if you have a two-card 16717 module (4 pods per card) in slot A with the slave card in slot B, pod index 5 equates to pod B1.

-threshall value

This command sets the logic threshold for ALL pods in the module.

-sethold label [bitpos] [? | value]

This command adjusts the setup/hold window (sampling position) for a specified "label". You can set the position for all bits in the label, or individually for each bit. The "value" is specified as the setup time in picoseconds. The hold time value is set automatically based on the setup time. Note that this option is only supported on 16715A and later logic analyzers.

Returns:

-assign ?

Returns all assigned pods.

Example:
```
A1,A2,C1,C2
```

-label ?

Returns information on all labels.

Example:
```
address,A1[15:0];A2[15:0]
data,C1[15:0]
read,C2[12]
write,C2[11]
control,C2[6,5,3:0]
```

Examples:

analyzer -n Analyzer<B> -mode ?

Sets Analyzer <B> as active analyzer, then queries the acquisition mode.

Returns:
```
Run ID: 1234567890
States: -4095..4096
Times: -1.0e-06..1.0e-06
5 labels
"ADDR" 32 bits unsigned integer
"DATA" 16 bits unsigned integer
"STAT" 5 bits unsigned integer
"Time" 64 bits signed integer timescale picoseconds
"State Number" N bits signed integer
```

analyzer -slot C -mode stfast

Sets Analyzer <C> as active analyzer, then sets acquisition mode to turbostate mode.

`analyzer -n Analyzer<C> -clock slave`

Sets Analyzer <C> as active analyzer, then sets state clock mode to slave.

`analyzer -depth ?`

Queries the acquisition depth of the active analyzer.

Returns:
`4000`

`analyzer -assign ?`

Queries which pods are assigned to the active analyzer.

Returns:
`A1,A2,B1,B2,C1,C2`

`analyzer -label address,îA1[15:0];A2[15:0]î,data,îC1[15:0]î`

Sets up the address and data labels in the active analyzer.

`analyzer -label -d address,data`

Deletes the labels address and data.

`analyzer -label -f myLabels.txt`

Loads all labels in myLabels.txt file into active analyzer.

`analyzer -threshold 2 0.8`

Set logic threshold for pod 2 to 0.8 V.

`analyzer -threshold 2 ?`

Query the logic threshold for pod 2.

`analyzer -sethold Label1 2 4000`

Set the setup time for bit 2 of "Label1" to 4000 ps.

`analyzer -sethold Label1 2500`

Set the setup time for all bits of "Label1" to 2500 ps.

`analyzer -sethold Label1 2 ?`

Query the setup time for bit 2 of "Label1".

Options for Data Query

This series of command options accesses the data captured by an active analyzer. The analyzer is set active by the -n name or -slot slot_id options.

These command options can also return information on the last data captured including data size and boundary ranges. You can then select which labels of data you are interested in and transfer all states or a partial range of data out the communication channel.

Syntax:

```
analyzer [-n name | -slot slot_id] [-i]
analyzer [-n name] -d [-l labellist | all] [-r start..end | all] [-t start..end |
all]
analyzer -eyefinder [run | load filename]
analyzer -sample [? | rate]
```

Options:

`-n name`

Sets the focus to the analyzer named "name".

`-slot slot_id`

Selects a specific analyzer located in "slot_id". The slot identifier is A-J for measurement modules and 1-4 for emulation modules.

`-i`

Queries for information on the last data captured.

```
-d [-l label1,label2...| all] [-r start..end | all]
[-t start...end | all]
```

Begin upload of binary data out of the analyzer. Use the -l option to list individual labels, -r to specify a range, and -t to specify a time period.

```
-d [-l label1,label2...| all] [-r start..end | all]
[-t start...end | all]
```

Begin upload of binary data out of the analyzer. Use the -l option to list individual labels, -r to specify a range, and -t to specify a time period.

`-eyefinder run`

Run eye finder setup.

`-eyefinder load filename`

Load eye finder results file.

`-sample [? | rate]`

In timing mode, this command sets the timing sample rate. The value "rate" is specified in floating-point seconds. For example, a 10 ns sample period is 10.0e-09. The value entered will be rounded down to the nearest supported sampling rate.

Returns: The -i information query structure returns the following:

NOTE: Transferring Transitional Timing Data. When capturing data in transitional timing mode, data is only stored when a transition occurs. Therefore, when accessing data captured by an active analyzer configured with transitional timing enabled, it is recommended that you transfer all states. Transferring a partial range of captured data may result in ambiguous data values until the first transition within that range is observed.

```
Run ID: 1234567890
States: -4095..4096
Times: -1.0e-06...1.0e-06
5 labels
"ADDR" 32 bits unsigned integer
"DATA" 16 bits unsigned integer
"STAT" 5 bits unsigned integer
"Time" 64 bits signed integer timescale picoseconds
"State Number" N bits signed integer
```

NOTE: To select which data is sent, the -d option must be accompanied by a range or time selection, and by a label selection.

A range selection looks like this:

```
-r start..end or -r all,
```

where start and end are integer state numbers. If the data has states from -4095..4096, there are 8K states. The trigger position is at state number 0.

The range can also be selected by time values, such as:

```
-t start..end or -t all
```

where start and end are floating-point values in units of seconds. The trigger location is always at time 0.0. So, to select from -1 microsecond to +1 microsecond:

```
-t -1.0e-06..1.0e-06
```

Finally, to select labels, the -l flag is used:

```
-l ADDR,DATA,STAT,Time or -l all
```

If a label contains white space, the label is enclosed in quotation marks:

```
-l "State Number","System Clock",ADDR
```

Once data is selected, a two-part binary data transfer occurs. First, a simple 8-byte header is sent, indicating how many states will be transferred, and how many bytes for each state will be sent. Then for each state, a row of bytes is sent containing the data for each of the selected labels as follows:

```
4 bytes - Number of records
4 bytes - Number of bytes per record
nrecords *bytes per record - Data
```

Each record contains one state or time of the data requested. For each label selected (-l option), there are an integer number of bytes containing the value. Labels are sorted in order by which they were requested, and if "all" is selected, they arrive in order by which they are listed in the -i query.

The number of bytes for each label is the lowest possible integer number of bytes given the bit width of the label. For example, a 17-bit label will require 3 bytes (24 bits), a 16-bit value will require 2 bytes.

Examples:

```
analyzer -n Analyzer<B> -i
```

Returns:

```
Run ID: 1234567890
States: -4095..4096
Times: -1.0e-06..1.0e-06
5 labels
"ADDR" 32 bits unsigned integer
"DATA" 16 bits unsigned integer
"STAT" 5 bits unsigned integer
"Time" 64 bits signed integer timescale picoseconds
"State Number" N bits signed integer
```

```
analyzer -slot C -d -l all -r all
<begin binary data transfer>
...
<end transfer>
```

Uploads data for all labels at all states.

```
analyzer -n Analyzer<C> -d -l all -t -0.001..0.001
<begin binary data transfer>
...
<end transfer>
```

Upload data for all labels, in the time range of -1 msec to +1 msec

```
analyzer -d -l addr,data -r -100..200
<begin binary data transfer>
...
<end transfer>
```

Upload specific data for labels "addr" and "data" in the range of -100 to 200 states.

```
analyzer -eyefinder load "/logic/eyefinder.dat"
```

Load previous eye finder results from file "/logic/eyefinder.dat".

```
analyzer -sample 20.0e09
```

Set timing sample rate to 20 ns.

Options for the Trigger Subsystem

This series of command options control the analyzer trigger subsystem. They allow control of the trigger position, occurrence counters on primary and secondary conditions, simple pattern matching with ANDed/ORed pairs, simple storage qualification, two level sequencing, simple durations and edge triggering.

The following options also allow you to recall up to 10 defined trigger setups from a recall buffer. This allows easy, fast switching of triggers between measurements.

Syntax:

```
analyzer [-n name] -trig condition [store condition2] followedby condition3
[store condition4]
analyzer [-n name] -trig condition1 [occurs X] [store condition2]
followedby condition3 [occurs Y] [store condition4]
analyzer [-n name] -trig position [percent | ? ]
```

Options:

```
-trig anything
```

Set to trigger on anything and store everything.

```
-trig recall n
```

Load a prestored trigger setup from the recall buffer "n".

```
-trig recall ìMacro Name
```

Recall stored trigger setup by its name.

```
-trig condition1 [store condition2]
```

Trace for a condition 1 with optional store.

```
-trig condition1 [store condition2] [followedby condition3 [store
condition4]]
```

Trace for condition 1 followed by a condition 2 (with optional store at each level).

```
-trig duration condition1 [< | >] time
```

Trace when you find a value occurring for the desired time.

```
-trig condition1 [occurs X]
```

Trace for condition 1 that occurs X times. See note below.

```
-trig condition1 [occurs X] [store condition2]
```

Trace for condition 1 that occurs X times with a conditional store. See note below.

```
-trig condition1 [occurs X] [store condition2] followedby condition3
[occurs Y]
```

Trace for condition 1 that occurs X times with a conditional store, followed by condition 2 that occurs Y times. See note below.

```
-trig condition1 [occurs X] [store condition2] followedby condition3
[occurs Y] [store condition4]
```

Trace for condition 1 that occurs X times with a conditional store, followed by condition 2 that occurs Y times with a conditional store. See note below.

```
-trig position [percent | ? ]
```

Controls the trigger position. The command uses an integer between 0 - 100 to represent the amount of data captured before trigger. To set trigger at start of trace, set percent to 100. To set trigger at end of trace, set percent to 0. For a trigger in the center, set percent to 50. See note below.

NOTE: Occurrence counts can NOT be used with duration triggers.

Conditions: A "condition" is a combination of Pattern, Range, and Edge definitions. Patterns and ranges are defined as hex, octal, or binary numbers with optional don't care digits. To specify the number base, a prefix is used:

```
#h: Hexadecimal
#q: Octal
#b: Binary
#e: Edge (see below)
```

The 'x' character denotes a don't care digit. So to define a simple pattern condition, we might use something like this:

```
ADDR=#hFFFFXXXX
```

The above is a pattern condition that will search for a state when the value of the ADDR label lies between 0xFFFF0000 and 0xFFFFFFFF.

To specify a range, two pattern specifiers are joined by a comma (,). For example, to specify the same condition above as a range:

```
ADDR=#hFFFF0000,#hFFFFFFFF
```

To search for an edge or a glitch, we use an "edge specifier", defined by "#e" followed by any combination of the following characters:

```
x don't care
r rising edge
f falling edge
t toggling edge
e either edge (same as toggling)
* glitch
g glitch (same as *)
```

Two conditions may be combined with an AND or an OR. For example:

```
ADDR=#hFFFF0000,#hFFFFFFFF and DATA=#exxxRxxxx
```

Would search for a rising edge in bit 5 of DATA while ADDR is within the range 0xFFFF0000 - 0xFFFFFFFF.

Condition examples:

Pattern and Range Examples:

`#hFFXX0022`

Hexadecimal number with 2 don't care digits (8 don't care bits)

`#q7777xxxx`

Octal number with 4 don't care digits (12 don't care bits)

`#b10110110xxxx0000`

Binary number with 4 don't care bits

`#hFF00,#hFFFF`

Range from 0xff00 to 0xffff

NOTE: Don't care digits are not allowed in ranges.

Edge Examples:

`#eXXXXRFEG`

Edge specifier with 4 don't care bits, then Rising, Falling, Either, and Glitch bits

Examples:

```
analyzer -n Analyzer<B> -trig addr=#h12e4c and ctl=#h00
```

Trigger when addr=0x12e4c and ctl=0.

```
analyzer -trig addr=#h12xx or addr=#h13xx store addr=#h1200,#h13ff
```

Setup trigger for default analyzer to start on addresses with don't cares and store everything in the range 12xx to 13xx of label named "addr".

```
analyzer -trig addr=#h210 followedby addr=#h344
```

Trigger on access to address 210 followed by access to address 344.

```
analyzer -trig recall=1
```

Loads trigger setup from the recall buffer 1.

```
analyzer -trig recall="Enter Main"
```

Recalls a trigger setup named "Enter Main".

```
analyzer -n MyTarget -trig duration status=#h22 > 30 ns
```

Trigger analyzer named "MyTarget" when label status has value 22 for more than 30 ns.

```
analyzer -trig duration rdwr!#h0 < 30 ns
```

Trigger when no rdwr is not 0 pattern is found for less than 30 ns.

```
analyzer -trig io=#exxxxxFxF and cycle=#h1
```

Trigger if bit 0 & 2 of label named "io" transition low while label cycle is at pattern binary 1.

```
analyzer -n mybus -trig ctl=5 occurs 3
```

Triggers on the third occurrence of label "ctl=5".

```
analyzer -trig addr_hi=0 and addr_lo=340 followedby ioreg=6 occurs 15
```

Triggers on the 15th occurrence of "ioreg=6" after finding "addr_hi=0" and "addr_lo=340".

```
analyzer -n mybus -trig position ?
100
```

Queries analyzer named "mybus" for its trigger position. It returns "100".

```
analyzer -trig position 33
```

Sets the trigger position of the active analyzer to 33 percent. Note: If an integer over 100 is set, the number will be set to 100. If a negitive number (below 0) is set, the number is set to 0.

scope

This command accesses the data captured by an active oscilloscope module. The scope is selected by name or slot id, and can be queried for information about data captured in the last run using the -i option, or data can be uploaded using the -d option. In addition to the entire data, data can also be uploaded from only channels of interest for a specific range of data.

A "channel" can be either a single digit channel number, as in 1,2,3, or 4, or the channel label name, such as "Ground" or "rd/wr". Default label names given to the channels are "Channel D1" where the "D" is actually the slot number of the card and the "1" is the scope channel number between 1 and 10 (if you have enough expansion cards).

The scope command has two series of options available:

- “Options to Access Data Capture” on page 56
- “Options to Access Trigger and Measurement Subsystems” on page 58

Options to Access Data Capture

Syntax:

```
scope [-n name | -slot slot_id] [-i] -d [-l channellist | all]
[-r range | all] [-t timerange | all] -c [channellist | all]
```

Options:

```
-n name
```

Selects the active scope module by name.

```
-slot slot_id
```

Selects a specific scope module by a slot_id. The slot identifier is A-J for measurement modules.

```
-i
```

Query information on last data captured.

```
-c [1,2,... | all]
```

Query names of available channels.

```
-d [-l ch1,ch2,... | all] [-r start..end | all] [-t start..end
| all]
```

Begins upload of binary data out of scope.

NOTE: The -n name option is used to specify a specific scope module. If there is only one active module, the -n name option is not required. However, if there are multiple scope modules active, you must use the -n name at least once to specify a module focus, then again each time you want to change the focus to another scope module.

Returns: -i information query structure returns the following:

```
Run ID: 374199271
States: -16383..16384
Times: -8.191740e-06.. 8.192260e-06
4 labels
"State Number" 32 bits signed integer
"Time" 64 bits signed integer timescale picoseconds
"Channel A1" 15 bits yincrement 2.5247e-04 (volts/bit) yorigin -1.6203e+00
"Channel A2" 15 bits yincrement 2.5247e-04 (volts/bit) yorigin -1.6203e+00
```

Analog data such as scope data is given in its unsigned integer format, and the -i information provides the scale factors needed to convert back to floating-point voltages. For "Channel E1" above, there are 15-bit integer values. To convert them to voltage, apply the following (where value is the 15-bit integer):

```
voltage = yorigin + yincrement*value
```

-c channel information query structure returns the following:

```
1: "Channel A1" 15 bits yincrement 2.5247e-04
(volts/bit) yoffset -1.6203e+00
2: "Channel A2" 15 bits yincrement 2.5247e-04
(volts/bit) yoffset -1.6203e+00
```

Examples:

```
scope -n Scope<E> -i
```

Query last data captured for scope named "Scope<E>".

Returns:

```
Run ID: 1250539440
States: -16383..16384
Times: -8.191659e-06..8.192341e-06
4 labels
"State Number" 32 bits signed integer
"Time" 64 bits signed integer timescale picoseconds
"Channel A1" 15 bits yincrement 2.5247e-04 (volts/bit)
yorigin -1.6203e+00
"Channel A2" 15 bits yincrement 2.5247e-04 (volts/bit)
```

```
yorigin -1.6203e+00
```

`scope -c`

Query available scope channels.

Returns:
```
1: "Channel A1" 15 bits yincrement 2.5247e-04 (volts/
bit) yoffset -1.6203e+00
2: "Channel A2" 15 bits yincrement 2.5247e-04 (volts/
bit) yoffset -1.6203e+00
```

`scope -n Scope<E> -d -c all -t all`

Upload all scope data.

Returns:
```
<begin binary data transfer>
...
<end transfer>
```

`scope -d 1"Ground" -r -100..200`

Upload specific data for channels "1" and "ground" in the range of -100 to 200 states.

Returns:
```
<begin binary data transfer>
...
<end transfer>
```

Options to Access Trigger and Measurement Subsystems

Syntax:

```
scope [-n name | -slot slot_id] [-c 1,2,...] [-l channel1,channel2,..]
-meas [-type | all ] [-range,... -tgmode]
```

Options:

These options to the scope command allow setting and querying of various measurement parameters and access to the automatic measurement results.

A "channel" can be either a single digit channel number, as in 1,2,3, or 4, or the channel label name, such as "Ground" or "rd/wr". Default label names given to the channels are "Channel D1" where the "D" is actually the slot number of the card and the "1" is the scope channel number between 1 and 10 (if you have enough expansion cards).

`-n name`

Selects the active scope named "name".

`-slot slot_id`

Selects a specific scope module by a slot_id. The slot identifier is A-J for measurement modules.

`-c channel number`

Selects the channel named channel number.

`-autoscale`

Autoscale the scope.

`-meas [type | all ]`

Query automatic measurement results. See "Automatic Measurement Types and Returned Value" below.

`-range [range | ?]`

Set or query channel range (vertical).

`-offset [offset | ?]`

Set or query channel offset.

`-trange [range | ?]`

Set or query display range (horizontal).

`-delay [delay | ?]`

Set or query display delay.

`-sweep [triggered | auto |?]`

Set or query triggered or auto sweep.

`-tglevel [N | ?]`

Set or query the channel trigger level.

`-tgsource [channel | ext | ?]`

Set or query the trigger source.

`-tgslope [rising | falling | ?]`

Set or query the trigger slope.

`-tgmode [edge | pattern | immediate | ?]`

Set or query the trigger mode.

Automatic Measurement Types and Returned Values

all	return structure with all measurement results.
falltime	.90% to 10% time of left-most falling edge. Falltime: 0.000000268200
risetime	10% to 90% time of leftmost rising edge. Risetime: 0.000000420800
frequency	Frequency: 9.9E37
preshoot	Preshoot: 0.000000000000
overshoot	Overshoot: 0.000000000000
period	Period: 9.9E37
pwidth	+Width: 9.9E37
nwidth	-Width: 0.000003408333
vamp	Vamp: 0.113105058670
vavg	Vavg: -0.058784030290
vbase	Vbase: -0.117573976517
vmax	Vmax: -0.004468917847
vmin	Vmin: -0.117573976517
vpp	Vpp: 0.113105058670
vtop	Vtop: -0.004468917847
vdcrms	Vdcrms: 0.060179378230
vacrms	Vacrms: 0.012887802882

To select which scope channel the measurement results come from, use the "-c channel" option as follows:

```
scope -c 1 -meas all
or
scope -1 ìChannel E2î -meas period
```

To query the current setting of any of the trigger options, use a "?" instead of a value. For example, to query the display time range:

```
scope -trange ?
```

To set the display range to 0.001 seconds (1 msec):

```
scope -trange 0.001
```

Examples:

```
scope -n Oscilliscope<B> -meas risetime
```

Query rise time of scope named "Oscilliscope<B>" -c 1.

Returns:

```
Risetime:0.004
```

```
scope -tgsource 3
```

Set trigger source to channel 3.

```
scope -delay ?
```

Query current timebase delay.

Returns:

```
0.00346
```

pattgen

This command provides access to the pattern generator module. It allows the user to to query or change the clock source, frequency, and delay. The internal clock can be run from 1 to 180 MHz (or 300 MHz in half channel mode). It also allows the user to load an ASCII stimulus file into the pattern generator module. The user can query a vector number for its value, or modify single vectors within a currently loaded stimulus file.

Syntax:

```
pattgen [-n name | -slot slot_id] -f vectorfile
pattgen [-n name] -v vector_num [label1=value1,label2=value2,...]
pattgen [-n name ] -clock [frequency | ext | ? ] -delay [delay | ? ]
```

Options:

`-n name`

Selects a pattern generator module. See the note below.

`-slot slot_id`

Selects a specific pattern generator module by a slot_id. The slot identifier is A-J for measurement modules.

`-f vectorfile`

Loads an ASCII stimulus file named "vectorfile" into the target module.

`-v vector_num [label1=value1,label2=value2,...]`

Queries single vectors, or modifies single vectors with new values for each specified label.

`-clock [frequency | ext | ? ]`

Sets clock source to external mode or sets internal clock frequency. Also queries for internal clock frequency.

`-delay [delay | ? ]`

Sets or queries for clock output delay. Delay is set with an integer between 1 and 14.

`-v -i vector_num [label1=value1, label2=value2, ...]`

Insert a new vector at a specific position.

`-v -d vector_num`

Delete a specific vector.

NOTE: The -n name option is used to specify a specific pattern generator module. If there is only one active module, the -n name option is not required. However, if there are multiple pattern generator modules active, you must use the -n name at least once to specify a module focus, then again each time you want to change the focus to another pattern generator module.

Returns: -v vector_num query information structure returns the following:

```
label1=value
label2=value
etc...
```

Examples:

```
pattgen -f mem_ctl
```

Loads vectors from the file named "mem_ctl".

```
pattgen -n Pattgen<B> -v 3
```

First sets the focus to the pattern generator module named "Pattgen<B>", then queries for the value of vector number 3.

Returns:

```
data=3
ctl=3
chip_sel=0
```

```
pattgen -v 3 chip_sel=1
```

Modify the value in vector 3 under label "chip_sel" to a value of 1.

```
pattgen -clock 35
```

Set to use internal clock at 35 MHz.

```
pattgen -clock ?
35
```

Queries for internal clock rate. Returns 35 MHz.

```
pattgen -clock ext
```

Sets clock source to external mode.

```
pattgen -delay 4
```

Sets clock output delay to setting number 4.

emulator

This command provides access to emulation probe HW modules. Processor control includes resetting the processor, breaking into the monitor, step, or starting the processor running (using the system "start" command or the -run flag). It can also download binary processor code into the target memory.

Syntax:

```
emulator [-n name | -slot slot_id] [-reset | -break | -run | -step]
```

Options:

`-n name`

Selects the emulator named "name". See the note below.

`-slot slot_id`

Selects the emulator in "slot_id". The slot identifier is 1-4 for emulation modules.

`-reset`

Resets the processor on the target system.

`-break`

Breaks the target systemís processor into the monitor.

`-run`

Runs the processor.

`-step`

Steps the processor.

NOTE: The -n name option is used to specify a specific emulation module. If there is only one active module, the -n name option is not required. However, if there are multiple emulation modules, you must use -n name at least once to specify an emulation module focus, then again each time you want to change the focus to another emulation module.

Examples:

```
emulator -n Emulator<1> -r
```

First sets the focus to the emulation module "Emulator<1>", then resets the processor on the target system.

```
emulator -break
```

Breaks the processor on the target system into the monitor.

```
emulator -run
```

Runs the processor on the target system.

```
emulator -step
```

Steps the processor on the target system.

4

Software Tool Commands

In this chapter you will find a description of remote control commands that act on the installed software tools.

- “compare” on page 71
- “fileout” on page 75

listing

This command accesses the data displayed by an active lister. The lister is accessed by it's logical name.

This command can return information on the last data captured including data size, labels, and boundary ranges. You can then select which labels of data you are interested in and transfer all states or a partial range of data out the communication channel.

Syntax:

```
listing [-n name] [-i] -d -l [labellist | all] -r [range | all]
```

Options:

`-n name`

Specifies a specific lister tool display by name.

`-i`

Query for information on the last data captured.

`-d -l [label1,label2,... | all]`

Begins upload of ASCII LBP data out of the lister for a list of specific labels, or all labels.

`-r [start..end | all]`

Specifies a range between start-state and end-state, or all states.

NOTE: The -n name option is used to specify a specific lister display. If there is only one lister display, the -n name option is not required. However, if there are multiple lister displays, you must use -n name at least once to specify a lister display focus, then again each time you want to change the focus to another lister display.

Returns: The -i query returns the following:

```
Run ID: 1799474489
States: -2032..2063
Times: -8.128000e-06..8.256000e-06
"State Number" 12 characters format Decimal
"Lab1" 4 characters format Hex
"Time" 11 characters format Absolute
```

NOTE: A maximum of 30,000 states can be transferred by this command.

Examples:

```
listing -n Lister<2> -i
```

Sets focus to Lister<2>, then queries for information on its last data captured.

Returned:

```
Run ID: 1799474489
States: -2032..2063
Times: -8.128000e-06..8.256000e-06
"State Number" 12 characters format Decimal
"Lab1" 8 characters format Hex
"Time" 11 characters format Absolute
```

`listing -n MEMBD -d -l all -r all`

Sets focus to Lister named MEMBD, then uploads data on all labels in all states.

Returns:

```
<begin ASCII transfer>
...
<end transfer>
```

`listing -d -l addr,data -r -100..200`

Uploads specific data for labels "addr" and "data" in the range of -100 to 200 states.

Returns:

```
<begin ASCII transfer>
...
<end transfer>
```

compare

This command accesses the SW compare tool. A compare tool that is active on the workspace automatically executes a compare against the reference buffer whenever an analyzer captures a new trace.

The -i option returns the number of differences found. If the number -1(-one) is returned, it means the compare has not been executed. The -l option returns a list of label pairs and their masks.

There are two ways to do a compare. One is to compare a dataset with a reference buffer, and another is to compare one dataset to another from another tool (perhaps FileIn from a simulation).

The more typical compare is against a reference. In this case, label pairs usually look like the following:

```
addr,addr_ref
```

Because it is possible to compare any two labels (for example, "ADDR,DATA"), it is possible to set a compare mask by selecting both pairs. For example, we have the following two label pairs:

```
ADDR,ADDR
and
ADDR,DATA
```

In order to set the mask on ADDR,DATA, we enter the following command and option:

```
compare -m ADDR,DATA=#hffff0000
```

If all label pairs are unique, masks can be set by their first label in the pair:

```
compare -m ADDR=#hffff0000
```

The comparison masks are values that are "ANDed" to the captured trace label before it is diffed with the reference buffer. Therefore, a "1" in a bit position means this bit is significant to compare and a "0" means this bit is a don't care.

The -d, -r, and -s options allow the user to control the depth of the compare, and then query the results of the last compare. The first use model would be to start the compare with the "-x" option, then give it an option to either stop after the "N" differences are found (or "N" matches if the compare was setup that way) or compare only a certain range of states. The user could then issue the "-s" option to query for the results of differences that were found.

NOTE: A maximum of 30000 differences can be reported for each invocation of the "-s" option.

A second typical use model would be to start the compare but stop it after the first difference was found using "compare -x -d 1". The user would then query the compare information to see how many differences were found using "compare -i". If "0" was returned, then no differences were found. If "1" was returned, the user could query the results to get the actual label values in the state that had differences using "compare -s". This would return something like "4,Ctl=3,Ctl_ref=04,Data=32,Data_ref=32".

A third typical use model would be to compare the states in ranges of 30000 and then cycle through the ranges to get the results of all the differences. In this case, the user would use "compare -x -r 0..29999", then "compare -i" to see how many differences were found. Again, use "compare -s" to unload all differences, then resume with a new compare in the next range of states using "compare -x -r 30000..59999".

Syntax:

```
compare [-n name] [-i] [-l] -m [label1=mask1,label2=mask2,...]
compare -x [-d {N | all }] [-r {start..end | all}]
compare -s
```

Options:

`-n name`

Sets focus to the active compare tool named "name".

`-i`

Query information on last comparison.

`-x`

Executes the compare.

`-l`

Lists current label pairs.

`-m [lab1=mask | lab1,lab12=mask]`

Query or set up label comparison masks.

`-x [-d {N | all }] [-r {start..end | all}]`

Executes the compare with the option to stop the compare after N matches. You can also have the compare only work in ranges of states.

```
-s
```

Queries for the results of the differences that were found.

NOTE: The -n name option is used to specify a specific compare tool. If there is only one compare tool, the -n name option is not required. However, if there are multiple compare tools, you must use -n name at least once to specify a compare tool focus, then again each time you want to change the focus to another compare tool.

Returns: The -i query returns the following:

```
67
```

The -s query returns the following:

```
state#,label1=value,label1_ref=value,label2=value,label2_ref=value, ...
```

The -l query structure returns the following:

```
label1,label1_ref (mask=0xff00)
label2,label2_ref
```

Examples:

```
compare -n DMA_Comp<1> -i
```

Sets focus to compare tool named "DMA_Comp<1>", then queries for status differences found.

Returns:

```
1
```

```
compare -m ctl=#hff00,-m data=#h00ff
```

Set up mask #hff00 on label ctl, and mask #h00ff on label data.

```
compare -m Lab1,Lab2=#hff00
```

Sets mask for a label pair using both primary and secondary labels.

```
compare -l
```

Lists the current label pairs and their masks. (If there are no masks, nothing is listed).

Returns:

```
Current label pairs:
Lab1, Lab1_ref (mask=0Xff00)
Lab1, Lab2_ref
```

```
compare -x
```

Re-execute compare.

```
compare -i
```

See if anything failed.

Returns:
```
0
```

```
compare -m data=#hff00 -x -i
```

Changes the mask for label data, executes a compare, and returns the number of differences.

Returns:
```
23
```

```
compare -x -d 5
```

Executes a compare until 5 differences are found.

```
compare -s
2,Lab=20,Lab1_ref=21
50,Lab1=0,Lab1_ref=1
1123,Lab1=30,Lab1_ref=31
```

Returns the results of the last 3 differences found.

```
compare -x -d 3 -r 0..1000
```

Executes a compare over states 0 through 1000, OR until 3 differences are found.

fileout

This command controls the saving of data from a fileout tool into a specified file, and exporting data.

Syntax:

```
fileout [-n name] [-f file] [-s]
fileout [-n name] -r [start..end | all]
```

Options:

`-n name`

Sets focus to a specific fileout tool named "name".

NOTE: The -n name option is used to specify a specific fileout tool. If there is only one fileout tool, the -n name option is not required. However, if there are multiple fileout tools, you must use -n name at least once to specify a fileout tool focus, then again each time you want to change the focus to another fileout tool.

`-f file`

Defines a filename named "file" to save to.

`-s`

Save data to file previously specified.

`-r [start..end | all]`

Select a range of states to export with fileout. Note: Ranging only works for the Fast Binary output file format.

NOTE: When a partial range is selected, the range is automatically expanded (by moving the starting state backward and the ending state forward) to meet internal 1024-sample block boundaries, and the output file will contain more samples than those specified.

Examples:

`fileout -n Fileout<1> -f pentium.out`

Sets focus to the fileout tool named "Fileout<1>", then defines the save filename to "pentium.out".

`fileout -s`

Save data to whatever file was defined with -f option. In this example, it was "pentium.out".

`fileout -r [0..1000]`

Exports first 1000 states in Fast Binary output file format.

Glossary

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

Publication Number: 5988-9067EN
January 1, 2003

Agilent Technologies