Programmer's Guide

Version 10-23-00

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

In This Book

This book is a programmer's guide for the **Remote Programming Interface (RPI)** for the Agilent 16700 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 book, you should have a general knowledge of programming in the Basic or C programming language. You should also have a basic understanding of making measurement with a logic analyzer.

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

In Chapter 1, you will find information on the setup of the logic analysis system and your remote compute.

Chapter 2 is a command reference for the sytem level commands.

Chapter 3 is a command reference for all the hardware modules.

Chapter 4 is a command reference for all the software tools.

Contents

1

Setup and Configuration

In the following 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

The Agilent Remote Programming Interface (RPI) allows you to create custom programs to control your Agilent 16600 or 16700 series logic analyzer. 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).*

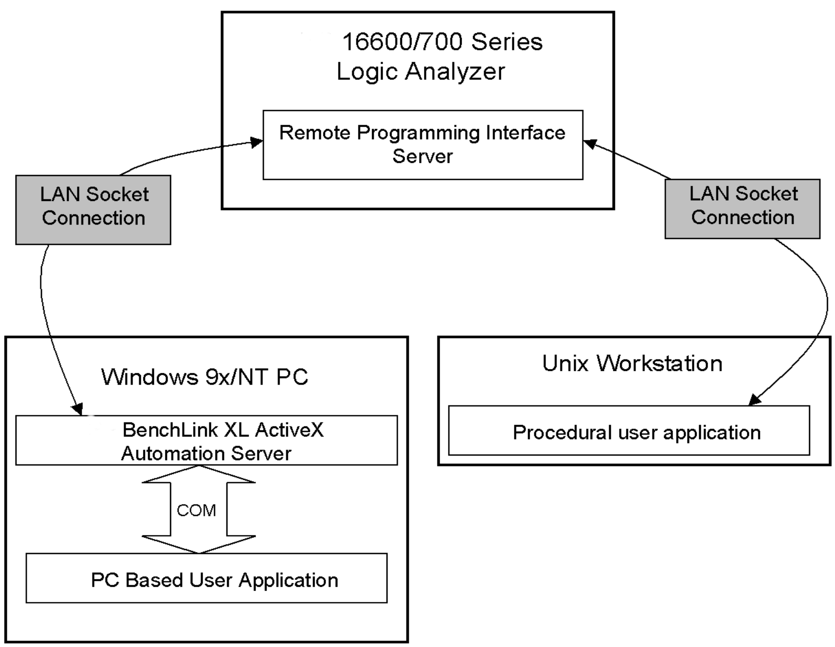

Figure 1. Remote Programming Interface Architecture

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 (i.e. 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 16700 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. 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.

For More Information. You can print the complete list of RPI commands by going into the Connectivity topic of the online help system and printing out the entire volume.

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.

Remote Programming Interface 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 book, the following general characteristics apply. For additional information on the Agilent Benchlink XL ActiveX Automation Server, refer to the help system included with the Excel Add-in Toolbar.

Agilent Benchlink XL ActiveX Automation Server. The Agilent Benchlink XL ActiveX Automation Server is based on the Microsoft Component Object Model (COM). The Agilent Benchlink XL 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 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 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 Oscilliscope<B> -c 1 -meas period risetime

where:

scope = is the base command,
-*n Oscilliscope<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 *Oscilliscope<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.

2

System Commands

In the following 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

Description: 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

Description: 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 it's 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

Description:

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

Description: 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

Description: 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

start

Description: 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 [-*g* | -*n name* | -*slot slot_id*] [-*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

Description:

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

Options:

with no option	Returns status of the frame. Returns either "*running*" or "*stopped*".
-v	Returns verbose status information instead of running/ stopped.
-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.
-n name	Returns the status of the active module named "*name*".
-g	Returns the status of all modules in the group run list.
-text	Retrieve the text messages from the Run Status display.
-clear	Clear the text messages in the Run Status display.

Returns:

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

Description:

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 [-*g* | -*n name* | -*slot slot_id*]

Options:

with no option	Stops all actively running modules.
-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.
-n name	Stops the actively running HW module named “*name*”.
-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

Description: 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

Description: 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

Description:

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.

> *WARNING:*
> *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.
-g	Waits until the group run group completes.
-slot slot_id	Waits until module in the indicated “*slot_id*” 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 the following chapter you will find a description of remote control commands that act on the installed hardware modules.

analyzer

Description:

This command accesses the data captured by an active analyzer. The analyzer is accessed by its logical name or slot id.

This command 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*] [-*i*]
analyzer [-*n name*] *-d* [-*l labellist* | *all*] [-*r start..end* | *all*] [-*t start..end* | *all*]

Options for Data Query

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.

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.

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.

Options for the Trigger Subsystem

The following options control the analyzer trigger subsystem. It allows 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*]

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 “Macro 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.

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.

scope

Description:

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).

Syntax:

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

Options to Access Data Capture

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 Query names of available channels.

-d [-l ch1,ch2,... | all] [-c 1,2,... | 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*] [-c 1,2,...] [-1 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"*.
-c channel number	Selects the channel named channel number
-autoscale	Autoscale the scope.
-c	
-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 “Channel E2” -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

Description: This command provides access to the pattern generator module. It allows the user to load an ASCII stimulus file into the pattern generator module. The user can also query a vector number for its value, or modify single vectors within a currently loaded stimulus file.

Syntax:

pattgen [-n name] -f vectorfile
pattgen [-n name] -v vector_num label1=value1,label2=value2,...

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.

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.

emulator

Description: 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 system's 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 the following chapter you will find a description of remote control commands that act on the installed software tools.

listing

Description:

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

Description:

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 (ie, "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. Therefor, a "1" in a bit position means this bit is significant to compare and a "0" means this bit is a don't care.

Syntax: compare [-*n name*] -*i* -*l* -*m* [*label1=mask1,label2=mask2,...*]

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,labl2=mask]*	Query or set up label comparison masks.

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 -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

fileout

Description: This command controls the saving of data from a fileout tool into a specified file.

Syntax: fileout [-*n name*] [-*f file*] [-*s*]

Options:

-n name	Sets focus to a specific fileout tool named *"name"*.
-f *file*	Defines a filename named *"file"* to save to.
-s	Save data to file previously specified.

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.

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*".

Index