Programmer's Guide

Publication Number 54620-97012
October 1995 (pdf version Dec 1998)

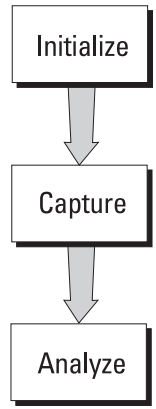
For Safety information, Warranties, and Regulatory information, see the pages behind the Index.

HP 54620A/C Logic Analyzer

Programming the Logic Analyzer

When you attach an interface module to the rear of the HP 54620A or HP 54620C (color) Logic Analyzer, the analyzer becomes programmable. That is, you can hook a controller (such as a PC or workstation) to the analyzer, and write programs on that controller to automate analyzer setup and data capture. Both HP-IB (also known as GP-IB, IEEE-488) and RS-232-C interfaces are available.

The following figure shows the basic structure of every program you will write for the analyzer.



Initialize

To ensure consistent, repeatable performance, you need to start the program, controller, and analyzer in a known state. Without correct initialization, your program may run correctly in one instance and not in another. This might be due to changes made in configuration by previous program runs or from the front panel of the logic analyzer.

- Program initialization defines and initializes variables, allocates memory, or tests system configuration.
- Controller initialization ensures that the interface to the analyzer (either HP-IB or RS-232) is properly setup and ready for data transfer.
- Analyzer initialization sets the channel configuration and labels, threshold voltages, trigger specification and mode, timebase, and acquisition type.

Capture

Once you initialize the analyzer, you can begin capturing data for analysis. Remember that while the analyzer is responding to commands from the controller, it is not performing acquisitions. Also, when you change the analyzer configuration, any data already captured is most likely invalid.

To collect data, you use the :DIGitize command. This command clears the waveform buffers and starts the acquisition process. Acquisition continues until acquisition memory is full, then stops. The acquired data is displayed by the analyzer, and the captured data can be measured, stored in trace memory in the analyzer, or transferred to the controller for further analysis. Any additional commands sent while :DIGitize is working are buffered until :DIGitize is complete.

You could also start the analyzer running, then use a wait loop in your program to ensure that the analyzer has completed at least one acquisition before you make a measurement. HP does not recommend this because the needed length of the wait loop may vary, causing your program to fail. :DIGitize, on the other hand, ensures that data capture is complete. Also, :DIGitize, when complete, stops the acquisition process so that all measurements are on displayed data, not on a constantly changing data set.

Analyze

After the analyzer has completed an acquisition, you can find out more about the data, either by using the analyzer measurements or by transferring the data to the controller for manipulation by your program. Built-in measurements include frequency, duty cycle, period, positive and negative pulse width, channel-to-channel delay, and setup and hold time.

Using the :WAVeform commands, you can transfer the data to your controller. You may want to display the data, compare it to a known good measurement, or simply check logic patterns at various time intervals in the acquisition.

In This Book

The *HP 54620A/C Programmer's Guide* is your introduction to programming the HP 54620A or HP 54620C (color) Logic Analyzer using an instrument controller. This book, with the *HP 54620A/C Programmer's Reference*, provides a comprehensive description of the analyzer's programmatic interface. The *Programmer's Reference* is supplied as a Microsoft Windows Help file on a 3.5" diskette.

To program the HP 54620A/C, you need an interface module, such as the HP 54650A, 54651A, or 54652B. You also need an instrument controller that supports either the IEEE-488 or RS-232-C interface standards, and a programming language capable of communicating with these interfaces.

This book contains the following information:

Chapter 1, "Introduction to Programming," gives a general overview of logic analyzer programming.

Chapter 2, "Programming Getting Started," shows a simple program, explains its operation, and discusses considerations for data types.

Chapter 3, "Programming over HP-IB," discusses the general considerations for programming the instrument over an HP-IB interface.

Chapter 4, "Programming over RS-232-C," discusses the general considerations for programming the instrument over an RS-232-C interface.

Chapter 5, "Programming and Documentation Conventions," describes the conventions used in representing the syntax of commands throughout this book and the *HP 54620A/C Programmer's Reference*, and gives an overview of the analyzer command set.

Chapter 6, "Status Reporting," discusses the analyzer status registers and how to use them in your programs.

Chapter 7, "Installing and Using the Programmer's Reference," tells how to install the *HP 54620A/C Programmer's Reference* online help file in Microsoft Windows, and explains help file navigation.

Chapter 8, "Programmer's Quick Reference," lists all the commands and queries available for programming the logic analyzer.

For information on analyzer operation, see the *HP 54620A/C User's Guide*. For information on interface configuration, see the documentation for the analyzer interface module and the interface card used in your controller (for example, the HP 82341A interface for IBM PC-compatible computers).

Contents

1

Introduction to Programming

Introduction to Programming

Chapters 1 and 2 introduce the basics for remote programming of a logic analyzer. The programming instructions in this manual conform to the IEEE 488.2 Standard Digital Interface for Programmable Instrumentation. The programming instructions provide the means of remote control.

To program the HP 54620A Logic Analyzer or HP 54620C (color) Logic Analyzer, you must add either an HP-IB (for example, HP 54650A) or RS-232-C (for example, HP 54651A or HP 54652B) interface to the rear panel.

You can perform the following basic operations with a controller and a logic analyzer:

- Set up the instrument.
- Make measurements.
- Get data (waveform, measurements, configuration) from the logic analyzer.
- Send information (pixel image, configurations) to the logic analyzer.

Other tasks are accomplished by combining these basic functions.

Languages for Program Examples

The programming examples for individual commands in this manual are written in HP BASIC or C.

Talking to the instrument

Computers acting as controllers communicate with the instrument by sending and receiving messages over a remote interface. Instructions for programming normally appear as ASCII character strings embedded inside the output statements of a “host” language available on your controller. The input statements of the host language are used to read in responses from the logic analyzer.

For example, HP BASIC uses the OUTPUT statement for sending commands and queries. After a query is sent, the response is usually read in using the ENTER statement.

Messages are placed on the bus using an output command and passing the device address, program message, and terminator. Passing the device address ensures that the program message is sent to the correct interface and instrument.

The following HP BASIC statement sends a command which sets the threshold voltage for channels 0 through 7 to TTL:

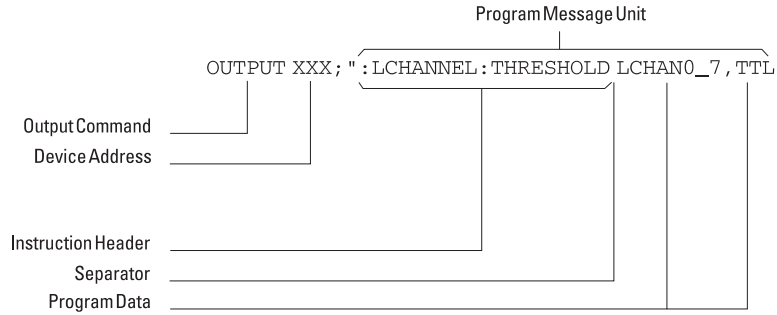
```
OUTPUT < device address > ;
  ":LCHANNEL:THRESHOLD LCHAN0_7,TTL"<terminator>
```

The < device address > represents the address of the device being programmed. Each of the other parts of the above statement are explained in the “Program Message Syntax” which follows.

Program message syntax

To program the instrument remotely, you must understand the command format and structure expected by the instrument. The IEEE 488.2 syntax rules govern how individual elements such as headers, separators, program data, and terminators may be grouped together to form complete instructions. Syntax definitions are also given to show how query responses are formatted. Figure 1 shows the main syntactical parts of a typical program statement.

Figure 1



Program Message Syntax

Output Command

The output command is entirely dependent on the programming language. Throughout this manual, HP BASIC is used in most of the individual command exampless. If you are using other languages, you must find the equivalents of HP BASIC commands like OUTPUT, ENTER, and CLEAR to convert the examples. For example, the example progam in chapter 2 is written in C, and uses the library function IOOUTPUTS to send a command. The instructions listed in this manual are always enclosed in quotation marks in the example programs.

Device Address

The location where the device address must be specified also depends on the programming language you are using. In some languages, this may be specified outside the output command. In HP BASIC, this is always specified after the keyword OUTPUT. The examples in this manual assume the logic analyzer is at device address 707. When writing programs, the address varies according to how the bus is configured.

Instructions

Each instruction (for both commands and queries) normally appears as a string embedded in a statement of your host language, such as BASIC, Pascal, or C. The only time a parameter is not meant to be expressed as a string is when the instruction's syntax definition specifies <block data>. There are only a few instructions that use block data. See the :WAVeform:DATA? query for an example of a command that uses block data.

Instructions are composed of two main parts:

- The header, which specifies the command or query to be sent.
- The program data, which provides additional information needed to clarify the meaning of the instruction.

Instruction Header

The instruction header represents the operation to be performed by the instrument, and is one or more mnemonics separated by colons (:). The command tree in chapter 5 illustrates how all the mnemonics can be joined together to form a complete header (see chapter 5, "Programming and Documentation Conventions").

The example in figure 1 is a command. Queries are indicated by adding a question mark (?) to the end of the header. Many instructions can be used as either commands or queries, depending on whether or not you have included the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

White Space (Separator)

White space is used to separate the instruction header from the program data. If the instruction does not require any program data parameters, you do not need to include any white space. In this manual, white space is defined as one or more spaces. ASCII defines a space to be character 32 (in decimal).

Program Data

Program data are used to clarify the meaning of the command or query. They provide necessary information, such as whether a function should be on or off, or which waveform is to be displayed. Each instruction's syntax definition shows the program data, as well as the values they accept. The "Program Data Syntax Rules" topic in this chapter describes all of the general rules about acceptable values.

When there are more than one data parameter, they are separated by commas (,). Spaces can be added around the commas to improve readability.

Header Types

There are three types of headers:

- Simple command headers
- Compound command headers
- Common command headers

Simple command header Simple command headers contain a single mnemonic. :AUToscale and :DIGitize are examples of simple command headers typically used in this instrument. The syntax is:

```
<program mnemonic><terminator>
```

Simple command headers must occur at the beginning of a program message; if not, they must be preceded by a colon.

When program data must be included with the simple command header (for example, :BLANk LCHan0), white space is added to separate the data from the header. The syntax is:

```
<program mnemonic><separator><program data><terminator>
```

Compound command header Compound command headers are a combination of two program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example:

To execute a single function within a subsystem:

```
:<subsystem>:<function><separator><program data><terminator>
```

(For example :DISPlay:TEXT BLANk)

Common command header Common command headers control IEEE 488.2 functions within the instrument (such as clear status). Their syntax is:

```
*<command header><terminator>
```

No space or separator is allowed between the asterisk (*) and the command header. *CLS is an example of a common command header.

Combining commands from the same subsystem

To execute more than one function within the same subsystem a semi-colon (;) is used to separate the functions:

```
:<subsystem>:<function><separator><data>;
     <function><separator><data><terminator>
```

(For example :TIMebase:REFerence LEFT;VERNier ON)

Duplicate mnemonics

Identical function mnemonics can be used for more than one subsystem. For example, the function mnemonic DELAY may be used to set the trigger delay or measure channel-to-channel delay:

```
:TIMEBASE:DELAY 30 US
```

sets the trigger delay to 30 μs.

```
:MEASURE:DELAY
```

measures the delay betwen predefined start and end channels (see :MEASure:DEFine:DELay).

:TIMebase and :MEASure are subsystem selectors and determine which meaning of DELAY is used.

Query command

Command headers immediately followed by a question mark (?) are queries. After receiving a query, the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued. When read, the answer is transmitted across the bus to the designated listener (typically a controller). For example, the query :TIMebase:RANGE? places the current time base setting in the output queue. In HP BASIC, the controller input statement:

```
ENTER < device address > ;Range
```

passes the value across the bus to the controller and places it in the variable Range.

Query commands are used to find out how the instrument is currently configured. They are also used to get results of measurements made by the instrument. For example, the command :MEASure:PERiod? instructs the instrument to measure the period of the current measurement source channel and places the result in the output queue.

The output queue must be read before the next program message is sent. For example, when you send the query :MEASure:PERiod? you must follow that query with an input statement. In HP BASIC, this is usually done with an ENTER statement immediately followed by a variable name. This statement reads the result of the query and places the result in a specified variable.

Read the Query Result First

Sending another command or query before reading the result of a query causes the output buffer to be cleared and the current response to be lost. This also generates a query interrupted error in the error queue.

Program header options

Program headers can be sent using any combination of uppercase or lowercase ASCII characters. Instrument responses, however, are always returned in uppercase.

Program command and query headers may be sent in either long form (complete spelling), short form (abbreviated spelling), or any combination of long form and short form. For example:

```
TIMEBASE:DELAY 1US  - long form
TIM:DEL 1US  - short form
```

Programs written in long form are easy to read and are almost self-documenting. The short form syntax conserves the amount of controller memory needed for program storage and reduces the amount of I/O activity.

Command Syntax Programming Rules

The rules for the short form syntax are described in chapter 5, “Programming and Documentation Conventions.”

Program data syntax rules

Program data conveys a variety of types of parameter information related to the command header. At least one space must separate the command header or query header from the program data.

```
<program mnemonic><separator><data><terminator>
```

When a program mnemonic or query has multiple program data, a comma separates sequential program data.

```
<program mnemonic><separator><data>,<data><terminator>
```

For example, :DISPlay:PIXel 10,12,2 has three program data: 10, 12, and 2.

There are two main types of program data used in commands: character and numeric program data.

Character Program Data

Character program data conveys parameter information as alpha or alphanumeric strings. For example, the :TIMebase:MODE command can be set to normal or delayed. The character program data in this case may be NORMal or DELayed. The command :TIMebase:MODE DELayed sets the time base mode to delayed.

The available mnemonics for character program data are always included with the instruction's syntax definition. When sending commands, either the long form or short form (if one exists) may be used. Uppercase and lowercase letters may be mixed freely. When receiving query responses, uppercase letters are used exclusively.

Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMebase:RANGe requires the desired full-scale range to be expressed numerically.

For numeric program data, you have the option of using exponential notation or using suffix multipliers to indicate the numeric value. The following numbers are all equal:

28 = 0.28E2 = 280e-1 = 28000m = 0.028K = 28e-3K

When a syntax definition specifies that a number is an integer, it means that the number should be whole. Any fractional part would be ignored, truncating the number. Numeric data parameters that accept fractional values are called real numbers.

All numbers are expected to be strings of ASCII characters. Thus, when sending the number 9, you would send a byte representing the ASCII code for the character “9” (which is 57). A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). This is taken care of automatically when you include the entire instruction in a string.

Embedded Strings

Embedded strings contain groups of alphanumeric characters which are treated as a unit of data by the logic analyzer. For example, this line of text written to the advisory line of the instrument with the :SYSTem:DSP command is an embedded string:

```
:SYSTEM:DSP"This is a message."
```

Embedded strings may be delimited with either single (’) or double (") quotation marks. These strings are case-sensitive, and spaces act as legal characters—just like any other character.

Program message terminator

The program instructions within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted in the HP-IB interface, or a combination of the two. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

New Line Terminator Functions

The NL (New Line) terminator has the same function as an EOS (End Of String) and EOT (End Of Text) terminator.

Selecting multiple subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

```
<program mnemonic><data>;
   :<program mnemonic><data><terminator>

:TRIGGER:MODE NORMAL;:TIMEBASE:RANGE 1
```

Combining Compound and Simple Commands

Multiple commands may be any combination of compound and simple commands.

2

Programming Getting Started

Programming Getting Started

This chapter explains how to:

- Set up the instrument
- Retrieve setup information and measurement results
- Digitize a waveform
- Pass data to the controller

Languages for Program Examples

The programming examples in this manual are written in HP BASIC or C.

Initialization

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. HP BASIC provides a CLEAR command which clears the interface buffer:

```
CLEAR 707  ! initializes the interface of the instrument
```

When you are using HP-IB, CLEAR also resets the logic analyzer's parser. The parser is the program that reads in the instructions you send to it.

After clearing the interface, initialize the instrument to a preset state:

```
OUTPUT 707;"*RST"  ! initializes the instrument to a preset
state.
```

Information for Initializing the Instrument

The actual commands and syntax for initializing the instrument are discussed in the online *HP 54620A/C Programmer's Reference*.

Refer to your controller manual and programming language reference manual for information about initializing the interface.

Autoscale

The :AUToscale feature performs a very useful function on unknown waveforms by setting up the instrument logic thresholds, initial channel settings, and time base.

The syntax for the autoscale function is:

```
:AUTOSCALE<terminator>
```

Setting up the instrument

A typical logic analyzer setup would set the threshold voltages, turn on the appropriate channels, and set up the sweep speed, delay time, trigger mode, trigger type, and trigger specification. An example of the commands sent to the logic analyzer is as follows:

```
:LCHANNEL:THRESHOLD LCHAN0_7, TTL<terminator>
:LCHANNEL:THRESHOLD LCHAN8_15, ECL<terminator>
:BLANK LCHANNEL7<terminator>
:BLANK LCHANNEL12<terminator>
:BLANK LCHANNEL13<terminator>
:TIMEBASE:RANGE 100US;REFERENCE CENTER;DELAY 1.5US<terminator>
:TRIGGER:TYPE EDGE;EDGE LCHANNEL3,FALLING<terminator>
:TRIGGER:MODE NORMAL<terminator>
```

This example sets channels 0 through 7 to TTL thresholds, sets channels 8 through 15 to CMOS thresholds, and turns off channels 7, 12, and 13 (the :BLANk command). It sets the sweep speed to 100 µs/div, with time reference at center and positive delay of 1.5 µs. Finally, it sets the trigger mode to normal with an edge trigger on the falling edge of channel 3.

Example program

This C program demonstrates the basic command structure used to program the logic analyzer, and how that command structure is embedded into the C language.

```
/*HP Instrument C program using the HP-IB interface                    */

#include <stdio.h>   /* Include file for printf, etc */
#include <stdlib.h>  /* Include file for malloc, etc */
#include "CHPIB.H"   /* HPIB Library constant declarations */
#include "CFUNC.H"   /* HPIB library function prototypes */

/* Globals used throughout the program */
long isc;
long analyzer;
int  error;

/* errorhandle - This function prints an error message of the screen    */
/*                 regarding the command that caused the error.         */
void errorhandle( char *function, char *cmdcause )
{
  if ( error != NOERR )
  {
    printf("HPIB error in call to %s with %s, error = %s\n",
            function, cmdcause, errstr(error) );
    exit(1);
  }
}

/* sendstringcmd - this function interfaces to the actual hpib output   */
/*                 functions.  It also checks for errors and calls the  */
/*                 errorhandler function above                          */
void sendstringcmd( char *cmd)
{
  error = IOOUTPUTS( analyzer, cmd, strlen(cmd) );
  errorhandle("IOOUTPUTS", cmd);
}

int main()
{
  isc = 7;                                    /* Assign 7 to isc global */
  analyzer = 707;            /* Assign device address (707) to analyzer */
  error = IORESET ( isc );                            /* Reset the isc */
```

```
  errorhandle( "IORESET");                        /* Report any problems */
  error = IOTIMEOUT (isc, 5.0);                     /* 5 second timeout  */
  errorhandle( "IOTIMEOUT");                      /* Report any problems */
  error = IOCLEAR (isc);                                 /* Clear the isc */
  errorhandle( "IOCLEAR");                        /* Report any problems */

  sendstringcmd("*RST");                         /* RESET the instrument */

  /* set trigger mode to NORMAL */
  sendstringcmd ( ":TRIGGER:MODE NORMAL");

  /* set trigger type to Simple edge */
  sendstringcmd ( ":TRIGGER:TYPE EDGE" );

  /* set trigger to rising edge on channel 0 */
  sendstringcmd (":TRIGGER:EDGE LCHANNEL0, RISING");

  /* set thresholds, time base and delay */
  sendstringcmd (":autoscale");

  return 0;
} /* end main */
```

Program overview

The first few lines of the program include header files that define function prototypes for standard C libraries and specialized libraries that handle the HP-IB interface. The program then defines variables for the interface select code (`isc`), analyzer address (`analyzer`), and error handling.

The `errorhandle()` function takes two strings defining the program function and analyzer command string. If the global `error` variable is non-zero, it prints (to the controller screen) an error message that includes this information. It then forces the program to exit.

The `sendstringcmd()` function encapsulates data tranmission to the instrument with error handling. First it calls the HP-IB library function `IOOUTPUTS` to send the specified command string (passed through `cmd`) to the analyzer. The result of the `IOOUTPUTS` call is returned to the error variable, so the `errorhandle` function is called to process the error (if one occurred).

The `main()` function initializes the interface select code and analyzer address variables. It then calls HP-IB library functions `IORESET`, `IOTIMEOUT`, and `IOCLEAR` to initialize the HP-IB card (and check for proper initialization). Then the `sendstringcmd` function is called several times to reset the analyzer, set the trigger mode to normal, define an edge trigger on the rising edge of channel 0, and invoke the autoscale feature.

Using the :DIGitize command

The :DIGitize command captures data that satisfyies the specifications set up during analyzer configuration. When the digitize process is complete, the acquisition is stopped. The captured data can then be measured by the instrument or transferred to the controller for further analysis. The captured data consists of two parts: the waveform data record and the preamble.

Ensure New Data is Collected

After changing the logic analyzer configuration, the waveform buffers are cleared. Before running a measurement, send the DIGITIZE command to the analyzer to ensure new data has been collected.

When you send the :DIGitize command to the analyzer, the input to all analyzer channels is captured and put into acquisition memory based on the current time base and threshold settings and trigger specification. To transfer the acquired waveform data to the controller, you must specify the :WAVeform parameters for the waveform data prior to sending the :WAVeform:DATA? query.

Set :TIMebase:MODE to NORMal when Using :DIGitize

:TIMebase:MODE must be set to NORMal to perform a :DIGitize or a WAVeform subsystem query. A "Settings conflict" error message will be returned if these commands are executed when MODE is set to DELayed. Sending the *RST (reset) command will also set the time base mode to normal.

The number of data points comprising a waveform is either 2048 or 8192, depending on whether the analyzer is in glitch mode. See the :ACQuire subsystem information in the *HP 54620A/C Programmer's Reference* online help file for more information.

The following program example shows a typical setup:

```
OUTPUT 707;":ACQUIRE:TYPE GLITCH"<terminator>
OUTPUT 707;":WAVEFORM:SOURCE LCHAN0_15"<terminator>
OUTPUT 707;":WAVEFORM:FORMAT BYTE"<terminator>
OUTPUT 707;":WAVEFORM:POINTS 512"<terminator>
OUTPUT 707;":DIGITIZE"<terminator>
OUTPUT 707;":WAVEFORM:PREAMBLE?"<terminator>
GOSUB Read_Preamble
OUTPUT 707;":WAVEFORM:DATA?"<terminator>
GOSUB Read_WaveData
```

This setup places the instrument into glitch mode, so there will be 2048 samples available. This means that when the :DIGitize command is received, the command will execute until all 2048 samples have been acquired. Here, however, the number of waveform points to be returned to the controller (upon a :WAVeform:DATA? query) is set to 512. This must always be an integer divisor of the number of samples available. The data for all 16 channels will be returned (:WAVeform:SOURce), in byte format (:WAVeform:FORMat).

The :WAVeform:PREamble? query returns the parameters of the acquired data, including the acquisition mode, number of points, and time interval between samples. These parameters are important for interpreting the data returned by the :WAVeform:DATA? query.

After receiving the :WAVeform:DATA? query, the instrument will start passing the waveform information when addressed to talk. The digitized waveforms are passed from the instrument to the controller by sending a numerical representation of each digitized point. The format of the numerical representation is controlled with the :WAVeform:FORMat command and may be selected as BYTE or WORD. The :WAVeform:BYTeorder command specifies whether the data should be sent MSB first or LSB first when WORD is the selected format.

The amount of data returned by the :WAVeform:DATA query depends on how many channels are selected to return data (the :WAVeform:SOURce command), whether the data is returned in byte or word format (the :WAVeform:FORMat command), and whether the acquisition was performed in glitch or normal mode (the :ACQuire:TYPE command).

The easiest method of transferring a digitized waveform depends on data structures, formatting available, and I/O capabilities.

For more information, see the waveform subsystem commands and corresponding program code examples in the online *HP 54620A/C Programmer's Reference*.

Aborting a Digitize Operation Over HP-IB

When using HP-IB, a digitize operation may be aborted by sending a Device Clear over the bus (CLEAR 707).

Receiving information from the instrument

After receiving a query (command header followed by a question mark), the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or until another command is issued. When read, the answer is transmitted across the interface to the designated listener (typically a controller). The input statement for receiving a response message from an instrument's output queue typically has two parameters; the device address, and a format specification for handling the response message. For example, to read the result of the query command :TRIGger:TYPE you would execute the HP BASIC statement:

```
ENTER <device address> ;Setting$
```

where <device address> represents the address of your device. This would enter the current setting for the channel one coupling in the string variable Setting$.

All results for queries sent in a program message must be read before another program message is sent. For example, when you send the query :TRIGger:ADVanced:DURation?, you must follow that query with an input statement. In HP BASIC, this is usually done with an ENTER statement.

Sending another command before reading the result of the query causes the output buffer to be cleared and the current response to be lost. This also causes an error to be placed in the error queue.

Executing an input statement before sending a query causes the controller to wait indefinitely.

The format specification for handling response messages depends on both the controller and the programming language.

String variables

The instrument output may be numeric or character data depending on what is queried. Refer to the specific commands for the formats and types of data returned from queries.

Express String Variables Using Exact Syntax

In HP BASIC, string variables are case-sensitive and must be expressed exactly the same each time they are used.

Address Varies According to Configuration

For the example programs in the help file, assume that the device being programmed is at device address 707. The actual address varies according to how you have configured the bus for your own application.

The following example shows the data being returned to a string variable:

```
10 DIM Rang$[30]
20 OUTPUT 707;":TIMEBASE:RANGE?"
30 ENTER 707;Rang$
40 PRINT Rang$
50 END
```

If the current time base setting is 10 ms/div, the controller displays:

```
+1.00000E-01
```

Numeric variables

The following example shows the data being returned to a numeric variable:

```
10 OUTPUT 707;":TIMEBASE:RANGE?"
20 ENTER 707;Rang
30 PRINT Rang
40 END
```

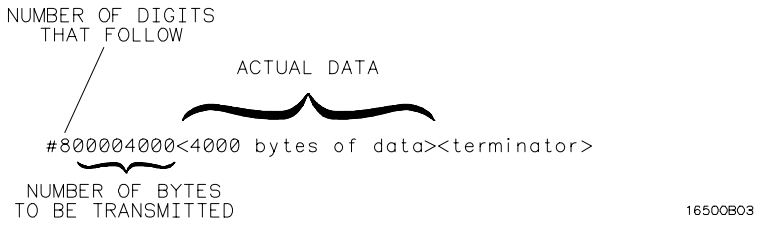
If the time base is set to 1 s/div, the controller displays:

```
1
```

Definite-length block response data

Definite-length block response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. The syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 4000 bytes of data, the syntax would be:



The “8” states the number of digits that follow, and “00004000” states the number of bytes to be transmitted.

Multiple queries

You can send multiple queries to the instrument in a single program message. When sending multiple queries in this way, you must then read them back in a single program message. You can do this by either reading them back into a string variable or into multiple numeric variables. For example, you could read the result of the query :TIMebase:RANGe?;DELay? into the string variable Results$ with the command:

```
ENTER 707;Results$
```

When you read the result of multiple queries into string variables, each response is separated by a semicolon. For example, the response of the query :TIMebase:RANGe?;DELay? would be:

```
<range_value>; <delay_value>
```

Use the following program message to read the query :TIMebase:RANGe?;DELay? into multiple numeric variables:

```
ENTER 707;Result1,Result2
```

Instrument status

Status registers track the current status of the instrument. By checking the instrument status, you can find out whether an operation has been completed, whether the instrument is receiving triggers, and more. Chapter 6, “Status Reporting” explains how to check the status of the instrument.

3

Programming over HP-IB

Programming over HP-IB

This section describes the HP-IB interface functions and some general concepts. In general, these functions are defined by IEEE 488.1. They deal with general interface management issues, as well as messages which can be sent over the interface as interface commands.

For more information about connecting the controller to the logic analyzer, see the documentation for the HP-IB interface card you are using.

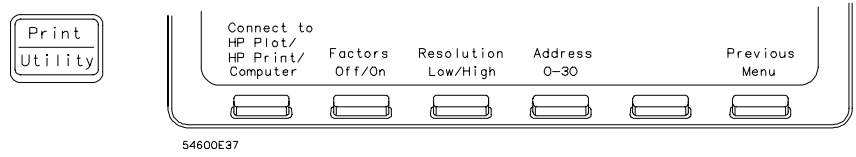
Interface capabilities

The interface capabilities of the logic analyzer, as defined by IEEE 488.1, are SH1, AH1, T5, L4, SR1, RL1, PP0, DC1, DT1, C0, and E2.

Addressing

By using the front-panel controls, the instrument must be placed in the "Connect to Computer" mode. Figure 2 is the Print/Utility menu after the HP 54650A HP-IB interface has been installed on the rear panel of the logic analyzer. Use this menu to set the HP-IB address for the logic analyzer and "Connect to Computer."

Figure 2



Print/Utility Menu with HP-IB

- Each device on the HP-IB resides at a particular address ranging from 0 to 30.
- The active controller specifies which devices talk and which listen.
- An instrument may be talk addressed, listen addressed, or unaddressed by the controller.

If the controller addresses the instrument to talk, the instrument remains configured to talk until it receives an interface clear message (IFC), another instrument's talk address (OTA), its own listen address (MLA), or a universal untalk command (UNT).

If the controller addresses the instrument to listen, the instrument remains configured to listen until it receives an interface clear message (IFC), its own talk address (MTA), or a universal unlisten command (UNL).

Communicating over the bus

Because HP-IB can address multiple devices through the same interface card, the device address passed with the program message must include not only the correct interface select code, but also the correct instrument address.

Interface Select Code (Selects Interface)

Each interface card has a unique interface select code. This code is used by the controller to direct commands and communications to the proper interface. The default is typically "7" for HP-IB controllers.

Instrument Address (Selects Instrument)

Each instrument on an HP-IB must have a unique instrument address between decimal 0 and 30. The device address passed with the program message must include not only the correct instrument address, but also the correct interface select code.

```
DEVICE ADDRESS = (Interface Select Code * 100) + (Instrument Address)
```

For example, if the instrument address for the logic analyzer is 4 and the interface select code is 7, when the program message is passed, the routine performs its function on the instrument at device address 704.

Logic Analyzer Device Address

The examples in this manual and in the online *HP 54620A/C Programmer's Reference* assume the logic analyzer is at device address 707.

See the documentation for your HP-IB interface card for more information about select codes and addresses.

Lockout

You can use the SYSTem:LOCK ON command to disable front-panel control while a program is running. By default, the instrument accepts and executes bus commands, and the front panel is entirely active.

Restore Front-Panel Control

The :SYSTem:LOCK OFF command will restore front-panel control. Cycling power also restores front-panel control.

With HP-IB, the instrument is placed in the lockout mode by sending the local lockout command (LLO). The instrument can be returned to local by sending the go-to-local command (GTL) to the instrument.

Bus commands

The following commands are IEEE 488.1 bus commands (ATN true). IEEE 488.2 defines many of the actions taken when these commands are received by the instrument.

Device clear

The device clear (DCL) or selected device clear (SDC) commands clear the input and output buffers, reset the parser, and clear any pending commands. If either of these commands is sent during a digitize operation, the digitize operation is aborted.

Interface clear (IFC)

The interface clear (IFC) command halts all bus activity. This includes unaddressing all listeners and the talker, disabling serial poll on all devices, and returning control to the system controller.

Programming over RS-232-C

Programming over RS-232-C

This section describes the interface functions and some general concepts of the RS-232-C. The RS-232-C interface on this instrument is Hewlett-Packard's implementation of EIA Recommended Standard RS-232-C, "Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange." With this interface, data is sent one bit at a time and characters are not synchronized with preceeding or subsequent data characters. Each character is sent as a complete entity without relationship to other events.

IEEE488.2 Operates with IEEE 488.1 or RS-232-C

IEEE 488.2 is designed to work with IEEE 488.1 as the physical interface. When RS-232-C is used as the physical interface, as much of IEEE 488.2 is retained as the hardware differences will allow. Messages for IEEE 488.1, such as DCL, GET, and END are not available.

Interface operation

The logic analyzer can be programmed with a controller over RS-232-C using either a minimum three-wire or extended hardwire interface. The operation and exact connections for these interfaces are described in more detail in the following sections. When you are programming the logic analyzer over RS-232-C with a controller, you are normally operating directly between two DTE (Data Terminal Equipment) devices as compared to operating between a DTE device and a DCE (Data Communications Equipment) device.

When operating directly between two RS-232-C devices, you must take certain considerations into account. For three-wire operation, an XON/XOFF software handshake must be used to handle handshaking between the devices. For extended hardwire operation, handshaking may be handled either with XON/XOFF or by manipulating the CTS and RTS lines of the logic analyzer interface module. For both three-wire and extended hardwire operation, the DCD and DSR inputs to the logic analyzer interface module must remain high for proper operation.

With extended hardwire operation, a high on the CTS input allows the logic analyzer to send data and a low on this line disables the logic analyzer data transmission. Likewise, a high on the RTS line allows the controller to send data, and a low on this line signals a request for the controller to disable data transmission. Because three-wire operation has no control over the CTS input, internal pull-up resistors in the logic analyzer interface module ensure that this line remains high for proper three-wire operation.

Cables

Selecting a cable for the RS-232-C interface depends on your specific application. The following paragraphs describe which lines of the logic analyzer are used to control the operation of the RS-232-C bus relative to the logic analyzer. To locate the proper cable for your application, refer to the reference manual for your controller. This manual should address the exact method your controller uses to operate over the RS-232-C bus.

Minimum Three-wire Interface with XON/XOFF Software Handshake

With a three-wire interface, the *software* (as compared to interface hardware) controls the data flow between the logic analyzer and the controller. This provides a much simpler connection between devices

because you can ignore hardware handshake requirements. The logic analyzer uses the following connections on its RS-232-C interface for three-wire communication:

- Pin 7 SGND (Signal Ground)
- Pin 2 TD (Transmit Data from logic analyzer)
- Pin 3 RD (Receive Data into logic analyzer)

These connections assume you have an interface module with a 25-pin RS-232 connector. If the RS-232 connector on your module is not a 25-pin connector, refer to the documentation that came with the module for connection information.

The TD (Transmit Data) line from the logic analyzer must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the logic analyzer must connect to the TD line on the controller. Internal pull-up resistors in the logic analyzer interface module ensure the DCD, DSR, and CTS lines remain high when you are using a three-wire interface.

No Hardware Means to Control Data Flow

The three-wire interface does not provide a hardware means to control data flow between the controller and the logic analyzer. XON/XOFF protocol is the only means to control this data flow.

Extended Interface with Hardware Handshake

With the extended interface, both the software and the hardware can control the data flow between the logic analyzer and the controller. This allows you to have more control of data flow between devices. The logic analyzer uses the following connections on its RS-232-C interface for extended interface communication (on a 25-pin connector):

- Pin 7 SGND (Signal Ground)
- Pin 2 TD (Transmit Data from logic analyzer)
- Pin 3 RD (Receive Data into logic analyzer)

The additional lines you use depend on your controller's implementation of the extended hardwire interface.

- Pin 4 RTS (Request To Send) is an output from the logic analyzer which can be used to control incoming data flow.

- Pin 5 CTS (Clear To Send) is an input to the logic analyzer which controls data flow from the logic analyzer.
- Pin 6 DSR (Data Set Ready) is an input to the logic analyzer which controls data flow from the logic analyzer within two bytes.
- Pin 8 DCD (Data Carrier Detect) is an input to the logic analyzer which controls data flow from the logic analyzer within two bytes.
- Pin 20 DTR (Data Terminal Ready) is an output from the logic analyzer which is enabled as long as the logic analyzer is turned on.

The TD (Transmit Data) line from the logic analyzer must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the logic analyzer must connect to the TD line on the controller.

The RTS (Request To Send) line is an output from the logic analyzer, and can be used to control incoming data flow. A high on the RTS line allows the controller to send data, and a low on this line signals a request for the controller to disable data transmission.

The CTS (Clear To Send), DSR (Data Set Ready), and DCD (Data Carrier Detect) lines are inputs to the logic analyzer which control data flow from the logic analyzer (Pin 2). Internal pull-up resistors in the logic analyzer assure the DCD and DSR lines remain high when they are not connected.

If DCD or DSR are connected to the controller, the controller must keep these lines and the CTS line high to enable the logic analyzer to send data to the controller. A low on any one of these lines will disable the logic analyzer data transmission. Dropping the CTS line low during data transmission will stop logic analyzer data transmission immediately. Dropping either the DSR or DCD line low during data transmission will stop logic analyzer data transmission, but as many as two additional bytes may be transmitted from the logic analyzer.

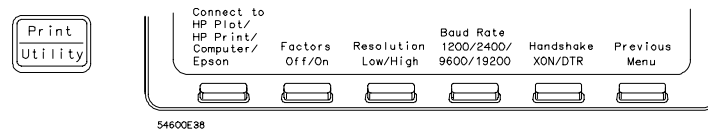
Buffering and Handshaking

Your program needs to account for proper handling of handshaking and buffering of data between the controller and the logic analyzer. In some cases, such as some dialects of BASIC, this support is built into the programming language. In others, such as C, providing these features is the programmer's responsibility. There are many third-party data communications function libraries that provide data communications support for those languages that do not provide such support directly.

Configuring the interface

By using the front-panel menus, the RS-232-C interface can be placed in either the printer mode or the computer mode. The printer mode should be used when you want the instrument to talk directly to a printer over RS-232-C without the aid of a computer. The controller mode is used when the instrument will operate with a computer over RS-232-C. Figure 3 is the logic analyzer Print/Utility menu after the RS-232-C interface module has been installed on the rear panel. Use this menu to "Connect to Computer," and assign the baud rate and handshake protocol. The HP 54652B module has a slightly different set of menus.

Figure 3



Print/Utility Menu with HP 54651A RS-232 Interface Module Installed

Interface capabilities

The baud rate, stop bits, parity, handshake protocol, and data bits must be configured exactly the same for both the controller and the logic analyzer to properly communicate over the RS-232-C bus. The analyzer RS-232-C interface capabilities are as follows:

- Baud Rate: 1200, 2400, 9600, or 19.2 k
- Stop Bits: 1
- Parity: None
- Handshake Protocol: DTR or XON/XOFF
- Data Bits: 8

Handshake Protocol

DTR (Data Terminal Ready) With a three-wire interface, selecting DTR for the handshake protocol does not allow the sending or receiving device to control data flow. No control over the data flow increases the possibility of missing data or transferring incomplete data.

With an extended hardwire interface, selecting DTR allows a hardware handshake to occur. With hardware handshake, hardware signals control data flow.

XON/XOFF XON/XOFF stands for Transmit On/Transmit Off. With this mode the receiver (controller or logic analyzer) controls data flow and can request that the sender (logic analyzer or controller) stop data flow. By sending XOFF (ASCII 17) over its transmit data line, the receiver requests that the sender disables data transmission. A subsequent XON (ASCII 19) allows the sending device to resume data transmission.

A controller sending data to the logic analyzer should send no more than 32 bytes of data after an XOFF.

The logic analyzer will not send any data after an XOFF is received until an XON is received.

Data Bits

Data bits are the number of bits sent and received per character that represent the binary code of that character.

Information is stored in bytes (8 bits at a time) in the logic analyzer. Data can be sent and received just as it is stored, without the need to convert the data.

Lockout Command

Use the :SYSTem:LOCK ON command to lockout the front-panel controls. When this function is on, all controls (except the power switch) are entirely locked out. Local control can only be restored by sending the command :SYSTem:LOCK OFF.

Restoring Local Control

The :SYSTem:LOCK OFF command will restore front-panel control. Cycling power will restore local control, but will also reset certain RS-232-C states.

5

Programming and Documentation Conventions

Programming and Documentation Conventions

This chapter describes conventions used in programming the instrument, as well as conventions used in the online *HP 54620A/C Programmer's Reference* and the remainder of this manual. This chapter also contains a detailed description of the command tree and command tree traversal.

Command Set Organization

The command set is divided into common commands, root level commands and eight sets of subsystem commands. Each of the ten groups of commands is described in the *HP 54620A/C Programmer's Reference*, which is supplied as an online help file for Microsoft Windows. See chapter 8 for information about installing and using the help file.

The commands are shown using both uppercase and lowercase letters. As an example, AUToscale indicates that the entire command name is AUTOSCALE. To speed up the transfer, the short form AUT is also accepted by the logic analyzer. Each command listing contains a description of the command and its arguments and the command syntax. Some commands have a programming example.

The subsystems are listed below:

- SYSTem—controls some basic functions of the logic analyzer.
- ACQuire—sets the parameters for acquiring and storing data.
- DISPlay—controls how waveforms, the graticule, and text are displayed and written on the screen, including the color palette for the HP 54620C (color) logic analyzer.
- LCHannel—controls all logic analyzer functions associated with individual channels or groups of channels, including channel color for the HP 54620C (color) logic analyzer.
- MEASure—selects the automatic measurements to be made and controls time markers.
- TIMebase—controls all horizontal sweep functions.
- TRIGger—controls the trigger modes and parameters for each trigger type.
- WAVeform—provides access to waveform data.

The command tree

The following command tree shows all of the logic analyzer commands and their relationships to each other. The IEEE 488.2 common commands are not listed as part of the command tree because they do not affect the position of the parser within the tree. When a program message terminator (<NL>, linefeed—ASCII decimal 10) or a leading colon (:) is sent to the instrument, the parser is set to the "root" of the command tree.

Command Types

The commands for this instrument can be categorized into three types:

- Common commands
- Root level commands
- Subsystem commands

Common commands These commands are defined by IEEE 488.2, and control some functions that are common to all IEEE 488.2 instruments.

Common commands are independent of the tree, and do not affect the position of the parser within the tree. These commands differ from root level commands in that root level commands place the parser back at the root of the command tree.

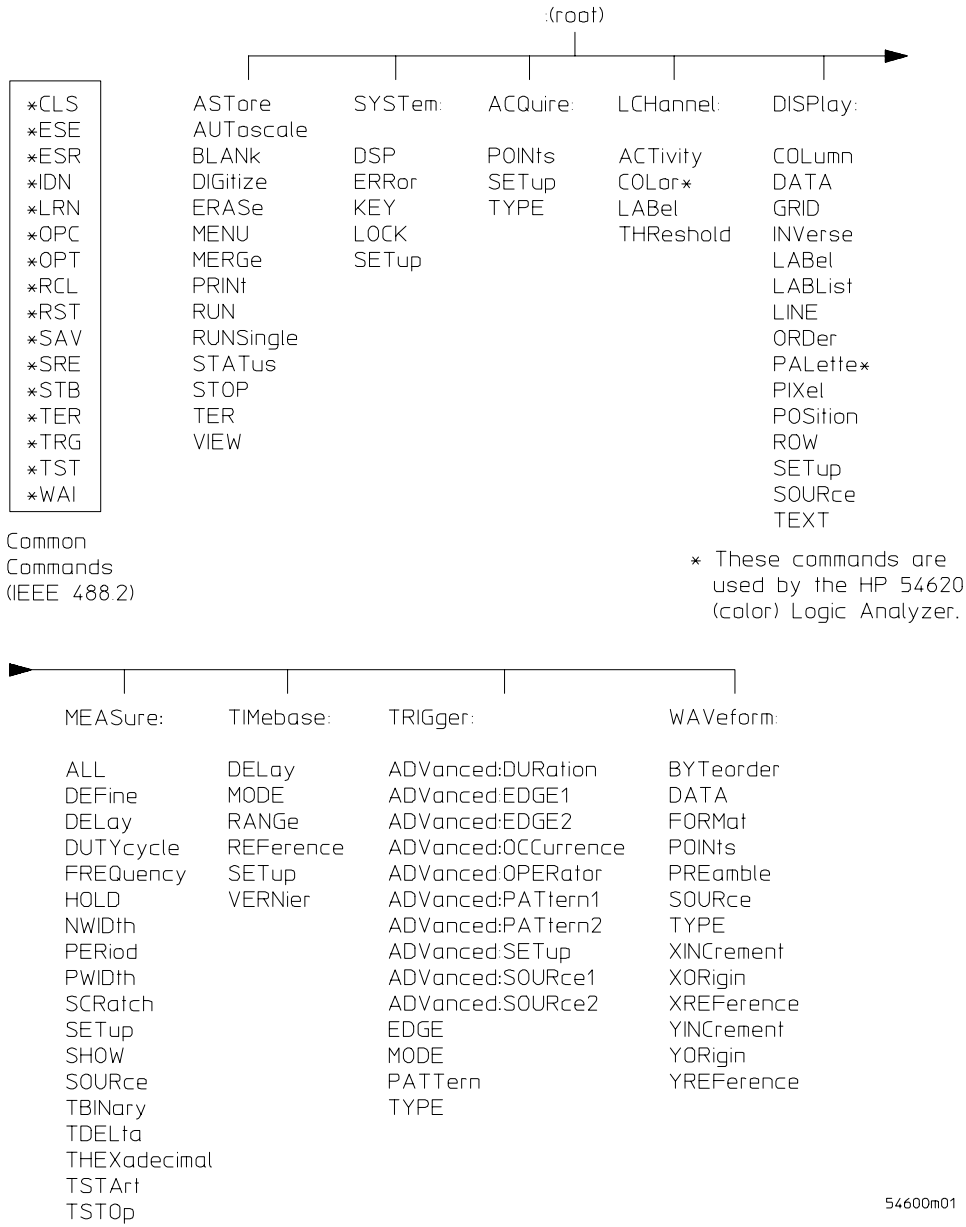
Example:

```
*RST
```

Root level commands These commands control many of the basic functions of the instrument, and reside at the root of the command tree. Root level commands are always parsable if they occur at the beginning of a program message, or are preceded by a colon.

Example:

```
:AUTOSCALE
```


:(root)
*CLS
*ESE
*ESR
*IDN
*LRN
*OPC
*OPT
*RCL
*RST
*SAV
*SRE
*STB
*TER
*TRG
*TST
*WAI
Common Commands (IEEE 488.2)
ASTore
AUToscale
BLANk
DIGitize
ERASe
MENU
MERGe
PRINt
RUN
RUNSingle
STATus
STOP
TER
VIEW
SYSTem:
DSP
ERRor
KEY
LOCK
SETup
ACQuire:
POINts
SETup
TYPE
LCHannel:
ACTivity
COLor*
LABel
THReshold
DISPlay:
COLumn
DATA
GRID
INVerse
LABel
LABList
LINE
ORDer
PALette*
PIXel
POSition
ROW
SETup
SOURce
TEXT
* These commands are used by the HP 54620C (color) Logic Analyzer.
MEASure:
ALL
DEFine
DELay
DUTYcycle
FREQuency
HOLD
NWIDth
PERiod
PWIDth
SCRatch
SETup
SHOW
SOURce
TBINary
TDELta
THEXadecimal
TSTArt
TSTOp
TIMebase:
DELay
MODE
RANGe
REFerence
SETup
VERNier
TRIGger:
ADVanced:DURation
ADVanced:EDGE1
ADVanced:EDGE2
ADVanced:OCCurrence
ADVanced:OPERator
ADVanced:PATTern1
ADVanced:PATTern2
ADVanced:SETup
ADVanced:SOURce1
ADVanced:SOURce2
EDGE
MODE
PATTern
TYPE
WAVeform:
BYTeorder
DATA
FORMat
POINts
PREamble
SOURce
TYPE
XINCrement
XORigin
XREFerence
YINCrement
YORigin
YREFerence
54600m01

Subsystem Commands

Each subsystem set of commands are grouped together under a common node of the command tree, such as the :TIMebase commands. Only one subsystem may be selected at any given time. When the instrument is initially turned on, the command parser is set to the root of the command tree, so no subsystem is selected.

Tree Traversal Rules

Command headers are created by traversing down the command tree. A legal command header from the command tree would be :TIMebase:RANGe. This is called a compound header. A compound header consists of two or more mnemonics separated by colons. The created mnemonic contains no spaces. The following rules apply to traversing the tree:

- A leading colon or a <program message terminator> (either an <NL> or EOI true on the last byte) places the parser at the root of the command tree. The leading colon is the first character of a program header.
- Executing a subsystem command places the logic analyzer in that subsystem until a leading colon or a <program message terminator> is found. In the command tree, use the last mnemonic in the compound header as a reference point (for example, RANGe). Then find the last colon above that mnemonic (TIMebase:). That is the point where the parser resides. You can send any command below that point within the current program message (for example, DELay) without sending the mnemonics that appear above them.

Examples

The OUTPUT statements in the examples are written using HP BASIC. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

Example 1:

```
OUTPUT 707;":TIMEBASE:RANGE 0.5 ;DELAY 0"
```

The colon between TIMEBASE and RANGE is necessary because TIMEBASE:RANGE is a compound command. The semicolon between the RANGE command and the DELAY command is the required program message unit separator. The DELAY command does not need TIMEBASE preceding it because the TIMEBASE:RANGE command sets the parser to the TIMEBASE node in the tree.

Example 2:

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER ; DELAY 0.00001"
```

or

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER"
OUTPUT 707;":TIMEBASE:DELAY 0.00001"
```

In the first line of example 2, the “subsystem selector” is implied for the DELAY command in the compound command. The DELAY command must be in the same program message as the REFERENCE command because the program message terminator places the parser back at the root of the command tree.

A second way to send these commands is by placing TIMEBASE: before the DELAY command as shown in the second part of example 2.

Example 3:

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER ; :LCHANNEL:LABEL 'Q0'"
```

The leading colon before LCHANNEL tells the parser to go back to the root of the command tree. The parser can then see the LCHANNEL:LABEL command.

Truncation rules

These are the truncation rules for mnemonics used in headers and alpha arguments:

The mnemonic is the first four characters of the keyword.

If

the fourth character is a vowel,

then

the mnemonic is the first three characters of the keyword.

This rule is not applied if the length of the keyword is exactly four characters.

Some examples of how the truncation rules are applied to various commands are shown in the following table.

Mnemonic Truncation

Long Form	Short Form
RANGE	RANG
PATTERN	PATT
TIMEBASE	TIM
DELAY	DEL
TYPE	TYPE

Infinity representation

The representation of infinity is 9.9E+37. This is also the value returned when a measurement cannot be made.

Sequential and overlapped commands

IEEE 488.2 distinguishes between sequential and overlapped commands. Sequential commands finish their task before the next command begins to execute. Overlapped commands run concurrently. Commands following an overlapped command may be started before the overlapped command is completed. All of the commands are sequential.

Response generation

As defined by IEEE 488.2, query responses may be buffered for the following conditions:

- When the query is parsed by the instrument.
- When the controller addresses the instrument to talk so that it can read the response.

The responses to a query are buffered when the query is parsed.

Notation conventions and definitions

The following conventions and definitions are used in this manual and the online *HP 54620A/C Programmer's Reference* in descriptions of remote operation:

Conventions

< > Angle brackets enclose words or characters that symbolize a program code parameter or an interface command.

::= "is defined as." For example, <A> ::= <B> indicates that <A> can be replaced by <B> in any statement containing <A>.

| "or." Indicates a choice of one element from a list. For example, <A> | <B> indicates <A> or <B>, but not both.

... An ellipsis (trailing dots) indicates that the preceding element may be repeated one or more times.

[ ] Square brackets indicate that the enclosed items are optional.

{ } When several items are enclosed by braces, one, and only one of these elements must be selected.

Definitions

d ::= A single ASCII numeric character, 0-9.

n ::= A single ASCII non-zero, numeric character, 1-9.

<NL> ::= Newline or Linefeed (ASCII decimal 10).

<sp> ::= <white space>

<white space> ::= 0 through 32 (decimal) except linefeed (decimal 10). The nominal value is 32 (the space character).

Program Examples

The program examples for commands in the online *HP 54620A/C Programmer's Reference* were written using the HP BASIC programming language. The programs always assume the logic analyzer is at address 7 and the interface is at address 7 for a program address of 707. If a printer is used, it is always assumed to be at address 701.

In these examples, give special attention to the ways in which the command or query can be sent. The way the instrument is set up to respond to a command or query has no bearing on how you send the command or query. That is, you can send the command or query using the long form or short form (if a short form exists for that command). You can send the command or query using uppercase (capital) letters or lowercase (small) letters. Also, the data can be sent using almost any form you wish. If you are sending a time base range value of 100 ms, it can be sent using a decimal (.1), or an exponential (1e-1 or 1.0E-1), or a suffix (100 ms or 100MS).

The following examples let you send any of these commands to set the sweep speed to 100 ms:

- Commands in long form using the decimal format:

```
OUTPUT 707;":TIMEBASE:RANGE .1"
```

- Commands in short form using an exponential format:

```
OUTPUT 707;":TIM:RANG 1E-1"
```

- Commands using lowercase letters, short forms, and a suffix:

```
OUTPUT 707;":tim:rang 100 ms"
```

Including the Colon Is Optional

In these examples, including the colon as the first character of the command is optional. The space between RANGE and the argument is required.

6

Status Reporting

Status Reporting

IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting on the interface. There are also instrument-defined structures and bits.

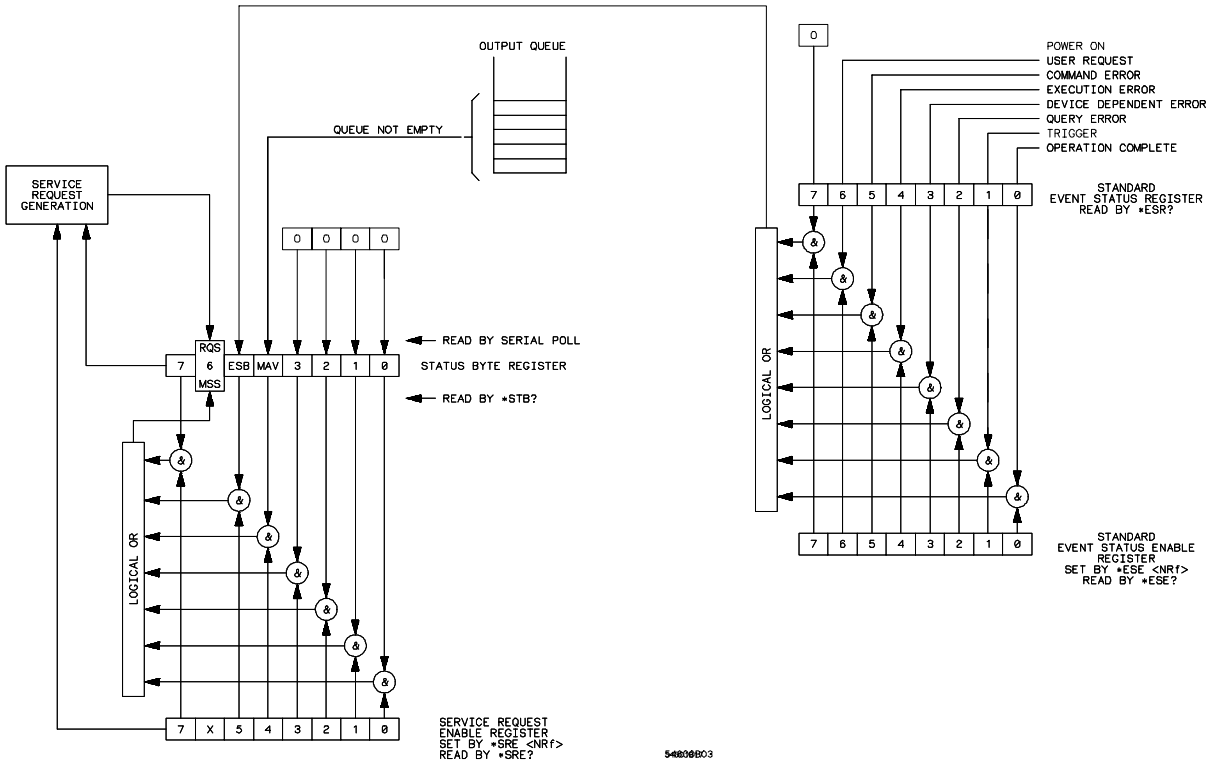
Bits in the status byte act as summary bits for data structures that reside behind them.

- For queues, the summary bit is set if the queue is not empty.
- For registers, the summary bit is set if any enabled bit in the event register is set.
- Events are enabled with the corresponding event enable register.

Events captured by an event register remain set until the register is read or cleared. Registers are read with their associated commands.

The *CLS command clears all event registers and all queues except the output queue. If *CLS is sent immediately following a program message terminator, the output queue is also cleared.

Figure 4



Status Reporting Data Structures

Bit Definitions

MAV—message available. Indicates whether there is a response in the output queue.

ESB—event status bit. Indicates if any of the conditions in the Standard Event Status Register are set and enabled.

MSS—master summary status. Indicates whether the device has a reason for requesting service. This bit is returned for the *STB? query.

RQS—request service. Indicates whether the device is requesting service. This bit is returned during a serial poll. RQS is set to 0 after it is read via a serial poll (MSS is not reset by *STB?).

URQ— user request. Indicates whether a front-panel key has been pressed.

CME— command error. Indicates whether the parser detected an error.

EXE — execution error. Indicates whether a parameter was out of range, or was inconsistent with the current settings.

DDE— device specific error. Indicates whether the device was unable to complete an operation for device dependent reasons.

QYE — query error. Indicates whether the protocol for queries has been violated.

RQC— request control. Indicates whether the device is requesting control. The logic analyzer never requests control.

OPC— operation complete. Indicates whether the device has completed all pending operations.

TRG—trigger. Indicates whether a trigger has been received.

Operation complete (*OPC)

The IEEE 488.2 structure provides one technique that you can use to determine whether any operation is finished. The *OPC command, when sent to the instrument after the operation of interest, sets the OPC bit in the Standard Event Status Register when all pending device operations have finished. If the OPC and RQS bits have been enabled, a service request is generated.

```
OUTPUT 707;"*SRE 32 ; *ESE 1"       !enables an OPC service request
OUTPUT 707;":DIG CHAN1 ; *OPC"      !initiates data acquisition,
                                    !and
                                    !generates an SRQ when the
                                    !acquisition is complete
```

Trigger Bit (TER)

The Trigger (TER) bit indicates whether the device has received a trigger. The TRG event register will remain set after receiving a trigger until it is cleared either by reading it or by using the *CLS command. If your application needs to detect multiple triggers, the TER event register must be cleared after each one.

If you are using the Service Request to interrupt a program or controller operation when the trigger bit is set, you must clear the event register each time it has been set.

```
OUTPUT 707;"*SRE 32"   ! enables event status register.
                       ! the next trigger will generate an SRQ.
OUTPUT 707;"*ESE 2"    ! enables event status register
OUTPUT 707;":TER?"     ! queries the TRG event register, thus
ENTER 707;A$           ! clearing it.
                       ! the next trigger can now generate an
                       ! SRQ
```

Status Byte

If the device is requesting service (RQS set), and the controller serial polls the device, the RQS bit is cleared. The MSS bit (read with *STB?) is not cleared by reading it. The entire status byte is not cleared when read, but the RQS bit is cleared.

Serial poll

This logic analyzer supports the IEEE 488.1 serial poll feature. When a serial poll is requested of the instrument, the RQS bit is returned on bit 6 of the status byte.

Using Serial Poll

The service request can be used by conducting a serial poll of all instruments on the bus. For this procedure, assume that there are two instruments on the bus: a logic analyzer at address 7 and a printer at address 1. It is assumed that you are operating on Interface Select Code 7.

The program command for serial poll using HP BASIC is Stat=SPOLL(707). The address 707 is the address of the logic analyzer in this example. The command for checking the printer is Stat=SPOLL(701) because the address of that instrument is 01 on bus address 7. This command reads the contents of the HP-IB Status Register into the variable called Stat. At that time bit 6 of the variable Stat can be tested to see if it is set (bit 6=1).

You can conduct the serial poll operation using the following steps:

1. Enable interrupts on the bus. This allows the controller to "see" the SRQ line.
2. If the SRQ line is high (an instrument is requesting service) then check the instrument at address 1 to see if bit 6 of its status register is high.
3. Disable interrupts on the bus.
4. To check whether bit 6 of an instrument's status register is high, use the following command line.
   ```
   IF BIT (Stat, 6) then
   ```
5. If bit 6 of the instrument at address 1 is not high, check the instrument at address 7 to see if bit 6 of its status register is high.
6. As soon as the instrument with status bit 6 high is found, check the rest of the status bits to determine what is required.

The SPOLL(707) command causes much more activity on the bus than simply reading the register. This command:

- Clears the bus
- Automatically addresses the talker and listener
- Sends SPE (serial poll enable) and SPD (serial poll disable) bus commands
- Reads the data

For more information about serial poll, refer to your controller manual and programming language reference manuals.

After the serial poll is completed, if the the RQS bit in the logic analyzer Status Byte Register was set, it is reset. Once a bit in the Status Byte Register is set, it remains set until the status is cleared with a *CLS command, or until the instrument is reset. If these bits are not reset, they cannot generate another SRQ.

7

Installing and Using the Programmer's Reference

Installing and Using the Programmer's Reference

The *HP 54620A/C Programmer's Reference* is supplied as an online help file readable with the Microsoft Windows 3.1 help viewer. A second diskette contains the help file as an ASCII text file and three sample programs for the HP 54620A/C.

This chapter:

- Describes how to install the help file on your system.
- Discusses the text and program files.
- Explains how you can get the programs and help file via the Internet.

To install the help file under Microsoft Windows

The help file requires Microsoft Windows 3.1 and MS-DOS 3.3 or greater running on an IBM-compatible PC. The file uses the Microsoft Windows help viewer, WINHELP.EXE.

1. Start your PC and start Microsoft Windows.
2. Insert the 3.5" floppy disk labeled "HP 54620 Pgmr's Reference with Example Programs for Windows 3.1 Applications" into the appropriate diskette drive (A: or B:) of your PC.
3. Select **`File | Run`** from the Program Manager, then type in the following:
   ```
   <drive>:\install
   ```
   where **`<drive>`** is either **`A:`** or **`B:`**.
4. Follow the onscreen instructions to complete the installation.

   The installer copies the help file to a directory named **`\hp54620`**. You can choose a different directory if desired. It also creates a Program Manager group and icon that you can use to open the help file with the Microsoft Windows help viewer.

To use the help text and example program files

The help file is available as an ASCII text file that can be browsed with a text editor or text search tools. Also, there are sample programs (in ASCII text format) that show how to use the HP 54620A/C commands with the analyzer.

- **Copy the help file text or sample programs from the diskette labeled "HP 54620 Pgmr's Reference with Example Programs for Non-Windows Applications" to your system for viewing.**

  See the file README.TXT on the diskette for more information about the diskette contents.

To get updated help and program files via the Internet

The latest versions of the help and example program files are available via anonymous ftp. You must have a connection to the Internet and have ftp software.

1. Log on to your Internet service.
2. Connect to host col.hp.com using anonymous ftp.

   A sample set of commands might be as follows:

   ```
   $ ftp col.hp.com
   Name: anonymous
   Password: myname@mycompany.com
   ```

3. Change to the directory containing HP 54620A/C files.

   ```
   ftp> cd dist/hp54620
   ```

4. Get sample programs or updated help files from the directory as desired.

   For example, if you want the latest version of the HP 54620A/C Programmer's Reference online help file, you set the transfer mode to binary and get the file:

   ```
   ftp> binary
   ftp> get hp54620.hlp
   ```

   Check the README file for more information about the files in this directory.

   If you have trouble making the connection, or need more information on ftp, see your network administrator.

To start the help file

- To open the help file under Microsoft Windows, double-click the "HP 54620 Help" icon in the "HP 54620 Prog Ref" program group in the Program Manager.

  The help file requires the program WINHELP.EXE for Microsoft Windows 3.1. The properties for the Program Manager icon are set to expect this file in the Windows directory.

To navigate through the help file

- **Navigate through the help file by clicking on highlighted text and buttons.**

  See your Microsoft Windows documentation for more information, or select `Help | How to Use Help` in the Help window.

Programmer's Quick Reference

Introduction

The Programmer's Quick Reference provides the commands and queries with their corresponding arguments and returned formats for the HP 54620A/C Logic Analyzer. The arguments for each command list the minimum argument required. The part of the command or query listed in uppercase letters refers to the short form of that command or query. The long form is the combination of the uppercase and lowercase letters. Any optional parameters are listed at the end of each parameter listing.

Conventions

The following conventions are used in this quick reference:

< >	Indicates that words or characters enclosed in angle brackets symbolize a program code parameter or an HP-IB command.
::= "is defined as."	<A>::= <B> indicates that <A> can be replaced by <B> in any statement containing <A>.
| "or"	Indicates a choice of one element from a list. For example, <A> | <B> indicates <A> or <B>, but not both.
...	Indicates that the element preceding the ellipses may be repeated one or more times.
[ ]	Indicates that the bracketed items are optional.
{ }	Indicates that when items are enclosed by braces, one, and only one of the elements may be selected.
{N,..,P}	Indicates the selection of one integer between N and P inclusive.

Suffix multipliers

The following suffix multipliers are available for arguments:

EX :: = 1E18	M :: = 1E-3
PE :: = 1E15	U :: = 1E-6
T :: = 1E12	N :: = 1E-9
G :: = 1E9	P :: = 1E-12
MA :: = 1E6	F :: = 1E-15
K :: = 1E3	A :: = 1E-18

For more information regarding specific commands or queries, please refer to the online *HP 54620A/C Programmer's Reference*.

Commands and Queries

The following tables facilitate easy access to each command and query for the HP 54620A/C Logic Analyzer. The commands and queries are broken into separate categories with each entry alphabetized.

The arguments for each command list the minimum argument required. The part of the command or query listed in uppercase letters refers to the short form of that command or query. The long form is the combination of the uppercase and lowercase letters.

Command	Query	Query Returns
n/a	**:ACQuire:POINts?**	<value> <new line> <value> ::= 16-bit integer in NR1 format, either +2048 or +8192
n/a	**:ACQuire:SETup?**	ACQuire:TYPE <type>;POINts <points> <new line> <type> ::= {NORMal | GLITch} <points> ::={+2048 | +8192}
:ACQuire:TYPE <value> <value> ::= {AUTO | NORMal | GLITch}	:ACQuire:TYPE?	<value> <new line> <value> ::= {AUTO | NORMal | GLITch}
:ASTore	n/a	n/a
:AUToscale	n/a	n/a
:BLANk <source> <source> ::= {LCHANnel0,..,LCHANnel15 | PMEMory1 | PMEMory2}	n/a	n/a
***CLS**	n/a	n/a
:DIGitize	n/a	n/a
:DISPLAY:COLumn <number> <number> ::= 16-bit integer (0 through 63 for HP 54620A) (2 through 63 for HP 54620C)	:DISPLAY:COLumn?	<value> <new line> <value> ::= 16-bit integer in NR1 format (0 through 63 for HP 54620A) (2 through 63 for HP 54620C)
:DISPlay:DATA #800016256 <Binary block> <binary block> ::= 16256 bytes of data in IEEE-488.2 format	:DISPlay:DATA?	#800016256<Binary block> <new line> <binary block> ::= 16256 bytes of data in IEEE-488.2 format
:DISPlay:GRID <value> <value> ::= {ON | OFF | FRAMe | FULL}	:DISPlay:GRID?	<value><new line> <value> ::= {OFF | FRAMe | FULL}
:DISPlay:INVerse <value> <value> ::={ON | OFF} ON ::= inverse video display OFF ::= normal display	:DISPlay:INVerse?	<value> <new line> <value> ::= {ON | OFF} ON ::= inverse video display is on OFF ::= inverse video display is off
:DISPlay:LABel <value> <value> ::= {ON | OFF}	:DISPlay:LABel?	<value> <new line> <value> ::= {ON | OFF}

Command	Query	Query Returns
:DISPlay:LABList #80000525 <label string> <label string> ::= a time-ordered list of 75 labels. Each label name is six characters followed by a comma.	:DISPlay:LABList?	#80000525<label string> <new line> 525 ::= the size in data bytes of the label string. <label string> ::= a time-ordered list of 75 labels. Each label name is six characters followed by a comma.
:DISPlay:LINE <string> <string> ::= ascii text	n/a	n/a
:DISPlay:ORDer <order list> ::= one 16-bit integer in NR1 format for each channel on the screen. Adjacent NR1 numerals need to be separated by a comma.	:DISPlay:ORDer?	<value> <new line> <value> ::= Sixteen 16-bit integers in NR1 format separated by commas.
:DISPlay:PIXel <x coordinate>, < y coordinate>, <intensity> <x coordinate> ::= 0 to 500 <y coordinate> ::= 0 to 275 <intensity> = 0 to clear pixel, 1 for half-bright, 2 for full-bright.	:DISPlay:PIXel? <x coordinate>, <y coordinate>	<intensity> <new line> <intensity> ::= integer in a NR1 format 0 for pixel off 1 for pixel with half-bright on 2 for pixel with full-bright on
:DISPlay:PALette <arg> (HP 54620C only) <arg> ::= 16-bit integer in NR1 format	:DISPlay:PALette?	<arg> ::= 16-bit integer in NR1 format This command is used by the HP 54620C. These values match the front-panel order: 0 Default 1 Alternate 1 2 Alternate 2 3 Alternate 3 4 Inverse 1 5 Inverse 2 6 Monochrome
n/a	**:DISPlay: POSition?**	#800000064 <order string> <new line> 64 ::= the number of bytes in the string <order string> ::= wave height ch. 0, bottom pixel location ch. 0 (main screen), wave height ch. 1, bottom pixel location ch. 1 (main screen), ... wave height ch. 15, bottom pixel location ch. 15 (main screen), wave height ch. 0, bottom pixel location ch. 0 (delay window), wave height ch. 1, bottom pixel location ch. 1 (delay window), ... wave height ch. 15, bottom pixel location ch. 15 (delay window).
:DISPlay:ROW <row number> <row number> ::= a 16-bit integer in a NR1 format (1 - 20)	:DISPlay:ROW?	<row number> <new line> <row number> ::= a 16-bit integer in NR1 format
n/a	**:DISPlay:SETup?**	:DISPlay:ROW <row>; COLumn <column>; INVerse <inverse>; GRID <grid>;SOURce <source> :LABel <label> <newline> <row> ::= {1,..,20} <column> ::= {0,..,63} (HP 54620A) <column> ::= {2,..,63} (HP 54620C) <inverse> ::= {ON | OFF} <grid> ::= {FRAMe | FULL | OFF} <source> ::= {PMEMory1 | PMEMory2} <label> ::= {ON | OFF}
:DISPlay:SOURce <value> <value> ::= {PMEMory1 | PMEMory2} PMEMory1 ::= pixel memory 1 PMEMory2 ::= pixel memory 2	:DISPlay:SOURce?	<value> <new line> <value> ::= {PMEMory1 | PMEMory2}

Command	Query	Query Returns
:DISPlay:TEXT BLANk	n/a	n/a
:ERASe [value] [value] ::= {PMEMory 1 | PMEMory2}	n/a	n/a
***ESE**	*ESE?	<mask> <new line> <mask> ::= integer in NR1 format
n/a	***ESR?**	<status> <new line> <status> ::= 16-bit integer in NR1 format
n/a	***IDN?**	HEWLETT-PACKARD, <instrument>, 0, <X.XX.XX> <new line> <instrument> ::= 54620A or 54620C (color logic analyzer) <X.XX.XX> ::= the software revision number string, for example, A.01.00
:LCHannel:ACTivity (clears the cumulative edge register)	:LCHannel:ACTivity?	<edges>, <levels> <newline> <edges> ::= presence of edges (32-bit integer in NR1 format) <levels> ::= logical highs or lows (32-bit integer in NR1 format)
:LCHannel:COLor <source>,<color> (HP 54620C only) <source> ::= {LCHANnel0,..,LCHANnel15} <color> ::= {COLor1,..,COLor4}	:LCHannel:COLor <source>	<source> ::= {LCHANnel0,..,LCHANnel15} <color> ::= {COL1,..,COL4}
:LCHannel:LABel <sourcetext>, <string> <sourcetext> ::= {LCHAN0,..,LCHAN15} <string> ::= normal ascii text (6 chars. maximum)	:LCHannel:LABel?<sourcetext>	<string><new line> <string> ::= normal ascii text
:LCHannel:THReshold <channel group>, <threshold type>[, <value>] <channel group> ::= {LCHAN0_7 | LCHAN8_15 | TRIG_IN} <threshold type> :::= {CMOS | ECL | TTL | USERdef} < value> ::= voltage for USERdef (float 32 NR3) [Voltype] [Voltype] ::= V, MV (-3), UV (-6)	:LCHannel:THReshold? <channel group>	<threshold type> [,<value>] <new line> <threshold type> ::= {CMOS | ECL | TTL | USERdef} <value> ::= voltage for USERdef (float 32 48:NR3)
n/a	***LRN?**	#80000500 <learn string> <new line> <learn string> ::= 500 bytes of data
n/a	**:MEASure:ALL?**	<value list> <new line> <value list> ::= frequency, period, duty cycle, positive width, negative width, setup, hold, delay (all are in floating point number NR3 format)
:MEASure:DEFine:DELay <chan1>, <edge1>, <chan2>, <edge2> <chan1> ::= {LCHANnel0,..,LCHANnel15} <chan2> ::= {LCHANnel0,..,LCHANnel15} <edge1> ::= {RISing, FALLing} <edge2> ::= {RISing, FALLing}	:MEASure:DEFine:DELay?	<chan1>, <edge1>, <chan2>, <edge2> <new line> <chan1> ::= {LCHANnel0,..,LCHANnel15} <chan2> ::= {LCHANnel0,..,LCHANnel15} <edge1> ::= {RISing, FALLing} <edge2> ::= {RISing, FALLing}
:MEASure:DEFine:SETup <chan1>, <chan2>, <edge2> <chan1> ::= {LCHANnel0,..,LCHANnel15} <chan2> ::= {LCHANnel0,..,LCHANnel15} <edge2> ::= {RISing, FALLing}	:MEASure:DEFine:SETup?	<chan1>, <chan2>, <edge2><new line> <chan1> ::= {LCHANnel0,..,LCHANnel15} <chan2> ::= {LCHANnel0,..,LCHANnel15} <edge2> ::= {RISing, FALLing}

Command	Query	Query Returns
:MEASure:DEFine:HOLD <chan1>, <edge1>, <chan2> <chan1> ::= {LCHANnel0,..,LCHANnel15} <chan2> ::= {LCHANnel0,..,LCHANnel15} <edge1> ::= {RISing, FALLing}	:MEASure:DEFine:HOLD?	<chan1>, <edge1>, <chan2><new line> <chan1> ::= {LCHANnel0,..,LCHANnel15} <chan2> ::= {LCHANnel0,..,LCHANnel15} <edge1> ::= {RISing, FALLing}
:MEASure:DELay	:MEASure:DELay?	<value><new line> <value> ::= floating point number in NR3 format
:MEASure:DUTYcycle	:MEASure: DUTYcycle?	<value><new line> <value> ::= floating point number in NR3 format
:MEASure:FREQuency	:MEASure:FREQuency?	<value><new line> <value> ::= floating point number in NR3 format
:MEASure:HOLD	:MEASure:HOLD?	<value><new line> <value> ::= floating point number in NR3 format
:MEASure:NWIDth	:MEASure:NWIDth?	<value><new line> <value> ::= floating point number in NR3 format
:MEASure:PERiod	:MEASure:PERiod?	<value><new line> <value> ::= floating point number in NR3 format
:MEASure:PWIDth	:MEASure:PWIDth?	<value><new line> <value> ::= floating point number in NR3 format
:MEASure:SCRatch	n/a	n/a
:MEASure:SETup	:MEASure:SETup?	<value><new line> <value> ::= floating point number in NR3 format
:MEASure:SHOW <value> <value> ::= {ON | OFF}	:MEASure:SHOW?	<value> <new line> <value> ::= {ON | OFF}
:MEASure:SOURce <source text> <source text> ::= {LCHANnel0,..,LCHANnel15}	:MEASure:SOURce?	<source text> <new line> <source text> ::= {LCHANnel0,..,LCHANnel15}
n/a	**:MEASure:TBINary? <value>** <value> ::= {STARt | STOP}	<string> <new line> <string> ::= ascii text (19 characters long)
n/a	**:MEASure:TDELta?**	<value> <new line> <value> ::= difference between start and stop markers (floating point number in NR3 format)
:MEASure:TSTArt <value> [suffix] <value> ::= floating point number in NR3 format [suffix] ::= {S | MS (-3) | US (-6) | NS (-9) | PS (-12)}	:MEASure:TSTArt?	<value> <new line> <value> ::= floating point number in NR3 format
:MEASure:TSTOp <value> [suffix] <value> ::= floating point number in NR3 format [suffix] ::= {S | MS (-3) | US (-6) | NS (-9) | PS (-12)}	:MEASure:TSTOp?	<value><new line> <value> ::= floating point number in NR3 format
n/a	**:MEASure:THEXadecimal? <value>** <value> ::= {STARt | STOP}	<string > <new line> <string> ::= ascii text
:MENU <selection> <selection> ::= 16-bit integer	:MENU?	<value> <new line> <value> ::= 16-bit integer in NR1 format
:MERGe <pixel memory text> <pixel memory text> ::= {PMEMory1 | PMEMory2}	n/a	n/a
***OPC**	*OPC?	<value> <new line> <value> ::= "1" as a16-bit integer in NR1 format
n/a	***OPT?**	0 <new line>

Command	Query	Query Returns
n/a	**:PRINt? [enhancement]** [enhancement] ::= [HIRes[,PARallel[,PCLcolor]]] HIRes ::= contains both half-bright and full-bright display information. PARallel ::= print output from parallel port. PCLcolor ::= basic color output using CMY driver	<print data>
***RCL <value>** <value> ::= {1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 }	n/a	n/a
***RST**	n/a	n/a
:RUN	n/a	n/a
:RUNSingle	n/a	n/a
***SAV <value>** <value> ::= {1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 }	n/a	n/a
***SRE <mask>** <mask> ::= values defined in table below: Bit 7 Weight 128 Enables: Not Used Bit 6 Weight 64 Enables: RQS - Request Service Bit 5 Weight 32 Enables: ESB - Event Status Bit Bit 4 Weight 16 Enables: MAV - Message Available Bit 3 Weight 8 Enables: Not used Bit 2 Weight 4 Enables: Not used Bit 1 Weight 2 Enables: Not used Bit 0 Weight 1 Enables: Not used	*SRE?	<mask> <new line> <mask> ::= {0,..,255} (16-bit integer in NR1 format)
n/a	**:STATus? <source>** <source> ::= {LCHANnel0,..,LCHANnel15 | PMEMory1 | PMEMory2}	<value> <new line> <value> ::= {ON | OFF}
n/a	***STB?**	<value> <new line> <value> ::= values defined in table below Bit 7 Weight 128 Name Condition: ---- Bit 6 Weight 64 Name Condition: RQS/MS Bit 5 Weight 32 Name Condition: ESB Bit 4 Weight 16 Name Condition: MAV Bit 3 Weight 8 Name Condition: ---- Bit 2 Weight 4 Name Condition: ---- Bit 1 Weight 2 Name Condition: ---- Bit 0 Weight 1 Name Condition: ----
:STOP	n/a	n/a
:SYSTem:DSP <normal ascii text> <normal ascii text> ::= quoted message string	n/a	n/a
n/a	**:SYSTem:ERRor?**	<value> <new line> <value> ::= 16-bit integer in NR1 format
:SYSTem:KEY <value> <value> ::= 16-bit integer in NR1 format	:SYSTem:KEY?	<keycode><new line> <keycode> ::= 16-bit integer in NR1 format.
:SYSTem:LOCK <value> <value> ::= {ON | OFF}	:SYSTem:LOCK?	<value> <new line> <value> ::= {ON | OFF}
:SYSTem:SETup <setup> <setup> ::= #800000500 <setup data string> <setup data string> ::= binary block data	:SYSTem:SETup?	<setup> <new line> <setup> ::= #800000500 <setup data string> <setup data string> ::= binary block data

Command	Query	Query Returns
n/a	**:TER?**	<value> <new line> <value> ::= {1 | 0} <16-bit integer in NR1 format> (0 = no trigger; 1 = trigger occurred)
TIMebase:DELay <delay value> <delay value> ::= floating point number [suffix] [suffix] ::= {S | MS (-3) | US (-6) | NS (-9) | PS (-12)}	:TIMebase:DELay?	<value><new line> <value> ::= floating point number in NR3 format
TIMebase:MODE <value> <value> ::= {NORMal | DELayed}	:TIMebase:MODE?	<value> <new line> <value> ::= {NORMal | DELayed}
:TIMebase:RANGe <range value> [suffix] <range value> ::= 50 ns to 50 s in NR3 format [suffix] ::= { S | MS (-3) | US (-6) | NS (-9) | PS (-12)}	:TIMebase:RANGe?	<range value> <new line> <range value> ::= 50 ns to 50 s (floating point number in NR3 format)
:TIMebase:REFerence <value> <value> ::= {LEFT | CENTer | RIGHt}	:TIMebase:REFerence?	<value> <new line> <value> ::= {LEFT | CENTer | RIGHt}
n/a	**:TIMebase:SETup?**	<current time base setup> ::= TIMebase:MODE <mode>; RANGe <range value>; DELay <delay>; REFerence <reference>; VERNier <vernier> <new line> <mode> ::= {NORMal | DELayed} <delay> ::= time from trigger to delay reference in seconds NR3 format <range value> ::= 50 ns to 50 s in NR3 format <reference> ::= {LEFT | CENTer | RIGHt} <vernier> ::= {ON | OFF}
:TIMebase:VERNier <value> <value>:::= {ON | OFF}	:TIMebase:VERNier?	<value> <new line> <value> ::= {ON | OFF}
:TRIGger:ADVanced:DURation <duration> **[suffix]** <duration> ::= 32-bit floating point number (16 ns - 8,388.60 seconds) in NR3 format [suffix] ::= {S | MS (-3) | US (-6) | NS (-9) | or PS (-12)}	:TRIGger:ADVanced:DURation?	<duration> <new line> <duration> ::= 32-bit floating point number in NR3 format
:TRIGger:ADVanced:EDGE1 <rising value>, **<falling value>** <rising value> ::= 32-bit integer in NR1 format or <string> <falling value> ::= 32-bit integer in NR1 format or <string> <string> ::= 0xnnnnn n ::= {0 | 1 | 2 | 4 | 8}	:TRIGger:ADVanced:EDGE1?	<rising value>, <falling value> <new line> <rising value> ::= 32-bit integer in NR1 format <falling value> ::= 32-bit integer in NR1 format
:TRIGger:ADVanced:EDGE2 <rising value> , **<falling value>** <rising value> ::= 32-bit integer in NR1 format or <string> <falling value> ::= 32-bit integer in NR1 format or <string> <string> ::= 0xnnnnn n ::= {0 | 1 | 2 | .... 8 | 9 | A | B | C | D | E | F}	:TRIGger:ADVanced:EDGE2?	<rising value>, <falling value> <new line> <rising value> ::= 32-bit integer in NR1 format <falling value> ::= 32-bit integer in NR1 format
:TRIGger:ADVanced:OCCurrence <occurrence> <occurrence> ::= 32-bit integer in NR1 format or <string> <string> ::= 0xnnnnn n ::= {0 | 1 | 2 | .... 8 | 9 | A | B | C | D | E | F}	:TRIGger:ADVanced OCCurrence?	<occurrence> <new line> <occurrence> ::= 32-bit integer in NR1 format

Command	Query	Query Returns
:TRIGger:ADVanced:OPERator <operator> <operator> ::= {AND | OR | THEN | ENTer | EXIT | OCCurrence | GREaterthan | LESSthan}	:TRIGger:ADVanced:OPERator?	<operator> <new line> <operator> ::= {AND | OR | THEN | ENTer | EXIT | OCCurrence | GREaterthan | LESSthan}
:TRIGger:ADVanced:PATtern1 <value>, <mask> <value> ::= 32-bit integer in NR1 format or <string> <mask> ::= 32-bit integer in NR1 format or <string> <string> ::= 0xnnnnn n ::= {0 | 1 | 2 | .... 8 | 9 | A | B | C | D | E | F}	:TRIGger:ADVanced:PATtern1?	<value>, <mask> <new line> <value> ::= 32-bit integer in NR1 format <mask> ::= 32-bit integer in NR1 format
:TRIGger:ADVanced:PATtern2 <value>, <mask> <value> ::= 32-bit integer in NR1 format or <string> <mask> ::= 32-bit integer in NR1 format or <string> <string> ::= 0xnnnnn n ::= {0 | 1 | 2 | .... 8 | 9 | A | B | C | D | E | F}	:TRIGger:ADVanced:PATtern2?	<value>, <mask> <new line> <value> ::= 32-bit integer in NR1 format <mask> ::= 32-bit integer in NR1 format
:TRIGger:ADVanced:SOURce1 <source> <source> ::= {PATtern1 | NPATtern1 | PATtern2 | NPATtern2 | EDGE1 | EDGE2 | PAT1ANDEDGE1 | PAT2ANDEDGE2}	:TRIGger:ADVanced:SOURce1?	<source> <new line> <source> ::= {PATtern1 | NPATtern1 | PATtern2 | NPATtern2 | EDGE1 | EDGE2 | PAT1ANDEDGE1 | PAT2ANDEDGE2}
:TRIGger:ADVanced:SOURce2 <source> <source> ::= {PATtern1 | NPATtern1 | PATtern2 | NPATtern2 | EDGE1 | EDGE2 | PAT1ANDEDGE1 | PAT2ANDEDGE2}	:TRIGger:ADVanced:SOURce2?	<source> <new line> <source> ::= {PATtern1 | NPATtern1 | PATtern2 | NPATtern2 | EDGE1 | EDGE2 | PAT1ANDEDGE1 | PAT2ANDEDGE2}
:TRIGger:EDGE <trigger source>, <trigger edge> <trigger source> ::= {LCHANnel0,..,LCHANnel15 | EXTernal | TRIG_IN} <trigger edge> ::= {RISing | FALLing | EITHeredge}	:TRIGger:EDGE?	<trigger source>,<trigger edge> <new line> <trigger source> ::= {LCHANnel0,.., LCHANnel15 | EXTernal} <trigger edge> ::= {RISing | FALLing | EITHeredge}
:TRIGger:MODE <trigger mode> <trigger mode> ::= {NORMal | AUTO}	:TRIGger:MODE?	<trigger mode> <new line> <trigger mode> ::= {NORMal | AUTO}
:TRIGger: PATTern <value>, <mask> [, <source>, <edge>] <value> ::= 32-bit integer in NR1 format or <string> <mask> ::= 32-bit integer in NR1 format or <string> <string> ::= 0xnnnnn n ::= {0 | 1 | 2 | .... 8 | 9 | A | B | C | D | E | F} <source> ::= {LCHANnel0,..,LCHANnel15 | EXTernal | TRIG_IN} <edge> ::= {RISing | FALLing | EITHeredge}	:TRIGger:PATTern?	<value>, <mask> [, <source>, <edge>] <new line> <value> ::= 32-bit integer in NR1 format <mask> ::= 32-bit integer in NR1 format <string> ::= 0xnnnnn n ::= {0 | 1 | 2 | .... 8 | 9 | A | B | C | D | E | F} <source> ::= {LCHANnel0,..,LCHANnel15 | EXTernal} <edge> ::= {RISing | FALLing | EITHeredge}

Command	Query	Query Returns
n/a	**:TRIGger:SETup?**	TRIGger:MODE <mode>;TYPE <type>;EDGE <channel>,<edge>;PATTern <value>, <mask> [,<channel>,<edge>];ADVanced:PATtern1 <value>,<mask>;PATtern2 <value>,<mask>;EDGE1 <rising value>, <falling value>;EDGE2 <rising value>, <falling value>;OPERator <operator>; SOURce1 <source>;SOURce2 <source>; DURation <duration>;OCCurrence <occurrence> <new line> <mode> ::= {NORMal | AUTO} <type> ::= {EDGE | PATTern |ADVanced} <channel> ::= {LCHANnel0,..,LCHANnel15 | EXTernal} <edge> ::= {RISing | FALLing | EITHeredge} <value> ::= 32-bit integer in NR1 format <mask> ::= 32-bit integer in NR1 format <rising value> ::= 32-bit integer in NR1 format <falling value> ::= 32-bit integer in NR1 format <operator> ::= {AND | OR | THEN | ENTer | EXIT | OCCurrence | GREaterthan | LESSthan} <source> ::= {PATtern1 | NPATtern1 | PATtern2 | NPATtern2 | EDGE1 | EDGE2 | PAT1ANDEDGE1 | PAT2ANDEDGE2} <duration> ::= 32-bit floating point number in NR3 format <occurrence> ::= 32-bit integer in NR1 format
TRIGger:TYPE <trigger type> <trigger type> ::= {EDGE | PATTern | ADVanced}	:TRIGger:TYPE?	<trigger type> <new line> <trigger type> ::= {EDGE | PATTern | ADVanced}
***TRG**	n/a	n/a
n/a	***TST?**	<result> <new line> <result> ::= 16-bit integer in NR1 format 0 ::= Pass Other ::= Fail
:VIEW <sourcetext> <source text> ::= {LCHANnel0,..,LCHANnel15 | PMEMory1 | PMEMory2}	n/a	n/a
***WAI**	n/a	n/a
:WAVeform:BYTeorder <value> <value> ::= {MSBFirst | LSBFirst}	:WAVeform:BYTeorder?	<value> <new line> <value> ::= {MSBFirst | LSBFirst}
n/a	**:WAVeform:DATA?**	<binary block length bytes>, <binary data> <newline> <binary block length bytes> ::= #8000ddddd <ddddd> ::= {16384 | 08192 | 04096}
:WAVeform:FORMat <value> <value> ::= {BYTE | WORD}	:WAVeform:FORMat?	<value> <new line> <value> ::= {BYTE | WORD}
:WAVeform:POINts <value> <value> ::= 16-bit integer in NR1 format	:WAVeform:POINts?	<value> <new line> <value> ::= 16-bit integer in NR1 format

Command	Query	Query Returns
n/a	**:WAVeform:PREamble?**	<waveform format>, <waveform type>, <points>, <count>, <xincrement>, <xorigin>, <xreference>, <yincrement>, <yorigin>, <yreference> <new line> <waveform format> ::= 16-bit integer in NR1 format, 0 = BYTE, 1 = WORD <waveform type> ::= 16-bit integer in NR1 format, 0 = GLITch, 1 = NORMal <points> ::= 16-bit integer in NR1 format (an even divisor of the number of acquisition points, which is 2K or 8K) <count> ::= 16-bit integer in NR1 format (always 1) <xincrement> ::= 32-bit floating point number in NR3 format (sample rate * (acq points / wave points) <xorigin> ::= 32-bit floating point number in NR3 format (main time base delay - (trace point in buffer (0-2K or 0-8K)) * sample rate) <xreference> ::= 16-bit integer in NR1 format (always 0) <yincrement> ::= 32-bit floating point number in NR3 format (always 0.0) <yorigin> ::= 32-bit floating point number in NR3 format (always 0.0) <yreference> ::= 16-bit integer in NR1 format (always 0)
:WAVeform:SOURce <source> <source> ::= {LCHAN0_7 | LCHAN8_15 | LCHAN0_15}	:WAVeform:SOURce?	<source> <newline> <source> ::= {LCHAN0_7 | LCHAN8_15 | LCHAN0_15}
n/a	**:WAVeform:TYPE?**	<value> <new line> <value> {NORMal | GLITch}
n/a	**:WAVeform:XINCrement?**	<value> <new line> <value> ::= floating point number in NR3 format
n/a	**:WAVeform:XORigin?**	<value> <new line> <value> ::= floating point number in NR3 format
n/a	**:WAVeform:XREFerence?**	<value> <new line> <value> ::=16-bit integer in NR1 format (always 0)
n/a	**:WAVeform:YINCrement?**	<value> <new line> <value> ::= floating point number in NR3 format (always 0.0)
n/a	**:WAVeform:YORigin?**	<value> <new line> <value> ::= floating point number in NR3 format (always 0.0)
n/a	**:WAVeform:YREFerence?**	<value> <new line> <value> ::=16-bit integer in NR1 format (always 0)

Index

Document Warranty

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

Hewlett-Packard shall not be liable for errors contained herein or for damages in connection with the furnishing, performance, or use of this material.

Safety

This apparatus has been designed and tested in accordance with IEC Publication 348, Safety Requirements for Measuring Apparatus, and has been supplied in a safe condition. This is a Safety Class I instrument (provided with terminal for protective earthing). Before applying power, verify that the correct safety precautions are taken (see the following warnings). In addition, note the external markings on the instrument that are described under "Safety Symbols."

Warning

- Before turning on the instrument, you must connect the protective earth terminal of the instrument to the protective conductor of the (mains) power cord. The mains plug shall only be inserted in a socket outlet provided with a protective earth contact. You must not negate the protective action by using an extension cord (power cable) without a protective conductor (grounding). Grounding one conductor of a two-conductor outlet is not sufficient protection.
- Only fuses with the required rated current, voltage, and specified type (normal blow, time delay, etc.) should be used. Do not use repaired fuses or short-circuited fuseholders. To do so could cause a shock of fire hazard.
- Service instructions are for trained service personnel. To avoid dangerous electric shock, do not perform any service unless qualified to do so. Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.
- If you energize this instrument by an auto transformer (for voltage reduction), make sure the common terminal is connected to the earth terminal of the power source.
- Whenever it is likely that the ground protection is impaired, you must make the instrument inoperative and secure it against any unintended operation.
- Do not operate the instrument in the presence of flammable gasses or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.
- Do not install substitute parts or perform any unauthorized modification to the instrument.
- Capacitors inside the instrument may retain a charge even if the instrument is disconnected from its source of supply.
- Use caution when exposing or handling the CRT. Handling or replacing the CRT shall be done only by qualified maintenance personnel.

Safety Symbols

Instruction manual symbol: the product is marked with this symbol when it is necessary for you to refer to the instruction manual in order to protect against damage to the product.

Hazardous voltage symbol.

Earth terminal symbol: Used to indicate a circuit common connected to grounded chassis.

WARNING

The Warning sign denotes a hazard. It calls attention to a procedure, practice, or the like, which, if not correctly performed or adhered to, could result in personal injury. Do not proceed beyond a Warning sign until the indicated conditions are fully understood and met.

CAUTION

The Caution sign denotes a hazard. It calls attention to an operating procedure, practice, or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product. Do not proceed beyond a Caution symbol until the indicated conditions are fully understood or met.

Hewlett-Packard
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901

Product Warranty

This Hewlett-Packard product has a warranty against defects in material and workmanship for a period of three years from date of shipment. During the warranty period, Hewlett-Packard Company will, at its option, either repair or replace products that prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Hewlett-Packard.

For products returned to Hewlett-Packard for warranty service, the Buyer shall prepay shipping charges to Hewlett-Packard and Hewlett-Packard shall pay shipping charges to return the product to the Buyer. However, the Buyer shall pay all shipping charges, duties, and taxes for products returned to Hewlett-Packard from another country.

Hewlett-Packard warrants that its software and firmware designated by Hewlett-Packard for use with an instrument will execute its programming instructions when properly installed on that instrument. Hewlett-Packard does not warrant that the operation of the instrument software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by the Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. Hewlett-Packard specifically disclaims the implied warranties of merchantability or fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are the buyer's sole and exclusive remedies. Hewlett-Packard shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Assistance

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales Office.

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology, to the extent allowed by the Institute's calibration facility, and to the calibration facilities of other International Standards Organization members.

About this edition

This is the first edition of the *HP 54620A/C Logic Analyzer Programmer's Guide.*

Publication number 54620-97012

Printed in USA.

Edition dates are as follows:

First edition, October 1995

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by you. The dates on the title page change only when a new edition is published.

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

The following list of pages gives the date of the current edition and of any changed pages to that edition.

All pages original edition