Hewlett-Packard's Logic Analyzer and Personal Computer Programming Series

HEWLETT PACKARD

16500 and HP Vectra/IBM PC Using HP's HP-IB Interface Cards

Contents

Introduction

Today's engineers, whether in manufacturing or research, are becoming more conscious of their productivity. The increasingly competitive global marketplace requires ever shorter design cycles. Product qualification and production test require fast, repeatable procedures. These requirements are causing engineers to rely heavily on computer control of test and measurement instruments.

The personal computer is now commonplace on the design engineer's workbench. A number of software packages allow the designer to model a circuit design and evaluate its performance before any components are assembled. The ability to program a personal computer for use as an instrument controller can build on these modeling and design capabilities. With instrument control capability, the designer can acquire real data from a prototype and compare it with the expected performance based on a model. From this design verification role, production test becomes a natural extension.

The Personal Computer

When IBM introduced the IBM PC/XT, they set the standard for personal computers and small technical computers. That unit was followed by the IBM PC/AT that increased the computing power of those machines. Many manufacturers have followed the pattern of the IBM computers and have produced "compatibles."

This application note describes how the IBM PC/AT, the HP Vectra, and the HP 150 personal computers are programmed to converse with the HP 16500A Logic Analysis System via the HP-IB (IEEE 488). The HP-IB interface cards discussed are the HP 14857A for the HP 150 and the HP 61062A and HP 82990A for both the HP Vectra and the IBM PC. Other HP-IB interface cards are available, but those cards require different software drivers and the information contained in this note does not apply.

Programming Logic Analyzers

There are four basic functions that can be done via the HP-IB:

- program the logic analyzer
- retrieve setup information and measurement results
- pass acquired data to the controller
- send data to the logic analyzer

Of course this is not a complete list of the functions that can be performed by the controller and the logic analyzer. More complex tasks are performed by combining these four basic functions.

This application note is not intended to be all inclusive. It isn't a tutorial on the HP-IB, programming a personal computer, or programming a logic analyzer. It is intended to demonstrate the basic concepts of HP-IB communication and provide sufficient examples to get you started. The programs included in this application note work on the HP 150 or IBM PC/AT (HP Vectra) as written. The user should be familiar with GWBASIC or BASICA and the HP-IB Command Library for MS-DOS that is supplied with the HP-IB interface. The user should also be familiar with the programming information supplied with the logic analyzer.

Although the example programs and the text are based on BASIC, the text and examples are valuable for those programming in other languages. The HP-IB subroutine calls remain the same, and the methods used to pass information to/from the logic analyzer still apply.

This application note specifically addresses programming for the HP 16500A Logic Analysis System. The command syntax and logic analyzer data format are also applicable to the HP 1650A and HP 1651A Logic Analyzers with only minor changes.

Subroutine Calls

A number of subroutine calls from the HP-IB command library are used in the example programs. It is helpful to understand the function of these calls and therefore each should be reviewed. The following calls are used in the programs:

IOCLEAR	IOEOI	IOOUTPUTS
IOENTER	IOEOL	IORESET
IOENTERA	IOMATCH	IOTIMEOUT
IOENTERS	IOOUTPUT	

Command Syntax

To program the Logic Analysis System via the HP-IB, you must have some understanding of the command format and structure expected by the Logic Analysis System. A complete set of rules is contained in the HP-IB and programming sections of the user manual set. A short review is included here.

The Command Message

The Logic Analysis System is remotely programmed on the HP-IB with ASCII data messages. The data messages contain a string of device dependent commands (program commands) and an EOL (end of line) terminator. The general command syntax is:

```
<SUBSYSTEM SELECTOR><COMMAND HEADER><ARGUMENT><EOL>
```

Subsystem Selector

Commands are divided into logical groups that control specific functions within the Logic Analysis System. Before issuing a command to control a function, the desired SUBSYSTEM must first be referenced. This allows identical command headers to be used in more than one function. For example, the command header TMAXIMUM may be used to query (ask) for the maximum time interval between the X and O markers in the oscilloscope module, the logic analyzer module, or the high speed timing module:

Oscilloscope - :MARKER:TMAXIMUM?
Logic Analyzer - :MACHINE1:SLIST:TMAXIMUM?
High Speed Timing - :WAVEFORM:TMAXIMUM?

In these examples, the MARKER, MACHINE, SLIST, and WAVEFORM subsystem selectors are used to specify the usage of the TMAXIMUM command header.

Command Header

The command header is the actual program function you wish to perform. The command headers represent the functions available within a selected subsystem. In the above examples, TMAXIMUM is the command header.

Command headers and subsystem selectors may be any combination of uppercase or lowercase ASCII characters and may be used in either their longform (complete spelling) or shortform. Both of the following examples query the Logic Analyzer module (HP 16510A) for the maximum X to O marker time for machine 1:

longform - :MACHINE1:SLIST:TMAXIMUM?
shortform -:MACH1:SLIS:TMAX?

Programs written in longform are more easily read and understood. The shortform syntax conserves the amount of controller memory required for program storage. The shortform also minimizes the amount of I/O activity required to pass commands and data. Rules for shortform abbreviations are included in the programming manuals for the Logic Analysis System.

Multiple commands may be sent as part of a single string if the command headers are separated from the preceding argument by a semicolon. The total string length cannot exceed the 256 character HP-IB buffer length. If a command string exceeds the maximum allowable length (an EOL or EOI is not received), the instrument will record the error and flush the input buffer without executing any of the commands. This results in the Logic Analysis System not being setup as the programmer expects.

Argument

Numeric argument - some command headers require that the argument be a number. Logic Analysis System menu selection, for example, requires that the desired module and menu selection be expressed numerically. The Logic Analysis System will recognize numbers in real, integer, or scientific notation (exponential) forms.

Alpha argument - for some program commands you must select one argument from a list of several options. The selected argument may be represented by either its longform or its shortform.

EOL Terminator

The EOL terminator may be either a LF (line feed), EOI (end-or-identify), or a combination of the two. Carriage return is not required prior to the LF, and if sent by the controller, it is ignored. EOI is an HP-IB control line, and when used, is asserted simultaneously with the last byte of the data message.

The instructions within a command message are only executed after the EOL is received.

Command/Query

Command headers followed by a "?" (question mark), in place of the normal argument, are queries. Queries instruct the Logic Analysis System to interrogate the current setting of the command header and place the information in the HP-IB output buffer. When the Logic Analysis System is addressed to talk, the contents of the output buffer are passed across the bus.

Sending the command string :MENU? places the current module number and menu number in the output buffer. Addressing the Logic Analysis System to talk (handled by the HP-IB subroutines) passes these two values across the HP-IB to the controller. Queries will be covered in more detail later.

Mainframe Commands

Mainframe commands are the highest command level and the Logic Analysis System executes them as soon as it parses them. These commands do not require a subsystem selector, and can be issued at any time. A mainframe command will not change the current subsystem selector. When a mainframe command finishes execution, the system will return to the previously selected subsystem. Examples of mainframe commands are:

```
CARDCAGE     SELECT
EOI ON/OFF   START/STOP
MENU
```

Selecting A Module

The Logic Analysis System is built on a modular architecture. When commands are sent over the bus to the Logic Analysis System, they must be directed toward a specific measurement module or software option. The SELECT command (a mainframe command) is used to specify a module that will have control for all subsequent commands and queries. You do not have to issue another SELECT command until you select a different control module. Only one module may be in control at any time.

If a valid command string is issued by the controller, and that command string is not defined for the selected module, an error condition will occur. You can use the SELECT query to verify what module is currently in control.

Using the HP 14857A/61062A/ 82990A HP-IB Command Libraries

The HP-IB command library contains a number of subroutines that allow the controller to bidirectionally communicate with the Logic Analysis System. These subroutines automatically handle bus protocol and control, simplifying HP-IB programming.

One of the HP-IB library files is called SETUP.BAS. This file contains the code required to load the necessary assembly language subroutine calls required to use the HP-IB library. Your program must be appended to this file, or the SETUP.BAS file must be merged with your program. When you run your program, SETUP.BAS will initialize the HP-IB interface and start the communications in a known state. All of the GWBASIC/BASICA examples included in this note should be appended to SETUP.BAS before you attempt to execute them.

Addressing the I/O Interface and Logic Analysis System

Each interface card (in the PC) has its own Interface Select Code or ISC. This address is used by the subroutine calls to direct commands and communications to the proper interface. The HP 150 uses address 7 for its HP-IB port and cannot be changed. The HP Vectra or IBM PC can use multiple interfaces, but each card must be assigned a unique ISC. The hardware ISC can be selected, or changed, by using the switches found on the interface card. Because some address combinations may result in internal controller bus conflicts, please refer to the installation section of the HP-IB Command Library Manual.

Parameters passed to the HP-IB subroutine calls must always be passed as variables (not as constants or expressions). The example programs use address 7 for the interface select code and "ISC" for the variable name.

```
ISC = 7
```

A call to the HP-IB subroutine IOCLEAR( ), using ISC as the parameter (ie. IOCLEAR(ISC)), would cause ALL devices connected to the bus at interface select code 7 to respond to the HP-IB bus command DCL (Universal Device Clear).

Device Address

Each instrument on the HP-IB must have a unique address that is between decimal 00 and 30. The method used to set the Logic Analysis System to a particular bus address is covered in the HP 16500A Mainframe Operating Manual.

The HP 14857A/61062A/82990A HP-IB interfaces reserve address 30 for the system controller, which prevents this address from being used by bus instruments.

In the examples, the device address for the Logic Analysis System has been chosen as the power up default of address 07. The reference variable passed to a subroutine call must include not only the device address, but also the correct interface select code. For the example programs, the variable name "LOGIC" will contain the appropriate bus address information. For example:

```
ISC = 7
DEVICE.ADDRESS = 07
LOGIC = 707 = (ISC * 100) + (DEVICE.ADDRESS)
```

When "LOGIC" is passed to a HP-IB subroutine, the subroutine will perform its function on the instrument at address 07. Calling the subroutine IOCLEAR( ), and passing LOGIC as the parameter (ie. IOCLEAR(LOGIC)), causes only the instrument at address 07 to be issued the bus command SDC (Selected Device Clear).

Initialization and Setting Default Values

Unexpected operation, or bus hangups, can be avoided by initializing both the HP-IB interface card and the Logic Analysis System at the beginning of a program. The initialization process will assure that both the instrument and the interface are in a known state.

CALL IORESET(ISC) sets the HP-IB interface to:

- interface timeout = no time limit
- interface EOI mode is enabled
- cr/lf selected as EOL
- lf established as the IOMATCH default

CALL IOCLEAR(LOGIC) clears the Logic Analysis System by:

- terminate any bus communication in progress (UNTalk,UNListen)
- clear all serial poll status bits
- clear the input and output HP-IB buffers
- clear the error queue and key register
- stop any measurement or acquisition processes

Once the HP-IB interface and Logic Analysis System are initialized, the I/O timeout time should be reduced, otherwise the controller will 'hang' if the instrument does not respond. If the instrument does not respond within the allotted time, the controller will abort the I/O operation and report an error condition. Once again, all parameters passed to an HP-IB subroutine are variables. To specify an I/O timeout limit of 10 seconds, use the following:

```
TIME = 10
CALL IOTIMEOUT(ISC,TIME)
```

Initialization Sequence

To conserve space, none of the example program listings will contain the initialization sequence. The following block of code should be inserted at line 1000 for each example program. For simplicity, all user programs in this note will start at line number 1000. The SETUP.BAS file uses line numbers 5 to 570.

```
1000 ISC = 7 : LOGIC = 707
1010 TIME = 10
1020 CALL IORESET(ISC)
1030 CALL IOTIMEOUT(ISC,TIME)
1040 CALL IOCLEAR(LOGIC)
1050 '
```

Error Reporting

When you are developing programs, it is advisable to report any errors that might occur. If a HP-IB subroutine detects an error, it places the number that corresponds to that error into a variable called "PCIB.ERR." Your program can examine PCIB.ERR to find the error number and the string "PCIB.ERR$" to find the error message text.

After each HP-IB subroutine call, the contents of PCIB.ERR should be verified to be zero (no error condition reported). If the variable is not zero, then the error message can be printed by transferring control, via the BASIC error trapping facilities, to an error handling routine in SETUP.BAS. This control tranfer looks like:

```
IF PCIB.ERR <> 0 THEN ERROR PCIB.BASERR
```

To improve readability, the error detecting/reporting has been removed from the example programs. If you have occasion to include any of these examples in your own programs, be sure to add the appropriate error detecting/reporting. Once program development is complete, some of the error checks can safely be eliminated.

Talking to the Logic Analysis System

As mentioned earlier, commands are sent to the Logic Analysis System as ASCII strings. Strings are transmitted to the bus by calling the HP-IB subroutine IOOUTPUTS( ) and passing the device address (ie. LOGIC), the name of the string variable containing the command string, and the length of the command string. The HP-IB subroutine then passes the command string to the correct interface and instrument. The EOL or cr lf is automatically appended to the end of the output message and EOI is asserted on the bus.

The following example program selects the System Utilities menu and changes the display background color to violet:

```
1000 Initialization Sequence
 . . .
1060 COMMAND$ = ":MENU 0,3; SETCOLOR 1,80,60,80"
1070 LENGTH = LEN(COMMAND$)
1080 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1090 '
1100 END
```

Since mainframe commands can be issued at any time, no SELECT command is required. You could also send each command (MENU and SETCOLOR) separately, but this adds more subroutine calls and the corresponding controller overhead. Program execution is faster if multiple commands are sent with a single HP-IB subroutine call.

Passing Computed Arguments

Occasionally you may want to pass derived or computed arguments as part of the command string. The simplest way to accomplish this is to convert the numeric argument to an ASCII string and then concatenate it to the command string. For example:

```
1000 Initialization Sequence
 . . .
1060 '
1070 ' select the mainframe module
1080 '
1090 COMMAND$ = ":SELECT 0"
1100 LENGTH = LEN(COMMAND$)
1110 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1120 '
1130 MODSKEW = 2E-8                ' set skew to 20 ns
1140 '
1150 COMMAND$ = ":INTERMODULE:SKEW5  " + STR$(MODSKEW)
1160 LENGTH = LEN(COMMAND$)
1170 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1180 '
1190 END
```

The STR$( ) function in line 1150 converts the argument into a string that can be added to the command header. The resulting string (what is actually loaded into COMMAND$) is ":INTER-MODULE:SKEW5 .00000002." Remember to include a space at the end of the command header string, otherwise the argument will be appended directly behind the command header without an intervening space. A missing space between the command header and the argument will cause an error condition.

The same skew information could be sent by passing ":INTERMODULE:SKEW5 " as a string and then sending MODSKEW as a real number. To retain the appearance of a single command (to the instrument), the EOL and EOI must be suppressed before you send MODSKEW. This version of the program would look like:

```
1000 Initialization Sequence
 . . .
1060 DISABLE = 0 : ENABLE = 1
1070 EOL$ = CHR$(10)
1080 '
1090 ' select the mainframe module
1100 '
```

```
1110 COMMAND$ = ":SELECT 0"
1120 LENGTH = LEN(COMMAND$)
1130 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1140 '
1150 ' disable EOI and don't append any EOLs
1160 '
1170 CALL IOEOI(ISC,DISABLE)
1180 CALL IOEOL(ISC,EOL$,DISABLE)
1190 '
1200 ' send the skew command
1210 '
1220 COMMAND$ = ":INTERMODULE:SKEW5 "
1230 LENGTH = LEN(COMMAND$)
1240 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1250 '
1260 CALL IOEOL(ISC,EOL$,ENABLE)          ' enable the appending of EOLs
1270 '
1280 MODSKEW = 2E-08                      ' set skew to 20 ns
1290 '
1300 CALL IOOUTPUT(LOGIC,MODSKEW)         ' send the skew value
1310 '
1320 END
```

As you can see, it takes many more subroutine calls to pass information in this manner. Since both EOI and EOL were supressed to avoid early termination of the command, EOL must be enabled before the last element of the command is sent. When the argument is sent, an EOL is sent with it, so the Logic Analysis System can execute the command.

Setting up the Logic Analysis System

When the Logic Analysis System is powered up, every installed module wakes up in some default configuration. By issuing a START command (from the front panel or over the bus) you could immediately make a measurement. That measurement may, or may not, be real useful because the default trace specification(s) (trigger event) are set up to trigger on just about anything. Let's take a closer look at the logic analyzer module and make some modifications to its default menus.

The power up menus for the logic analyzer module will allow you to make a 16 channel, transitional mode timing measurement. The trigger event is any pattern and eight acquisition channels will show up on the timing waveform display.

Let's modify the logic analyzer configuration menu and add a second 16 channel acquisition pod. On the logic analyzer format menu, let's assign eight channels of the new pod to a label called DATA. While we are on the format menu, let's also change the label for pod one to ADDR. The example program that would make these changes is listed below.

```
1000 Initialization Sequence
 . . .
1060 ' select the logic analyzer module (cardslot E)
1070 '
1080 COMMAND$ = ":SELECT 5"
1090 LENGTH = LEN(COMMAND$)
1100 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1110 '
1120 ' assign pods 1 and 2 to analyzer 1
1130 '
1140 COMMAND$ = ":MACHINE1:ASSIGN 1,2"
1150 LENGTH = LEN(COMMAND$)
1160 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1170 '
1180 ' change the existing pod 1 label
1190 '
1200 COMMAND$ = ":MACHINE1:TFORMAT:LABEL 'ADDR',POS,#HFFFF"
1210 LENGTH = LEN(COMMAND$)
1220 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1230 '
1240 ' add a label for pod 2 and assign acquisition channels
1250 '
1260 COMMAND$ = ":MACHINE1:TFORMAT:LABEL 'DATA',POS,0,#H00FF"
1270 LENGTH = LEN(COMMAND$)
1280 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1290 '
1300 END
```

The two label names and channel assignment commands (lines 1200 and 1260) could be passed as a single command string. The combined command string would look like:

```
1200 COMMAND$ = ":MACHINE1:TFORMAT:LABEL 'ADDR',POS,#HFFFF;
                    LABEL: 'DATA',POS,0,#H00FF"
```

Combining command strings reduces the amount of HP-IB and controller overhead since only a single HP-IB subroutine call is made. This is a valid programming technique when speed is the primary concern, but remember not to make the combined string so long that the interface buffer overflows. Combining multiple command strings does compromise the readability of your program, making "bug" isolation more difficult.

Receiving Information from the Logic Analysis System

When the Logic Analysis System receives a query (ie. command header followed by a question mark), the instrument interrogates the requested module function and places the answer in the HP-IB output buffer. The output message remains in the buffer until read by the controller or another query is received. When the instrument is addressed to talk (done by the HP-IB subroutines), the message is transmitted across the bus to the designated listener (controller).

The format of the returned ASCII string is dependent upon the current settings of the HEADER and LONGFORM commands. The general response format is:

```
[HEADER](ARGUMENT)(EOL)
```

The square brackets around HEADER indicate that this field is optional. The HEADER field identifies the argument that follows. The presence of a HEADER is controlled by the HEADER ON/OFF commands. If the header is turned off (ie. HEADER OFF), only the argument is returned by the query.

If a header is returned, its format is controlled by the LONGFORM ON/OFF commands. With a LONGFORM OFF command, the header will be in its short form, with trailing spaces added to make it a six character string. With a LONGFORM ON command, the entire header will be spelled out. Its length will vary depending on the particular query that was issued. For example, in response to a module select query (ie. :SELECT?), the following strings could appear based on current HEADER and LONGFORM selections.

(ARGUMENT)(EOL)	- header off
SEL (ARGUMENT)(EOL)	- header on/longform off
SELECT (ARGUMENT)(EOL)	- header on/longform on

Most arguments returned will be either real or integer numbers. In some instances, the query can return a string argument. The returned argument type is described in the programming manuals.

String Variables

Reading queries into string variables is simple and requires little attention to formatting. The only concern when reading a string is to insure that the maximum length dimension of the string is not exceeded.

A call to the HP-IB subroutine IOENTERS( ) is required to read data into a string variable. The subroutine call syntax is:

```
CALL IOENTERS(LOGIC,VARIABLE$,MAX.LENGTH,ACTUAL.LENGTH)
```

LOGIC is the variable that contains the HP-IB select code and bus address. VARIABLE$ is the name of the string variable that the data is to be read into. MAX.LENGTH is the maximum number of characters that can be read to the string (ie. MAX.LENGTH should be less than or equal to the dimension of VARIABLE$). ACTUAL.LENGTH is a number returned by the IOENTERS( ) subroutine that indicates the number of characters that were actually read into the string.

MAX.LENGTH and ACTUAL.LENGTH will be different if IOENTERS( ) encounters a line feed or EOI before MAX.LENGTH characters have transferred. Either of these terminating conditions may be defeated by using the appropriate subroutine calls (discussed later).

Arguments that are read in as strings, and are to be used in later computations, must be converted from a string representation with the VAL($) function. If you read a string that will be passed to VAL($), you must turn the HEADER OFF. The VAL($) function will return a zero if the first character of its input string is an alpha symbol (ie. the first character of HEADER).

Real Variables

Numeric output of the Logic Analysis System can be read into a real, or integer, variable. The controller's number builder ignores text characters and responds only to +, –, e, E, or the digits 0-9. If the Logic Analysis System outputs an alpha header followed by a numeric argument, the header is ignored and the numeric argument is read into the variable.

A single real number can be read into a real variable by calling the HP-IB subroutine IOENTER( ). The subroutine call syntax is:

```
CALL IOENTER(LOGIC,NUMBER.VARIABLE)
```

The controller commands the instrument at address LOGIC to talk and then puts its response into the real variable NUMBER.VARIABLE.

The following example program queries the 1 GHz Timing Analyzer module (HP 16515A) for the average X to O marker time. This example program assumes the timing analyzer has previously been set up with labels, channel assignments, trigger specifications, and marker instructions. The analyzer module is queried for the average X to O time and reads it into both a string variable (XOAVG$) and a real variable (XOAVG). The results are printed on the controller's screen.

```
1000 Initialization Sequence
 . . .
1060 ' reserve space for the command and response strings
1070 ' 1080 COMMAND$ = SPACE$(255) : XOAVG$ = SPACE$(30)
1090 '
1100 ' select the mainframe module
1110 '
1120 COMMAND$ = ":SELECT 0"
1130 LENGTH = LEN(COMMAND$)
1140 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1150 '
1160 ' turn on longform headers
1170 '
1180 COMMAND$ = ":SYSTEM:LONGFORM ON; HEADER ON"
1190 LENGTH = LEN(COMMAND$)
1200 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1210 '
1220 ' select the 1 GHz timing analyzer module (cardslot D)
1230 '
1240 COMMAND$ = ":SELECT 4"
1250 LENGTH = LEN(COMMAND$)
1260 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1270 '
1280 COMMAND$ = ":START"                ' acquire data
1290 LENGTH = LEN(COMMAND$)
1300 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
```

```
1310 '
1320 ' query for the average X to O marker time
1330 '
1340 COMMAND$ = ":WAVEFORM:TAVERAGE?"
1350 LENGTH = LEN(COMMAND$)
1360 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1370 '
1380 MAX = 30 : ACTUAL = 0
1390 '
1400 ' read the reply into a string
1410 '
1420 CALL IOENTERS(LOGIC,XOAVG$,MAX,ACTUAL)
1430 PRINT "Answer via string variable - ", XOAVG$
1440 '
1450 ' query for the average X to O marker time
1460 '
1470 COMMAND$ = ":WAVEFORM:TAVERAGE?"
1480 LENGTH = LEN(COMMAND$)
1490 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1500 '
1510 ' read the reply into a real variable
1520 '
1530 CALL IOENTER(LOGIC,XOAVG)
1540 PRINT "Answer via real variable - ", XOAVG
1550 '
1560 END
```

Note

Memory should be reserved for the string variables as shown in line 1080. If this is not done, or the strings are DIMensioned to 255, the program may overwrite the string(s).

Receiving Multiple Data Messages

Queries of subsystems (ie. the disc directory), or automatic measurements, may place more than one data message in the HP-IB output buffer. Multiple replies can be read into string variables, real variables, or real arrays. The format of the buffered messages is the same for either type of reply. The subsystem selector will be included in subsystem queries. The received message format is:

```
[SUBSYSTEM SELECTOR](EOL)
[HEADER](ARGUMENT)(EOL)[HEADER](ARGUMENT)(EOL) ...
```

The use of the HEADER and/or LONGFORM commands will modify the returned message data as previously discussed. The HEADER OFF command will delete the subsystem selector during subsystem queries.

Reading Multiple Replies Into String Variables

Reading into a string variable terminates when an EOL (default cr lf) or EOI is detected by the controller's HP-IB interface. Since cr lf is used as a separator within the message, it is necessary to disable the EOL terminator. This is done by calling the HP-IB subroutine IOMATCH( ):

```
DISABLE = 0
ENABLE = 1
EOL$ = CHR$(10)
CALL IOMATCH(ISC,EOL$,DISABLE)
```

Once the EOL terminator is disabled, the read statement will not terminate until a specified number of characters have been read, EOI is detected, or a timeout occurs. When you are passing multiple data messages, program the Logic Analysis System for EOI ON mode, otherwise read operations may not terminate until the I/O times out.

The maximum allowable string length in GWBASIC/BASICA is 255 characters. When transferring a large block of string data, it may be necessary to test the subroutine variable that indicates the number of characters that were actually read by the subroutine call (ie. ACTUAL.LENGTH). If that number is 255, then subsequent call(s) to IOENTERS( ) will be needed to complete the read. Each call to IOENTERS( ) should read the data into a different string variable. The following example requests that the Oscilloscope module (HP 16530A) run an automatic measurement and reads in all ten waveform parameters.

```
1000 Initialization Sequence
 . . .
1060 COMMAND$ = SPACE$(255) : WAVEDATA$ = SPACE$(255)
1070 '
1080 ENABLE = 1 : DISABLE = 0
1090 EOL$ = CHR$(10)
1100 '
1110 ' select the mainframe module and turn EOI on
1120 '
1130 COMMAND$ = ":EOI ON; SELECT 0"
1140 LENGTH = LEN(COMMAND$)
1150 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1160 '
```

```
1170 ' turn on longform headers
1180 '
1190 COMMAND$ = ":SYSTEM:LONGFORM ON; HEADER ON"
1200 LENGTH = LEN(COMMAND$)
1210 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1220 '
1230 ' select the oscilloscope module (cardslot B)
1240 '
1250 COMMAND$ = ":SELECT 2"
1260 LENGTH = LEN(COMMAND$)
1270 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1280 '
1290 ' autoscale the oscilloscope for optimum measurement results
1300 '
1310 COMMAND$ = ":AUTOSCALE"
1320 LENGTH = LEN(COMMAND$)
1330 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1340 '
1350 ' wait for autoscale to complete
1360 '
1370 COMMAND$ = "*WAI"
1380 LENGTH = LEN(COMMAND$)
1390 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1400 '
1410 ' query for all waveform parameters on channel 1
1420 '
1430 COMMAND$ = ":MEASURE:SOURCE CHANNEL1:ALL?"
1440 LENGTH = LEN(COMMAND$)
1450 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1460 '
1470 ' disable EOL checking and enable EOI
1480 '
1490 CALL IOMATCH(ISC,EOL$,DISABLE)
1500 CALL IOEOI(ISC,ENABLE)
1510 '
1520 ' read the waveform parameters
1530 '
1540 MAX = 255 : ACTUAL = 0
1550 CALL IOENTERS(LOGIC,WAVEDATA$,MAX,ACTUAL)
1560 '
1570 CALL IOMATCH(ISC,EOL$,ENABLE)
1580 '
1590 PRINT WAVEDATA$
1600 '
1610 END
```

The wait command (ie. *WAI) is used to allow the oscilloscope module enough time to complete the autoscale operation before its data is read. The wait command begins with an "*" since it is a common HP-IB command, as opposed to a Logic Analysis System command.

Reading Multiple Replies Into Real Variables

Multiple real number replies can be read into many single real variables if desired. Although possible, it's very inefficient since a subroutine call must be made for each variable to be read. A better method is to read the information into a real array.

A call to the HP-IB subroutine IOENTERA( ) is required to read data into a real array. The subroutine call syntax is:

```
CALL IOENTERA(LOGIC,REAL.ARRAY(1),MAX.LENGTH,ACTUAL.LENGTH)
```

LOGIC is the variable that contains the HP-IB addressing information. REAL.ARRAY is the name of the array that the data is to be read into, starting at array element one. MAX.LENGTH is the maximum number of array entries (ie. MAX.LENGTH should be less than or equal to the dimension of REAL.ARRAY). ACTUAL.LENGTH is a number returned by the IOENTERA( ) subroutine that indicates the number of entries that were actually read into the array.

The array must be dimensioned to be at least as large as the number of elements to be read from the Logic Analysis System. If the array is dimensioned larger than is required, the read operation will terminate upon receiving EOI or the number of expected entries in the array. Any headers in the response are ignored by the number builder. The following example program queries the Logic Analysis System for the identification of all installed modules.

```
1000 Initialization Sequence
 . . .
1060 OPTION BASE 1                 ' set base array index to 1
1070 DIM IDS(10)
1080 COMMAND$ = SPACE$(255)
1090 '
1100 COMMAND$ = ":CARDCAGE?"
1110 LENGTH = LEN(COMMAND$)
1120 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
```

```
1130 '
1140 ' read the card id codes
1150 '
1160 MAX = 10 : ACTUAL = 0
1170 CALL IOENTERA(LOGIC,IDS(1),MAX,ACTUAL)
1180 '
1190 FOR I = 1 TO ACTUAL
1200   PRINT IDS(I)
1210 NEXT I
1220 '
1230 END
```

Receiving Waveform Data

When an acquisition module acquires data, the data is passed from the Logic Analysis System to the controller by sending a numerical representation of the acquired data. The specific content of a modules data block varies based on the type of module and the kind of data acquired.

ASCII Waveform Data - Oscilloscope Module

The easiest way to read a digitized waveform from the oscilloscope module is to use the ASCII waveform data format and put the waveform data into a real array. Each point in the waveform is represented as a six digit integer whose value ranges form 0 to 16,383. Each of the data points is followed by a cr lf which delimits each entry into the array. The data that is passed can be used to determine the voltage at each point in the waveform. The numbers are passed starting with the leftmost point on the oscilloscope display.

The number of points in the waveform varies depending on the RECORD mode selected in the WAVEFORM subsystem. When RECORD FULL is selected, the full 4096 point waveform memory is returned. When RECORD WINDOW is selected, only the data displayed on the screen is returned. The POINTS? query (in the WAVEFORM subsystem) is used to determine how many data points are in the waveform when the RECORD WINDOW mode is selected.

This example program digitizes the signal present on channel 1's input, stores it in memory, passes it to the controller, and finally plots the information on the controller's display.

```
1000 Initialization Sequence
 . . .
1060 OPTION BASE 1
1070 COMMAND$ = SPACE$(255)
1080 ENABLE = 1 : DISABLE = 0
1090 '
1100 DIM WAVE(4096)
1110 '
1120 CALL IOEOI(ISC,ENABLE)
1130 '
1140 ' select the oscilloscope module (cardslot B)
1150 '
1160 COMMAND$ = ":SELECT 0"
1170 LENGTH = LEN(COMMAND$)
1180 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1190 '
1200 COMMAND$ = ":AUTOSCALE"
1210 LENGTH = LEN(COMMAND$)
1220 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1230 '
1240 COMMAND$ = "*WAI"
1250 LENGTH = LEN(COMMAND$)
1260 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1270 '
1280 ' specify data from channel 1, a full record in ASCII format
1290 '
1300 COMMAND$ = ":WAVEFORM:SOURCE CHANNEL1"
1310 LENGTH = LEN(COMMAND$)
1320 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1330 '
1340 COMMAND$ = ":WAVEFORM:RECORD FULL"
1350 LENGTH = LEN(COMMAND$)
1360 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1370 '
1380 COMMAND$ = ":WAVEFORM:FORMAT ASCII"
1390 LENGTH = LEN(COMMAND$)
1400 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1410 '
1420 ' run the oscilloscope
1430 '
1440 COMMAND$ = ":START"
1450 LENGTH = LEN(COMMAND$)
1460 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1470 '
1480 ' query for the waveform data
```

```
1490 '
1500 COMMAND$ = ":WAVEFORM:DATA?"
1510 LENGTH = LEN(COMMAND$)
1520 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1530 '
1540 ' read the waveform data
1550 '
1560 MAX = 4096 : ACTUAL = 0
1570 CALL IOENTERA(LOGIC,WAVE(1),MAX,ACTUAL)
1580 '
1590 ' determine the controller type and select the graphics screen
1600 '
1610 ON ERROR GOTO 1650
1620 SCREEN 3                         ' HP 150 graphics screen
1630 CLS 0
1640 GOTO 1680
1650 RESUME 1660                      ' must be a HP Vectra (IBM PC)
1660 CLS
1670 SCREEN 2                         ' HP Vectra (IBM PC) graphics screen
1680 ON ERROR GOTO 0                  ' disable error branching
1690 '
1700 WINDOW (0,0)-(4096,16383)        ' set screen bounds
1710 '
1720 ' display the waveform
1730 '
1740 FOR I = 1 TO ACTUAL
1750   PSET(I,WAVE(I)),7              ' plot each point
1760 NEXT I
1770 '
1780 END
```

Note

The graphics screen of the HP 150 differs from that of the IBM PC and the HP Vectra. Lines 1610-1670 determine the controller and clears the appropriate graphics screen.

This example program assumes that the timebase, vertical sensitivity, trigger, etc. have been setup prior to running this program.

Binary Data Transfers

When using the binary data transfer format, the data stream will begin with an ASCII byte count. The byte count tells how many data bytes are in the data stream. The byte count field has the format "#8DDDDDDDD". The "D"s define an eight digit field that contains the byte count. Leading zeros are used to fill out the field to eight digits.

Binary transfer is the fastest method for moving information across the bus, but it requires considerably more controller processing time to make use of the information. The ASCII data transfer (that we saw above) loads each data point into an array element. Each array element can be read and processed by the controller in a fairly straightforward manner.

When transferring data from the Logic Analysis System in binary format, the HP 14857A/61062A/82990A HP-IB interfaces don't support reading the data directly into an array. However, the data can be read into strings. GWBASIC/BASICA string commands cannot be used to interpret the data since the commands assume an ASCII representation of the contents of the strings. To correctly reconstruct the binary information, the data must be read directly from controller memory using the PEEK command.

String variables are actually stored in the controller as two separate elements: the string descriptor and the actual data string. The string descriptor is made up of three parts. The first element of a string descriptor contains the length of the string. The second element contains the low byte of the memory address where the actual data string is located and the third element contains the high byte of the memory address.

To find the address of the string descriptor, use the VARPTR( ) command. PEEKing at the returned address with an offset of one returns the low byte of the data string's location. Using an offset of two returns the high byte of the data string's location.

Assuming that 255 binary bytes have been stored in the string DATA$, the following block of code will find the starting memory address of the data string and store it in the variable ADDR.PTR.

```
LOW.ADR = PEEK(VARPTR(DATA$) + 1)
HIGH.ADR = PEEK(VARPTR(DATA$) + 2)
ADDR.PTR = (256 * HIGH.ADR) + LOW.ADR
```

Once the memory location of the data string portion of DATA$ is known, it's a simple matter to PEEK at this address and read the binary data. You can increment the address pointer to look at other binary data in the string. For example:

```
FOR I = 0 TO 254
  BYTE = PEEK(ADDR.PTR + I)
  PRINT BYTE
NEXT I
```

Dimensioning the Input Strings

The number of 255 byte strings required for storing a module's data block is a function of the data block length. The number of bytes of data to be transfered is contained in the ASCII string at the beginning of the data block. One method of determining the number of data bytes to be transferred is to read the ASCII string, dimension an array of strings to the appropriate size, and perform read operations to fill the array.

An easier method is to simply dimension an array of strings large enough to accommodate the maximum number of bytes expected, and simply terminate the read when the EOI occurs. This may not make the most efficient use of controller memory, but it is easy to implement.

Binary Acquisition Data - Logic Analyzer Module

This example program causes the logic analyzer module to setup a trace and acquire data. The data is loaded into an array of strings in the controller.

A call to the HP-IB subroutine IOGETTERM( ) is found in line 1710. This subroutine returns a number in the variable EOI.CHECK that indicates why a read statement terminated. If the read is terminated by an EOI, then the Logic Analysis System is finished sending data and the loop may be exited. If the read terminated because the 255 character buffer is full, then additional reads are required.

Note

IOMATCH() must be disabled to prevent a read from terminating prematurely when the binary data appears as a decimal 10 (ie. a line feed).

```
1000 Initialization Sequence
 . . .
1060 OPTION BASE 1
1070 DISABLE = 0 : ENABLE = 1
1080 EOL$ = CHR$(10)
1090 COMMAND$ = SPACE$(255)
1100 '
1110 ' setup an array of strings for the logic analyzer data (14532 bytes)
1120 '
1130 DIM TRACE$[57], ADDR.PTR(57)
1140 '
1150 FOR I = 1 TO 57
1160 TRACE$[I] = SPACE$(255)
1170 NEXT I
1180 '
1190 ' select the logic analyzer module (cardslot E)
1200 '
1210 COMMAND$ = ":SELECT 5"
1220 LENGTH = LEN(COMMAND$)
1230 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1240 '
1250 ' assign pods and labels
1260 '
1270 COMMAND$ = ":MACHINE1:ASSIGN 5,3,1"
1280 LENGTH = LEN(COMMAND$)
1290 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1300 '
1310 COMMAND$ = ":MACHINE1:TFORMAT:LABEL 'DATA',POS,
     #HFFFF,#HFFFF,0"
1320 LENGTH = LEN(COMMAND$)
1330 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1340 '
1350 COMMAND$ = ":MACHINE1:TFORMAT:LABEL 'ADDR',POS,0,0,
     #H03FF"
1360 LENGTH = LEN(COMMAND$)
1370 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1380 '
1390 ' select glitch mode operation
1400 '
1410 COMMAND$ = ":MACHINE1:TTRACE:AMODE GLITCH"
1420 LENGTH = LEN(COMMAND$)
1430 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1440 '
1450 ' run the logic analyzer
1460 '
```

```
1470 COMMAND$ = ":START"
1480 LENGTH = LEN(COMMAND$)
1490 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1500 '
1510 COMMAND$ = "*WAI"
1520 LENGTH = LEN(COMMAND$)
1530 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1540 '
1550 ' disable EOL checking and enable EOI
1560 '
1570 CALL IOMATCH(ISC,EOL$,DISABLE)
1580 CALL IOEOI(ISC,ENABLE)
1590 '
1600 ' query for the timing data
1610 '
1620 COMMAND$ = ":SYSTEM:DATA?"
1630 LENGTH = LEN(COMMAND$)
1640 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1650 '
1660 ' read the timing data
1670 '
1680 FOR I = 1 TO 57
1690   MAX = 255 : ACTUAL = 0
1700   CALL IOENTERS(LOGIC,TRACE$[I],MAX,ACTUAL)
1710   CALL IOGETTERM(ISC,EOI.CHECK)
1720   IF (EOI.CHECK AND 4) THEN GOTO 1740
1730 NEXT I
1740 '
1750 DEF SEG                                   ' reset controller data register
1760 '
1770 ' compute the data string memory locations
1780 '
1790 FOR I = 1 TO 57
1800   LOW.BYTE = PEEK(VARPTR(TRACE$[I]) + 1)
1810   HIGH.BYTE = PEEK(VARPTR(TRACE$[I]) + 2)
1820   ADDR.PTR(I) = (256 * HIGH.BYTE) + LOW.BYTE
1830 NEXT I
1840 '
1850 DEF SEG
1860 '
1870 ' print the entire data block
1880 '
1890 FOR J = 1 TO 57
1900   FOR I = 0 TO 254
1910     PRINT "Byte # ",(256 * (J - 1) + I),HEX$(PEEK(ADDR.PTR[J] + I))
```

```
1920   NEXT I
1930 NEXT J
1940 '
1950 END
```

Sending Data to the Logic Analysis System

The Logic Analysis System can accept data by using the DATA command. The format of the data stream into the Logic Analysis System depends on the specific module receiving the data, but in general, is exactly like the output data stream for that module.

Usually, the data that is sent into the Logic Analysis System is from a file that has been filled by reading data out of the Logic Analysis System. Modifying a data file before loading it back into the Logic Analysis System can be a complex task and is not recommended.

If a DATA? query is issued with the HEADER ON, the command header (ie. DATA), the data block size, and the data itself is all in the file and is ready to be loaded into the Logic Analysis System. As you might expect, this is the easiest way to input data. Issuing an appropriate number of calls to the IOOUTPUTS( ) subroutine is all that is required to transfer this data block.

If the HEADER was OFF when the data was read from the Logic Analysis System, the "DATA " command (including the trailing space) must precede the actual data stream. The simplest way to accomplish this is to concatenate "DATA " to the front of the actual data stream. If the data stream fills a 255 character buffer, then there is no room to perform the concatenation since the GWBASIC/BASICA string length would be exceeded.

If there is insufficient room to perform the concatenation, the DATA command must be issued immediately before the data stream. The EOL and EOI terminators must be disabled to prevent the DATA command's termination before receiving the data.

Regardless of how the DATA header got where it needs to be, the EOL and EOI must be turned off when sending multiple strings, otherwise the cr lf could be interpreted as valid data by the Logic Analysis System. Before sending the last string, enable the EOI so the write operation will terminate properly.

This example program will read a stimulus program from the Pattern Generator module (HP 16520A) into the controller. Once the stimulus program is in the controller, the program will be deleted from pattern generator memory and then loaded back into the Logic Analysis System. We will assume that a six statement stimulus program has already been written and is currently in the pattern generator memory.

```
1000 Initialization Sequence
 . . .
1060 OPTION BASE 1
1070 DISABLE = 0 : ENABLE = 1
1080 EOL$ = CHR$(10)
1090 COMMAND$ = SPACE$(255)
1100 '
1110 ' setup an array of strings to hold the stimulus program
1120 ' (1496 bytes)
1130 '
1140 DIM PROG$[6]
1150 '
1160 FOR I = 1 TO 6
1170   PROG$[I] = SPACE$(255)
1180 NEXT I
1190 '
1200 ' select the mainframe module
1210 '
1220 COMMAND$ = ":SELECT 0"
1230 LENGTH = LEN(COMMAND$)
1240 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1250 '
1260 ' turn headers off
1270 '
1280 COMMAND$ = ":SYSTEM:HEADER OFF"
1290 LENGTH = LEN(COMMAND$)
1300 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1310 '
1320 ' select the pattern generator module (cardslot C)
1330 '
1340 COMMAND$ = ":SELECT 3"
1350 LENGTH = LEN(COMMAND$)
1360 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1370 '
1380 ' disable EOL checking and enable EOI
1390 '
1400 CALL IOMATCH(ISC,EOL$,DISABLE)
```

```
1410 CALL IOEOI(ISC,ENABLE)
1420 '
1430 ' query for the stimulus program
1440 '
1450 COMMAND$ = ":SYSTEM:DATA?"
1460 LENGTH = LEN(COMMAND$)
1470 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1480 '
1490 ' read the stimulus program
1500 '
1510 FOR I = 1 TO 6
1520   MAX = 255 : ACTUAL = 0
1530   CALL IOENTERS(LOGIC,PROG$[I],MAX,ACTUAL)
1540   CALL IOGETTERM(ISC,EOI.CHECK)
1550   IF (EOI.CHECK AND 4) THEN GOTO 1570
1560 NEXT I
1570 '
1580 ' delete the stimulus program from pattern generator memory
1590 '
1600 COMMAND$ = ":LISTING:REMOVE ALL"
1610 LENGTH = LEN(COMMAND$)
1620 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1630 '
1640 ' disable EOI and EOL for binary transfer
1650 '
1660 CALL IOEOL(ISC,EOL$,DISABLE)
1670 CALL IOEOI(ISC,DISABLE)
1680 '
1690 ' tell the pattern generator a stimulus program is coming
1700 '
1710 COMMAND$ = ":SYSTEM:DATA  "
1720 LENGTH = LEN(COMMAND$)
1730 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1740 '
1750 ' load the stimulus program
1760 '
1770 FOR I = 1 TO 5
1780   LENGTH = LEN(PROG$[I])
1790   CALL IOOUTPUTS(LOGIC,PROG$[I],LENGTH)
1800 NEXT I
1810 '
1820 ' enable EOL and EOI for the last string
1830 '
1840 CALL IOEOL(ISC,EOL$,ENABLE)
1850 CALL IOEOI(ISC,ENABLE)
```

```
1860 '
1870 ' write the last string
1880 '
1890 LENGTH = LEN(PROG$[6])
1900 CALL IOOUTPUTS(LOGIC,PROG$[6],LENGTH)
1910 '
1920 ' look at the pattern generator listing menu
1930 '
1940 COMMAND$ = ":MENU 3,1"
1950 LENGTH = LEN(COMMAND$)
1960 CALL IOOUTPUTS(LOGIC,COMMAND$,LENGTH)
1970 '
1980 END
```

Note

In line 1710, the command string must end with a space, otherwise an error condition will result (the space is a separator between the command and the data stream).

HEWLETT
PACKARD

HP PART NUMBER: 5952-4241

Printed in U.S.A.