HP 8719C, 8720C, and 8722A/C network analyzers

QuickC Programming Guide

for use with MS-DOS personal computers

HP Part No. 08720-90155
Printed in USA November 1992

Contents

Figures

Tables

1

Programming Basics

Introduction

This programming guide is an introduction to remote operation of the HP 8719C, 8720C and 8722A/C network analyzers using an HP Vectra personal computer (or IBM compatible), using a MS-DOS® operating system, with the HP 82335A HP-IB command library and Microsoft® QuickC 2.5. It is a tutorial introduction using C programming examples. This document is closely associated with the *HP-IB Programming Reference (HP part number 08720-90160)*, which provides complete programming information in a concise format. Included in the *HP-IB Programming Reference* is an alphabetical list of HP-IB mnemonics and their explanations.

The reader should become familiar with the operation of the network analyzer before controlling it over HP-IB. This document is not intended to teach C programming or to discuss HP-IB theory except at an introductory level. Refer to the documents listed below which are better suited to these tasks. 8

- For more information concerning the operation of the network analyzer, refer to the following:

  User's Guide (HP part number 08720-90136)

  Operating and Programming Reference Section (within the HP 8719C, 8720C, 8722A/C Operating Manual, HP part number 08720-90135)

- For more information concerning HP-IB and C, refer to the following:

  HP-IB Programming Reference (HP part number 08720-90160)

  Using the HP 82335A HP-IB Interface and Command Library Manual (HP part number 82335-90005)

  Microsoft QuickC: Up and Running

  Microsoft QuickC: Tool Kit

  C for Yourself

Caution

The programming examples found in this guide are for example purposes only. They may require modification to work with your particular personal computer.

Start-up

Required equipment

To run the examples in this programming guide, the following equipment is required:

1. 3* HP 8719C, 8720C, or 8722A/C network analyzer.
2. The following computer equipment:
   - HP Vectra personal computer (or compatible) with Microsoft QuickC 2.5
   - HP 82335A HP-IB interface card or a compatible IEEE 488 interface card
   - MS-DOS 3.3 or higher
   - 512 Kbytes of memory. 9
3. HP 10833A/B/C/D HP-IB cables to interconnect the computer, the analyzer, and any peripherals.
4. Calibration kit and appropriate test port cables.
5. A device under test (DUT), such as the bandpass filter supplied with your instrument (HP part number 0955-0446).

Configuring the HP 82335A HP-IB interface card

Configure the HP 82335A HP-IB interface card according to its respective manual. The HP-IB interface cannot share the same memory address, nor the same interrupt level with another card. If an expanded memory manager is used, make sure it does not use the same memory space as the HP-IB interface.

The example programs in this guide assume the HP-IB interface card is configured with select code 7. If it is not, the variables `ISC`, `INSTR`, and `DISPLAY`, in the following programs, will have to be changed to reflect a different select code.

Using other computer equipment

Other versions of C can be substituted for QuickC. The programs in this guide are designed to be easily translated into other language versions.

9The programs in this guide are specific to the HP 82335A HP-IB command library which is part of the HP 82335A HP-IB interface card. Other IEEE 488 cards can be used; however, their I/O command library maybe different and the following example programs may have to be translated accordingly.

Note

There maybe some HP 82335A command library commands that may not have an equivalent command in other IEEE 488 I/O command libraries. In this case, consult the manufacturer or program an equivalent command.

Configuring Microsoft QuickC

It is important to configure Microsoft QuickC properly for correct operation with the HP 82335A HP-IB command library and the example programs in this guide. The following steps should be verified before continuing.

1. Installing QuickC.

   When installing QuickC, choose either the small or large memory model. Refer to the QuickC manual for detailed installation information.

2. Copying the HP 82335A HP-IB command library.

   It is assumed that QuickC is installed in the "qc25" directory on the default drive. Copy the following HP 82335A HP-IB command library files to their corresponding directories already created by the installation of QuickC.

CLHPIB.LIB	—> qc25/lib/CLHPIB.LIB
CHPIB.H	—> qc25/include/CHPIB.H
CFUNC.H	—> qc25/include/CFUNC.H

3. Customizing QuickC for the HP 82335A command library.

   Load QuickC by typing "QC" at the prompt. Activate the options menu by clicking the mouse on the menu bar or by pressing the [ALT][O] keys. Select the "environment" menu. Enter the following directory names:

binary and help files	—> qc25/bin
include files	—> qc25/include
library files	—> qc25/lib

   Select <OK> when done.

   Again from the Options menu, select the "make" menu. Select the "linker flags" option. Enter the following for

   ```
   "GLOBAL FLAGS:  Stack Size":  4096
   ```

   Also from this menu, enter the following for

   ```
   "CUSTOM FLAGS:  GLOBAL":  qc25/lib/clhpib.lib
   ```

   Select <OK> when done.

Powering up the system

1. Set up the network analyzer as shown in Figure 1-1

   Connect the instrument to the computer with an HP-IB cable. The instrument has only one HP-IB interface, but it occupies two addresses: one for the instrument and one for the display. The display address is the instrument address with the least significant bit complemented. The default addresses are 16 for the instrument and 17 for the display. Devices on the HP-IB cannot occupy the same address as the network analyzer.

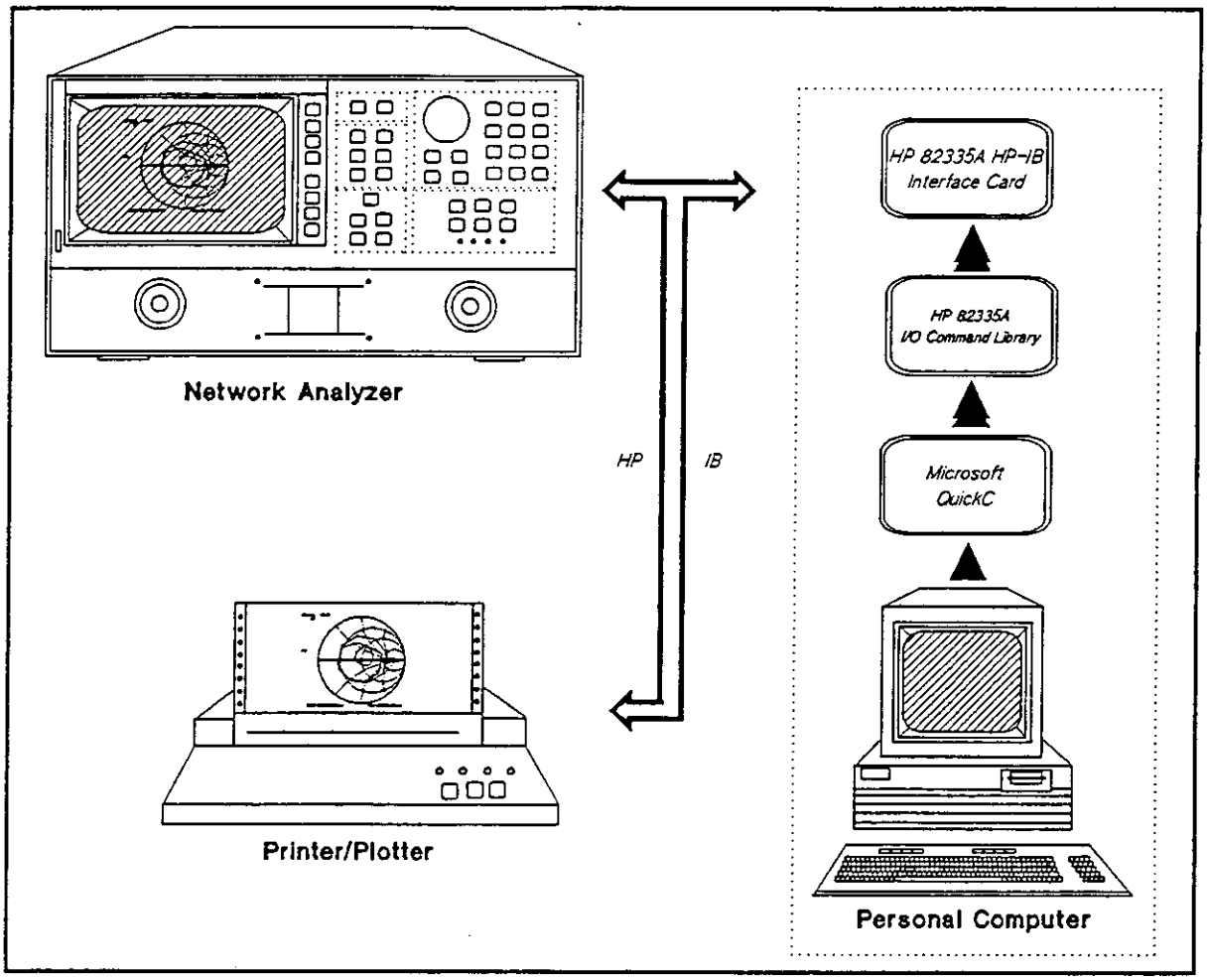

Figure 1-1. HP-IB connections in a typical setup

2. Turn the network analyzer on.

   To verify the instrument HP-IB address, press (LOCAL) SET ADDRESSES and ADDRESS: INSTRUMENT. If the address has been changed from the default value 16, return it to 16 for the examples in this document by pressing (1) (6) (x1) and then presetting the instrument. Make sure the instrument is in either USE PASS CONTROL or TALKER/LISTENER mode, as indicated under the (LOCAL) menu. These are the only modes in which the network analyzer will accept commands over HP-IB.

Programming techniques

Introduction

The following example programs introduce the interfacing capabilities of the instrument with HP-IB and a computer. Each example program contains a description of the program, the program listing, a line by line explanation, and detailed instructions for running the program. Note that line numbers aren't used in C programs but are included in the program listings for functional explanations. For clarity, the HP-IB command library function names are shown in upper case. Remember that C is a case-sensitive language.

There are four basic steps in designing a program in C to send a simple command to the instrument:

1. Create a source file. This is a C program in text form. This file must contain the proper "#include" definitions and variable declarations, including the "main" function declaration.
2. Include in the main declaration the command to be sent. The command is sent with the HP-IB command library function IOOUTPUTS.
3. Compile and link the source file.
4. Run the compilied program.

These steps form the basis for programs that do more than send simple commands. More complex programs should contain additional routines, such as error checking and initializing, to support full I/O capabilities of the instrument.

Error checking

Error checking should be performed after every HP-IB library call. Each HP-IB command library call returns a value corresponding to the error status of the current operation. An error handler routine is a convenient way of checking errors. For example, the following error routine could be used:

```
void error_handle (int error, char *routine)
{
     if (error !=NOERR)
     {    /* If there is an error, print an error */
          /* message and exit */
          printf ("HP-IB error in call to
                %s: %d, %s\n", routine, error,
               errstr(error));
          exit(1);
     }
     return;    /* No error, so return normally */
}
```

Using the above error routine, a command, for example, can be sent to the instrument as follows:

```
error_handle (IOOUTPUTS (716L,"PRES;",5), "IOOUTPUTS");
```

`error_handle`: Passes an error number and appropriate display string to the error handling routine.

`IOOUTPUTS(716L,"PRES;",5)`: Executes the HP-IB string data output command. 716L is the address. The data is directed to interface 7 (HP-IB) and out to the device at address 16 (the instrument). The "L" is required by the routine, which expects a long-integer. The command "`PRES;`" is the factory preset command for the instrument. "5" indicates that the command sent is five characters long.

Note

Throughout this guide, the term HP-IB library command refers to an HP 82335A HP-IB command library function. The term instrument command refers to a set of commands which the instrument is programmed to process. All HP-IB library commands begin with the prefix `IO-`. For instance, `IOOUTPUTS` is an HP-IB library command, while `PRES` is an instrument command.

Each instrument command sent is executed automatically upon receipt, taking precedence over manual control. A command applies only to the active channel, except where functions are coupled between channels, just as with front panel operation. Most commands are equivalent to front panel functions.

The network analyzer automatically goes into remote mode when sent a command with the HP-IB command library `IOOUTPUTS` statement. This turns on the remote (R) and listen (L) HP-IB status indicators. In remote mode, all front panel keys except the local key are ignored. Pressing the [LOCAL] key returns the instrument to manual operation, unless the HP-IB library command `IOLLOCKOUT (7L)` has been issued. This command puts the instrument into local lockout. The only way to get out of local lockout is to execute the `IOLLOCAL(7L)` command, or to cycle instrument power, which will return to local operation.

The debug mode can be used to aid in trouble-shooting systems. When the debug mode is on, incoming HP-IB commands scroll across the instrument display. To turn the mode on manually, press [LOCAL], `HP-IB DIAG ON`. To turn it on over HP-IB, execute:

```
error_handle (IOOUTPUTS (716L,"DEBUON;",7), "IOOUTPUTS")
```

Command interrogate

When the operator has changed a setting from the front panel, the computer can find out the value of the new setting using the command interrogate function. If a question mark is appended to the root of a command, the value of that function is sent. For instance, `POWE -20 DB` sets the output power to −20 dBm, and `POWE?` outputs the current RF output power at the test port.

On/off commands can be also be interrogated. The reply is a one if the function is on, a zero if it is off. Similarly, if a command controls a function that is underlined on the network analyzer display when active, interrogating that command yields a one if the command is underlined, a zero if it is not. For example, there are nine options on the format menu, but only one is underlined at a time. The underlined option will return a one when interrogated.

Held commands

When the network analyzer is executing a command that cannot be interrupted, it will hold off processing new HP-IB commands. It will fill the 16 character input buffer, and then halt HP-IB until the held command has completed execution. This action will be transparent to a program unless HP-IB timeouts have been set with the `IOTIMEOUT (7L,timeout_value)` command.

While a held command is executing, the instrument will still service the HP-IB interface commands, such as `IOSPOLL(716L,response_variable)`, `IOCLEAR (716L)`, and `IOABORT(7L)`. Executing `IOCLEAR(716L)` or `IOCLEAR(7L)` will abort a command hold off, leaving the held command to complete execution as if it had begun from the front panel. These commands also clear the input buffer, destroying any commands received after the held command. If the network analyzer has halted the bus because its input buffer was full, `IOABORT(7L)` will release the bus.

Operation complete

Occasionally, there is a need to find out when certain operations have been completed. For instance, a program should not have the operator connect the next calibration standard while the instrument is still measuring the current one.

To provide such information, the network analyzer has an "Operation Complete" reporting mechanism that will indicate when certain key commands have completed operation. The mechanism is activated by sending either `OPC` or `OPC?` immediately before a command. Not all commands can be preceded with `OPC` or `OPC?` If the operation complete mechanism was interrogated with `OPC?`, the network analyzer will output a one when the command completes execution. If `OPC` is used, the instrument will not output a response indicating an operation complete, instead it will set bit 0 of the status byte (see Figure 4-4)

The following procedure, when called, only returns when it is has received a response following an `OPC()` command:

```
void opc ()
{
   int   one=1;
   char  reply;

   error_handle (IOENTERS (716L,&reply,&one), "IOENTERS");
}
```

For example, the following line sequence will not continue until a single sweep has been completed:

```
error_handle (IOOUTPUTS (716L,"SWET3S;OPC?;SING;",17), "IOOUTPUTS");
opc ();
```

The first line causes the instrument to single sweep for 3 seconds. Before the single sweep command is executed, it is preceded with `OPC?` which asks the instrument to output a one when done with the following command. The next line calls the procedure `OPC()` which will wait until the instrument outputs a response.

Note that to use the `OPC ()` routine, an instrument command must be preceded with `OPC?` and not `OPC`.

Preparing for HP-IB control

At the beginning of a program, the instrument has to be taken from an unknown state and brought under computer control. One way to do this is with an abort/clear sequence. IOABORT(7L) is used to halt bus activity and return control to the computer. IOCLEAR(716L) will then prepare the instrument to receive commands by clearing syntax errors, the input command buffer, and any messages waiting at the output. The abort/clear sequence makes the instrument ready to receive HP-IB commands.

The next step is to set it to a known state. The most convenient way to do this is to send PRES, which returns the instrument to the factory preset state. If the factory preset cannot be used and the status reporting mechanism is going to be used, CLES can be sent to clear all of the status reporting registers and their enables. The user preset can be recalled by RECA5.

For example, the following initialize routine can be used to set up the instrument:

```
void initialize ()
{
   error_handle (IOABORT (7L), "IOABORT");
   error_handle (IOCLEAR (716L), "IOCLEAR");
   error_handle (IOTIMEOUT (7L,15.0), "IOTIMEOUT");
   error_handle (IOOUTPUTS (716L,"PRES;",5), "IOOUTPUTS");
}
```

This routine brings the network analyzer to a known state, ready to respond to HP-IB control. The IOTIMEOUT (7L,15.0) sets a timeout of 15 seconds, long enough for most commands. The timeout value passed must be a floating point integer, so a decimal point must be included.

The network analyzer will not respond to HP-IB commands unless the remote line is asserted. When the remote line is asserted and the network analyzer is addressed to listen, it automatically goes into remote mode. Remote mode means that all the front panel keys are disabled except [LOCAL] and the line power switch. IOABORT (7L) asserts the remote line, which remains asserted until a IOLOCAL (7L) statement is executed. Another way to assert the remote line is to execute IOREMOTE (716L). This statement asserts remote operation and addresses the network analyzer to listen.

Measurement Programming

The previous sections outlined some basic programming techniques, along with demonstrating how to send commands to the network analyzer. The next step is to organize the commands into a measurement sequence. A typical measurement sequence is shown in Figure 1-2.

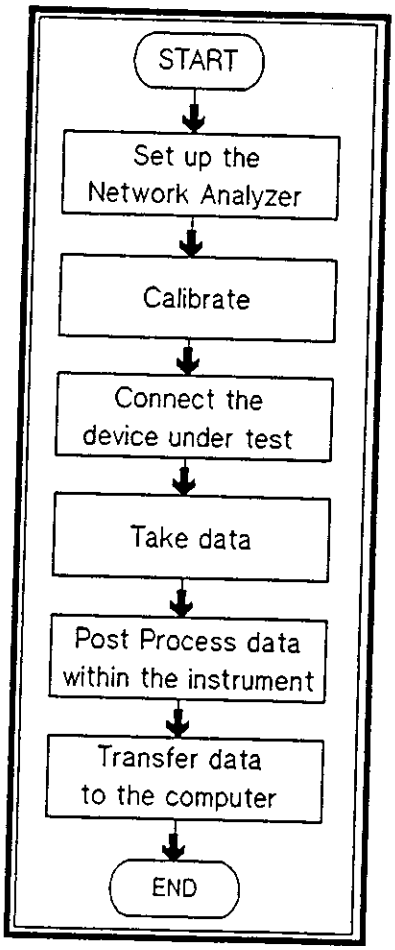

Figure 1-2. Typical Measurement Sequence

- Set up the network analyzer

  Define the measurement by setting all of the basic measurement parameters. These include all the stimulus parameters: sweep type, frequency, sweep time, number of points, and RF power level. They also include the parameter to be measured, and both IF averaging and IF bandwidth. These parameters define the way data is gathered and processed within the instrument. Each time one of the above parameters is changed, a new sweep must be triggered.

  There are other parameters that can be set within the instrument that do not affect data gathering directly, such as smoothing, setting trace resolution (scale), or performing trace math. These functions are classed as post processing functions: they can be changed with the instrument in hold mode, and the data will correctly reflect the current state.

  The save/recall registers is a rapid way of setting up an entire instrument state.

- Calibrate

Measurement calibration is normally performed once the instrument state has been defined. Measurement calibration is not required to make a measurement, but it does improve the measurement accuracy.

There are several ways to calibrate the instrument.

- The simplest is to stop the program and have the operator perform the calibration from the front panel.
- Alternatively, the computer can be used to guide the operator through the calibration, as discussed in Example 2A, S_{11} 1-port calibration and Example 2B, Full 2-port calibration.
- The last option is to transfer calibration data from a previous calibration back into the instrument as discussed in Example 6B, Reading calibration data.

- Connect the device under test

  The computer can be used to prompt the operator to connect and adjust the device and it can be also used to speed the adjustment process by setting up such functions as limit testing, bandwidth searches, and trace statistics.

- Take data

  Once the device is connected and adjusted, measure its frequency response, and store the data within the instrument so that there is a valid trace to analyze.

  The single sweep command `SING` is designed to ensure a valid sweep. All stimulus changes are completed before the sweep is started, and the HP-IB hold state is not released until the formatted trace is displayed. When the sweep is completed the instrument is put into hold, freezing the data inside the instrument. A single sweep can be preceded with `OPC?`; therefore, it is easy to determine when the sweep has been completed.

  The number of groups commands, `NUMGn`, is designed to work the same as a single sweep, except that it triggers n sweeps. This is useful, for example, in making a measurement with an averaging factor n (n can take on values between 1 and 999.) Both single sweep and number of groups restart averaging.

- Post process data within the instrument

  With valid data to operate on, the post-processing functions can be used. Referring ahead to Figure 2-5, any function that affects the data after the error correction stage can be used. The most useful functions are trace statistics, marker searches, electrical delay offset, time domain, and gating. If a 2-port calibration is active, then any of the four S-parameters can be viewed without taking a new sweep.

- Transfer data to the computer

  Lastly, read the results out of the instrument. All the data output commands are designed to ensure that the data transmitted reflects the current state of the instrument:

  - `OUTPDATA`, `OUTPRAWn`, and `OUTPFORM`, etc. will transmit entire data traces.
  - `OUTPLIML`, `OUTPLIMM`, and `OUTPLIMF` will transmit limit testing results.
  - `OUTPMARK` will transmit the currently active marker's results. This command will activate a marker if one is not already selected.
  - `OUTPMSTA` will transmit the statistics that have been calculated for data between the active marker and the delta reference marker. If there is no delta reference, the entire trace data is used.

- OUTPMWID will transmit the results of a bandwidth search.

Data transfer is discussed further in Examples 3A through 3C.

2

Basic Programming Examples

Example 1: Setting up a basic measurement

In general, the procedure for setting up measurements on the network analyzer via HP-IB follows the same sequence as if the setup was performed manually. There is no required order, as long as the desired frequency range, number of points and power level are set prior to performing the calibration.

This example illustrates how a basic measurement can be set up. The program will first select the desired S-parameter, the measurement format, and then the frequency range. Performing calibrations is described later.

Since the standard network analyzer has a frequency resolution of 100 kHz, it is required that all of the data points in the sweep be at some integer multiple of 100 kHz (10.1 MHz, 2.0011 GHz, 19.9985 GHz for example). Therefore, the actual frequencies that are set may be slightly different from those specified by the user. By interrogating the instrument, the user can determine the actual values of the start and stop frequencies. Note that if you are using a network analyzer which has 1 Hz frequency resolution, (Option 001), the actual frequencies set by the analyzer will be identical to those specified using the START and STOP commands.

Caution

The programming examples found in this guide are for **example** purposes only. They may require modification to work with your particular personal computer.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 1  */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:
70:  void error_handle (int, char *);
80:  void output (char *);
90:  void initialize(void);
100:
110: void error_handle (int error, char *routine)
120: {
130:    if (error !=NOERR)
140:    {
150:      printf ("HP-IB error in call to %s: %d, %s\n",
                routine, error, errstr(error));
```

```
160:       exit(1);
170:     }
180:     return;
190: }
200:
210: void output (char *sendstr)
220: {
230:     error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
240: }
250:
260: void initialize ()
270: {
280:     error_handle (IOTIMEOUT (isc,5.0), "IOTIMEOUT");
290:     error_handle (IOABORT (isc), "IOABORT");
300:     error_handle (IOCLEAR (isc), "IOCLEAR");
310:     output ("PRES;");
320: }
330:
340: main ()
350: {
360:     char  cmd[80];
370:     float f_start,f_stop;
380:
390:     initialize ();
400:     output ("CHAN1;S11;LOGM;");
410:     output ("CHAN2;S11;PHAS;DUACON;");
420:     printf ("Enter start frequency (GHz): ");
430:     scanf ("%f",&f_start);
440:     printf ("Enter stop frequency (GHz): ");
450:     scanf ("%f",&f_stop);
460:     sprintf (cmd,"STAR%fGHZ;STOP%fGHZ;",f_start,f_stop);
470:     output (cmd);
480:     output ("CHAN1;AUTO;CHAN2;AUTO;");
490: }
```

Figure 2-1. Sample Program: Basic Programming Measurement

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.

Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.
Line 90	Function prototype for the `initialize ()` routine.
Line 110	Define a routine that handles errors returned from the HP-IB command library I/O functions.
Line 130	Check to see if there is an error.
Line 150	An error has occurred, so display a message and halt.
Line 180	No error has occurred, so return.
Line 210	Define a routine that outputs string commands and performs error trapping.
Line 230	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 260	Define a routine to initialize the instrument.
Line 280	Define a timeout value of five seconds.
Line 290	Abort any HP-IB transfers.
Line 300	Clear the instrument's HP-IB interface.
Line 310	Preset the instrument.
Line 340	`Main` declaration
Line 360	Declare the needed variables.
Line 390	Call the `initialize ()` routine.
Line 400	Output commands to the instrument to switch to channel one, and measure the log magnitude of S_{11}.
Line 410	Output commands to the instrument to switch to channel two, measure the phase of S_{11}, and display both channel one and channel two.
Line 420	Request the start frequency.
Line 430	Input the start frequency.
Line 440	Request the stop frequency.
Line 450	Input the stop frequency.
Line 460	Create an output string to contain commands to tell the analyzer what the start and stop frequencies are.
Line 470	Output this string.
Line 480	Autoscale both channel one and channel two.

Running the program

The program will set up a measurement of S_{11}, log magnitude on channel one, and S_{11}, phase on channel two, and turn on the dual channel display mode. When prompted for the start and stop frequencies, enter any value in GHz from .050 (50 MHz) to 13.5 GHz, 20 GHz, or 40 GHz. These will be entered in the instrument, and the actual frequencies that the instrument is set to will be displayed on the instrument.

Performing a measurement calibration

This section will demonstrate how to coordinate a measurement calibration over HP-IB. The HP-IB command sequence follows the same key sequence required to calibrate from the front panel: there is a command for each step.

The general key sequence is to select the calibration, measure the calibration standards, and then declare the calibration done. The actual sequence depends on the calibration kit and changes slightly for 2-port calibrations, which are divided into three calibration sub-sequences.

Calibration kits

The calibration kit definition tells the network analyzer what standards to use at each step of the calibration. The set of standards associated with a given calibration is termed a class. For example, measuring the short during a S_{11} 1-port calibration is one calibration step. All of the shorts that can be used for this calibration step make up the class, which is called class $S_{11}B$. For the 2.4 mm, 3.5 mm, and 7 mm calibration kits, class $S_{11}B$ has only one standard in it. For type-N calibration kits, class $S_{11}B$ has two standards in it: male and female shorts.

When doing a S_{11} 1-port calibration in 2.4 mm, 3.5 mm, or 7 mm, selecting SHORT automatically measures the short because there is only one standard in the class. When doing the same calibration in type-N, selecting SHORTS brings up a second menu, allowing the user to select which standard in the class is to be measured.

Doing an S_{11} 1-port calibration over HP-IB is very similar. In 2.4 mm, 3.5 mm, and 7 mm, sending `CLASS11B` will automatically measure the short. In type-N, sending `CLASS11B` brings up the menu with the male and female short options. To select a standard, use `STANA` or `STANB`. The `STAN` command is appended with the letters A through G, corresponding to the standards listed under softkeys 1 through 7, softkey 1 being the topmost softkey.

The `STAN` command can be preceded with `OPC?`. A command that calls a class can only be preceded with `OPC?` if that class has only one standard in it. If there is more than one standard in a class, the command that calls the class only brings up another menu, and there is no need to `OPC?` it.

Hence, both the manual and HP-IB calibration sequences depend heavily on which calibration kit is active.

Full 2-port calibrations

Each full 2-port measurement calibration is divided into three sub-sequences: transmission, reflection, and isolation. Each sub-sequence is treated like a calibration in its own right: each must be opened, have all the standards measured, and then be declared done. The opening and closing statements for the transmission sub-sequence are TRAN and TRAD. The opening and closing statements for the reflection sub-sequence are REFL and REFD. The opening and closing statements for isolation are ISOL and ISOD.

Example 2A: S_{11} 1-port calibration

To demonstrate coordinating a calibration over HP-IB, the following program does an S_{11} 1-port calibration using the HP 85052B 3.5 mm calibration kit. This program simplifies the calibration for the operator by giving explicit directions on the computer display.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 2A */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:
70:  void error_handle (int, char *);
80:  void output (char *);
90:  void opc (void);
100: void initialize (void);
110: void disp_prompt (char *);
120:
130: void error_handle (int error, char *routine)
140: {
150:    if (error !=NOERR)
160:    {
170:       printf ("HP-IB error in call to %s: %d,%s\n",
                routine, error, errstr(error));
180:       exit(1);
190:    }
200:    return;
210: }
220:
230: void output (char *sendstr)
240: {
250:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
260: }
270:
280: void opc ()
290: {
```

```
300:   int   one=1;
310:   char  reply;
320:
330:   error_handle (IOENTERS (instr,&reply,&one), "IOENTERS");
340: }
350:
360: void initialize ()
370: {
380:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
390:    error_handle (IOABORT (isc), "IOABORT");
400:    error_handle (IOCLEAR (isc), "IOCLEAR");
410:    output ("CLES;");
420: }
430:
440: void disp_prompt (char *prompt)
450: {
460:    char  ch;
470:
480:    printf ("%s",prompt);
490:    printf (", then press [RETURN]\n");
500:    ch=getche();
510: }
520:
530: main ()
540: {
550:    int  index;
560:
570:    initialize ();
580:    output ("CALK35MM;CLES;CALIS11;");
590:    disp_prompt ("Connect OPEN at port 1");
600:    output ("OPC?;CLASS11A;");
610:    opc ();
620:    disp_prompt ("Connect SHORT at port 1");
630:    output ("OPC?;CLASS11B;");
640:    opc ();
650:    disp_prompt ("Connect LOWBAND LOAD at port 1");
660:    output ("CLASS11C;OPC?;STANC;");
670:    opc ();
680:    disp_prompt ("Connect SLIDING LOAD at port 1");
690:    output ("STANB;");
700:    for (index=1; index<=5; index++)
710:    {
720:       disp_prompt ("Set SLIDE in position");
730:       output ("SLIS;");
740:    }
750:    output ("SLID;");
760:    printf("Computing calibration coefficients.\n");
770:    output ("DONE;OPC?;SAV1;");
780:    opc ();
790:    printf ("DONE\n");
```

```
800: }
```

Figure 2-2. Sample Program: S_{11} 1-Port Calibration

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.
Line 90	Function prototype for the `opc ()` routine.
Line 100	Function prototype for the `initialize ()` routine.
Line 110	Function prototype for the `disp_prompt ()` routine.
Line 130	Define a routine that handles errors returned from the HP-IB command library I/O functions.
Line 150	Check to see if there is an error.
Line 170	An error has occurred, so display a message and halt.
Line 200	No error has occurred, so return.
Line 230	Define a routine that outputs string commands and performs error trapping.
Line 250	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 280	Define a routine that when called will only return when it receives a response from the instrument. This routine is called after an `OPC?` command has been issued.
Line 310	Define a variable to hold the response.
Line 330	Input the response into the variable reply and do nothing with it.
Line 360	Define a routine to initialize the instrument.
Line 380	Define a timeout value of five seconds.
Line 390	Abort any HP-IB transfers.
Line 400	Clear the instrument's HP-IB interface.
Line 410	Clear the instrument's status.

Line 440	Define a routine to display a prompt and wait for (RETURN) to be pressed.
Line 480	Display the message prompt.
Line 500	Wait for a key to be pressed.
Line 530	`Main` declaration
Line 550	Declare the needed variables.
Line 560	Call the `initialize ()` routine.
Line 580	Select the 3.5 mm calibration kit, clear the instrument status, and open the S_{11} 1-port calibration.
Line 590	Display a prompt to connect an OPEN at port 1.
Line 600	Measure the standard. Since there is only one choice in this class, the `CLASS` command is OPC'able. Using the `OPC?` command causes the program to wait until the standard has been measured before continuing. This is very important because the prompt to connect the next standard should only appear after the first standard is measured.
Line 610	Wait for the `OPC?` command to return a response.
Line 620	Display a prompt to connect a SHORT at port 1.
Line 630	Measure the standard.
Line 640	Wait for the `OPC?` command to return a response.
Line 650	Display a prompt to connect a LOWBAND LOAD at port 1.
Line 660	Measure the standard. Since there is more than one standard in the loads class, the program must identify the specific standard within that class. The lowband load is the third softkey selection from the top in the menu; so to select a lowband load as the standard use the command `STANC`.
Line 670	Wait for the `OPC?` command to return a response.
Line 680	Display a prompt to connect a SLIDING LOAD at port 1.
Line 690	Select the appropriate softkey.
Line 700	It will require five different positions of the sliding load to properly characterize the directivity error term.
Line 720	Display a prompt to set the SLIDE.
Line 730	Measure the slide standard.
Line 750	Tell the instrument that the sliding load calibration has been completed.
Line 770	Affirm the completion of the calibration, and save the calibration.

Running the program

The program assumes that the test port being calibrated is a 3.5 mm, either male or female. The program interacts with the operator through the computer. When the measurement calibration is complete, it will display DONE.

Before running the program, set up the desired instrument state. This program does not modify the instrument state in any way. Run the program, and connect the standards as prompted. When the standard is connected, press (RETURN) to measure it.

Example 2B: Full 2-port measurement calibration

The following example shows how to perform a full 2-port measurement calibration using the HP 85052D calibration kit. In this example, the calibration process allows the removal of both the forward and reverse error terms, so that all four S-parameters of the device under test can be measured. Since the HP 85052D calibration kit is used, a broadband load will be used instead of the sliding load used in Example 2A. Use of the broadband load results in a more convenient calibration, since only one measurement is required for the load calibration, as opposed to five measurements when the sliding load is used.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 2B */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:
70:  void error_handle (int, char *);
80:  void output (char *);
90:  void opc (void);
100: void initialize (void);
110: void disp_prompt (char *);
120:
130: void error_handle (int error, char *routine)
140: {
150:    if (error !=NOERR)
160:    {
170:       printf ("HP-IB error in call to %s: %d,%s\n",
                routine,error, errstr(error));
180:       exit(1);
190:    }
200:    return;
210: }
220:
230: void output (char *sendstr)
240: {
```

```
250:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
260: }
270:
280: void opc ()
290: {
300:    int   one=1;
310:    char  reply;
320:
330:    error_handle (IOENTERS (instr,&reply,&one), "IOENTERS");
340: }
350:
360: void initialize ()
370: {
380:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
390:    error_handle (IOABORT (isc), "IOABORT");
400:    error_handle (IOCLEAR (isc), "IOCLEAR");
410:    output ("CLES;");
420: }
430:
440: void disp_prompt (char *prompt)
450: {
460:    char  ch;
470:
480:    printf ("%s",prompt);
490:    printf (", then press [RETURN]\n");
500:    ch=getche();
510: }
520:
530: main ()
540: {
550:    initialize ();
560:    output ("CALK35MM;CLES;CALIFUL2;REFL;");
570:    disp_prompt ("Connect OPEN at port 1");
580:    output ("OPC?;CLASS11A;");
590:    opc ();
600:    disp_prompt ("Connect SHORT at port 1");
610:    output ("OPC?;CLASS11B;");
620:    opc ();
630:    disp_prompt ("Connect BROADBAND LOAD at port 1");
640:    output ("CLASS11C;OPC?;STANA;");
650:    opc ();
660:    disp_prompt ("Connect OPEN at port 2");
670:    output ("OPC?;CLASS22A;");
680:    opc ();
690:    disp_prompt ("Connect SHORT at port 2");
700:    output ("OPC?;CLASS22B;");
710:    opc ();
720:    disp_prompt ("Connect BROADBAND LOAD at port 2");
730:    output ("CLASS22C;OPC?;STANA;");
740:    opc ();
750:    output ("REFD;");
```

```
760:    printf ("Computing reflection calibration coefficients\n");
770:    output ("TRAN;");
780:    disp_prompt ("Connect THRU [port 1 to port 2]");
790:    printf ("Measuring forward transmission\n");
800:    output ("OPC?;FWDT;");
810:    opc ();
820:    output ("OPC?;FWDM;");
830:    opc ();
840:    printf ("Measuring reverse transmission\n");
850:    output ("OPC?;REVT;");
860:    opc ();
870:    output ("OPC?;REVM;");
880:    opc ();
890:    output ("TRAD;");
900:    disp_prompt ("Isolate test ports");
910:    output ("AVERFACT16;AVERON;ISOL;");
920:    printf ("Measuring reverse isolation\n");
930:    output ("OPC?;REVI;");
940:    opc ();
950:    printf ("Measuring forward isolation\n");
960:    output ("OPC?;FWDI;");
970:    opc ();
980:    output ("ISOD;AVEROFF;");
990:    printf("Computing calibration coefficients\n");
1000:   output ("OPC?;SAV2;");
1010:   opc ();
1020:   printf ("DONE");
1030: }
```

Figure 2-3. Sample Program: Full 2-Port Measurement Calibration

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.
Line 90	Function prototype for the `opc ()` routine.
Line 100	Function prototype for the `initialize ()` routine.
Line 110	Function prototype for the `disp_prompt ()` routine.

Line 130	Define a routine that handles errors returned from the HP-IB command library I/O functions.
Line 150	Check to see if there is an error.
Line 170	An error has occurred, so display a message and halt.
Line 200	No error has occurred, so return.
Line 230	Define a routine that outputs string commands and performs error trapping.
Line 250	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 280	Define a routine that when called will only return when it receives a response from the instrument. This routine is called after an `OPC?` command has been issued.
Line 310	Define a variable to hold the response.
Line 330	Input the response into the variable reply and do nothing with it.
Line 360	Define a routine to initialize the instrument.
Line 380	Define a timeout value of five seconds.
Line 390	Abort any HP-IB transfers.
Line 400	Clear the instrument's HP-IB interface.
Line 410	Clear the instrument's status.
Line 440	Define a routine to display a prompt and wait for (RETURN) to be pressed.
Line 480	Display the message prompt.
Line 500	Wait for a key to be pressed.
Line 530	`Main` declaration
Line 550	Call the `initialize ()` routine.
Line 560	Select the 3.5 mm calibration kit, clear the instrument status, and open the full 2-port calibration.
Line 570	Display a prompt to connect an OPEN at port 1.
Line 580	Measure the standard.
Line 590	Wait for the `OPC?` command to return a response.
Line 600	Display a prompt to connect a SHORT at port 1.
Line 610	Measure the standard.
Line 620	Wait for the `OPC?` command to return a response.
Line 630	Display a prompt to connect a BROADBAND LOAD at port 1.
Line 640	Measure the standard.
Line 650	Wait for the `OPC?` command to return a response.
Line 660	Display a prompt to connect an OPEN at port 2.

Line 670	Measure the standard.
Line 680	Wait for the OPC? command to return a response.
Line 690	Display a prompt to connect a SHORT at port 2.
Line 700	Measure the standard.
Line 710	Wait for the OPC? command to return a response.
Line 720	Display a prompt to connect a BROADBAND LOAD at port 2.
Line 730	Measure the standard.
Line 740	Wait for the OPC? command to return a response.
Line 750	Close the reflection calibration sub-sequence.
Line 770	Open the transmission calibration sub-sequence.
Line 780	Display a prompt to connect a THRU connection.
Line 800	Measure forward transmission.
Line 810	Wait for the OPC? command to return a response.
Line 820	Measure forward load match.
Line 830	Wait for the OPC? command to return a response.
Line 850	Measure reverse transmission.
Line 870	Measure reverse load match.
Line 890	Close the transmission calibration sub-sequence.
Line 900	Display a prompt to isolate test ports.
Line 910	Define an averaging factor of 16, turn on averaging, and open the isolation calibration sub-sequence.
Line 930	Measure reverse isolation.
Line 960	Measure forward isolation.
Line 980	Close the isolation calibration sub-sequence, and turn off averaging.
Line 1010	Wait until the instrument calculates the calibration coefficients before continuing.

Running the program

The program assumes that the test ports being calibrated are 3.5 mm, either male or female, and that the HP 85052D 3.5 mm economy calibration kit is to be used (no sliding loads). The program will display DONE when the measurement calibration is complete.

Before running the program, set up the desired instrument state. This program does not modify the instrument state in any way. Run the program, and connect the standards as prompted. When the standard is connected, press (RETURN) to measure it.

```
140:    {
150:       printf ("HP-IB error in call to %s: %d, %s\n",
                  routine, error, errstr(error));
160:       exit(1);
170:    }
180:    return;
190: }
200:
210: void output (char *sendstr)
220: {
230:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
240: }
250:
260: void initialize ()
270: {
280:    error_handle (IOTIMEOUT (isc,5.0), "IOTIMEOUT");
290:    error_handle (IOABORT (isc), "IOABORT");
300:    error_handle (IOCLEAR (isc), "IOCLEAR");
310:    output ("CLES;");
320: }
330:
340: main ()
350: {
360:    float val[3];
370:    int   length=3;
380:
390:    initialize ();
400:    output ("SING;MARK1;SEAMIN;FORM5;OUTPMARK;");
410:    error_handle (IOENTERA (instr,val,&length), "IOENTERA");
420:    printf("Value 1: %f\nValue 2: %f\nStimulus: %f\n",
              val[0],val[1],val[2]);
430: }
```

Figure 2-4. Sample Program: Data Transfer Using Markers

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.

Line 90	Function prototype for the `initialize ()` routine.
Line 110	Define a routine that handles errors returned from the HP-IB command library I/O functions.
Line 130	Check to see if there is an error.
Line 150	An error has occurred, so display a message and halt.
Line 180	No error has occurred, so return.
Line 210	Define a routine that outputs string commands and performs error trapping.
Line 230	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 260	Define a routine to initialize the instrument.
Line 280	Define a timeout value of five seconds.
Line 290	Abort any HP-IB transfers.
Line 300	Clear the instrument's HP-IB interface.
Line 310	Clear the instrument's status.
Line 340	`Main` declaration
Line 360	Declare the needed variables.
Line 390	Call the `initialize ()` routine.
Line 400	Perform a single trace, turn on marker one, place marker one at the single trace minimum, and output the marker data.
Line 410	Input three values, the stimulus value, value one and value two.
Line 420	Display both values, and the stimulus.

Running the program

The values displayed by the computer should agree with the marker values, except that the second value displayed will be meaningless in some formats. To see the possibilities for different values, run the program three times: once in log magnitude format, once in phase format, and once in Smith chart format. To change display format, press (LOCAL), (FORMAT), and then select the desired format.

Trace transfer

Getting trace data with a computer can be broken down into three steps:

1. Setting up the receive array.
2. Telling the network analyzer to transmit the data.
3. Accepting the transferred data.

Data is always stored in values, to accommodate real/imaginary pairs, for each data point. Therefore, the receiving array has to be two elements wide, and as deep as the number of points. The memory space for this array must be declared before any data is to be transferred to the computer.

Data formats

The network analyzer can transmit data over HP-IB in five different formats. Of the five formats, Form 5 is the most appropriate for personal computers.

- Form 1

  Internal binary format. In this mode, each *point* takes 6 bytes. This means that a 201 point transfer takes 1,206 bytes. Re-formatting must be done in order to decode the information. This is the format the network analyzer uses to store data. Form 1 also has a four byte header.

- Form 2

  IEEE-754 32-bit floating point format. In this mode, each *number* takes 4 bytes. This means that a 201 point transfer takes 1608 bytes. This form also has a four byte header.

- Form 3

  IEEE-754 64-bit floating point format. In this mode, each *number* takes 8 bytes. This means that a 201 point transfer takes 3,216 bytes. This form also has a four byte header.

- Form 4

  ASCII data transfer format. In this mode, each *number* is sent as a 24 character string, each character being a digit, sign, or decimal point. Since there are two numbers per point, a 201 point transfer takes 9,648 bytes. This form does not have a four byte header.

- Form 5

  PC-compatible 32-bit floating point format. This mode is a modification of the IEEE-754 32-bit floating point format with the byte order reversed. Therefore, a 201 point transfer takes 1,608 bytes. This form also has a four byte header. In this mode, a MS-DOS personal computer can store data internally without reformatting it.

Data levels

Different levels of data can be read out of the instrument (Refer to Figure 2-5)

- Raw Data.

  This is basic measurement data with no error correction applied. If a full 2-port measurement calibration is ON, there are actually four raw arrays kept: one for each raw S-parameter. The data is read out with the command `OUTPRAW{n}`, where n ranges from 1 to 4. Normally, only raw 1 is available and it holds the current parameter. If a 2-port calibration is ON, the four arrays refer to S_{11}, S_{21}, S_{12}, and S_{22} respectively. This data is always in real/imaginary pairs.

- Error-corrected data.

  This is data with error-correction applied. The array corresponds to the currently measured parameter, and is always in real/imaginary pairs. The error-corrected data is read out with `OUTPDATA`. `OUTPMEMO` reads the trace memory if available, which is also error-corrected data. Note that neither raw nor error-corrected data reflect such post-processing functions as electrical delay offset, trace math, or time domain gating.

- Formatted data.

  This is the array of data actually being displayed. It reflects all post-processing functions. The units of the array read out depends on the current display format. Refer to Table 2-1 for the various units as a function of display format. The formatted data is read out with `OUTPFORM`.

- Calibration coefficients.

  The results of a calibration are arrays of calibration coefficients which are used in the error-correction routines. Each array corresponds to a specific error term in the error model. The calibration coefficients can be read out with `OUTPCALC{n}`, where n ranges from 1 to 12.

  Formatted data is generally the most useful, being the same information seen on the display. However if the post processing is unneeded or unwanted, as may be the case with smoothing, error-corrected data is more desirable. Error-corrected data also give you the opportunity to put the data into the instrument and apply post-processing at a later time.

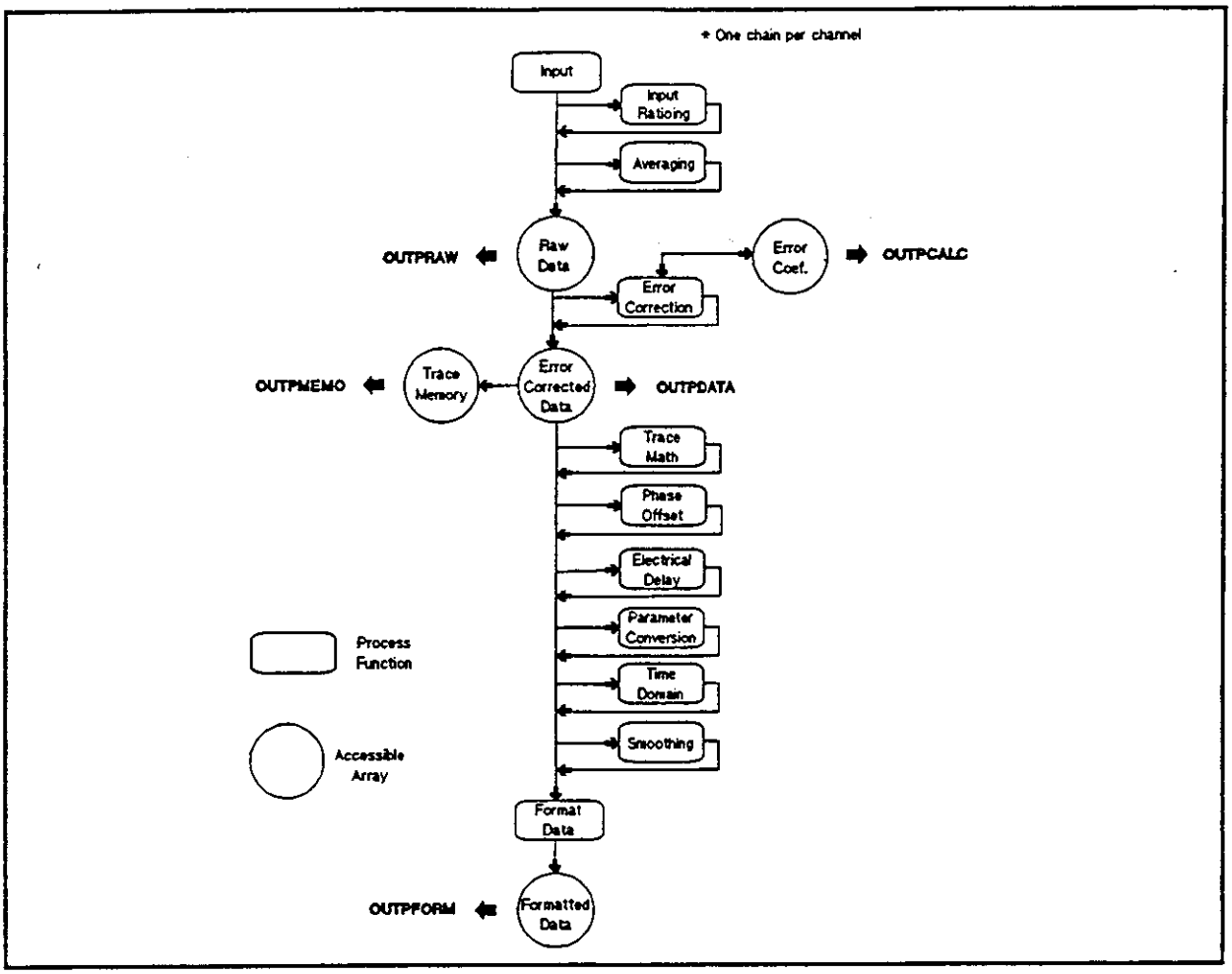

Figure 2-5. Data processing chain

Example 3B: Data transfer using Form 4 (ASCII transfer)

In Form 4, each number is sent as a 24 character string, each character being a digit or decimal point. Since there are two numbers per point, a 201 point transfer in Form 4 takes 9,648 bytes

```
1:   /*  HP 8719C, 8720C, 8722A  QuickC IPG Program 3B */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:
70:  void error_handle (int, char *);
80:  void output (char *);
```

```
 90:  void opc (void);
 100: void initialize (void);
 110:
 120: void error_handle (int error, char *routine)
 130: {
 140:    if (error !=NOERR)
 150:    {
 160:       printf ("HP-IB error in call to %s: %d, %s\n",
                   routine, error, errstr(error));
 170:       exit(1);
 180:    }
 190:    return;
 200: }
 210:
 220: void output (char *sendstr)
 230: {
 240:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
 250: }
 260:
 270: void opc ()
 280: {
 290:    int   one=1;
 300:    char  reply;
 310:
 320:    error_handle (IOENTERS (instr,&reply,&one), "IOENTERS");
 330: }
 340:
 350:
 360: void initialize ()
 370: {
 380:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
 390:    error_handle (IOABORT (isc), "IOABORT");
 400:    error_handle (IOCLEAR (isc), "IOCLEAR");
 410:    error_handle (IOMATCH (isc,'\n',0), "IOMATCH");
 420:    output ("PRES;");
 430: }
 440:
 450: main ()
 460: {
 470:    char  ascii_dat[550];
 480:    int   elements=550,i,j;
 490:
 500:    initialize();
 510:    output ("POIN 11;OPC?;SING;");
 520:    opc ();
 530:    output ("FORM4;OUTPFORM;");
 540:    error_handle (IOENTERS (instr,ascii_dat,&elements), "IOENTERS");
 550:    for (i=0; i<528; i=i+50)
 560:    {
 570:      printf ("\nPoint: %2d Value 1: ",(i/50)+1);
 580:      for (j=1; j<=24; j=j+1)
```

```
590:          printf ("%c",ascii_dat[i+j]);
600:       printf (" Value 2: ");
610:       for (j=1; j<=24; j=j+1)
620:          printf ("%c",ascii_dat[i+j+24]);
630:    }
640: }
```

Figure 2-6. Sample Program: Data Transfer Using Form 4

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.
Line 90	Function prototype for the `opc ()` routine.
Line 100	Function prototype for the `initialize ()` routine.
Line 120	Define a routine that handles errors returned from the HP-IB command library I/O functions
Line 140	Check to see if there is an error.
Line 160	An error has occurred, so display a message and halt.
Line 190	No error has occurred, so return.
Line 220	Define a routine that outputs string commands and performs error trapping.
Line 240	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 270	Define a routine that when called will only return when it receives a response from the instrument. This routine is called after an `OPC?` command has been issued.
Line 310	Define a variable to hold the response.
Line 320	Input the response into the variable reply and do nothing with it.
Line 360	Define a routine to initialize the instrument.
Line 380	Define a timeout value of 45 seconds.
Line 390	Abort any HP-IB transfers.

Line 400	Clear the instrument's HP-IB interface.
Line 410	Disable character matching. This command defines the character used by `IOENTERB` and `IOENTERS` for termination. The "lf" enter terminator should be turned off because "lf" is a valid binary value.
Line 420	Preset the instrument.
Line 450	`Main` declaration
Line 470	Define the needed variables.
Line 500	Call the `initialize ()` routine.
Line 510	Set the number of points sampled to eleven and perform a single trace.
Line 520	Wait for the `OPC?` command to return a response.
Line 530	Define the data format as Form 4 (ASCII transfer), and request the instrument's formatted data.
Line 540	Input the data from the instrument.
Line 550	Define a loop to display the data. Each point has 50 bytes associated with it. There are two values which occupy 24 bytes each, and two bytes to signal the termination of the string.
Line 580	Loop 24 times to display the first value.
Line 590	Display the first value.
Line 610	Loop 24 times to display the second value.
Line 620	Display the second value.

Running the program

Changing the display format will change the data sent with the `OUTPFORM` transfer. Refer to Table 2-1 for a list of how data is provided by format. The data from `OUTPFORM` reflects all the post processing such as time domain, gating, electrical delay, trace math, smoothing, etc.

Relating the data from a linear frequency sweep to frequency can be done by interrogating the start frequency, the frequency span, and the number of points. Given that information, the frequency of point N in a linear frequency sweep is just:

$F = Start\_frequency + (N\text{-}1)\ X\ Span/(Points\text{-}1)$

Alternatively, it is possible to read the frequencies directly out of the instrument with the `OUTPLIML` command. `OUTPLIML` reports the limit test results by transmitting the stimulus point tested, a number indicating the limit test results, and then the upper and lower limits at that stimulus point, if available. The number indicating the limit results is a -1 for no test, 0 for fail, and 1 for pass. If there are no limits available, zeros are transmitted as the upper and lower limits.

Example 3C: Data transfer using Form 5, PC-compatible 32-bit floating point format.

Form 5 transfers two numbers for each trace point. Each number is a four byte real number. Form 5 also has additional four byte header. The first two bytes are the ASCII characters "#A" that indicate that a fixed block length transfer follows, and the next two bytes form an integer containing the number of bytes in the block to follow. Thus, a 201 point transfer requires 1612 (201*4*2+4) bytes.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 3C */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:
70:  void error_handle (int, char *);
80:  void output (char *);
90:  void initialize (void);
100: void opc (void);
110:
120: void opc ()
130: {
140:     int   one=1;
150:     char  reply;
160:
170:     error_handle (IOENTERS (instr,&reply,&one), "IOENTERS");
180: }
190:
200:
210: void error_handle (int error, char *routine)
220: {
230:     if (error !=NOERR)
240:     {
250:       printf ("HP-IB error in call to %s: %d, %s\n",
                    routine, error, errstr(error));
260:       exit(1);
270:     }
280:     return;
290: }
300:
310: void output (char *sendstr)
320: {
330:     error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
340: }
350:
360: void initialize ()
```

```
370: {
380:    error_handle (IOTIMEOUT (isc,15.0), "IOTIMEOUT");
390:    error_handle (IOABORT (isc), "IOABORT");
400:    error_handle (IOCLEAR (isc), "IOCLEAR");
410:    error_handle (IOMATCH (isc,'\n',0), "IOMATCH");
420:    output ("CLES;");
430: }
440:
450: main ()
460: {
470:    int   bytes,i,length;
480:    float data[50];
490:    char  header[2];
500:
510:    initialize();
520:    output ("POIN 11;OPC?;SING;");
530:    opc ();
540:    output ("FORM5;OUTPFORM;");
550:    length=2;
560:    error_handle (IOENTERS (instr,header,&length), "IOENTERS");
570:    error_handle (IOENTERB (instr,&bytes,&length,1), "IOENTERB");
580:    printf ("Header: %s\nNumber of bytes: %d\n\n",header,bytes);
590:    length=bytes;
600:    error_handle (IOENTERB (instr,data,&length,1), "IOENTERB");
610:    for (i=0; i<=20; i=i+2)
620:       printf ("Point: %5d,   Value 1: %f,   Value 2: %f\n",
                  (i/2)+1,data[i],data[i+1]);
630: }
```

Figure 2-7. Sample Program: Data Transfer Using Form 5

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.
Line 90	Function prototype for the `initialize ()` routine.
Line 100	Function prototype for the `opc ()` routine.

Line 120	Define a routine that when called will only return when it receives a response from the instrument. This routine is called after an `OPC?` command has been issued.
Line 150	Define a variable to hold the response.
Line 170	Input the response into the variable reply and do nothing with it.
Line 210	Define a routine that handles errors returned from the HP-IB command library I/O functions
Line 230	Check to see if there is an error.
Line 250	An error has occurred, so display a message and halt.
Line 280	No error has occurred, so return.
Line 310	Define a routine that outputs string commands and performs error trapping.
Line 330	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 360	Define a routine to initialize the instrument.
Line 380	Define a timeout value of 15 seconds.
Line 390	Abort any HP-IB transfers.
Line 400	Clear the instrument's HP-IB interface.
Line 410	Disable character matching. This command defines the character used by `IOENTERB` and `IOENTERS` for termination. The "lf" enter terminator should be turned off because "lf" is a valid binary value.
Line 420	Clear the instrument's status.
Line 450	`Main` declaration
Line 470	Define the needed variables.
Line 510	Call the `initialize ()` routine.
Line 520	Set the number of points sampled to eleven and perform a single trace.
Line 530	Wait for the `OPC?` command to return a response.
Line 540	Define the data format as Form 5 (PC-compatible 32-bit floating point integer transfer), and request the instrument's formatted data.
Line 550	Define a variable, `length`, to contain the number of bytes to read from the instrument.
Line 560	Input the header.
Line 570	Input the number of bytes in the block to follow. Put this in the variable, `bytes`.
Line 580	Display the header and the number of bytes.
Line 590	Assign the variable, `length`, the number of bytes to read.
Line 600	Read the data.

Line 610	Loop enough times to display all the data points.
Line 620	Display the point number, value one and value two.

Running the program

Run the program. It will set the number of points to eleven, and display the header and number of bytes required to input the data. It will also show the two values associated with each of the eleven data points.

3

Advanced Programming Examples

Using list frequency mode

The list frequency mode allows selection of specific points or frequency spacing between points at which measurements are to be made. Sampling specific points reduces the measurement time, since additional time is not spent measuring device characteristics at unnecessary frequencies.

Example 4: Setting up a list frequency sweep

This example shows how to create a list frequency table and transmit it to the network analyzer. The command sequence for entering a list frequency table imitates the key sequence followed when entering a table from the front panel: there is a command for every key press. Editing a segment is also the same as the key sequence, but the network analyzer automatically re-orders each edited segment in order of increasing start frequency.

The list frequency table is also carried as part of the learn string. While it cannot be modified as part of the learn string, it can be stored and easily recalled.

This example takes advantage of the computer's capabilities to simplify creating, adding to, and editing the table. The table is entered and completely edited before being transmitted to the network analyzer.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 4A */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:  #include <graph.h>
40:
50:  #define isc     7L
60:  #define instr   716L
70:
80:  int   points[30],start[30],stop[30];
90:
100: void error_handle (int, char *);
110: void output (char *);
120: void initialize (void);
130: void getsegment (int);
140: int edit(void);
150:
160: void error_handle (int error, char *routine)
```

```
170: {
180:    if (error !=NOERR)
190:    {
200:       printf ("HP-IB error in call to %s: %d, %s\n",
                  routine, error, errstr(error));
210:       exit(1);
220:    }
230:    return;
240: }
250:
260: void output (char *sendstr)
270: {
280:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
290: }
300:
310: void initialize ()
320: {
330:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
340:    error_handle (IOABORT (isc), "IOABORT");
350:    error_handle (IOCLEAR (isc), "IOCLEAR");
360:    output ("PRES;");
370: }
380:
390: void getsegment (int number)
400: {
410:    _settextwindow (20,0,25,80);
420:    _clearscreen (_GWINDOW);
430:    _settextwindow (0,0,25,80);
440:    _settextposition (20,0);
450:    printf ("Start Frequency (GHz)? ");
460:    scanf ("%d",&start[number]);
470:    printf ("Stop Frequency (GHz)? ");
480:    scanf ("%d",&stop[number]);
490:    printf ("Number of Points? ");
500:    scanf ("%d",&points[number]);
510:    if (points[number] == 1) stop[number]=start[number];
520:    _settextposition (number+1,0);
530:    printf ("%d",number);
540:    _settextposition (number+1,20);
550:    printf ("%d",start[number]);
560:    _settextposition (number+1,40);
570:    printf ("%d",stop[number]);
580:    _settextposition (number+1,60);
590:    printf ("%d",points[number]);
600: }
610:
620: int edit()
630: {
640:    int   edit_t;
650:
660:    _settextposition (24,0);
```

```
670:    printf ("Edit which segment (0=exit)? ");
680:    scanf ("%d",&edit_t);
690:    return (edit_t);
700: }
710:
720: main ()
730: {
740:    int   edit_num,i;
750:    float segments;
760:    char  cmd[80];
770:
780:    initialize();
790:    output ("EDITLIST;");
800:    for (i=0; i < 30; i=i+1);
810:       output ("SDEL;");
820:    printf ("Number of segments? ");
830:    scanf ("%f",&segments);
840:    _clearscreen (_GCLEARSCREEN);
850:    _settextposition (1,0);
860:    printf ("SEGMENT");
870:    _settextposition (1,20);
880:    printf ("START");
890:    _settextposition (1,40);
900:    printf ("STOP");
910:    _settextposition (1,60);
920:    printf ("NUMBER OF POINTS\n");
930:    for (i=1; i<=segments; i=i+1)
940:       getsegment (i);
950:    edit_num=edit();
960:    while (edit_num != 0)
970:    {
980:       getsegment (edit_num);
990:      edit_num=edit ();
1000:   }
1010:   output ("EDITLIST;");
1020:        for (i=1; i<=segments; i=i+1)
1030:        {
1040:           sprintf (cmd,"SADD;STAR%dGHZ;STOP%dGHZ;POIN%d;SDON;",
                        start[i],stop[i],points[i]);
1050:           output (cmd);
1060:        }
1070:        output ("EDITDONE;LISFREQ;");
1080: }
```

Figure 3-1. Sample Program: Setting Up a List Frequency Sweep

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 30	Tell the compiler which file includes information on screen commands.
Line 50	Define a variable to contain the HP-IB interface select code, 7.
Line 60	Define a variable to contain the instrument address, 716.
Line 80	Define some global variables. These contain the start, stop, and number of points for each of the segments.
Line 100	Function prototype for the `error_handler ()` routine.
Line 110	Function prototype for the `output ()` routine.
Line 120	Function prototype for the `initialize ()` routine.
Line 130	Function prototype for the getsegment () routine.
Line 140	Function prototype for the `edit ()` routine.
Line 160	Define a routine that handles errors returned from the HP-IB command library I/O functions
Line 180	Check to see if there is an error.
Line 200	An error has occurred, so display a message and halt.
Line 230	No error has occurred, so return.
Line 260	Define a routine that outputs string commands and performs error trapping.
Line 280	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 310	Define a routine to initialize the instrument.
Line 330	Define a timeout value of 45 seconds.
Line 340	Abort any HP-IB transfers.
Line 350	Clear the instrument's HP-IB interface.
Line 360	Preset the instrument.
Line 390	Define a routine to input each of the segments.
Line 410	Define a window to include the bottom half of the screen.
Line 420	Delete the above window.
Line 430	Return the window back to full screen.
Line 440	Position the text.

Line 470	Input the start frequency.
Line 480	Input the stop frequency.
Line 500	Input the number of points.
Line 510	If the number of points is equal to one, then the stop frequency should equal the start frequency.
Line 520	Position the inputted text in a column format.
Line 530	Display the segment number.
Line 540	Position the next column.
Line 550	Display the start frequency.
Line 560	Position the next column.
Line 570	Display the stop frequency.
Line 580	Position the next column.
Line 590	Display the number of points.
Line 620	Define a routine to determine which segment to edit and return that segment value.
Line 660	Position the text to be displayed.
Line 680	Input the segment number to edit.
Line 690	Return the segment number.
Line 720	`Main` declaration
Line 740	Define the needed variables.
Line 780	Call the `initialize ()` routine.
Line 790	Edit the segment list in the instrument.
Line 800	Loop to delete the segment list.
Line 830	Input the number of segments to enter.
Line 840	Clear the screen.
Line 850	Position the text for the header.
Line 860	Display the SEGMENT header.
Line 880	Display the START header.
Line 900	Display the STOP header.
Line 920	Display the NUMBER OF POINTS header.
Line 930	Loop to input the segments.
Line 940	Input each segment.
Line 950	Determine which segment to edit.
Line 960	If the segment does not equal zero then continue, otherwise go to line 1010.
Line 980	Re-input the segment to be edited.

Line 990	Determine which segment to edit.
Line 1010	Edit the segment list in the instrument.
Line 1020	Loop to output the segment list to the instrument.
Line 1040	Create an appropriate string that adds a segment with the correct start and stop frequencies, and number of points.
Line 1050	Output this string.
Line 1070	Declare the editing done, and activate the frequency list sweep mode.

Running the program

The program displays the frequency list table as it is entered. During editing, the displayed table is updated as each line is edited. The table is not re-ordered. At the completion of editing, the table is entered into the instrument, and the list frequency mode turned on.

Any segments already in the frequency list table will be deleted by the program. Thus, new segments will be entered on top of the old ones.

Using limit lines

There are two steps to performing limit testing under HP-IB control. First, limits must be specified and loaded into the analyzer. Second, the limits are activated, the device is measured, and its performance to the specified limits is signaled by a pass or fail message on the display.

Example 5A: Setting up limit lines

This example shows how to create a limit table and transmit it. The command sequence for entering a limit table imitates the key sequence followed when entering a table from the front panel: there is a command for every key press. Editing a limit is also the same as the key sequence, but remember that the instrument automatically re-orders the table in order of increasing start frequency.

The limit table is also carried as part of the learn string. While it cannot be modified as part of the learn string, it can be stored and recalled with very little effort.

This example takes advantage of the computer's capabilities to simplify creating and editing the table. The table is entered and completely edited before being transmitted. To simplify the programming task, options such as entering offsets are not included.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 5A */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:  #include <graph.h>
```

```
40:
50:  #define isc     7L
60:  #define instr   716L
70:  #define fl      'FL'
80:  #define sl      'SL'
90:  #define sp      'SP'
100:
110: int   l_type[30],lower[30],stimulus[30],upper[30];
120:
130:
140: void error_handle (int, char *);
150: void output (char *);
160: void initialize (void);
170: void getlimit (int);
180: int edit(void);
190:
200:
210: void error_handle (int error, char *routine)
220: {
230:    if (error !=NOERR)
240:    {
250:       printf ("HP-IB error in call to %s: %d, %s\n",
                  routine, error, errstr(error));
260:       exit(1);
270:    }
280:    return;
290: }
300:
310: void output (char *sendstr)
320: {
330:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
340: }
350:
360: void initialize ()
370: {
380:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
390:    error_handle (IOABORT (isc), "IOABORT");
400:    error_handle (IOCLEAR (isc), "IOCLEAR");
410:    output ("PRES;");
420: }
430:
440: void getlimit (int number)
450: {
460:    _settextwindow (19,0,25,80);
470:    _clearscreen (_GWINDOW);
480:    _settextwindow (0,0,25,80);
490:    _settextposition (19,0);
500:    printf ("Stimulus Value (GHz)? ");
510:    scanf ("%d",&stimulus[number]);
520:    printf ("Upper Limit Value (dB)? ");
530:    scanf ("%d",&upper[number]);
```

```
540:    printf ("Lower Limit Value (dB)? ");
550:    scanf ("%d",&lower[number]);
560:    printf ("Limit Type (1=Flat, 2=Sloped, 3=Point)? ");
570:    scanf ("%d",&l_type[number]);
580:    _settextposition (number+1,0);
590:    printf ("%d",number);
600:    _settextposition (number+1,15);
610:    printf ("%d",stimulus[number]);
620:    _settextposition (number+1,30);
630:    printf ("%d",upper[number]);
640:    _settextposition (number+1,45);
650:    printf ("%d",lower[number]);
660:    _settextposition (number+1,60);
670:    switch (l_type[number])
680:    {
690:       case 1: printf ("FL");
700:               break;
710:       case 2: printf ("SL");
720:               break;
730:       case 3: printf ("SP");
740:               break;
750:    }
760: }
770:
780: int edit()
790: {
800:    int   edit_t;
810:
820:    _settextposition (24,0);
830:    printf ("Edit which limit (0=exit)? ");
840:    scanf ("%d",&edit_t);
850:    return (edit_t);
860: }
870:
880: main ()
890: {
900:    int   edit_num,i;
910:    float limits;
920:    char  cmd[80];
930:
940:    initialize();
950:    output ("EDITLIML;");
960:    for (i=0; i < 30; i=i+1)
970:    {
980:       output ("SDEL;");
990:    }
1000:   printf ("Number of limits? ");
1010:   scanf ("%f",&limits);
1020:   _clearscreen (_GCLEARSCREEN);
1030:   _settextposition (1,0);
1040:   printf ("LIMIT");
```

```
1050:   _settextposition (1,15);
1060:   printf ("STIMULUS");
1070:   _settextposition (1,30);
1080:   printf ("UPPER");
1090:   _settextposition (1,45);
1100:   printf ("LOWER");
1110:   _settextposition (1,60);
1120:   printf ("TYPE\n");
1130:   for (i=1; i<=limits; i=i+1)
1140:      getlimit (i);
1150:   edit_num=edit();
1160:   while (edit_num != 0)
1170:   {
1180:      getlimit (edit_num);
1190:      edit_num=edit ();
1200:   }
1210:   output ("EDITLIML;");
1220:   for (i=1; i<=limits; i=i+1)
1230:   {
1240:      sprintf (cmd,"SADD;LIMS%dGHZ;LIMU%dDB;LIML%dDB;",
                   stimulus[i],upper[i],lower[i]);
1250:      output (cmd);
1260:      switch (l_type[i])
1270:      {
1280:         case 1: output ("LIMTFL;SDON;");
1290:                 break;
1300:         case 2: output ("LIMTSL;SDON;");
1310:                 break;
1320:         case 3: output ("LIMTSP;SDON;");
1330:                 break;
1340:       }
1350:   }
1360:   output ("EDITDONE;LIMILINEON;LIMITESTON;");
1370: }
```

Figure 3-2. Sample Program: Setting Up Limit Lines

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the complier which file includes information on the HP-IB command library error constants.
Line 30	Tell the compiler which file includes information on screen commands.
Line 50	Define a variable to contain the HP-IB interface select code, 7.

Line 60	Define a variable to contain the instrument address, 716.
Line 70	Define a variable to contain the string 'FL' for flat line. This string is sent to the instrument when a flat line is desired.
Line 80	Define a variable to contain the string 'SL' for sloped line. This string is sent to the instrument when a sloped line is desired.
Line 90	Define a variable to contain the string 'SP' for single point. This string is sent to the instrument when a single point is desired.
Line 110	Define some global variables. These contain the stimulus, upper and lower limits, and the line type of the limit line.
Line 140	Function prototype for the `error_handler ()` routine.
Line 150	Function prototype for the `output ()` routine.
Line 160	Function prototype for the `initialize ()` routine.
Line 170	Function prototype for the `getlimit ()` routine.
Line 180	Function prototype for the `edit ()` routine.
Line 210	Define a routine that handles errors returned from the HP-IB command library I/O functions
Line 230	Check to see if there is an error.
Line 250	An error has occurred, so display a message and halt.
Line 280	No error has occurred, so return.
Line 310	Define a routine that outputs string commands and performs error trapping.
Line 330	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 360	Define a routine to initialize the instrument.
Line 380	Define a timeout value of 45 seconds.
Line 390	Abort any HP-IB transfers.
Line 400	Clear the instrument's HP-IB interface.
Line 410	Preset the instrument.
Line 440	Define a routine to input each of the limit lines.
Line 460	Define a window to include the bottom half of the screen.
Line 470	Delete the above window.
Line 480	Return the window back to full screen.
Line 490	Position the text.
Line 510	Input the stimulus.
Line 530	Input the upper limit.
Line 550	Input the lower limit.
Line 570	Input the line type.

Line 580	Position the inputted text in a column format.
Line 590	Display the limit line number.
Line 600	Position the next column.
Line 610	Display the stimulus.
Line 620	Position the next column.
Line 630	Display the upper limit.
Line 640	Position the next column.
Line 650	Display the lower limit.
Line 660	Position the next column.
Line 670	Display the appropriate limit line type corresponding to each of the three values.
Line 780	Define a routine to determine which limit line to edit and return that segment value.
Line 820	Position the text to be displayed.
Line 840	Input the limit line number to edit.
Line 850	Return the limit line number.
Line 880	`Main` declaration
Line 900	Define the needed variables.
Line 940	Call the `initialize ()` routine.
Line 950	Edit the limit line list in the instrument.
Line 960	Loop to delete the limit line list.
Line 1010	Input the number of limit lines to enter.
Line 1020	Clear the screen.
Line 1030	Position the text for the header.
Line 1050	Display the LIMIT number header.
Line 1060	Display the STIMULUS header.
Line 1080	Display the UPPER limit header.
Line 1100	Display the LOWER limit header.
Line 1120	Display the limit line TYPE header.
Line 1130	Loop to input the limit lines.
Line 1140	Input each limit line.
Line 1150	Determine which limit line to edit.
Line 1160	If the limit line number does not equal zero then continue, otherwise go to line 1220.
Line 1180	Re-input the limit line to be edited.
Line 1190	Determine which limit line to edit.

Line 1210	Edit the limit line list in the instrument.
Line 1220	Loop to output the limit line list to the instrument.
Line 1240	Create an appropriate string that adds a limit line with the correct stimulus, and upper and lower limits.
Line 1250	Output this string.
Line 1260	Determine which string to output to the instrument corresponding to the type of limit line (flat, sloped, or single point).
Line 1360	Declare the editing done, and activate the limit lines and test.

Running the program

The program displays the limit table as it is entered. During editing, the displayed table is updated as each line is edited. The table is not reordered. At the completion of editing, the table is entered, and limit testing mode is turned on. This example program will delete any existing limit lines before entering the new limits.

Example 5B: PASS/FAIL tests

This example demonstrates the use of the limit/search fail bits in event status register B to determine whether a device passes the specified limits. Limits can be entered manually, or by Example 5A.

The limit/search fail bits are set and latched when limit testing or marker search fails. There are four bits, one for each channel for both limit testing and marker search. Their purpose is to allow the computer to determine whether the test/search just executed was successful. To use them, clear event status register B, trigger the limit test or marker search, and then check the appropriate fail bit.

In the case of limit testing, the best way to trigger the limit test is to trigger a single sweep. By the time the SING command finishes, limit testing will have occurred. A second consideration when dealing with limit testing is that if the device is tuned during the sweep, it may be tuned into and then out of limit, causing a limit test pass when the device is not in fact within limits.

In the case of marker searches (max, min, target, and widths), outputting marker or bandwidth values automatically triggers any related searches. Hence, all that is needed is to check the fail bit after reading the data.

Several sweeps in a row should be performed before determining whether or not a device has passed. This gives confidence that the device has passed not due to settling or tuning. Upon running the program, the number of sweeps for qualification has to be entered. For slow sweeps, a small number such as two is appropriate. For very fast sweeps, where the device needs time to settle after tuning and the operator needs time to get away from the device, as many sweeps as six or more might be appropriate.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 5B */
2:
```

```
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:
70:  void error_handle (int, char *);
80:  void output (char *);
90:  void opc (void);
100: void initialize (void);
110: void disp_prompt (char *);
120:
130: void error_handle (int error, char *routine)
140: {
150:   if (error !=NOERR)
160:   {
170:     printf ("HP-IB error in call to %s: %d, %s\n",
               routine, error, errstr(error));
180:     exit(1);
190:   }
200:   return;
210: }
220:
230: void output (char *sendstr)
240: {
250:    error_handle (IOOUTPUTS(instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
260: }
270:
280: void opc ()
290: {
300:    int   one=1;
310:    char  reply;
320:
330:    error_handle (IOENTERS (instr,&reply,&one), "IOENTERS");
340: }
350:
360: void initialize ()
370: {
380:    error_handle (IOTIMEOUT (isc,15.0), "IOTIMEOUT");
390:    error_handle (IOABORT (isc), "IOABORT");
400:    error_handle (IOCLEAR (isc), "IOCLEAR");
410:    output ("CLES;");
420: }
430:
440: void disp_prompt (char *prompt)
450: {
460:    char  ch;
470:
480:    printf ("%s, then press [RETURN]\n",prompt);
490:    ch=getche();
```

```
500: }
510:
520: main ()
530: {
540:    float e_stat;
550:    int   fail_flag,i,stat,qual_tests;
560:
570:    initialize ();
580:    printf ("Number of test for qualification? ");
590:    scanf ("%d",&qual_tests);
600:    fail_flag=0;
610:    printf ("Test #: ");
620:    for (i=1; i<=qual_tests; i=i+1)
630:    {
640:       output ("OPC?;SING;");
650:       opc ();
660:       printf ("%d...",i);
670:       output ("ESB?;");
680:       error_handle (IOENTER (instr,&e_stat), "IOENTER");
690:       stat=e_stat;
700:       if (stat & 16) fail_flag=1;
710:    }
720:    if (fail_flag == 1) printf ("\nDEVICE FAILED!\n");
730:       else printf ("\nDEVICE PASSED!\n");
740: }
```

Figure 3-3. Sample Program: PASS/FAIL tests

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.
Line 90	Function prototype for the `opc ()` routine.
Line 100	Function prototype for the `initialize ()` routine.
Line 110	Function prototype for the `disp_prompt ()` routine.
Line 130	Define a routine that handles errors returned from the HP-IB command library I/O functions

Line 150	Check to see if there is an error.
Line 170	An error has occurred, so display a message and halt.
Line 200	No error has occurred, so return.
Line 230	Define a routine that outputs string commands and performs error trapping.
Line 250	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 280	Define a routine that when called will only return when it receives a response from the instrument. This routine is called after an `OPC?` command has been issued.
Line 310	Define a variable to hold the response.
Line 330	Input the response into the variable reply and do nothing with it.
Line 360	Define a routine to initialize the instrument.
Line 380	Define a timeout value of 15 seconds.
Line 390	Abort any HP-IB transfers.
Line 400	Clear the instrument's HP-IB interface.
Line 410	Clear the instrument's status.
Line 440	Define a routine to display a prompt and wait for (RETURN) to be pressed.
Line 480	Display the message prompt.
Line 490	Wait for a key to be pressed.
Line 520	`Main` declaration
Line 540	Declare the necessary variables.
Line 570	Call the `initialize ()` routine.
Line 590	Determine the number of tests needed for qualification.
Line 600	Set the `fail_flag` to zero. The `fail_flag` determines if the device has failed. A one represents a failure.
Line 610	Display the current test number text.
Line 620	Loop to test the device.
Line 640	Perform a single trace.
Line 650	Wait for the `OPC?` command to issue a response.
Line 660	Display the current test number.
Line 670	Request event status register B byte.
Line 680	Input the event status register B byte.
Line 690	Convert the floating point integer to an integer.
Line 700	If bit 4 is set, then the device has failed, so set the `fail_flag`.

Line 720 If the `fail_flag` is set, then the device has failed, otherwise, it has passed.

Running the program

Set up a limit table on channel one for a specific device either manually, or using the program in Example 5A. The recommended device is the bandpass filter supplied with the instrument (HP part number 0955-0446). Run the program, and enter the number of sweeps desired for pass qualification. After entering the number of sweeps, connect the filter. When enough sweeps in a row pass, the computer displays `DEVICE PASSED`. For the bandpass filter, the suggested limits are as follows.

Table 3-1. Suggested Limits

Seg	Stimulus (GHZ)	Upper (dB)	Lower (dB)	Type
1	8.0	-70	-200	FL
2	9.0	-70	-200	SP
3	9.4	-60	-200	SL
4	10.0	-3	-200	SP
5	10.2	0	-3	FL
6	10.3	0	-3	SP
7	10.5	-3	-200	SL
8	11.1	-60	-200	SP
9	11.5	-70	-200	FL
10	12.5	-70	-200	SP

These are only suggestions. Your filter may vary slightly, and the limits may need to be modified to allow the filter to pass. The program assumes a response calibration (thru calibration) or full 2-port calibration has been performed prior to running the program. Try causing the filter to fail by loosening the cables connecting the filter and then re-tightening them.

Storing and recalling instrument states

The following examples demonstrate ways of storing and recalling instrument states over HP-IB. Example 6A coordinates disk storage, while Example 6B shows an example of how to read calibration data. Example 6A can be easily applied to the coordination of printer and plotters.

There are three operating modes with respect to HP-IB, as set under the (LOCAL) menu: system controller mode, talker/listener mode, and pass control mode. System controller mode is used when no computer is present. The other two modes allow the computer to coordinate certain actions: in talker/listener mode the computer can control the network analyzer, as well as coordinate plotting and printing, and in pass control mode the computer can pass active control to the network analyzer so that it can plot, print, or load/store to disk. Peripheral control is the major difference between the two modes.

If the network analyzer is in pass control mode and receives a command telling it to plot, print, or store/load to disk, it sets bit one in the event status register to indicate that it needs control of the bus. If the computer then uses the HP-IB library control command, `IOPASSCTL` to pass control, the network analyzer will return control back to the computer when its operation is complete.

Control should not be passed to the network analyzer before it has set event status register bit one, (Request Active Control.) If the network analyzer receives control before the bit is set, control is immediately passed back.

While the network analyzer has control , it is free to address devices to talk and listen, as needed. The only functions denied it are the ability to assert the interface clear line (IFC), and the remote line (REN). These are reserved for the system controller. As active controller, the network analyzer can send messages to and back from printers, plotters, and disk drives.

Example 6A: Coordinating disk storage

Refering to Figure 3-4, a personal computer can transfer data to the instrument using any five array transfer formats. The instrument then can transfer this data to an external drive using CS-80 commands to store the data in LIF format. A LIF formatted disk can be converted to a DOS formatted disk through addition software.

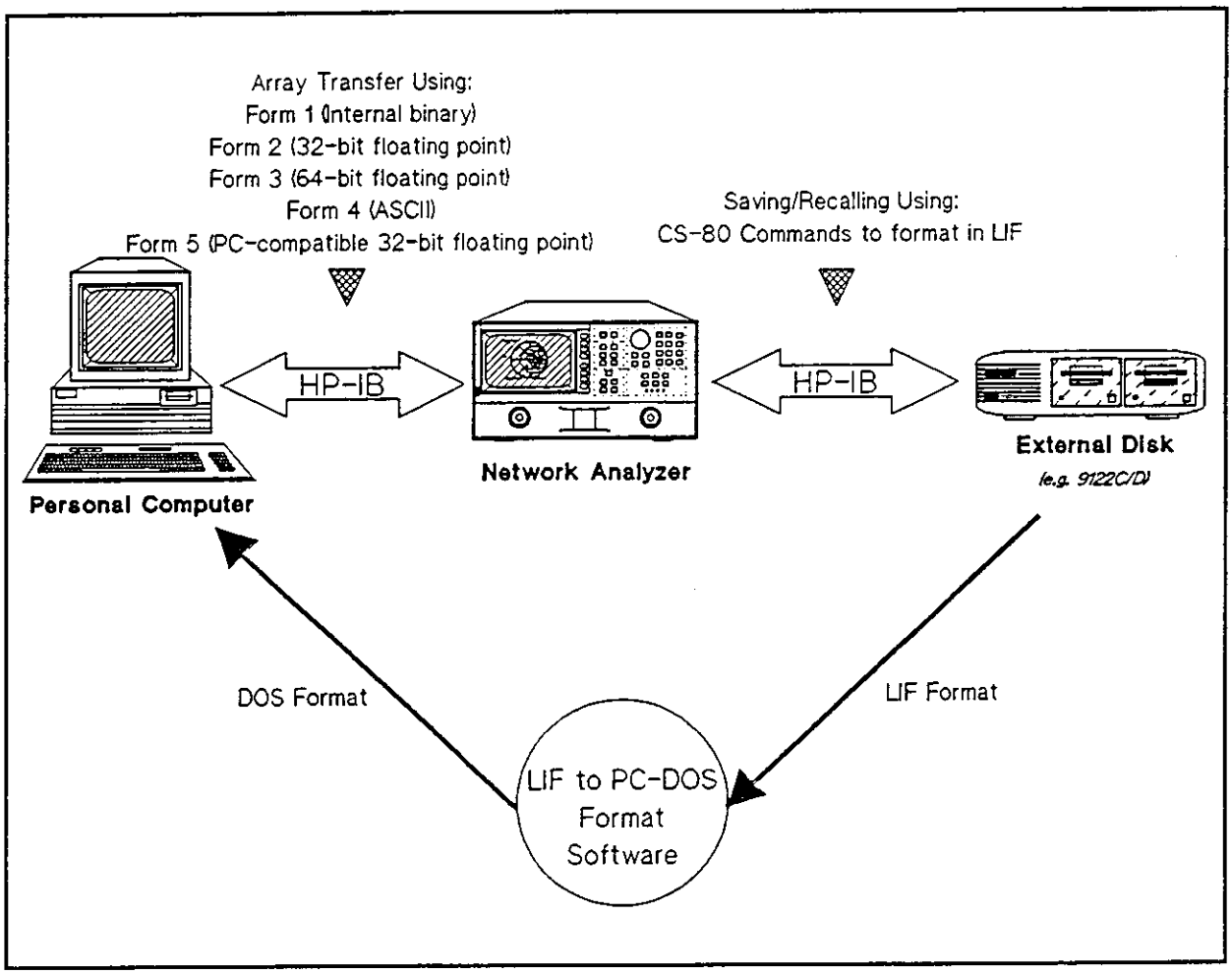

Figure 3-4. Data transfer paths

To have the instrument store an instrument state on disk, specify the state name by titling a file using TITFn, then specify a STORn of that file, where n is the file number, 1 to 5. On receipt of the store command, the instrument will request active control. When control is received, the instrument will store the instrument state on disk as defined under the [SAVE], STORE TO DISK, DEFINE STORE menu.

To have the network analyzer load a file from disk, title the file with TITFn, and then request a LOADn of that file. The best way of learning what the register titles on the disk are, is to use the sequence [RECALL], LOAD FROM DISK, READ FILE TITLES.

Note that the instrument assumes that the address of the disk drive is correctly stored in its HP-IB addresses menu under the ADDRESS: DISK entry. The default address for an external disk drive is 0.

The instrument command USEPASC puts the instrument in pass control. This is neccessary if the instrument is to receive control. When the computer has passed control and the instrument is ready to pass back control, the instrument will pass control to the address under [LOCAL], SET ADDRESSES, ADDRESS: CONTROLLER. This address should be the default 30 address. If this is not 30, change it by [3] [0] [x1].

Note

The default address can be changed with the library command IOCONTROL (716L,7,bus_address), where bus_address is the address of the controller. The library command IOSTATUS (716L,7,bus_address) will determine the current bus address.

The address of the controller must be properly set so that controller return is possible.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 6A */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:  #define quote   '"'
70:
80:  void error_handle (int, char *);
90:  void output (char *);
100: void opc (void);
110: void initialize (void);
120:
130: void error_handle (int error, char *routine)
140: {
150:     if (error !=NOERR)
160:     {
170:        printf ("HP-IB error in call to %s: %d, %s\n",
                    routine, error, errstr(error));
180:        exit(1);
190:     }
200:     return;
210: }
220:
230: void output (char *sendstr)
240: {
250:     error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
260: }
270:
280: void opc ()
290: {
300:     int   one=1;
310:     char  reply;
320:
330:     error_handle (IOENTERS (instr,&reply,&one), "IOENTERS");
340: }
350:
360: void initialize ()
370: {
```

```
380:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
390:    error_handle (IOABORT (isc), "IOABORT");
400:    error_handle (IOCLEAR (isc), "IOCLEAR");
410:    output ("CLES;");
420: }
430:
440: main ()
450: {
460:    int   hpib,stat;
470:    char  ch,cmd[80],name[8];
480:
490:    initialize ();
500:    output ("ADDRCONT30;USEPASC;");
510:    printf("File name to SAVE (up to 8 char.)? ");
520:    scanf("%s",&name);
530:    sprintf (cmd,"CLS;ESE 2;OPC?;TITF1%c%s%c;STOR1;",quote,name,quote);
540:    output (cmd);
550:    printf ("\nSaving on Disk...");
560:    do
570:       error_handle (IOSPOLL (instr,&stat), "IOSPOLL");
580:    while (!(stat & 32));
590:    error_handle (IOPASSCTL (instr), "IOPASSCTL");
600:    do
610:       error_handle (IOSTATUS (isc,4,&hpib), "IOSTATUS");
620:    while (hpib != 1);
630:    printf ("Done.\n\n");
640:    printf ("HIT [RETURN] to recall instrument state.\n");
650:    ch=getch();
660:    printf ("Loading...");
670:    sprintf (cmd,"CLS;ESE 2;OPC?;TITF1%c%s%c;LOAD1;",quote,name,quote);
680:    output (cmd);
690:    do
700:       error_handle (IOSPOLL (instr,&stat), "IOSPOLL");
710:    while (!(stat & 32));
720:    error_handle (IOPASSCTL (instr), "IOPASSCTL");
730:    do
740:       error_handle (IOSTATUS (isc,4,&hpib), "IOSTATUS");
750:    while (hpib != 1);
760:    printf ("Done.\n");
770: }
```

Figure 3-5. Sample Program: Coordinating Disk Storage

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 80	Function prototype for the `error_handler ()` routine.
Line 90	Function prototype for the `output ()` routine.
Line 100	Function prototype for the `opc ()` routine.
Line 110	Function prototype for the `initialize ()` routine.
Line 130	Define a routine that handles errors returned from the HP-IB command library I/O functions
Line 150	Check to see if there is an error.
Line 170	An error has occurred, so display a message and halt.
Line 200	No error has occurred, so return.
Line 230	Define a routine that outputs string commands and performs error trapping.
Line 250	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 280	Define a routine that when called will only return when it receives a response from the instrument. This routine is called after an `OPC?` command has been issued.
Line 310	Define a variable to hold the response.
Line 330	Input the response into the variable reply and do nothing with it.
Line 360	Define a routine to initialize the instrument.
Line 380	Define a timeout value of 45 seconds.
Line 390	Abort any HP-IB transfers.
Line 400	Clear the instrument's HP-IB interface.
Line 410	Clear the instrument's status.
Line 440	`Main` declaration
Line 460	Define the needed variables.
Line 490	Call the `initialize ()` routine.
Line 500	Set the instrument state as pass control.
Line 520	Input the file name to be saved.

Line 530	Prepare a string to clear the instrument's status, specify bit two of the event status register to be summarized by bit five of the status byte, specify a disk file, and save.
Line 540	Output this string.
Line 560	Loop until bit five of the status byte is set, thus indicating that the instrument is ready to take control.
Line 570	Read the status byte.
Line 590	Pass control to the instrument.
Line 600	Loop until the interface is active controller.
Line 610	Read the current interface status.
Line 640	Display a message that the program is ready to recall the saved instrument state.
Line 650	Wait for a key to be pressed.
Line 670	Prepare a string to clear the instrument's state, specify bit two of the event status register to be summarized by bit five of the status register, specify a disk file, and load.
Line 680	Output this string.
Line 690	Loop until bit five of the status byte is set, thus indicating that the instrument is ready to take control.
Line 700	Read the status byte.
Line 720	Pass control to the instrument.
Line 730	Loop until the interface is active controller.
Line 740	Read the current interface status.

Running the program

Put a formatted disk in the disk drive, and set the disk address, unit number, and volume number for that drive. Run the example and enter a valid file name. The program will save the current instrument state, wait for (RETURN) to be pressed, and load the previously saved state. When the program pauses, change the instrument state so that a change will be noticeable.

Example 6B: Reading calibration data

This example demonstrates how to read and write measurement calibration data.

The data used to perform measurement error correction is stored in up to twelve calibration coefficient arrays. Each array is a specific error coefficient and always contains a real and imaginary part corresponding to the each point in the sweep. The five data formats also apply to the transfer of calibration coefficient arrays.

A computer can read out the error coefficients using the command OUTPCALC{n}, where n can range from 1 to 12. Each calibration type uses only as many arrays as needed, starting with array one. Therefore, it is necessary to know the type of calibration that was used to produce the coefficients (ie. 1-port vs. 2-port). Attempting to read an array not being used in the current calibration causes the "REQUESTED DATA NOT CURRENTLY AVAILABLE" warning. Refer to the *HP-IB Programming Reference (HP part number 08720-90160)* for the definitions of calibration types, standard classes, and calibration coefficients.

A computer can also store calibration coefficients in the instrument. To do this, declare the type of calibration data about to be stored just as if you were about to perform that calibration. Then instead of calling up different classes, transfer the calibration coefficients using the INPUCALC{n} (n ranges from 1 to 12) command. When all the coefficients are in, activate the calibration by issuing the mnemonic SAVC, and take a sweep.

This example reads the response calibration coefficients from a response calibration, using Form 1, into an array, from which they can be examined, modified, stored, or put back into the instrument. In Form 1, each data point is sent out as it is stored inside the network analyzer, in a six byte binary string. Hence, it is a very fast transfer, using only 1,206 bytes to transfer 201 points, but it is difficult to interpret by the computer since it is not a standard data format. (Real/imaginary data uses the first two bytes for the imaginary fraction mantissa, the middle two bytes for the real fraction mantissa, the fifth byte for additional resolution when transferring raw data, and the last byte as the common power of two).

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 6B */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:
70:  void error_handle (int, char *);
80:  void output (char *);
90:  void initialize (void);
100: void opc (void);
110:
120: void opc ()
130: {
140:    int   one=1;
150:    char  reply;
160:
170:    error_handle (IOENTERS (instr,&reply,&one), "IOENTERS");
```

```
180: }
190:
200:
210: void error_handle (int error, char *routine)
220: {
230:    if (error !=NOERR)
240:    {
250:       printf ("HP-IB error in call to %s: %d, %s\n",
                  routine, error, errstr(error));
260:       exit(1);
270:    }
280:    return;
290: }
300:
310: void output (char *sendstr)
320: {
330:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
340: }
350:
360: void initialize ()
370: {
380:    error_handle (IOTIMEOUT (isc,15.0), "IOTIMEOUT");
390:    error_handle (IOABORT (isc), "IOABORT");
400:    error_handle (IOCLEAR (isc), "IOCLEAR");
410:    output ("CLES;");
420: }
430:
440: main ()
450: {
460:    int   bytes,i,length;
470:    char  ascii_dat[1300],carline='\n',ch,cmd[80],header[2],response;
480:
490:    initialize();
500:    output ("CORRON;OPC?;SING;");
510:    opc ();
520:    output ("CALIRESP?;");
530:    length=1;
540:    error_handle (IOENTERS (instr,&response,&length), "IOENTERS");
550:    if (response == '0')
560:    {
570:       printf ("Calibration Response data not available.\n");
580:       exit (1);
590:    }
600:    output ("FORM1;OUTPCALC01;");
610:    length=2;
620:    error_handle (IOMATCH (isc,'\n',0), "IOMATCH");
630:    error_handle (IOENTERS (instr,header,&length), "IOENTERS");
640:    error_handle (IOENTERB (instr,&bytes,&length,2), "IOENTERB");
650:    printf ("Header: %s\nNumber of bytes: %d\n\n",header,bytes);
660:    length=bytes;
670:    error_handle (IOENTERB (instr,ascii_dat,&length,1), "IOENTERB");
```

```
680:    printf ("\nData is now loaded using Form 1.\n\n");
690:    printf ("\nPress [RETURN] to re-transmit calibration\n");
700:    ch=getch();
710:    initialize ();
720:    output ("CALIRESP;FORM1;");
730:    error_handle (IOEOL (isc,&carline,0), "IOEOL");
740:    error_handle (IOEOI (isc,0), "IOEOI");
750:    output ("INPUCALCO1");
760:    length=2;
770:    error_handle (IOOUTPUTS (instr,header,length), "IOOUTPUTS");
780:    error_handle (IOOUTPUTB (instr,&bytes,length,2), "IOOUTPUTB");
790:    length=bytes;
800:    error_handle (IOOUTPUTB (instr,ascii_dat,length,1), "IOOUTPUTB");
810:    error_handle (IOEOL (isc,&carline,1), "IOEOL");
820:    error_handle (IOEOI (isc,1), "IOEOI");
830:    output ("SAVC;CONT;");
840: }
```

Figure 3-6. Sample Program: Reading Calibration Data

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.
Line 90	Function prototype for the `initialize ()` routine.
Line 100	Function prototype for the `opc ()` routine.
Line 120	Define a routine that when called will only return when it receives a response from the instrument. This routine is called after an `OPC?` command has been issued.
Line 150	Define a variable to hold the response.
Line 170	Input the response into the variable reply and do nothing with it.
Line 210	Define a routine that handles errors returned from the HP-IB command library I/O functions
Line 230	Check to see if there is an error.
Line 250	An error has occurred, so display a message and halt.

Line 280	No error has occurred, so return.
Line 310	Define a routine that outputs string commands and performs error trapping.
Line 330	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 360	Define a routine to initialize the instrument.
Line 380	Define a timeout value of 15 seconds.
Line 390	Abort any HP-IB transfers.
Line 400	Clear the instrument's HP-IB interface.
Line 410	Clear the instrument's status.
Line 440	`Main` declaration
Line 450	Define the needed variables.
Line 490	Call the `initialize ()` routine.
Line 500	Switch debug on, correction on, and perform a single trace.
Line 510	Wait for the `OPC?` command to issue a response.
Line 520	Request if calibration response is active.
Line 530	Define a variable, `length`, that contains the length of the inputted data. In this case, only one byte.
Line 540	Input the response.
Line 550	If the response is zero, then calibration response is not active and therefore its data is not available.
Line 580	Exit.
Line 600	Request calibration coefficients of array one, using form 1 (binary internal format).
Line 610	Set the `length` of the header and number of bytes of the block to follow to two.
Line 620	Disable character matching. This command defines the character used by `IOENTERB` and `IOENTERS` for termination. The "lf" enter terminator should be turned off because "lf" is a valid binary value.
Line 630	Input the header.
Line 640	Input the number of bytes in the block to follow.
Line 650	Display the header and the number of bytes in the block.
Line 670	Read in the calibration coefficient data.
Line 680	Display a confirmation that data has been loaded.
Line 700	Wait for a RETURN.
Line 710	Initialize the instrument to remote, in case the user has changed the instrument calibration.
Line 720	Open the calibration response menu. Prepare for a Form 1 transfer.

Line 730	Disable End of Line (EOL) character. This command defines the character used by IOOUTPUT, IOOUTPUTA, IOOUTPUTB, and IOOUTPUTS for termination. The "lf" enter terminator should be turned off because "lf" is a valid binary value.
Line 740	Disable End Or Identify (EOI).
Line 750	Request the analyzer to input calibration data.
Line 770	Output the header.
Line 780	Output the number of bytes in the block to follow.
Line 800	Output the calibration coefficients.
Line 810	Enable End of Line (EOL) character as the "lf" enter terminator.
Line 820	Enable End Or Identify (EOI).
Line 830	Create a calibration set based on the current error coefficient arrays loaded. Turn on the continuous sweep trigger mode.

Running the program

Before executing the program, perform a response calibration. Run the program, and when it pauses, perform a different calibration. When the program resumes, the old calibration data will be loaded in and activated.

The program will determine if response calibration data is active. If it is not, it will halt and display an appropriate message.

4

Miscellaneous/Reference Programming Examples

Example 7: Key Trapping

It is possible to sense operator action with the front panel keys. The user request bit, bit 6, in the event status register is set whenever a front panel key is pressed or the knob is turned, whether the instrument is in remote or local mode. Each key has a number associated with it. The number of the key last pressed can be read with the KOR? and the OUTPKEY commands. With KOR?, a knob turn is reported as a negative number encoded with the number of counts turned. With OUTPKEY, a knob turn is always reported as a negative one.

Refer to the *HP-IB Programming Reference (HP part number 08720-90160)* for the codes of the front panel keys.

In this example, the OUTPKEY command is used to re-define the top four and eighth softkeys.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 7  */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:
70:  void error_handle (int, char *);
80:  void output (char *);
90:  void initialize (void);
100:
110: void error_handle (int error, char *routine)
120: {
130:     if (error !=NOERR)
140:     {
150:        printf ("HP-IB error in call to %s: %d, %s\n",
                    routine, error, errstr(error));
160:        exit(1);
170:     }
180:     return;
190: }
200:
210: void output (char *sendstr)
220: {
230:     error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
240: }
250:
```

```
260: void initialize ()
270: {
280:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
290:    error_handle (IOABORT (isc), "IOABORT");
300:    error_handle (IOCLEAR (isc), "IOCLEAR");
310:    output ("PRES;");
320: }
330:
340: main ()
350: {
360:    int   keycode,status;
370:    float value;
380:
390:    initialize ();
400:    output ("CLES;ESE64;MENUOFF");
410:    printf ("Ready...\n");
420:    do
430:    {
440:      do
450:         error_handle (IOSPOLL (instr, &status), "IOSPOLL");
460:      while (!(status & 32));
470:      output ("OUTPKEY;");
480:      error_handle (IOENTER (instr, &value), "IOENTER");
490:      keycode=value;
500:      switch (keycode)
510:      {
520:         case 60: printf ("Calibration #1\n");
530:                  break;
540:         case 61: printf ("Test #1\n");
550:                  break;
560:         case 56: printf ("Calibration #2\n");
570:                  break;
580:         case 59: printf ("Test #2\n");
590:                  break;
600:         case 10: printf ("Abort \n");
610:                  break;
620:         default: printf ("*** UNDEFINED ***\n");
630:                  break;
640:      }
650:      output ("CLES;ESE64;");
660:    }
670:    while (keycode !=10);
680:    output ("MENUON;");
690: }
```

Figure 4-1. Sample Program: Key Trapping

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.
Line 90	Function prototype for the `initialize ()` routine.
Line 110	Define a routine that handles errors returned from the HP-IB command library I/O functions
Line 130	Check to see if there is an error.
Line 150	An error has occurred, so display a message and halt.
Line 180	No error has occurred, so return.
Line 210	Define a routine that outputs string commands and performs error trapping.
Line 230	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 260	Define a routine to initialize the instrument.
Line 280	Define a timeout value of 45 seconds.
Line 290	Abort any HP-IB transfers.
Line 300	Clear the instrument's HP-IB interface.
Line 310	Preset the instrument.
Line 340	`Main` declaration
Line 360	Declare the needed variables.
Line 390	Call the `initialize ()` routine.
Line 400	Clear the instrument's status, specify bit six of the event status register to be summarized by bit five of the status byte, and turn off the softkey menus.
Line 420	Do until `keycode` is equal to ten. `keycode` contains the current key code for the front key pressed.
Line 450	Loop until bit five of the status register is set. This indicates that bit six of the event status register is set.
Line 470	Request the key code of the key just pressed.
Line 480	Input the key code.

Line 490	Convert the key code to an integer.
Line 500	If the key code is any of the top four or eighth softkeys, then display an appropriate message, otherwise, the key pressed is undefined.
Line 650	Clear the status register and specify bit six of the event status register to be summarized by bit five of the status byte.
Line 680	Turn the softkey menus back on.

Running the program

The program will turn off the current softkey menu and trap the first four and eighth softkeys. When any of the first four softkeys are pressed, a softkey specific message is displayed on the screen. The eighth softkey is defined as ABORT, and will terminate the program.

Example 8: CRT Graphics

The following example program illustrates how to display graphics on the instrument. Graphics can be drawn by sending HP-GL (Hewlett-Packard Graphics Language) commands to the network analyzer display.

The display address is the instrument address with the least significant bit complemented. If the instrument address is 716 then the display address will be 717. In the following program, the routine `disp_output` sends output commands to address 717.

Note

This program uses the QuickC math include file and the HP-IB include file. Including both of these files causes QuickC to issue a warning that the variable `ERANGE` has been defined twice. For this example program, the warning can be disregarded.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 8  */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:  #include <math.h>
40:
50:  #define isc     7L
60:  #define instr   716L
70:  #define display 717L
80:
90:  void error_handle (int, char *);
100: void disp_output (char *);
110: void output (char *);
120: void initialize (void);
130:
140:
150: void error_handle (int error, char *routine)
```

```
160: {
170:    if (error !=NOERR)
180:    {
190:       printf ("HP-IB error in call to %s: %d, %s\n",
                  routine, error, errstr(error));
200:       exit(1);
210:    }
220:    return;
230: }
240:
250: void disp_output (char *sendstr)
260: {
270:    error_handle (IOOUTPUTS (display,sendstr,strlen(sendstr)),
                     "IOOUTPUTS");
280: }
290:
300: void output (char *sendstr)
310: {
320:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
330: }
340:
350: void initialize ()
360: {
370:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
380:    error_handle (IOABORT (isc), "IOABORT");
390:    error_handle (IOCLEAR (isc), "IOCLEAR");
400:    output ("PRES;");
410: }
420:
430: main ()
440: {
450:    int    x,y;
460:    double i=0;
470:    char   cmd[80];
480:
490:    initialize ();
500:    disp_output ("AF;CS;SP4;PU;PA 0,1500;PD;PA 5850,1500;");
510:    disp_output ("SP5;PU;PA 0,2000;PD;PA 5850,2000;");
520:    disp_output ("SP4;PU;PA 0,2500;PD;PA 5850,2500;");
530:    disp_output ("PU;SP2;PA 0,2000;");
540:    do
550:    {
560:       i=i+(3.14/10);
570:       x=i*370;
580:       y=(sin(i)*500)+2000;
590:       sprintf (cmd,"PD;PA%d,%d;",x,y);
600:       disp_output (cmd);
610:    }
620:    while (i < (3.14*5));
630:    disp_output ("SL.16,.20;SP5;PU;PA 700,1000;LBS I N E   W A V E\3;");
640:    disp_output ("SL.16,.20;SP1;PA 100,3950;LBHewlett Packard\3;");
```

```
650: }
```

Figure 4-2. Sample Program: CRT Graphics

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 30	Tell the compiler which file include information on the math functions.
Line 50	Define a variable to contain the HP-IB interface select code, 7.
Line 60	Define a variable to contain the instrument address, 716.
Line 70	Define a variable to contain the instrument display address, 717.
Line 90	Function prototype for the `error_handler ()` routine.
Line 100	Function prototype for the disp_output () routine.
Line 110	Function prototype for the `output ()` routine.
Line 120	Function prototype for the `initialize ()` routine.
Line 150	Define a routine that handles errors returned from the HP-IB command library I/O functions
Line 170	Check to see if there is an error.
Line 190	An error has occurred, so display a message and halt.
Line 220	No error has occurred, so return.
Line 250	Define a routine that outputs string commands to the instrument display and performs error checking.
Line 270	Send a string to the instrument located at the value of `display`, 717. Perform error checking.
Line 300	Define a routine that outputs string commands to the instrument and performs error trapping.
Line 320	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 350	Define a routine to initialize the instrument.
Line 370	Define a timeout value of 45 seconds.
Line 380	Abort any HP-IB transfers.
Line 390	Clear the instrument's HP-IB interface.
Line 400	Preset the instrument.

Line 430	`Main` declaration
Line 450	Define the needed variables.
Line 490	Call the `initialize ()` routine.
Line 500	Output the following command sequence to the instrument display: erase the user graphics display, turn off measurement display, select color four, pen up, pen position 0, 1500, pen down, pen draw to 5850,1500. This prepares the graphics display, and draws a single line.
Line 510	Draw a second line using color five.
Line 520	Draw a third line using color four.
Line 530	Position the pen to begin drawing a sine wave.
Line 540	Loop to draw a sine wave.
Line 560	The variable `i` is the radian sine value. `i` is increased by 1/10th of pi. For increased resolution, this value should be increased.
Line 570	Scale the x-axis.
Line 580	Scale the y-axis.
Line 590	Prepare a string to draw 1/20th of the period of a sine wave.
Line 600	Output this string to the instrument display.
Line 620	Loop until two and half periods have been completed.
Line 630	Select character size, color five, and place label "`S I N E W A V E`" at position 700, 1000 on the display.
Line 640	Select character size, color one, and place label "`Hewlett Packard`" at position 100, 3950 on the display.

Running the program

The program will display a sine wave along with two text labels on the instrument. The HP-GL commands can perform three basic types of functions:

- Label text,
- Change line types and colors, and
- Draw lines.

Using key trapping to take over the instrument's front panel along with HP-GL commands, a custom user interface can be easily created.

Status Reporting

The network analyzer has a status reporting mechanism that gives information about specific functions and events inside the instrument. The status byte is an eight bit register with each bit summarizing one aspect of the instrument state. For example, the error queue summary bit will always be set if there are any errors in the queue. The value of the status byte can be read with the IOSPOLL statement. This command does not automatically put the instrument in remote mode, thus giving the operator access to the front panel functions. The status byte can also be read by sending the command OUTPSTAT. Reading the status byte does not affect its value.

The status byte summarizes the error queue, as mentioned before. It also summarizes two event status registers that monitor specific conditions inside the instrument. The status byte also has a bit that is set when the instrument is issuing a service request over HP-IB, and a bit that is set when network analyzer has data to send out over HP-IB. See Figure 4-4 for a definition of the status registers.

Example 9A: Using the error queue

The error queue holds up to 20 instrument errors and warnings in the order that they occurred. Each time the analyzer detects an error condition, it displays a message on the CRT and puts the error in the error queue. If there are any errors in the queue, bit three of the status byte will be set. The errors can be read from the queue with the OUTPERRO command, which causes the analyzer to transmit the error number and error message of the oldest error in the queue.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 9A */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:
70:  void error_handle (int, char *);
80:  void output (char *);
90:  void initialize (void);
100:
110: void error_handle (int error, char *routine)
120: {
130:    if (error !=NOERR)
140:    {
150:       printf ("HP-IB error in call to %s: %d, %s\n",
                  routine, error, errstr(error));
160:       exit(1);
170:    }
180:    return;
```

```
190: }
200:
210: void output (char *sendstr)
220: {
230:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
240: }
250:
260: void initialize ()
270: {
280:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
290:    error_handle (IOABORT (isc), "IOABORT");
300:    error_handle (IOCLEAR (isc), "IOCLEAR");
310:    output ("PRES;");
320: }
330:
340: main ()
350: {
360:    int   hpib,length,stat;
370:    char  err_data[80];
380:
390:    initialize ();
400:    top:
410:    error_handle (IOLOCAL (instr), "IOLOCAL");
420:    do
430:       error_handle (IOSPOLL (instr,&stat), "IOSPOLL");
440:    while (!(stat & 8));
450:    output ("OUTPERRO;");
460:    length=80;
470:    error_handle (IOENTERS (instr,err_data,&length), "IOENTERS");
480:    printf ("%s",err_data);
490:    goto top;
500: }
```

Figure 4-3. Sample Program: Using the Error Queue

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.

Line 90	Function prototype for the `initialize ()` routine.
Line 110	Define a routine that handles errors returned from the HP-IB command library I/O functions
Line 130	Check to see if there is an error.
Line 150	An error has occurred, so display a message and halt.
Line 180	No error has occurred, so return.
Line 210	Define a routine that outputs string commands and performs error trapping.
Line 230	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 260	Define a routine to initialize the instrument.
Line 280	Define a timeout value of 45 seconds.
Line 290	Abort any HP-IB transfers.
Line 300	Clear the instrument's HP-IB interface.
Line 310	Preset the instrument.
Line 340	`Main` declaration
Line 360	Declare the needed variables.
Line 390	Call the `initialize ()` routine.
Line 400	Label this point as `top:`.
Line 410	Put the instrument into local mode.
Line 430	Loop until bit three of the status is set. This is the error queue bit.
Line 450	Request the error message.
Line 460	Define the `length` of the error message to be an arbitrary number, 80.
Line 470	Input the error message. On return, the variable `length` contains the actual length of the message.
Line 480	Display the error message.
Line 490	Loop unconditionally to `top:`.

Running the program

Preset the network analyzer and run the program. Nothing should happen at first. To get something to happen, press a blank softkey. The message "`CAUTION: INVALID KEY`" will appear followed by a second message "`NO ERRORS`". Hence, to clean the error queue, you can either loop until the no errors message is received, or until the bit in the status register is cleared. Note that throughout all this, the front panel is in local mode.

To break from the program loop, press <CTRL-Break>, followed by a blank softkey.

Because the error queue will keep up to 20 errors, it is important to clear out the error queue whenever errors are detected so that old errors are not associated with the current instrument

state. Not all messages displayed are put in the error queue: operator prompts and cautions are not included.

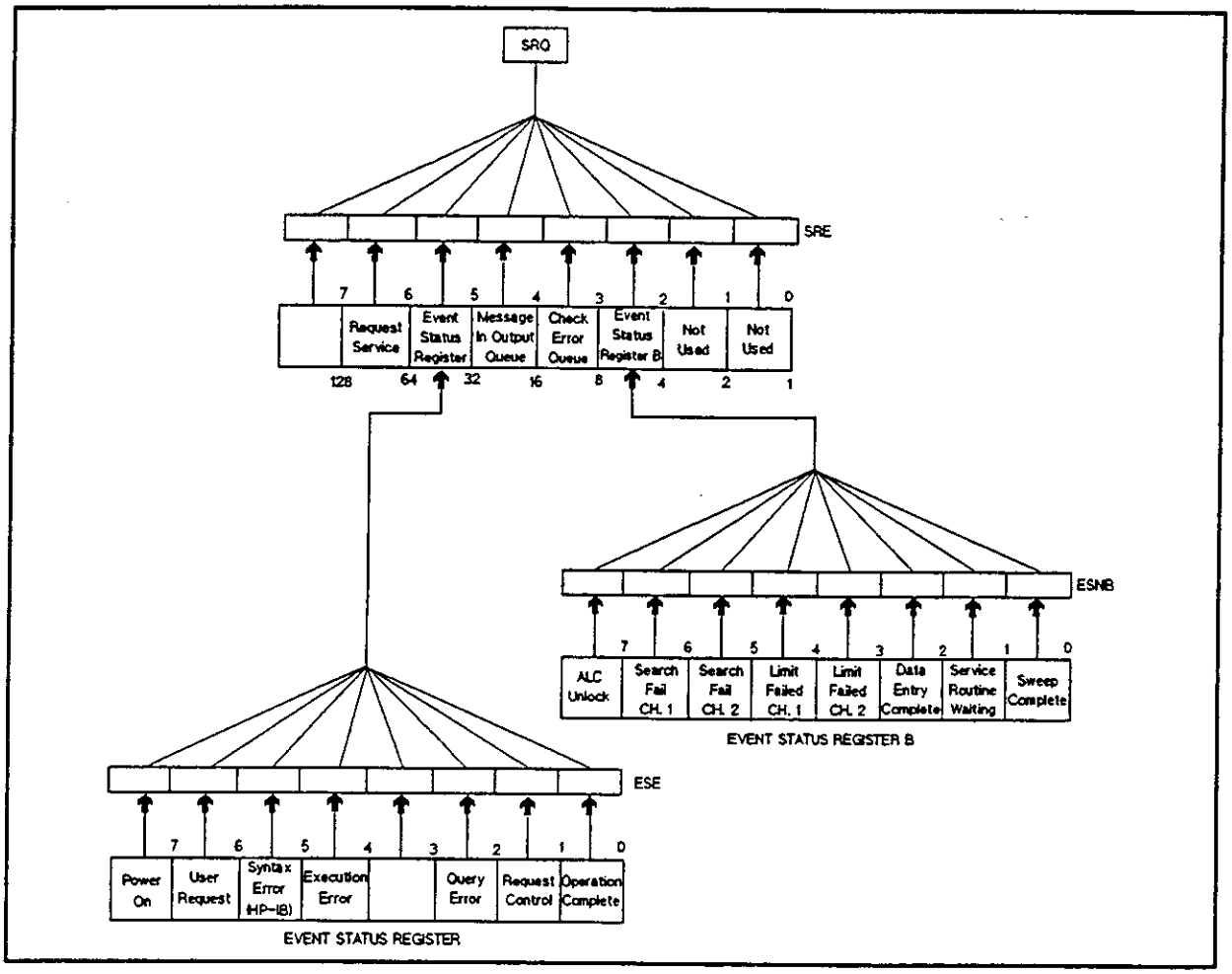

Figure 4-4. Status reporting system

Example 9B: Using the status registers

The other two key components of the status reporting system are the event status register, and event status register B. These eight bit registers consist of latched event bits. A latched bit is set at the onset of the monitored condition, and is cleared only by a read of the register or by clearing the status registers with CLES.

This program uses the instrument command KOR? to determine a key press. The keycode encoding with KOR? is as follows. Clockwise rotations of the knob are reported as numbers from -1 to -64, -1 being a very small rotation. Counterclockwise rotations are reported as the numbers -32767 to -32703, -32767 being a very small rotation. Hence, clockwise rotations don't need any decoding at all, and counterclockwise rotations can be decoded by adding 32,768. There are approximately 120 counts per knob rotation, and the sign of the count depends on the direction the knob was turned.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 9B */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
60:
70:  void error_handle (int, char *);
80:  void output (char *);
90:  void initialize (void);
100:
110:
120: void error_handle (int error, char *routine)
130: {
140:    if (error !=NOERR)
150:    {
160:       printf ("HP-IB error in call to %s: %d, %s\n",
                  routine, error, errstr(error));
170:       exit(1);
180:    }
190:    return;
200: }
210:
220: void output (char *sendstr)
230: {
240:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
250: }
260:
270: void initialize ()
280: {
290:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
300:    error_handle (IOABORT (isc), "IOABORT");
310:    error_handle (IOCLEAR (isc), "IOCLEAR");
320:    output ("PRES;");
330: }
340:
350: main ()
360: {
370:    float e_stat,keycode;
380:    int   stat;
390:
400:    initialize ();
410:    top:
420:    do
430:    {
440:       output ("ESR?;");
450:       error_handle (IOENTER (instr,&e_stat), "IOENTER");
460:       stat=e_stat;
470:    }
```

```
480:    while (!(stat & 64));
490:    output ("KOR?;");
500:    error_handle (IOENTER (instr,&keycode), "IOENTER");
510:    if (keycode >= 0) printf ("Key ");
520:       else if (keycode < -400.0) keycode=keycode+32768;
530:    printf ("code = %.0f\n",keycode);
540:    goto top;
550: }
```

Figure 4-5. Sample Program: Using the Status Registers

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.
Line 90	Function prototype for the `initialize ()` routine.
Line 120	Define a routine that handles errors returned from the HP-IB command library I/O functions
Line 140	Check to see if there is an error.
Line 160	An error has occurred, so display a message and halt.
Line 190	No error has occurred, so return.
Line 220	Define a routine that outputs string commands and performs error trapping.
Line 240	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 270	Define a routine to initialize the instrument.
Line 290	Define a timeout value of 45 seconds.
Line 300	Abort any HP-IB transfers.
Line 310	Clear the instrument's HP-IB interface.
Line 320	Preset the instrument.
Line 350	`Main` declaration
Line 370	Declare the needed variables.

Line 400	Call the `initialize ()` routine.
Line 410	Label this point as `top:`.
Line 420	Loop until bit six of the event status register is set.
Line 440	Request the event status register.
Line 450	Input the event status register.
Line 460	Convert floating point integer to an integer value.
Line 490	Request the key code of the key just pressed.
Line 500	Input the key code.
Line 510	If the key code is greater than or equal to zero, then no decoding is necessary, and the key pressed is not a knob count.
Line 520	Otherwise, the key code needs decoding.
Line 530	Display the key code.
Line 540	Loop unconditionally to the position of `top:`.

Running the program

Run the program. Pressing a front panel key causes the computer to display the keycode associated with that key. Note that since the instrument is in remote mode, the normal function of that key is not executed. In effect, we have taken over the front panel and can now re-define the keys.

To break from the program loop, press <CTRL-Break>.

Example 10: Passing data to other application programs

The following example creates a formatted data file that can be sent to/recalled from an application-specific program.

The program performs two data transfers from the instrument. The first, using `OUTPLIML` with Form 4 (ASCII transfer format), reads out limit data to obtain the stimulus frequencies. `OUTPLIML` reads out the stimulus frequency, result, upper limit, and lower lower limit of limit data. Since stimulus frequency is only needed, the other values are discarded.

The second data transfer uses `OUTPFORM` with Form 5 (PC-compatible 32-bit floating point format) to read out magnitude data.

```
1:   /*  HP 8719C, 8720C, 8722A/C  QuickC IPG Program 10 */
2:
5:   #include <stdio.h>
10:  #include <cfunc.h>
20:  #include <chpib.h>
30:
40:  #define isc     7L
50:  #define instr   716L
```

```
60:
70:  void error_handle (int, char *);
80:  void output (char *);
90:  void pad (int);
100:
110: void error_handle (int error, char *routine)
120: {
130:    if (error !=NOERR)
140:    {
150:       printf ("HP-IB error in call to %s: %d,  %s\n",
                  routine, error, errstr(error));
160:       exit(1);
170:    }
180:    return;
190: }
200:
210: void output (char *sendstr)
220: {
230:    error_handle (IOOUTPUTS (instr,sendstr,strlen(sendstr)), "IOOUTPUTS");
240: }
250:
260: void pad (int pad_num)
270: {
280:    char  pad[40];
290:
300:    error_handle (IOENTERS (instr,pad,&pad_num), "IOENTERS");
310: }
320:
330: main ()
340: {
350:    char  ascii_dat[15],filename[10],header[2],reply;
360:    int   bytes,elements,i,one=1;
370:    float data,points,seek_len;
380:    FILE  *f_ptr;
390:
400:    error_handle (IOTIMEOUT (isc,45.0), "IOTIMEOUT");
410:    error_handle (IOABORT (isc), "IOABORT");
420:    error_handle (IOCLEAR (isc), "IOCLEAR");
430:    error_handle (IOMATCH (isc,'\n',0), "IOMATCH");
440:    output ("OPC?;SING;");
450:    error_handle (IOENTERS (instr,&reply,&one), "IOENTERS");
460:    output ("POIN?;");
470:    error_handle (IOENTER (instr,&points), "IOENTER");
480:    printf ("Save under what DOS file name? ");
490:    gets (filename);
500:    f_ptr=fopen (filename,"w+");
510:    if (f_ptr != NULL)
520:    {
530:       printf ("\n\nSaving ... ");
540:       output ("FORM4;OUTPLIML;");
550:       elements=15;
```

```
560:        for (i=0; i<points; i++)
570:        {
580:           pad(2);
590:           error_handle (IOENTERS (instr,ascii_dat,&elements), "IOENTERS");
600:           pad(38);
610:           fprintf (f_ptr,"%s,                                    \n",ascii_dat);
620:        }
630:        fseek (f_ptr,17L,SEEK_SET);
640:        output ("FORM5;OUTPFORM;");
650:        elements=2;
660:        error_handle (IOENTERS (instr,header,&elements), "IOENTERS");
670:        error_handle (IOENTERB (instr,&bytes,&elements,1), "IOENTERB");
680:        elements=4;
690:        for (i=0; i<points; i++)
700:        {
710:            error_handle (IOENTERB (instr,&data,&elements,1), "IOENTERB");
720:            fprintf (f_ptr,"%f,",data);
730:            error_handle (IOENTERB (instr,&data,&elements,1), "IOENTERB");
740:            fprintf (f_ptr,"%f",data);
750:            seek_len=i+1;
760:            seek_len=(seek_len*48)+17;
770:            fseek(f_ptr,seek_len,SEEK_SET);
780:        }
790:        fclose (f_ptr);
800:        printf ("done!\n");
810:    }
820:        else printf ("Could not open file\n");
830: }
```

Figure 4-6. Sample Program: Passing Data to Other Application Programs

Program explanation

Line 5	Tell the compiler which file includes information on the standard I/O routines.
Line 10	Tell the compiler which file includes information on the HP 82335A HP-IB command library I/O functions.
Line 20	Tell the compiler which file includes information on the HP-IB command library error constants.
Line 40	Define a variable to contain the HP-IB interface select code, 7.
Line 50	Define a variable to contain the instrument address, 716.
Line 70	Function prototype for the `error_handler ()` routine.
Line 80	Function prototype for the `output ()` routine.
Line 90	Function prototype for the `pad ()` routine.
Line 110	Define a routine that handles errors returned from the HP-IB command library I/O functions

Line 130	Check to see if there is an error.
Line 150	An error has occurred, so display a message and halt.
Line 180	No error has occurred, so return.
Line 210	Define a routine that outputs string commands and performs error trapping.
Line 230	Send a string to the instrument located at the value of `instr`, 716. Perform error checking.
Line 260	Define a routine that reads a specified number of bytes from the instrument located at the value of `instr`, 716. The passed variable `pad_num` determines the number of bytes to read and discard.
Line 330	`Main` declaration
Line 350	Declare the needed variables.
Line 400	Define a timeout value of 45 seconds.
Line 410	Abort any HP-IB transfers.
Line 420	Clear the instrument's HP-IB interface.
Line 430	Disable character matching. This command defines the character used by `IOENTERB` and `IOENTERS` for termination. The "lf" enter terminator should be turned off because "lf" is a valid binary value.
Line 440	Perform a single trace.
Line 450	Determine when the trace has been completed.
Line 460	Ask for the number of points sampled.
Line 470	Input the number of points sampled.
Line 480	Determine the filename to save the formatted data.
Line 500	Open the filename.
Line 510	If there are no errors in opening the file, continue, otherwise go to line 820.
Line 530	Displaying the "`Saving ...` " message.
Line 540	Ask for the results of a limit test in Form 4.
Line 550	Define a variable, `elements`, to contain the length of the sent data values.
Line 560	Loop and read the data.
Line 580	Disregard the first two characters.
Line 590	Input the stimulus value.
Line 600	Disregard the next 38 characters, which contain the results of a limit test, and the upper and lower limits of limit data.
Line 610	Output the stimulus values to a file.

Line 630	Prepare to write to the file the second and third columns, the first and second data values. Specifically, offset seventeen characters from the beginning of the file.
Line 640	Ask for the trace data in Form 5.
Line 650	Define the variable, `elements`, to contain the length of the header and number of bytes in the data block to follow.
Line 660	Input the header.
Line 670	Input the number of bytes in the data block to follow.
Line 680	Define the length of each point. Since each point is sent as a 32-bit floating point integer, the length of each point is four bytes long.
Line 690	Loop and read the data.
Line 710	Input the first value.
Line 720	Output the first value to the second column in the file.
Line 730	Input the second value.
Line 740	Output the second value to the third column in the file.
Line 750	The variable `seek_len` contains the offset from the beginning of the file to the placement of the second column.
Line 770	Position the file pointer to the next line of the second column.
Line 790	Close the file.
Line 800	Display "done!".
Line 820	If the file could not be opened, display an appropriate message.

Running the program

Set up the instrument with a DUT connected. Run the program. The program will ask for a filename to save data. Note that if this filename exists, it will be erased with no warning.

The stored data is formatted in three columns separated by commas. The first column is the stimulus frequency, the second column is the first value, and the third column is the second value. Refer to Table 2-1 for the units on value one and value two with respect to the display format.

Note that since the program does not store data in arrays, but rather in a file, it is not limited to the amount of memory available to it. Thus it is not limited to the number of points sampled.

Manual Part No.
08720-90155