NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF AN IBM-PC/AT
AS A GPIB CONTROLLER

by

George H. Self, Jr.

December 1986

Thesis Advisor: Prof. J. P. Powers

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

IMPLEMENTATION OF AN IBM-PC/AT AS A GPIB CONTROLLER

12 PERSONAL AUTHOR(S)

Self, George H., Jr.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1986 December	80

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	GPIB controller, GPIB, data collection system, instrumentation, computer controlled instrumentati

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis integrates an IBM-PC/AT microcomputer with five pieces of standard laboratory test equipment via a GPIB. A menu-driven program prompts the user to operate the test equipment from the keyboard on the PC. The user can perform a wide variety of tasks with this program and the program can be modified to perform other specific tasks desired by the user.

Two subroutines were developed to demonstrate the utility of this system and the use of the programming guidelines that were developed. A subroutine to collect waveform data from a digital oscilloscope and to plot the waveform with a plotter and a subroutine to generate a Bode plot of the transfer function for a two port network were developed.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. J. P. Powers	22b TELEPHONE (Include Area Code) (408) 646-2679	22c OFFICE SYMBOL 62Po

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited

Implementation of an IBM-PC/AT as a GPIB Controller

by

George H. Self, Jr.
Lieutenant, United States Coast Guard
B.S., U.S. Coast Guard Academy, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1986

ABSTRACT

This thesis integrates an IBM-PC/AT microcomputer with five pieces of standard laboratory test equipment via a GPIB. A menu-driven program prompts the user to operate the test equipment from the keyboard on the PC. The user can perform a wide variety of tasks with this program and the program can be modified to perform other specific tasks desired by the user.

Two subroutines were developed to demonstrate the utility of this system and the use of the programming guidelines that were developed. A subroutine to collect waveform data from a digital oscilloscope and to plot the waveform with a plotter and a subroutine to generate a Bode plot of the transfer function for a two port network were developed.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

This thesis investigated the use of an IBM-PC/AT microcomputer as a system controller for a set of programmable test equipment. It is the third in a series of theses that address the subject of using programmable test equipment for simple lab tests. The PC controls the equipment via a General Purpose Interface Bus (GPIB). Previous theses written at the Naval Postgraduate School (NPS) [Refs. 1 and 2] give detailed information about the GPIB and the test equipment used.

The HP-85 microcomputer used in the previous theses was replaced by the IBM-PC. The HP-85 is programmed to control via its Hewlett Packard Instrument Bus (HPIB) in BASIC. This limits the use of the HP-85 to programs developed on it in its particular version of BASIC. The programs written for the HP-85 are not transportable to different computers. The use of peripheral equipment such as printers and plotters is also restricted, making the HP-85 less flexible and powerful.

The PC can be programmed in a variety of languages, such as FORTRAN, PASCAL, and C to control equipment on a GPIB. The PC can also be used with a variety of peripherals to print, plot, store, manipulate, and display data. The use of the MS-DOS operating system also gives the PC a lot more flexibility as programs developed for the PC can be run on many similar computers that use MS-DOS.

The project undertaken here was to establish control of the various pieces of test equipment and operate them using an interactive menu-driven program running on the PC. The scope of control was to enable

the simple electronic engineering laboratory exercises taught at NPS to be implemented on the system. The user steps through a series of device menus to operate the test equipment and rarely has to adjust controls on the test equipment.

II. HARDWARE

This study made use of the following test equipment:

1. TEK PS 5010 Programmable Power Supply

2. TEK DM 5010 Programmable Digital Multimeter

3. TEK DC 5009 Programmable Universal Counter/Timer

4. TEK 5223 Digitizing Oscilloscope

5. WAVETEK MODEL 270 Programmable Function Generator

Detailed information about these five pieces of equipment is available in Ref. 1 and Ref. 2 as well as the manufacturers' technical documentation for the equipment.

This equipment was connected to the PC via a National Instruments IEEE-488 Instrument Interface. This chapter describes the pertinent hardware issues of the computer and the control board as they relate to this thesis.

A. IBM-PC/AT MICROCOMPUTER

In this study a PC was used as the GPIB controller. The large 30 megabyte capacity of the hard disk is needed to support the storage for a full featured programming language compiler. Without it, compiling a large program in a high level language degenerates into floppy swapping and becomes a real burden during the development of a large application program.

The PC does not have a GPIB connector and the required circuitry as standard equipment. It does come with several I/O slots that can accept a number of GPIB interfaces made by such companies as Tektronix, Capital Equipment Corporation, Hewlett Packard, and National Instruments to name a few.

B. NATIONAL INSTRUMENTS MODEL GPIB-PC2

A National Instruments Model GPIB-PC2 interface board was used to provide the hardware and software interface between the PC and the test equipment on the GPIB. Figure 2.1 is a photograph of this circuit board. It fits into one of the small slots of the PC and enables the PC to communicate with devices on the GPIB. Figure 2.2 is a functional block diagram of this circuit board.

The switches and jumpers on the card are used to configure it to work in a particular PC environment. The factory default setting of a base I/O address of 2B8, DMA Channel 1, and Interrupt Line (IRQ) for a GPIB TLC (Talker/Listener/Controller) of 7. These can be changed as needed when other devices already using these settings have been previously installed in a PC. The GPIB-PC User Manual contains detailed information on how to change these settings.

This circuit board can support up to sixteen devices on the GPIB and work in conjunction with another GPIB-PC2 card installed in the same PC to control another sixteen devices. This gives the PC the capacity to control up to thirty-two devices without another computer in the system. Figure 2.3 is a block diagram of how this system was configured.

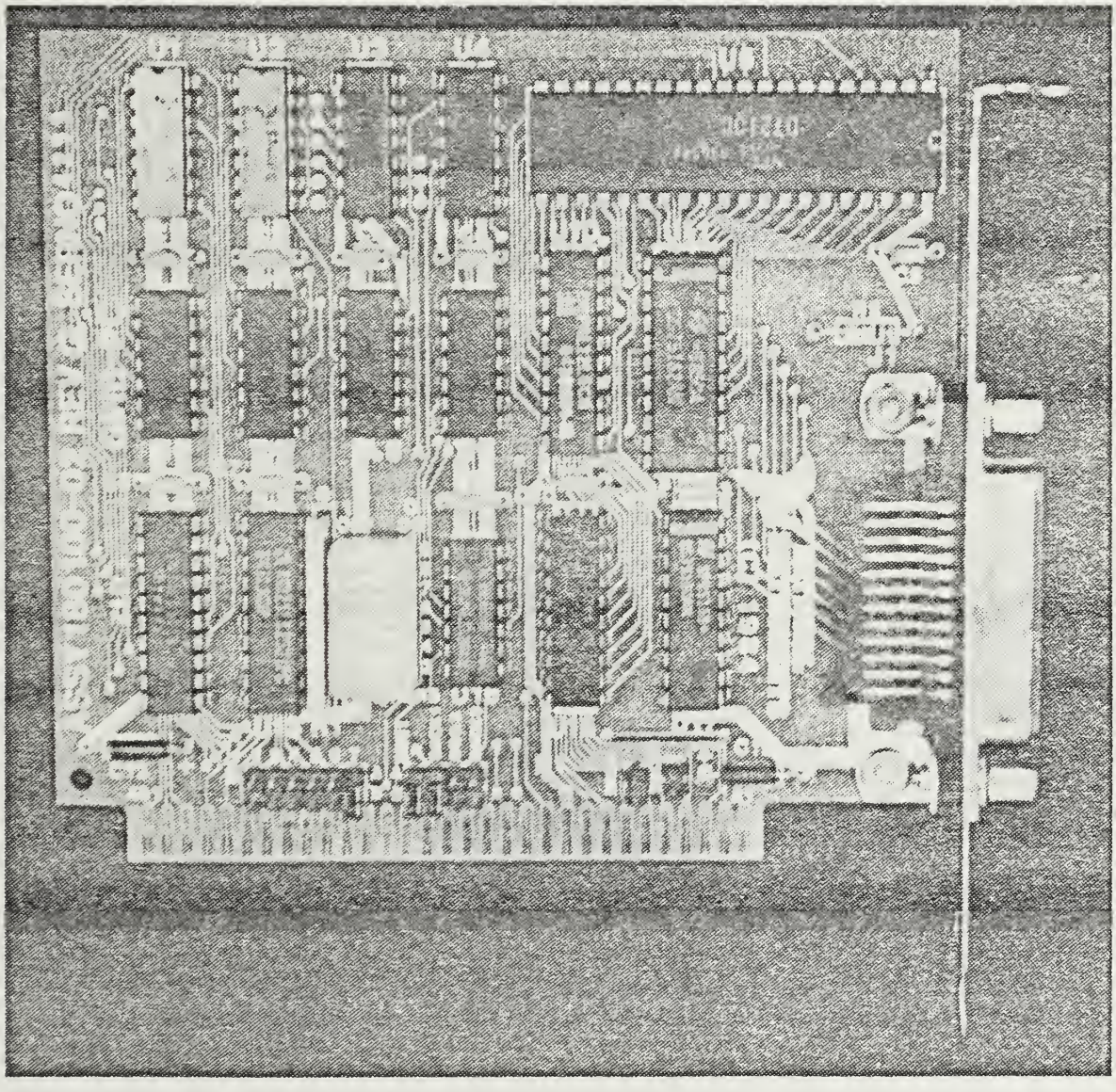

Figure 2.1 National Instruments GPIB-PC2 Circuit Board

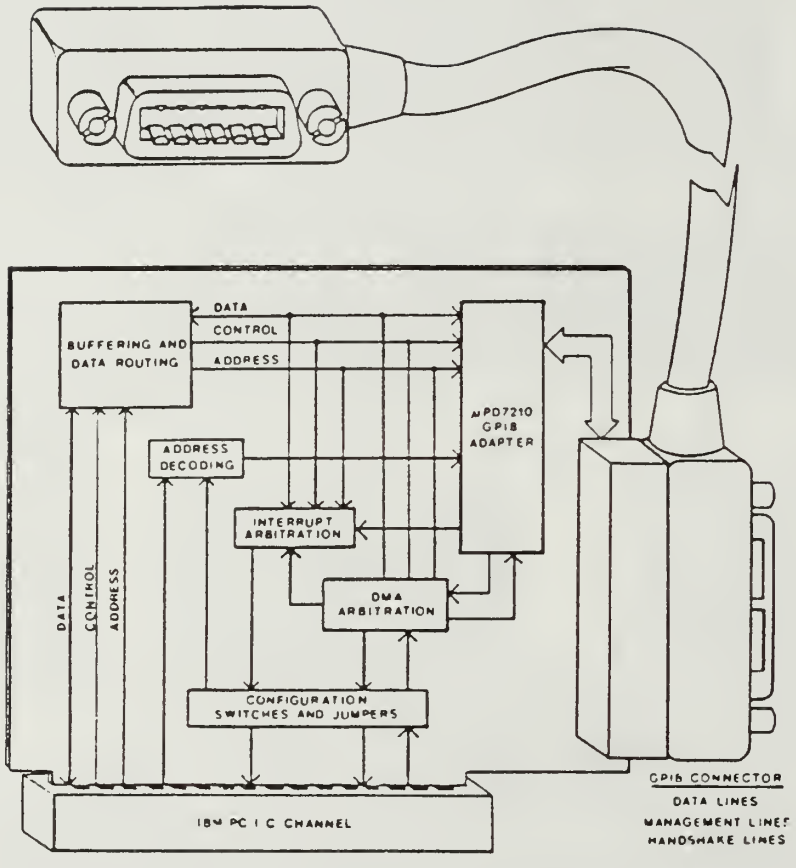

Figure 2.2 GPIB-PC2 Functional Block Diagram (From Ref. 3)

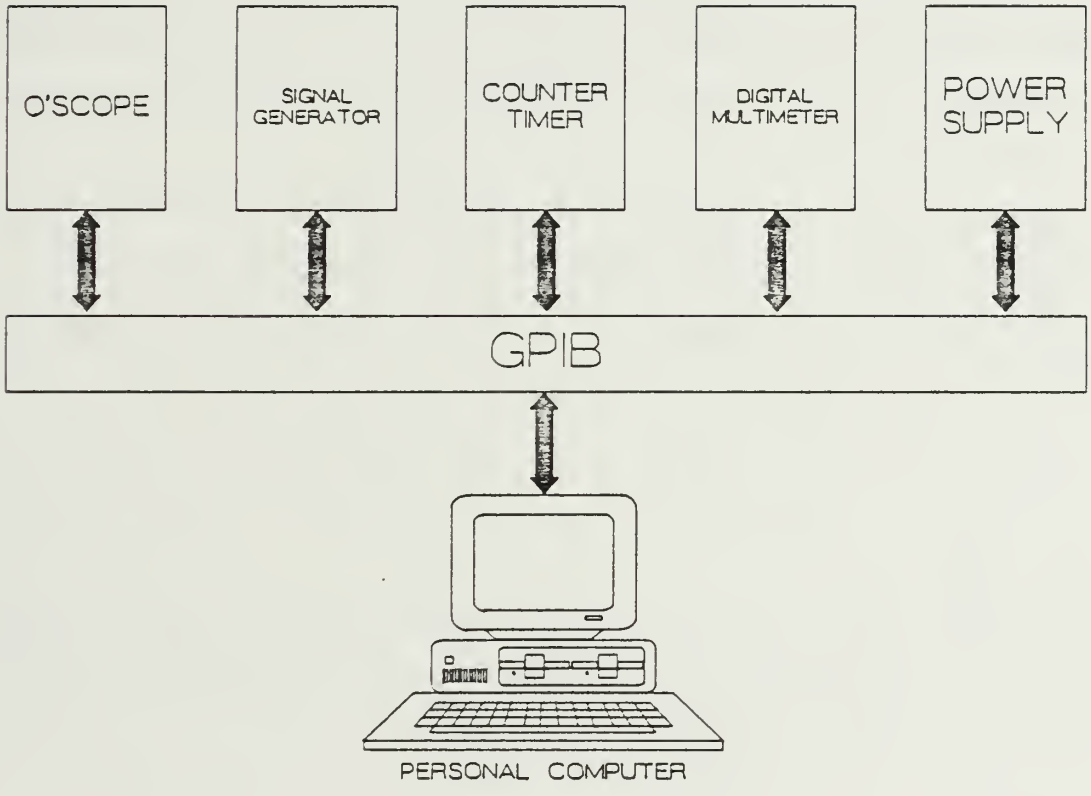

Figure 2.3 Block Diagram of Developed System

III. SOFTWARE

The circuit board provided by National Instruments (NI) comes with a software package that provides the DOS handler, language interface, installation package, and system configuration programs. The following programs were provided by NI:

1. GPIB.COM
2. IBSTART.BAT
3. MKCFG.EXE
4. IBSTA.EXE
5. IBSTB.EXE
6. IBDIAG.EXE
7. IBTEST.BAT
8. IBIC.EXE

Additional files are delivered for each language support option requested. To support programs written in Microsoft-FORTRAN 3.2 the following files were provided:

1. MFIB.OBJ
2. DECL.FOR
3. DFSAMP.FOR
4. BFSAMP.FOR

This chapter describes how these programs work and how software was developed to enable the PC to act as a GPIB controller.

A. DOS HANDLER

The file called GPIB.COM is loaded when the PC boots up. GPIB.COM is required to reside on the default boot drive to enable it to be installed during boot up. The term 'handler' is used by National Instruments to refer to a loadable device driver. DOS uses the DEVICE= command in a file called CONFIG.SYS to load the desired device drivers when the PC first boots up. GPIB.COM contains the software needed to operate the GPIB-PC2 circuit board as a GPIB I/O device.

B. INSTALLATION, CONFIGURATION, AND START-UP

The file IBSTART.BAT is a DOS BATCH command file that installs the software provided by NI. It copies the needed files off the floppy disk from NI and puts them on the desired disk. In this study the default disk drive is the PC hard disk, C:\, and the NI files were copied to a sub-directory, C:\GPIB-PC. GPIB.COM and IBCONF.EXE were then copied to the default drive C:\ as these files must reside in the default drive to operate properly. IBSTART.BAT adds the DOS command DEVICE=GPIB.COM to CONFIG.SYS by using the file MKCFG.EXE. The file IBDIAG.EXE is used to test the hardware before the associated software is installed.

Once the hardware and handler are installed, the file IBTEST.BAT is used to test both the hardware and software for proper installation and operation. This test is done in two parts by running IBSTA.EXE and IBSTB.EXE. All the tests are menu-driven and take only a few minutes to execute.

C. IBCONF

The file IBCONF.EXE is very useful even after the system is initially installed, as it allows devices to be added and deleted from the GPIB very easily. This routine helps to handle the specific details of setting up such things as GPIB addresses, system mnemonics, and end of instruction characters.

IBCONF runs as an interactive menu-driven program that has the user specify the device characteristics needed by the handler to properly address and communicate with a device on the GPIB. Figure 3.1 shows the first menu that is displayed when the program is run. Once a device on the bus is selected, the second menu (shown in Figure 3.2) is displayed and the user can change the GPIB attributes of device as required. If a change is made to a device's attributes in IBCONF, the PC must be rebooted so that the modified handler can be re-installed by DOS.

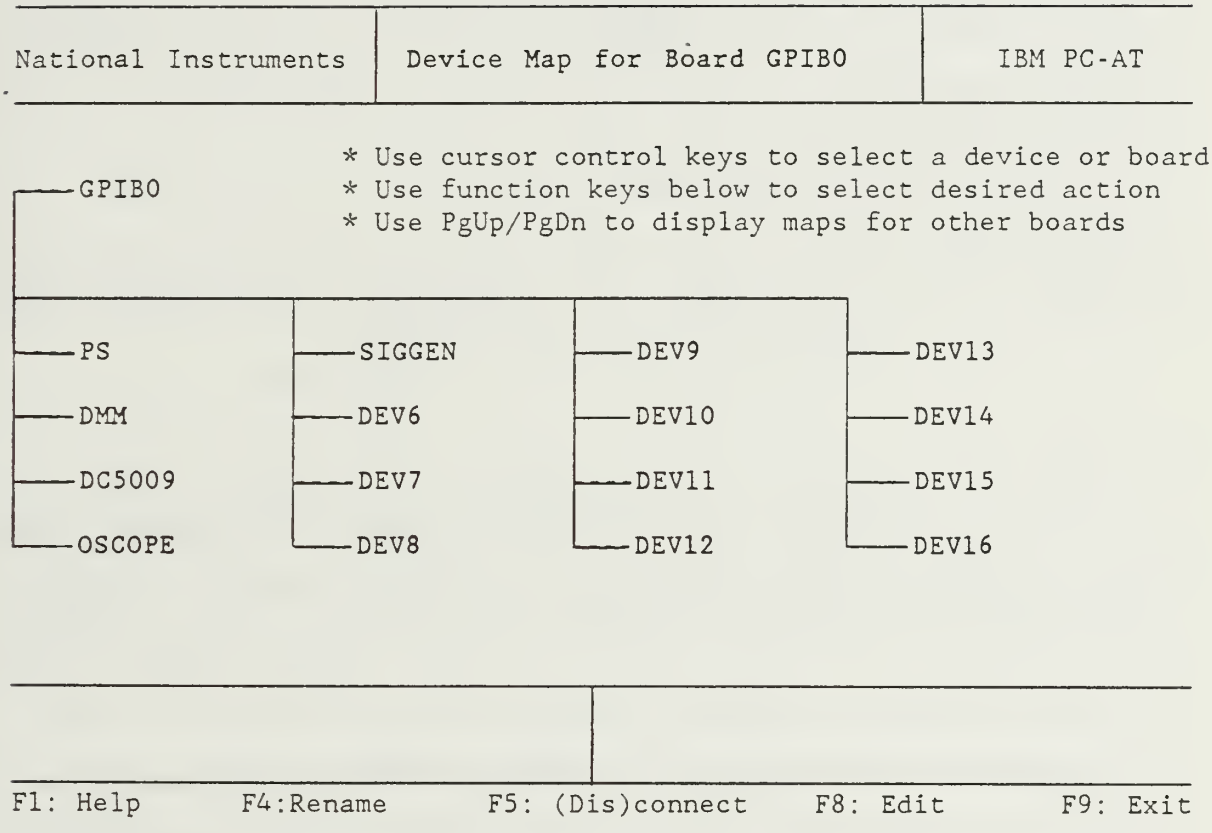

Figure 3.1 First Menu in IBCONF

National Instruments	Device Characteristics	IBM PC-AT
Device: PS Access:GPIB0	SELECT (use right/left arrow keys):	
Primary GPIB Address ..... 16H Secondary GPIB Address ..... NONE Timeout setting ..... T10s EOS byte ..... 00H Terminate Read on EOS ..... no Set EOI with EOS on Write .. no Type of compare on EOS ..... 7-bit Set EOI w/last byte of Write yes	00H to 1EH	
F1: Help F4: Explain Field F6: Reset Value F9: Return to Map		

Figure 3.2 Second Menu in IBCONF

D. USE OF IBIC

IBIC is the Interface Bus Interactive Control Program (IBIC). This program provides keyboard control of the GPIB and connected equipment. IBIC functions include most IEEE-488 commands, the functions supplied for specific language interfaces, and some functions specific to IBIC.

The functions allow the user to send a specific command to a device, to receive data (in the form of character strings) from the devices, and to display the data received on the PC screen. Data can also be saved to a file named in DOS path name convention. This allows a user to generate the correct command sequence to perform a specific task. It was used extensively in this study as the commands for each device are peculiar to that specific device. Practice is required for a user to become familiar with a device's command structure and IBIC is a good practice tool.

When IBIC is running, messages appear on the screen prompting the user to enter commands, data, or request help as needed. The commands used most are IBFIND, IBWRT, IBRD.

IBFIND is used to select a device on the GPIB. For example, to select a digital multimeter with the device mnemonic DMM installed via IBCONF the user could enter the following at the colon prompt:

```
: IBFIND DMM
```

IBWRT is used to send command strings to devices over the GPIB. To have the multimeter read resistance the following command might be entered at the DMM: prompt:

```
DMM: IBWRT "OHMS"
```

IBRD is used to read data from a device over the GPIB and display it on the screen. The number of bytes to be read are specified when the command is used as shown in the following example to read fifty bytes:

```
DMM: IBRD 50
```

The previous command results in the following display when the multimeter is measuring the resistance of an open circuit:

```
[2900] ( end rqs cmpl )
count: 9
31  2E  45  25  39  39  3B  0D      1.E+99;*
0A                                 *
DMM:
```

The first line of the above message is the status word IBSTA that describes the status of the GPIB in two forms: a hexadecimal value followed by a mnemonics list. The second line contains the actual number of bytes received from the device over the GPIB, in this case 9.

The next two lines contain the received characters and their ASCII codes. In this example 1.E+99 represents the infinite impedance of an open circuit. The two asterisks represent the two small diamond characters that actually appear on the display. These represent the carriage return and line feed indicated by the ASCII codes 0D and 0A in the third line of the display. The fourth line is the prompt, DMM:, for the next command.

More specific information and additional examples are contained in Reference 3 Section 5.

E. PROGRAMMING LANGUAGE INTERFACE

MFIB.OBJ is the Microsoft FORTRAN 3.2 language interface that enables that particular version of a FORTRAN application program to use subroutine calls that make use of the handler supplied by NI. Programs compiled with MS-FORTRAN 3.2 are linked with MFIB.OBJ to produce an executable file. MFIB.OBJ must <u>not</u> be the first file named in the link list when linking the application program.

Similar to IBIC, the most commonly used subroutines and functions are IBFIND, IBWRT, and IBRDF.

IBFIND is a function used to find the address of a device that has been installed on the GPIB via IBCONF. The integer returned is assigned to a variable that must be used in all references to that device in GPIB subroutine and function calls. The following is an example of how to use IBFIND in a FORTRAN program to assign the address of the multimeter to the integer variable DMM:

```
DMM = IBFIND ('DMM ')
```

(The software provided by NI requires the last character in a string be a blank to indicate the end of a string.)

IBWRT writes data to a GPIB device. It has three parameters: the device address, the data to be sent contained in an integer vector, and the number of bytes to be sent. The following is an example to command a digital multimeter to read resistance from a FORTRAN program:

```
WRT(1) = ICHAR('O') + ICHAR('H')*256
WRT(2) = ICHAR('M') + ICHAR('S')*256
CALL IBWRT (DMM,WRT,4)
```

NI requires character strings to be entered as shown to be compatible with the handler GPIB.COM. Characters are represented in FORTRAN programs run on the PC in memory as two bytes in low order byte then high order byte convention. The first two lines above squeeze two characters into one sixteen bit word and convert the characters to high order byte then low order byte convention. This word is transmitted over the GPIB as two sequential eight bit bytes that contain the character codes in the correct order to be used by devices on the GPIB.

Writing code in this way for every command string is very tedious. The subroutine STRING, shown in the following FORTRAN program listing, was written to put character strings into the integer array format required to be used with IBWRT. An explanation of this subroutine follows:

```
      SUBROUTINE STRING (INPUT,LENGTH,WRT)
C***** THIS CONVERTS CHARACTER STRINGS INTO REQUIRED FORM FOR IBWRT **
      CHARACTER*1 INPUT(30)
      INTEGER LENGTH,I,J,K,WRT(512)
      J= 1
      DO 10 I=1,LENGTH,2
            K= I+1
            WRT(J)= ICHAR(INPUT(I)) + (ICHAR(INPUT(K))*256)
```

```
                J= J+1
10      CONTINUE
        RETURN
        END
```

The code:

```
        SUBROUTINE STRING (INPUT,LENGTH,WRT)
        CHARACTER*1 INPUT(30)
        INTEGER LENGTH,I,J,K,WRT(512)
        J= 1
```

establishes the subroutine STRING with the formal parameters: INPUT, LENGTH, and WRT. INPUT is the character string to be modified and LENGTH is the number of characters in the STRING. WRT is the integer array returned by STRING to be used with IBWRT. I, J, and K are the indices of the arrays INPUT and WRT.

The code:

```
        DO 10 I=1,LENGTH,2
                K= I+1
                WRT(J)= ICHAR(INPUT(I)) + (ICHAR(INPUT(K))*256)
                J= J+1
10      CONTINUE
```

takes the elements of INPUT in pairs and performs the operation needed to generate the elements of WRT.

IBRDF reads data to a file. It has two parameters: the device name and the filename under which the data is stored. An example of how to read a resistance value from the multimeter at address DMM and store it in a file called DATA on the A: disk in a FORTRAN program follows:

```
        CALL IBRDF (DMM,'A:DATA ')
```

The DECL.FOR is a file of FORTRAN variable declarations recommended for use by NI and is included as Appendix A. The three global variables

IBSTA, IBERR, and IBCNT must be used in all FORTRAN programs to enable the GPIB status, any detected bus errors, and the number of bytes transmitted during a message to be available from the handler. These variables are updated after each subroutine call to reflect the status of the most recently referenced device and the status of the GPIB.

IBSTA is the status word returned by all functions. This contains information about the GPIB status. It can be used to check for proper bus operation and bus status such as I/O complete or a device requests service.

IBERR is the error variable containing the error code when an error is detected. The error codes indicate such problems DOS errors, invalid arguments to function calls, or file system errors.

The IBCNT variable is updated after each read or write is executed. It contains the number of bytes transferred during the last read or write.

The program listing in Appendix B was generated making use of these variables and functions. There are many more functions and variables available. A user can also write new functions and specify new variables if needed.

The two files DFSAMP.FOR and BFSAMP.FOR are example programs provide by NI that show how to write application programs that make use of the subroutines provided in MFIB.OBJ.

F. PLOTTING PACKAGE

The plotting package SlideWrite Plus produced by Advanced Graphics Software was used for all waveform and data plots in this study. It is one of the many plotting packages available for use with the PC.

IV. DEVELOPMENT OF THE SYSTEM CONTROLLER

The goal of this study was to develop a system for use in student laboratory environment as a teaching aid. Students at NPS could run programs on a computer connected to various pieces of lab test equipment. The student would perform circuit tests and demonstrations of class room theory through menus displayed on the computer monitor. Students would set up and control instruments by typing in responses on the computer keyboard. Previously written and stored programs could put waveforms on the monitor or the digitized oscilloscope, check data values at test points for correct values, and record and store data automatically for later use. Off-the-shelf software would be used for analyzing and plotting the recorded data.

With these goals in mind a set of design guidelines were developed. This chapter describes these guidelines and details the software that was developed.

A. DESIGN GUIDELINES

Design of this system was undertaken as a top-down, structured programming implementation. A top-down structure was chosen early in the study. This enabled a gradual system development starting with the most essential program features first and then progressing to more complicated functions. To this end, step-wise refinement was used extensively.

The concept of modular programming was followed. All new functions were added as subroutines that would not affect the code previously

written and tested. These new functions were kept simple so as to be short. This limited the size of a program module that had to be written, tested, and debugged to a manageable size.

All control of the test equipment was to be via the computer keyboard. The student would not have to adjust the equipment by hand. Menus on the computer monitor would give instructions to the student as required.

B. EARLY EFFORTS WITH BASICA

National Instruments provides a handler package to be used with BASICA. (BASICA is an enhanced BASIC written for the IBM/PC by Microsoft.) Early in this study, this handler was used with some simple test programs written in BASICA. These programs were found to be very slow during run time since BASICA is an interpretative language. In addition these BASICA programs were not transportable to all IBM/PC compatible computers. Due to these limitations another programming language was selected.

C. SELECTION OF FORTRAN 77

A language having separate compilation was desired to facilitate the modular programming style desired. It was anticipated that the analysis on data collected by the system could be handled most advantageously by using the some of the many subroutines already developed and available in FORTRAN libraries. A FORTRAN 77 subset compiler suitable for use on an IBM/PC was available in the lab for use and was selected as the system development language. This was Microsoft-FORTRAN 3.2.

D. THE DEVELOPED PROGRAM - GPIBX

The final program developed, GPIBX, is included as Appendix B. Once compiled as an executable file it requires around 100K bytes of RAM. Execution of most commands appears to be immediate to the user. The demonstration subroutines entail some delay due to the number of operations being performed sequentially and the disk I/O for storage being performed.

The program presents the user with a series of menus that enable the user to remotely operate most of the front panel controls available on the different pieces of test equipment. After a piece of equipment is selected, a new menu specific to that piece of equipment is presented. The user can then select a particular operation to perform or setting to adjust. Specific values for voltages or frequencies can be entered via the PC keyboard. Numbers can be entered in scientific notation to save time.

The program only accepts certain responses depending on the menu selections available. It prompts the user to try again when invalid responses are entered. Settings outside the range of the equipment are ignored.

Control of the different pieces of equipment is accomplished one level at a time. First, the device is selected from a main menu. Then the user selects a particular feature of the device to adjust or operate. The menus presented to the user proceed in one level of control each time. The user can back-up one level of control at any time. To switch devices a user must back out to the main menu level and then select another device. This prevents a user from jumping around the

menus and inadvertently adjusting the wrong piece of equipment. A single point of return for each subroutine enforces this run time operation.

E. PROGRAMMING PROBLEMS ENCOUNTERED

MS-FORTRAN 3.2 was not the best language to use for development of this system. The bulk of the programming required character string manipulations. A full featured ANSI FORTRAN 77 compiler may have been satisfactory but MS-FORTRAN 3.2 is a subset and does not have some of the FORTRAN 77 character string handling features that were needed.

The first programming problem found was the lack of substring support. Device commands are typically a string of characters specifying an operation and a data value to use. Building these strings could be handled in FORTRAN 77 by concatenating substrings. Since neither substrings nor string concatenation is available in MS-FORTRAN 3.2, blocks of code had to be written for each operation similar to the following listing of a subroutine in GPIBX. An explanation follows:

```
        SUBROUTINE FREQ
C***** THIS MAKES THE SIGGEN OUTPUT A SPECIFIED FREQUENCY *****
        INTEGER DVM,I,WRT(512)
        CHARACTER*1 FREQ(13),INPUT(11)
C
        FREQ(1)= 'F'
        FREQ(13)= 'I'
C
        DVM= IBFIND ('SIGGEN ')
        WRITE (*,10)
10      FORMAT('0',9X,'ENTER DESIRED FREQUENCY AS XX.XEX (.01Hz-12MHz)')
C
        READ (*,20) INPUT
20      FORMAT(11A1)
C
        DO 30 I= 1,11
                FREQ(I+1)= INPUT(I)
30      CONTINUE
```

```
C
      CALL STRING (FREQ,13,WRT)
      CALL IBWRT (DVM,WRT,13)
      RETURN
      END
```

The code:

```
      SUBROUTINE FREQ
      INTEGER DVM,I,WRT(512)
      CHARACTER*1 FREQ(13),INPUT(11)
```

establishes the subroutine FREQ and declares the variables used in the subroutine. DVM (DeVice Mnemonic) is a variable used to contain the integer representation of a device address on the GPIB. I is a variable used for the index of the arrays declared. WRT is the integer array containing the command string to be sent over the GPIB by the subroutine IBWRT. It has 512 as its upper dimension limit to enable it to accept as large a string as needed.

The code:

```
FREQ(1)= 'F'
FREQ(13)= 'I'
```

puts a 'F' in the first element of FREQ and an 'I' in the last element of FREQ. The WAVETEK MODEL 270 Programmable Function Generator has a command string format for selecting frequencies that require the first character to be a 'F' followed by characters that specify the desired frequency, such as '100000' or '1E5' for 100 KHz. If the last character in the string is an 'I' then the command is executed immediately. The code:

```
DVM= IBFIND('SIGGEN ')
```

uses the function IBFIND to place the integer representation of the address of the device SIGGEN (signal generator) in the variable DVM.

The code:

```
        WRITE (*,10)
10      FORMAT('0',9X,'ENTER DESIRED FREQUENCY AS XX.XEX (.01Hz-12MHz)')
        READ (*,20) INPUT
20      FORMAT(11A1)
```

generates an on-screen menu that requests the user to enter a frequency via the keyboard.

The code:

```
        DO 30 I= 1,11
                FREQ(I+1)= INPUT(I)
30      CONTINUE
```

builds command strings using previously stored characters and those entered by the user.

The code:

```
        CALL STRING (FREQ,13,WRT)
        CALL IBWRT (DVM,WRT,13)
        RETURN
        END
```

uses the subroutine STRING to put the command string FREQ in the form required for use with the subroutine IBWRT. IBWRT sends the integer array WRT that represents the command string in FREQ to the device identified by the variable DVM. The 13 is the number of bytes to be transferred and is required for both STRING and IBWRT.

The subroutine IBWRT has three parameters. It requires the GPIB address of where the message is going, the integer vector to be passed, and the number of characters represented in the integer vector being passed. MS-FORTRAN 3.2 does not have an intrinsic function to return the length of a string. The programmer must keep track of the length of strings being passed. In some cases the length of a string depends on run-time input from the user. To cover all the possible cases the

character string variables used to input command strings are larger than always needed. This adversely affects both memory requirements and run-time efficiencies.

The most difficult problem encountered was reading data returned from a device. The subroutine IBRDF reads data from a device into a file. If the file did not previously exist, a new one is created. Once the data has been copied into the file, IBRDF closes the file.

The book, Structured FORTRAN 77 For Engineers And Scientists (Ref. 4), was heavily relied on as a FORTRAN 77 reference. It does not list binary files as one of the file types supported. After many unsuccessful attempts of reading the data from one of these files a call was made to the Microsoft Users Hotline. A Microsoft representative recommended opening the file as a BINARY type of file explaining that MS-FORTRAN 3.2 does support binary files. The following is an example of FORTRAN code that was used to successfully open files for use in the system controller program:

```
OPEN (1,FILE='A:DATA',STATUS='OLD',FORM='BINARY')
```

These BINARY files contained numeric data that could not be read with formatted read statements. Characters have to be read out one at a time into arrays before the data is available for numeric processing. Some data files can have over 5,000 characters and take a lot of time to be read into arrays.

The data format used by specific devices can cause problems as well. The waveform data from the TEK 5223 Oscilloscope comes over the GPIB as a string of numbers separated by commas. The numbers are one, two, or

three digits long and can have a minus sign in front. It is not possible to read this string of characters with a formatted read statement to pull out the separate data values. A great deal of code was written just to get this data in a usable form.

V. DEMONSTRATION SUBROUTINES

Once the basic objectives of this study were met, further work was done to investigate the issues involved with developing software modules to perform a sequence of the basic operations already developed. To this end a subroutine to generate a BODE plot of a two port network and a subroutine to plot the waveforms shown on the digitizing oscilloscope were written. This chapter discusses these two subroutines and highlights some of the problems encountered.

A. BODE PLOT SUBROUTINE

The BODE plot subroutine is selected in the SIGNAL GENERATOR MENU. This subroutine generates and records the data necessary to make a plot of input frequency vs. gain magnitude for a two port network. The user can select the starting and stopping frequencies to be swept while data is being recorded. The user also selects the number of points to be taken, up to 400. More points could be taken but the plotting software, SlideWrite Plus, can plot a maximum of only 400 total data points on a graph. This corresponds to one line of 400 points or two lines of 200, etc.

The rms voltage of the output and the corresponding input frequency are recorded for each frequency generated. Figure 5.1 is block diagram of how the system is configured to conduct this test. The magnitude of the transfer gain is calculated as $20*\log(V_{out}/V_{in\ rms})$ and recorded in a file with the corresponding input frequency. This data is then plotted with the SlideWrite Plus plotting package. The circuit shown in

Figure 5.2 was examined with this subroutine and the corresponding Bode Plot is shown in Figure 5.3. The input signal is assumed constant over all frequencies of the sweep.

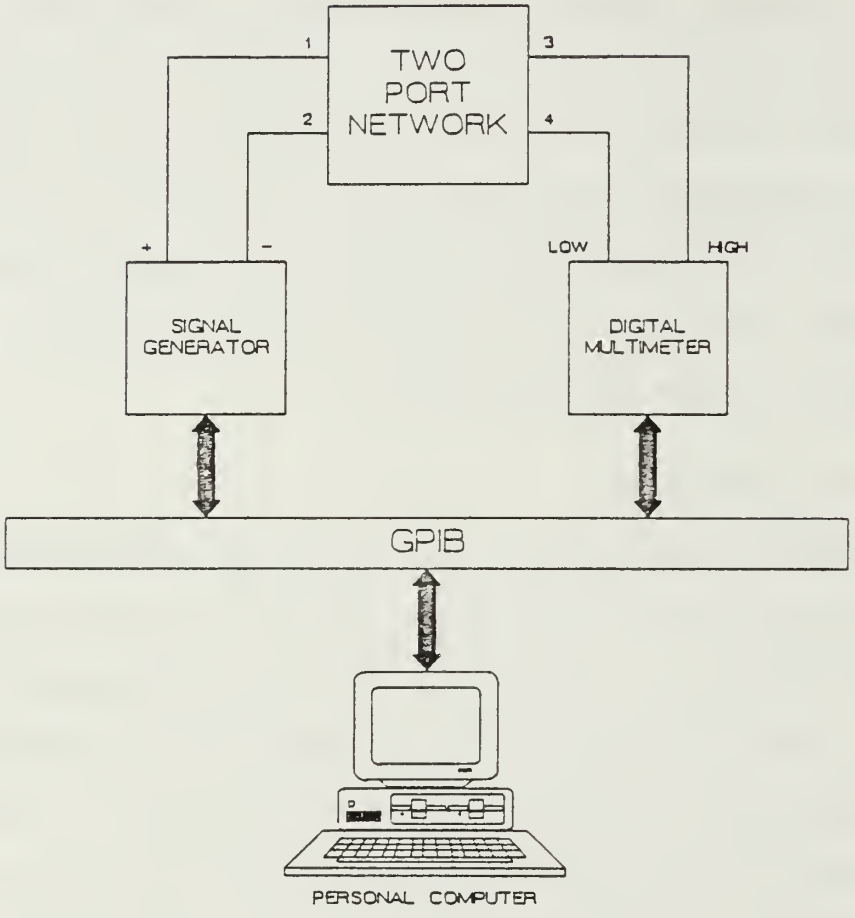

Figure 5.1 Block Diagram of Bode Plot Test

The Bode Plot shown in Figure 5.3 appears to represent the response of a Low-Pass Filter. The circuit is made up of passive elements so the gain is always zero DB or less. The circuit gain decreases as the input frequency increases. There appears to be a resonant frequency up at around 1 MHz. This is probably due to some stray reactances present in the circuit components that have little effect at the lower frequencies.

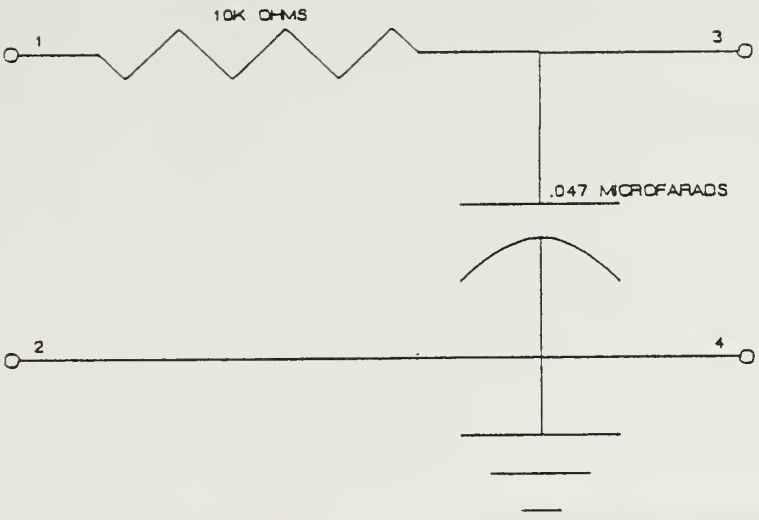

Figure 5.2 Low-Pass Filter

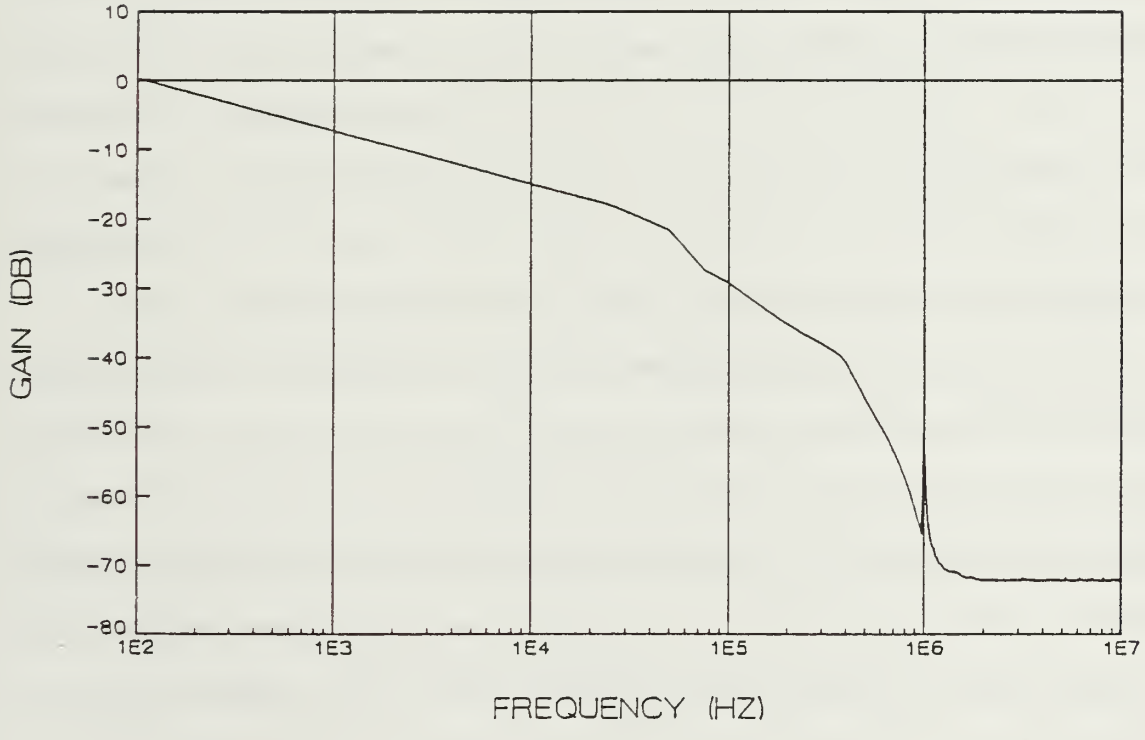

Figure 5.3 Bode Plot of the Low-Pass Filter Circuit

Caution must be used when using this Bode Plot subroutine as some circuits tested were observed to load the signal generator at various frequencies and cause the input voltage to fall off. The change in input voltage may be very slight and not visible on the oscilloscope. The input voltage should always be checked with a voltmeter to be sure it remains constant. This problem could be corrected by using two voltmeters connected to the GPIB or by using a multiplexer to enable one voltmeter to take two different readings at the same frequency.

B. WAVEFORM RECORDER SUBROUTINE

The waveform recording routine is selected in the OSCILLOSCOPE MENU. The oscilloscope samples the input signal on either input channel and digitizes the amplitude. The number of sample data points generated depends on the sweep rate selected and the number of channels being displayed. For one dual trace amplifier in use and a sweep rate of not less than .1 msec/div, 512 points are taken representing a period of time equal to ten time divisions (one full oscilloscope screen). For sweep rates less than .1 msec/div, 1024 points are taken. The increase in the number of points is to avoid aliasing by under-sampling.

The plotting software, SlideWrite Plus, can plot a maximum of 400 points. The waveform recording subroutine selects 400 out of 512 or 341 out of 1024 points for plotting. The 400 out of 512 are selected by taking the first four of every five points up to 500 points and ignoring the last twelve points. The 341 points are selected by taking the first of every three data points up to 1024 points. Either number of points

do a fine job of representing a five inch wide oscilloscope trace with a few cycles of a waveform displayed.

No scaling information is available from the oscilloscope via the GPIB. The user must enter the volts/div and time/div to allow the subroutine to properly scale the data. The numbers from the oscilloscope represent only the voltage amplitude information. Each vertical scale division is 100 units, so a value of 250 represents 2.5 divisions above the y-axis and a value of -320 represents 3.2 division below the y-axis. The waveform recorder subroutine uses the volts/div to scale the numbers transferred from the oscilloscope into volts.

This subroutine also calculates the time scaling information by using the time/div entered by the user and the user's response to questions about how many waveforms are being displayed. Both pieces of information dictate how many data points are sampled by the oscilloscope during one display trace.

More expensive oscilloscopes have the scaling information available over the GPIB.

Figure 5.4 shows a plot of a 1 KHz sine wave output by the WAVETEK MODEL 270 into Channel 1 of the TEK Oscilloscope. The signal has a peak amplitude of five volts. The volt/div is set on five volts/div and the time/div is set on .2 msec/div. This waveform is represented by 512 points sent from the oscilloscope and has been plotted with 400 points. The plot is a very smooth sine wave.

Figure 5.5 is a plot of the waveform shown on the oscilloscope when a 1 MHz square wave is output from the signal generator into Channel 1 of the oscilloscope. The volts/div is set on two volts/div and the

time/div is set on 0.5 microseconds/div. The plot is a very accurate representation of the waveform shown on the oscilloscope. The slight distortion of this high frequency signal is clearly visible in the plot.

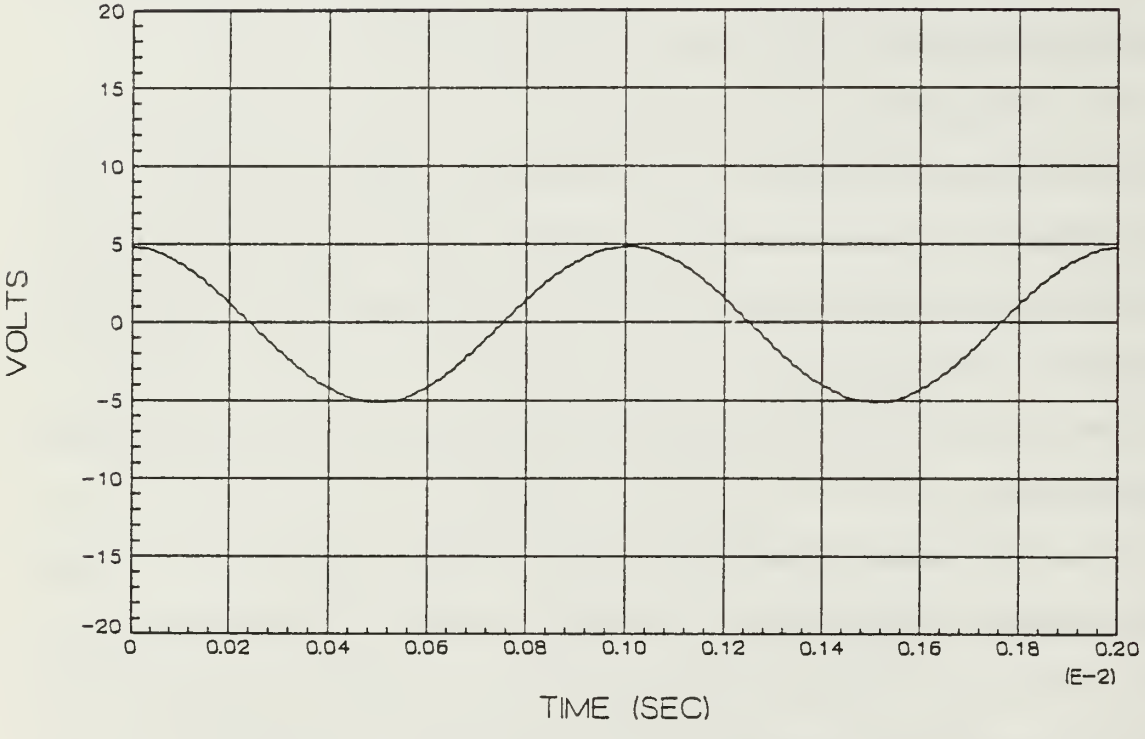

Figure 5.4 Plot of 1 KHz Sine Wave Shown on O'Scope

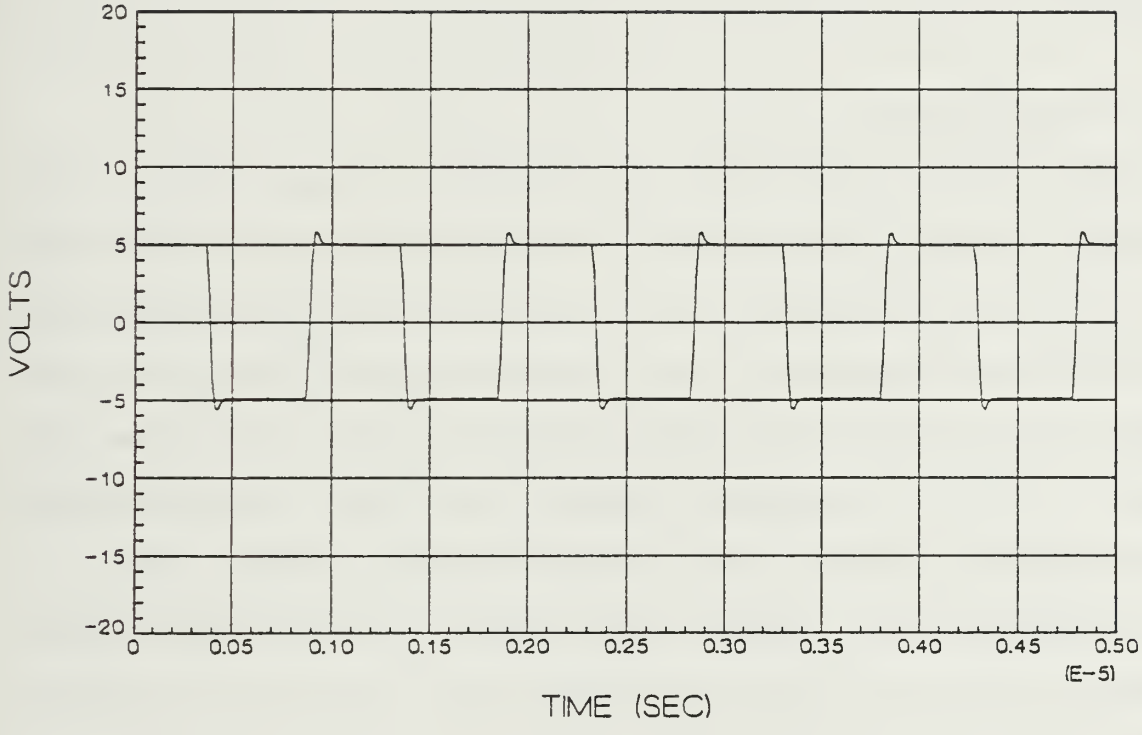

Figure 5.5 Plot of 1 MHz Square Wave Shown on O'Scope

VI. PROGRAMMING GUIDELINES

While developing this system certain steps of the development were repeated. A set of steps evolved to be performed every time a new function was implemented in software. This chapter outlines these steps and details items to watch for when developing modules to use in the system.

A. OUTLINE THE TASK

The task to be performed was outlined on paper. This included drawing a schematic diagram clearly showing test points. Each test point was labeled to indicate the test equipment connections. The connections of the test equipment to be used, such as Channel A, Ground, High, and Guard were indicated. Labels for these points were assigned variable names in the system development language. These four connection points were labeled in FORTRAN 77 as CHANA, GRND, HIGH, and GUARD.

Sketching a block diagram of the process to be performed is very helpful. This can be done using standard flow charting techniques. A block diagram of the process helps to show how control in the program flows and where data flows during execution.

B. MODEL TASK WITH IBIC

The program IBIC was run to develop the command strings necessary to perform the task outlined. These command strings are unique for each piece of equipment and are detailed in each equipment's technical

reference manual. Using IBIC allows for interactively entering command strings, executing them, and observing the results.

C. GENERATE CODE

Once the necessary command strings have been generated, program code in the programming language of choice can be written. This study used Microsoft FORTRAN 3.2, a subset of ANSI FORTRAN 77. Many of the issues discussed apply to other languages as well.

Globally scoped variables were avoided. The handler from NI uses three global variables: IBSTA (status word), IBERR (GPIB error code), and IBCNT (the number of bytes sent). These should be the only globally scoped variables used. Using only locally scoped variables avoids the problems of side-effects and indiscriminate access that global variables are subject to. This speeds program development time, improves program readability, and facilitates software maintenance.

Program modules were developed separately from the existing system program. This kept the amount of text to review with the editor being used to a minimum. For this study Wordstar was used as the text editor and allowed the user to view twenty-five lines of programming code in one screen. The program modules under development were written as subroutines. This ensured the proper identification of formal parameters required to be passed back and forth when the module was integrated into the existing system program.

Previously developed subroutines were used when applicable. The editor can then copy blocks of text into files. The MS-FORTRAN Metacommand $INCLUDE can be used to include other files for compilation with

the one under development. This helps to keep the size of the file being edited to a minimum.

Once the module is written and compiled, it should be tested extensively. By using the windowing environment of DESQview the system program GPIBX, IBIC, and the developed module were run in parallel. This facilitated testing the module as it allows observing the real time interaction of the module with the system program without actually changing the system program. The two programs are still separate and cannot change the operation of either's code.

D. INTEGRATE NEW CODE INTO GPIBX

When the new module performs correctly, it can be implemented in the system program, GPIBX.FOR. The editor moves the necessary block of code from the developed module file into the GPIBX.FOR file. The necessary modifications were made such as the text of menus, additional function options, the addition of subroutine calls and returns, and the addition of comment lines to document the new subroutines.

All subroutines developed in this study have a single point of return. This gives up some flexibility in programming but helps program readability. This is why menu selection proceeds in one level at a time and out one level at a time.

VII. CONCLUSIONS AND RECOMMENDATIONS

This thesis took a close look at the GPIB interface circuitry and software made by National Instruments to enable an IBM-PC to be a GPIB system controller. Software was developed to implement interactive control of the test equipment from the computer keyboard. A subroutine to enable waveform data acquisition from the TEK 5223 Digitizing Oscilloscope and to plot the data using software provided by Advanced Graphics Software Inc. on a HP740A plotter was developed. Another subroutine to generate a Bode Plot for a two-port system was developed.

A. CONCLUSIONS

The basic electronic laboratory equipment used at NPS are manual versions of the test equipment used in this study. This study has shown how an interactive program could be developed to allow automation of several of the processes involved in executing basic laboratory exercises such as data acquisition, waveform plotting, and Bode Plots. The results obtained point out several concluding points:

1. Selection of a system development language is key. MS-FORTRAN 3.2 doesn't support enough of the FORTRAN 77 extensions to make development as straight-forward as possible. The bulk of programming involves string manipulations. A language such as C is probably better suited to this application.

2. Selection of an IBM-PC based computer enabled use of several different software packages for the IBM-PC and its compatibles. Editing the software was performed with WORDSTAR by MICROPRO. Operating the computer as a development system was done with DESQVIEW by QUARTERDECK. Plotting of data was done with SLIDEWRITE PLUS by ADVANCED GRAPHICS SOFTWARE. Using a widely supported computer such as the PC makes an extensive amount of software available giving any system development undertaken a lot of tools to use.

3. Use of a window based operating system such as Desqview allowed for several different IBIC sessions to be run at what appears to be the same time to the user; they actually run one at a time. This allowed for quick investigation of the necessary command strings to have a specific task performed by a particular piece of test equipment. Several pieces of equipment can be operated at the same time this way. The developed program, GPIBX, can also be run and the interaction of the GPIBX and the test equipment can be observed. A windows environment greatly speeds system development.

4. Great care must be used when selecting software packages to be used with the system. A plotting/graphics package written by Enertronics was first tried to handle the plotting requirements. This software was not able to generate logarithmic plots as advertised. It also did not plot as many data points per line as stated in the manual. These deficiencies led to the use of SlideWrite Plus for plotting.

5. Similar caution must be used when selecting GPIB devices as well. The TEK oscilloscope sends back graphic data as a string of ASCII characters. It requires a lot of program code to put these characters in a form usable in FORTRAN to express numeric data.

6. Developing a program that is menu driven and allows the user to specify a series of operations, tests , and measurements for the GPIB controller to perform is beyond the scope of what a student can do as a thesis assignment. Such a system is technically feasible. NI recently began marketing just such a software/hardware package called Labview. It is written for an Apple Macintosh computer and may not have the I/O flexibility needed to make use of the different peripherals required for a specific development.

7. The GPIB connector is made to allow stacking several connectors at the same connection point. As a result, the cable feeds in at a right angle to the connector. On the IBM-PC the arrangement of the connector in the back of the computer is such that the cable binds up against the computer housing. A GPIB socket extender would eliminate this annoying problem.

B. RECOMMENDATIONS

Further thesis work should be done evaluating some of the new software packages available at this time or this system could be developed into a fully interactive lab teaching aid for the elementary labs taught at NPS. As work progressed more sophisticated labs could be automated.

The development language should be changed to one having good character string handling primitives. Selection of a language should be based on an evaluation of the documentation available, portability of compiled programs to run on different but compatible computers, run time speed of compiled programs, and the programming experience of the programmer.

The local operation of devices on the GPIB should always be available to allow students the opportunity to investigate their circuits outside the control of the computer program. Too much automation would be detrimental to a student's understanding of how an instrument works and what its capabilities and limits are. Observation of the operation of test equipment provides a lot of real world experience in applying concepts developed in different classes.

A proposed student work station that could be used in an automated lab environment is shown in Figure 7.1. The block diagram shown in Figure 7.2 shows how these work stations could be connected to a single computer, printer, and plotter. The cost of personal computers used for this application may be low enough to have a dedicated personal computer for each work station. Having a computer at each work station would prevent a single computer failure from stopping the work of all students.

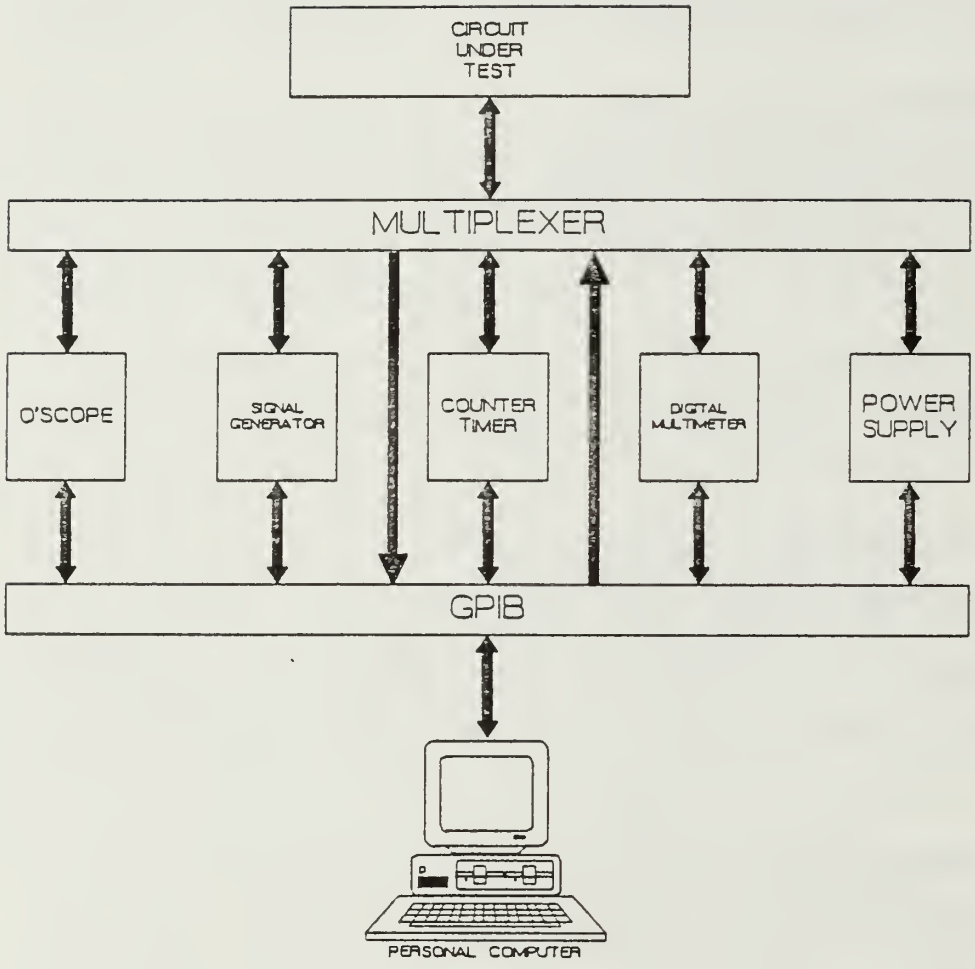

Figure 7.1 Proposed Student Work Station

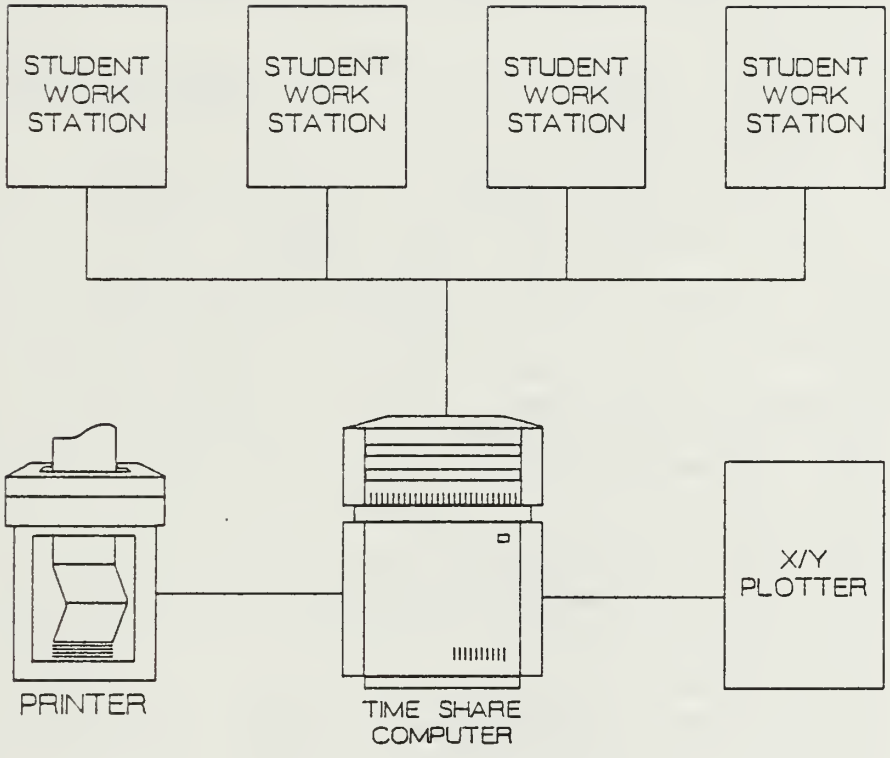

Figure 7.2 Proposed Time Share System

APPENDIX A

DECL.FOR LISTING

This is a listing of the Fortran variable declarations recommended for use and provided by National Instruments. This listing was provided by National Instruments with the exception of the list of variables appearing as all capital letters.

```
c Microsoft FORTRAN Declarations

$storage:2

c You must include the following common declarations
c in your program.
c
c status variables declared common by the language interface
c ibsta - status word
c iberr - GPIB error code
c ibcnt - number of bytes sent

        common /ibglob/ ibsta, iberr, ibcnt

c Optionally include the following declarations in your
c  program.
c
c GPIB Commands and meanings
c     UNL - GPIB unlisten command
c     UNT - GPIB untalk command
c     GTL - GPIB go to local
c     SDC - GPIB selected dev clear
c     PPC - GPIB ppoll configure
c     GET - GPIB group execute trig'r
c     TCT - GPIB take control
c     LLO - GPIB local lock out
c     DCL - GPIB device clear
c     PPU - GPIB ppoll unconfigure
c     SPE - GPIB serial poll enable
c     SPD - GPIB serial poll disable
c     PPE - GPIB ppoll enable
c     PPD - GPIB ppoll disable

        integer UNL, UNT, GTL, SDC, PPC, GET, TCT
        integer LLO, DCL, PPU, SPE, SPD, PPE, PPD

c GPIB status bit vector :
c       global variable ibsta and wait mask
c      ERR (hex 8000) - Error detected
c      TIMO (hex 4000) - Timeout
c      END (hex 2000) - EOI or eos detected
c      SRQI (hex 1000) - SRQ detected by CIC
```

```
c      RQS (hex 800) - Device needs service
c      CMPL (hex 100) - I/O completed
c      LOK  (hex   80) - Local lockout state
c      REM  (hex  40)  -  Remote  state
c      CIC  (hex   20) - Controller-in-charge
c      ATN  (hex   10) - Attention asserted
c      TACS (hex    8) - Talker active
c      LACS (hex    4) - Listener active
c      DTAS (hex    2) - Device trigger state
c      DCAS (hex    1) - Device clear state

        integer ERR, TIMO, END, SRQI, RQS, CMPL, LOK
        integer REM, CIC, ATN, TACS, LACS, DTAS, DCAS

c Error messages returned in common variable iberr
c     EDVR = 0 DOS error
c     ECIC = 1 Function requires board to be CIC
c     ENOL = 2 Write function detected no Listeners
c     EADR = 3 Interface board not addressed correctly
c     EARG = 4 Invalid argument to function call
c     ESAC = 5 Function requires board to be SAC
c     EABO = 6 I/O operation aborted
c     ENEB = 7 Non-existent interface board
c     EOIP = 10 I/O operation started before previous operation completed
c     ECAP = 11 No capability for operation
c     EFSO = 12 File system operation error
c     EBUS = 14 Command error during device call
c     ESTB = 15 Serial Poll status byte lost
c     ESRQ = 16 SRQ remains asserted

        integer EDVR,ECIC,ENOL,EADR,EARG,ESAC,EABO
        integer ENEB,EOIP,ECAP,EFSO,EBUS,ESTB,ESRQ

c EOS mode bits
c     BIN (hex 1000) - Eight bit compare
c     XEOS (hex 800) - Send EOI with eos byte
c     REOS (hex 400) - Terminate read on eos

        integer BIN, XEOS, REOS

c Timeout values and meanings
c     TNONE  =  0  Infinite  timeout  (disabled)
c     T10us   = 1 Timeout of 10 us (ideal)
c     T30us  =  2  Timeout  of  30  us  (ideal)
c     T100us  = 3 Timeout of 100 us (ideal)
c     T300us  = 4 Timeout of 300 us (ideal)
c     T1ms    = 5 Timeout of 1 ms (ideal)
c     T3ms    = 6 Timeout of 3 ms (ideal)
c     T10ms  =  7  Timeout  of  10  ms  (ideal)
c     T30ms  =  8  Timeout  of  30  ms  (ideal)
c     T100ms  = 9 Timeout of 100 ms (ideal)
```

```
c     T300ms = 10 Timeout of 300 ms (ideal)
c     T1s    = 11 Timeout of 1 s (ideal)
c     T3s    = 12 Timeout of 3 s (ideal)
c     T10s   = 13 Timeout of 10 s (ideal)
c     T30s   = 14 Timeout of 30 s (ideal)
c     T100s  = 15 Timeout of 100 s (ideal)
c     T300s = 16 Timeout of 300 s (ideal)
c     T1000s = 17 Timeout of 1000 s (maximum)
        integer TNONE,T10us,T30us,T100us,T300us
        integer T1ms,T3ms,T10ms,T30ms,T100ms
        integer T300ms,T1s,T3s,T10s,T30s
        integer T100s,T300s,T1000s

c  Miscellaneous
c     S - specifies sense of PPR
c     LF - ASCII line feed character

        integer  S,LF

c Variables passed in to GPIB function examples
c     cmd    - command buffer
c     rd     - read data buffer
c     wrt    - write data buffer
c     bname  - board name buffer
c     bdname - board or device name buffer
c     flname - file name buffer
c     bd     - board or device number
c     dvm    - device number
c     v      - "value" parameter
c     cnt    - byte count for transfers
c     mask - events to be waited for
c     spr    - serial poll response byte
c     ppr    - parallel poll response byte

        integer       cmd(10),rd(512),wrt(512)
        character*8  bname,bdname
        character*50 flname
        integer      bd,dvm,v,cnt,mask
        integer      spr,ppr

C***** THESE DECLARATIONS ARE NEEDED TO RUN GPIBX *****
C       ---SOME DECLARATIONS...
C               SELECTION:       OPTION SELECTED BY USER
C               ERROR2SELECTION: ERROR MESSAGE FOR INCORRECT SELECTION BY USER
C               PS:              POWER SUPPLY SUBROUTINE
C               DMM:             DIGITAL MULTIMETER SUBROUTINE
C               COUNTER:         COUNTER/TIMER SUBROUTINE
C               SIGGEN:          SIGNAL GENERATOR SUBROUTINE
C               OSCOPE:          OCCILISCOPE  SUBROUTINE
C
        CHARACTER*1 SELECTION
```

```
        CHARACTER*50 ERROR2SELECTION
        ERROR2SELECTION = 'ERROR...INVALID SELECTION, TRY AGAIN.'
C
C
c GPIB Commands: values

        data UNL/63/,UNT/95/,GTL/01/,SDC/04/,PPC/05/
        data GET/08/,TCT/09/, LLO/17/,DCL/20/,PPU/21/
        data SPE/24/,SPD/25/,PPE/96/,PPD/112/

c GPIB status bit vector: values
c To check for error in ibsta - if (ibsta .LT. 0)...
c       data ERR/-32768/

        data TIMO/16384/,END/8192/,SRQI/4096/
        data RQS/2048/,CMPL/256/,LOK/128/,REM/64/,CIC/32/
        data ATN/16/,TACS/8/,LACS/4/,DTAS/2/,DCAS/1/

c Iberr error messages: values

        data       EDVR/0/,ECIC/1/,ENOL/2/,EADR/3/,EARG/4/
        data      ESAC/5/,EABO/6/,ENEB/7/,EOIP/10/,ECAP/11/
        data       EFSO/12/,EBUS/14/,ESTB/15/,ESRQ/16/

c EOS mode bit values

        data BIN/4096/,XEOS/2048/,REOS/1024/

c Timeout values

        data TNONE/0/,T10us/1/,T30us/2/,T100us/3/,T300us/4/
        data T1ms/5/,T3ms/6/,T10ms/7/,T30ms/8/,T100ms/9/
        data T300ms/10/,T1s/11/,T3s/12/T10s/13/,T30s/14/
        data T100s/15/,T300s/16/,T1000s/17/

c Miscellaneous values

        data S/08/,LF/10/
```

APPENDIX B

GPIBX PROGRAM LISTING

This is a listing of the developed system controller program called GPIBX.

```
$INCLUDE: 'DECL.FOR'
C
C       GPIBX.FOR
C
C       IBM-PC/AT GPIB CONTROLLER PROGRAM
C
C       MAIN PROGRAM
C
C***** THIS CALL INITIALIZES THE GPIB BUS *****
        CALL IBINIT (IBSTA)
C
1       CALL CLEAR
2       CALL MAINMENU
        READ (*,10) SELECTION
10      FORMAT (1A1)
        IF (SELECTION .EQ. 'P') THEN
                CALL PS
        ELSEIF (SELECTION .EQ. 'D') THEN
                CALL DMM
        ELSEIF (SELECTION .EQ. 'C') THEN
                CALL COUNTER
        ELSEIF (SELECTION .EQ. 'S') THEN
                CALL SIGGEN
        ELSEIF (SELECTION .EQ. 'O') THEN
                CALL OSCOPE
        ELSEIF (SELECTION .EQ. 'F') THEN
                CALL FUNCMU
        ELSEIF (SELECTION .NE. 'X') THEN
                WRITE (*,20) ERROR2SELECTION
20              FORMAT ('0',1A50)
                GOTO 2

        ELSE
                STOP
        ENDIF
        GOTO 1
        END
C
C
        SUBROUTINE CLEAR
C***** THIS CLEARS THE SCREEN *****
C       ---SOME DECLARATIONS...
C
                CHARACTER*1 C1, C2, C3, C4
```

```
                INTEGER*2 IC(4)
                EQUIVALENCE (C1,IC(1)),(C2,IC(2)),(C3,IC(3)),(C4,IC(4))
                DATA IC/16#1B,16#5B,16#32,16#4A/
C
        WRITE (*,1) C1,C2,C3,C4
1       FORMAT (1X,4A1)
        RETURN
        END
C
C
        SUBROUTINE MAINMENU
C***** THIS PUT THE MAIN MENU ON THE SCREEN *****
C       ---SOME DECLARATIONS...
C       MENUDIS: MENU DISPLAY
        CHARACTER*50 MENUDIS,PS5010,DM5010,DC5009,WAVTEK,TEK,EXIT,FUNC
        MENUDIS = '        *** MAIN MENU ***'
        PS5010 = 'P........POWER SUPPLY'
        DM5010 = 'D........DIGITAL MULTIMETER'
        DC5009 = 'C........COUNTER/TIMER'
        WAVTEK = 'S........SIGNAL GENERATOR'
        TEK =    'O........OSCILLOSCOPE'
        FUNC =  'F........special  FUNCTIONS'
        EXIT =  'X........EXIT  PROGRAM'
C
        WRITE (*,10) MENUDIS
10      FORMAT ('0',1A50)
        WRITE (*,20) PS5010
20      FORMAT ('0',1A50)
        WRITE (*,20) DM5010
        WRITE (*,20) DC5009
        WRITE (*,20) WAVTEK
        WRITE (*,20) TEK
        WRITE (*,20) FUNC
        WRITE (*,20) EXIT
        WRITE (*,30)
30      FORMAT ('0',9X,'ENTER YOUR SELECTION.')
C
        RETURN
        END
C
C
        SUBROUTINE PS
C***** THIS IS THE DRIVER FOR THE POWER SUPPLY *****
C       ---SOME DECLARATIONS...
C       PSSELECT: POWER SUPPLY MENU SELECTION
C
        CHARACTER*1 PSSELECT
C
C
1       CALL CLEAR
2       WRITE (*,10)
```

```
        WRITE (*,20)
        WRITE (*,30)
        WRITE (*,40)
        WRITE (*,50)
        WRITE (*,60)
        WRITE (*,65)
C
10      FORMAT ('0','*** POWER SUPPLY MENU ***')
20      FORMAT ('0','1........SET VOLTAGES')
30      FORMAT ('0','2........SET CURRENT')
40      FORMAT ('0','3........ENABLE OUTPUT')
50      FORMAT ('0','RET......RETURN TO MAIN MENU')
60      FORMAT ('0','X........EXIT PROGRAM')
65      FORMAT ('0',9X,'ENTER YOUR SELECTION.')
C
        READ (*,70) PSSELECT
70      FORMAT (1A1)
        IF (PSSELECT .EQ. '1') THEN
                CALL SETVOLT
        ELSEIF (PSSELECT .EQ. '2') THEN
                CALL SETCURRENT
        ELSEIF (PSSELECT .EQ. '3') THEN
                CALL OUTONOFF
        ELSEIF (PSSELECT .EQ. ' ') THEN
                RETURN
        ELSEIF (PSSELECT .EQ. 'X') THEN
                STOP
        ELSE
                WRITE (*,80)
80              FORMAT ('0',' INVALID INPUT, TRY AGAIN')
                GOTO 2
        ENDIF
        GOTO 1
        END
C
C
        SUBROUTINE DMM
C***** THIS IS THE DRIVER FOR THE DIGITAL MULTIMETER *****
        CHARACTER*1 ACDC(4),ACV(3),DCV(3),DIODE(5),OHMS(4),DMMSEL
        INTEGER WRT(512),DVM
C
        ACV(1)= 'A'
        ACV(2)= 'C'
        ACV(3)= 'V'
C
        ACDC(1)= 'A'
        ACDC(2)= 'C'
        ACDC(3)= 'D'
        ACDC(4)= 'C'
C
        DCV(1)= 'D'
```

```
        DCV(2)= 'C'
        DCV(3)= 'V'
C
        DIODE(1)= 'D'
        DIODE(2)= 'I'
        DIODE(3)= 'O'
        DIODE(4)= 'D'
        DIODE(5)= 'E'
C
        OHMS(1)= 'O'
        OHMS(2)= 'H'
        OHMS(3)= 'M'
        OHMS(4)= 'S'
C
1       CALL CLEAR
2       WRITE (*,10)
        WRITE (*,20)
        WRITE (*,30)
        WRITE (*,40)
        WRITE (*,50)
        WRITE (*,60)
        WRITE (*,70)
        WRITE (*,80)
        WRITE (*,90)
C
10      FORMAT ('0','*** DIGITAL MULTIMETER MENU ***')
20      FORMAT ('0','1........DC VOLTS')
30      FORMAT ('0','2........OHMS')
40      FORMAT ('0','3........AC VOLTS rms')
50      FORMAT ('0','4........AC+DC VOLTS rms')
60      FORMAT ('0','5........DIODE TEST')
70      FORMAT ('0','RET......RETURN TO MAIN MENU')
80      FORMAT ('0','X........EXIT')
90      FORMAT ('0','ENTER YOUR SELECTION')
C
        READ (*,100) DMMSEL
100     FORMAT (1A1)
C
        DVM= IBFIND ('DMM ')
        IF (DMMSEL .EQ. '1') THEN
                CALL STRING (DCV,3,WRT)
                CALL IBWRT (DVM,WRT,3)
        ELSEIF (DMMSEL .EQ. '2') THEN
                CALL STRING (OHMS,4,WRT)
                CALL IBWRT (DVM,WRT,4)
        ELSEIF (DMMSEL .EQ. '3') THEN
                CALL STRING (ACV,3,WRT)
                CALL IBWRT (DVM,WRT,3)
        ELSEIF (DMMSEL .EQ. '4') THEN
                CALL STRING (ACDC,4,WRT)
                CALL IBWRT (DVM,WRT,4)
```

```
        ELSEIF (DMMSEL .EQ. '5') THEN
                CALL STRING (DIODE,5,WRT)
                CALL IBWRT (DVM,WRT,5)
        ELSEIF (DMMSEL .EQ. ' ') THEN
                RETURN
        ELSEIF (DMMSEL .EQ. 'X') THEN
                STOP
        ELSE
                WRITE (*,110)
110             FORMAT ('0','INVALID SELECTION, TRY AGAIN')
                GOTO 2
        ENDIF
        GOTO 1
        END
C
C
        SUBROUTINE COUNTER
C***** THIS IS THE UNDEVELOPED DRIVER FOR THE COUNTER/TIMER *****
C       ---SOME DECLARATIONS...
99      RETURN
        END
C
C
        SUBROUTINE SIGGEN
C***** THIS IS THE SIGNAL GENERATOR DRIVER *****
C       ---SOME DECLARATIONS...
C       SIGSEL: SIGGEN MENU SELECTION
C
        CHARACTER*1 SIGSEL
C
1       CALL CLEAR
2       WRITE (*,10)
        WRITE (*,20)
        WRITE (*,30)
        WRITE (*,40)
        WRITE (*,50)
        WRITE (*,60)
        WRITE (*,65)
        WRITE (*,70)
        WRITE (*,80)
        WRITE (*,90)
C
10      FORMAT ('0','*** SIGNAL GENERATOR MENU ***')
20      FORMAT ('0','1........FREQUENCY')
30      FORMAT ('0','2........AMPLITUDE')
40      FORMAT ('0','3........FUNCTION')
50      FORMAT ('0','4........OFFSET')
60      FORMAT ('0','5........OUTPUT ENABLE')
65      FORMAT ('0','6........SWEEP FREQUENCIES')
70      FORMAT ('0','RET......RETURN TO MAIN MENU')
80      FORMAT ('0','X........EXIT')
```

```
90      FORMAT ('0',9X,'ENTER YOUR SELECTION')
C
        READ (*,100) SIGSEL
100     FORMAT (1A1)
C
        IF (SIGSEL .EQ. '1') THEN
                CALL FREQ
        ELSEIF (SIGSEL .EQ. '2') THEN
                CALL AMP
        ELSEIF (SIGSEL .EQ. '3') THEN
                CALL FUNC
        ELSEIF (SIGSEL .EQ. '4') THEN
                CALL OFFSET
        ELSEIF (SIGSEL .EQ. '5') THEN
                CALL SIGOUT
        ELSEIF (SIGSEL .EQ. '6') THEN
                CALL SWEEP
        ELSEIF (SIGSEL .EQ. ' ') THEN
                RETURN
        ELSEIF (SIGSEL .EQ. 'X') THEN
                STOP
        ELSE
                WRITE (*,110)
110             FORMAT ('0','INVALID RESPONSE, TRY AGAIN.')
                GOTO 2
        ENDIF
        GOTO 1
        END
C
C
        SUBROUTINE FREQ
C***** THIS MAKES THE SIGGEN OUTPUT A SPECIFIED FREQUENCY *****
        INTEGER DVM,I,WRT(512)
        CHARACTER*1 FREQ(13),INPUT(11)
C
        FREQ(1)= 'F'
        FREQ(13)= 'I'
C
        DVM= IBFIND ('SIGGEN ')
        WRITE (*,10)
10      FORMAT ('0',9X,'ENTER DESIRED FREQUENCY AS XXX.XEX (.01Hz-12MHz)')
C
        READ (*,20) INPUT
20      FORMAT (11A1)
C
        DO 30 I= 1,11
                FREQ(I+1)= INPUT(I)
30      CONTINUE
C
        CALL STRING (FREQ,13,WRT)
        CALL IBWRT (DVM,WRT,13)
```

```
        RETURN
        END
C
C
        SUBROUTINE AMP
C***** THIS MAKES THE SIGGEN OUTPUT A SPECIFIED AMPLITUDE *****
        INTEGER DVM,I,WRT(512)
        CHARACTER*1 AMP(13),INPUT(11)
C
        AMP(1)= 'A'
        AMP(13)= 'I'
C
        DVM= IBFIND ('SIGGEN ')
        WRITE (*,10)
10      FORMAT ('0',9X,'ENTER DESIRED AMPLITUDE AS XX.XEX (FREE FORMAT)')
        WRITE (*,15)
15      FORMAT ('0',9X,'SIGNAL AMPLITUDE IS Vpp FROM 10mV TO 10.0V')
C
        READ (*,20) INPUT
20      FORMAT (11A1)
C
        DO 30 I= 1,11
                AMP(I+1)= INPUT(I)
30      CONTINUE
C
        CALL STRING (AMP,13,WRT)
        CALL IBWRT (DVM,WRT,13)
        RETURN
        END
C
C
        SUBROUTINE FUNC
C***** THIS MAKES THE SIGGEN OUTPUT A SPECIFIED WAVEFORM *****
        INTEGER DVM,I,WRT(512)
        CHARACTER*1 INPUT,C0(3),C1(3),C2(3),C3(3),C4(3)
C
        C0(1)= 'C'
        C0(2)= '0'
        C0(3)= 'I'
C
        C1(1)= 'C'
        C1(2)= '1'
        C1(3)= 'I'
C
        C2(1)= 'C'
        C2(2)= '2'
        C2(3)= 'I'
C
        C3(1)= 'C'
        C3(2)= '3'
        C3(3)= 'I'
```

```
C
      C4(1)= 'C'
      C4(2)= '4'
      C4(3)= 'I'
C
      DVM= IBFIND ('SIGGEN ')
C
1     CALL CLEAR
2     WRITE (*,10)
      WRITE (*,20)
      WRITE (*,30)
      WRITE (*,40)
      WRITE (*,50)
      WRITE (*,60)
      WRITE (*,70)
      WRITE (*,80)
10    FORMAT ('0','*** SIGNAL GENERATOR FUNCTION MENU ***')
20    FORMAT ('0','1........SINE WAVE')
30    FORMAT ('0','2........TRIANGLE WAVE')
40    FORMAT ('0','3........SQUARE WAVE in phase with sync output')
50    FORMAT ('0','4........SQUARE WAVE out of phase with sync output')
60    FORMAT ('0','5........DC OUTPUT VOLTAGE')
70    FORMAT ('0','RET......RETURN TO SIGNAL GENERATOR MAIN MENU')
80    FORMAT ('0','X........EXIT PROGRAM')
C
      READ (*,90) INPUT
90    FORMAT (1A1)
C
      IF (INPUT .EQ. '1') THEN
              CALL STRING (C0,3,WRT)
              CALL IBWRT (DVM,WRT,3)
      ELSEIF (INPUT .EQ. '2') THEN
              CALL STRING (C1,3,WRT)
              CALL IBWRT (DVM,WRT,3)
      ELSEIF (INPUT .EQ. '3') THEN
              CALL STRING (C2,3,WRT)
              CALL IBWRT (DVM,WRT,3)
      ELSEIF (INPUT .EQ. '4') THEN
              CALL STRING (C3,3,WRT)
              CALL IBWRT (DVM,WRT,3)
      ELSEIF (INPUT .EQ. '5') THEN
              CALL STRING (C4,3,WRT)
              CALL IBWRT (DVM,WRT,3)
      ELSEIF (INPUT .EQ. ' ') THEN
              RETURN
      ELSEIF (INPUT .EQ. 'X') THEN
              STOP
      ELSE
              WRITE (*,100)
100           FORMAT ('0',9X,'INVALID ENTRY, TRY AGAIN')
              GOTO 2
```

```
        ENDIF
        GOTO 1
        END
C
C
        SUBROUTINE OFFSET
C***** THIS ENTERS A DC OFFSET FOR THE SIGGEN OUTPUT *****
        INTEGER DVM,I,WRT(512)
        CHARACTER*1 D(6),INPUT(4)
C
        D(1)= 'D'
        D(6)= 'I'
C
        DVM= IBFIND ('SIGGEN ')
        WRITE (*,10)
10      FORMAT ('0',9X,'ENTER DESIRED OFFSET AS -5.00 TO +5.00 VOLTS')
C
        READ (*,20) INPUT
20      FORMAT (4A1)
C
        DO 30 I= 1,4
                D(I+1)= INPUT(I)
30      CONTINUE
C
        CALL STRING (D,6,WRT)
        CALL IBWRT (DVM,WRT,6)
        RETURN
        END
C
C
        SUBROUTINE SIGOUT
C***** THIS TOGGLES THE SIGGEN OUTPUT *****
        CHARACTER*1 P0(3),P1(3),INPUT
        INTEGER WRT(512),DVM
C
        DVM = IBFIND ('SIGGEN ')
C
        P0(1)= 'P'
        P0(2)= '0'
        P0(3)= 'I'
C
        P1(1)= 'P'
        P1(2)= '1'
        P1(3)= 'I'
C
1       WRITE (*,10)
10      FORMAT ('0',9X,'SIGNAL GENERATOR OUTPUT ON? ENTER Y OR N')
C
        READ (*,20) INPUT
20      FORMAT (1A1)
        IF (INPUT .EQ. 'Y') THEN
```

```
                CALL STRING (P1,3,WRT)
                CALL IBWRT (DVM,WRT,3)
        ELSEIF (INPUT .EQ. 'N') THEN
                CALL STRING (P0,3,WRT)
                CALL IBWRT (DVM,WRT,3)
        ELSE
                WRITE (*,30)
30              FORMAT ('0','INVALID RESPONSE, TRY AGAIN')
                GOTO 1
        ENDIF
        RETURN
        END
C
C
        SUBROUTINE SWEEP
C***** THIS SWEEPS THRU A RANGE OF FREQUENCIES *****
        CHARACTER*8 INPUTA,INPUTB
        CHARACTER*1 FREQ(10),TEMP(8),INPUTC
        INTEGER I,J,K,L,DVM,STEPS,WRT(512),TEMPA
        REAL STARTF,STEPF,VALUE
C
        FREQ(1) = 'F'
        FREQ(10) = 'I'
C
        DVM = IBFIND ('SIGGEN ')
C
        CALL CLEAR
        WRITE (*,10)
10      FORMAT ('0','SWEEP FREQUENCIES FUNCTION')
        WRITE (*,20)
20      FORMAT ('0','ENTER STARTING FREQUENCY AS XXX.XEX (.01Hz-12MHz)')
        READ (*,30) INPUTA
30      FORMAT (1A8)
        WRITE (*,40)
40      FORMAT ('0','ENTER FREQUENCY STEP SIZE AS XXX.XEX (.01Hz-12MHz)')
        READ (*,30) INPUTB
        WRITE (*,50)
50      FORMAT ('0','ENTER NUMBER OF STEPS 1-99')
        READ (*,60) STEPS
60      FORMAT (1I2)
C
        READ (INPUTA,65) STARTF
65      FORMAT (BNF8.0)
        READ (INPUTB,65) STEPF
C
        READ (INPUTA,80) TEMP
80      FORMAT (8A1)
C
        DO 70 I=1,STEPS
                DO 90 J=1,8
                        FREQ(J+1)=TEMP(J)
```

```
90              CONTINUE
                CALL STRING (FREQ,10,WRT)
                CALL IBWRT (DVM,WRT,10)
                WRITE (*,95) STARTF
95              FORMAT ('0','FREQUENCY IS NOW ',F10.2,' Hz')
                WRITE (*,100) I
100             FORMAT ('0','STEP #',I2,
     +      ' COMPLETE. ENTER RET TO CONTINUE OR ANY OTHER KEY TO END')
                READ (*,110) INPUTC
110             FORMAT (1A1)
                IF (INPUTC .NE. ' ') THEN
                        RETURN
                ELSE
                        STARTF = STARTF + STEPF
                        VALUE = STARTF
                        DO 120 K=1,8
                                TEMPA = INT(STARTF/(10**(8-K)))
                                TEMP(K) = CHAR (TEMPA + 48)
                                STARTF = STARTF - (TEMPA*10**(8-K))
120                     CONTINUE
                        STARTF = VALUE
                ENDIF
70      CONTINUE
        RETURN
        END
C
C
        SUBROUTINE OSCOPE
C***** THIS IS THE OSCOPE DRIVER *****
        DVM = IBFIND ('OSCOPE ')
C
1       CALL CLEAR
2       WRITE (*,10)
        WRITE (*,20)
        WRITE (*,30)
        WRITE (*,40)
        WRITE (*,50)
        WRITE (*,60)
        WRITE (*,70)
C
10      FORMAT ('0','*** OSCOPE MENU ***')
20      FORMAT ('0','1........RECORD WAVEFORM DATA')
30      FORMAT ('0','2........GO TO LOCAL')
40      FORMAT ('0','3........REMOTE ENABLE')
50      FORMAT ('0','RET........RETURN TO MAIN MENU')
60      FORMAT ('0','X........EXIT')
70      FORMAT ('0','ENTER YOUR SELECTION')
C
        READ (*,100) SELECTION
100     FORMAT (1A1)
C
```

```
        IF (SELECTION .EQ. '1') THEN
                CALL RECORD
        ELSEIF (SELECTION .EQ. '2') THEN
                CALL IBLOC (DVM)
        ELSEIF (SELECTION .EQ. '3') THEN
                DVM = IBFIND ('OSCOPE ')
        ELSEIF (SELECTION .EQ. ' ') THEN
                RETURN
        ELSEIF (SELECTION .EQ. 'X') THEN
                STOP
        ELSE
                WRITE (*,110)
110             FORMAT ('0','INVALID SELECTION, TRY AGAIN')
                GOTO 2
        ENDIF
        GOTO 1
        END
C
C
        SUBROUTINE RECORD
C***** THIS IS THE WAVEFORM RECORDING DRIVER *****
        CHARACTER*1 SEL, ACCESM(4), ASCII(2), RECORM(13), RELALM(7)
        CHARACTER*15 REC, ACCESS, RELALL,FILNAM
        INTEGER WRT(512),DVM
        REAL TIMDIV,VOLDIV
C
        ASCII(1) = 'A'
        ASCII(2) = 'S'
        REC = 'SEL SAVE.R;C?'
        RELALL = 'REL ALL'
C
        DVM = IBFIND ('OSCOPE ')
1       CALL CLEAR
C
        WRITE (*,800)
800     FORMAT ('0','THIS PROGRAM CREATES A DATA FILE THAT CAN BE')
        WRITE (*,810)
810     FORMAT (' ','BE USED WITH THE SLIDE WRITE PLUS PROGRAM TO')
        WRITE (*,820)
820     FORMAT (' ','PLOT THE WAVEFORMS ON THE OSCOPE. ADJUST THE')
        WRITE (*,830)
830     FORMAT (' ','SCOPE FOR A PROPERLY TRIGGERED TRACE, NO JITTER')
        WRITE (*,840)
840     FORMAT (' ','USE THE HORIZ CONTROL TO LEFT JUSTIFY THE TRACES')
C
2       WRITE (*,10)
10      FORMAT ('0','SELECT WAVEFORM DESIRED')
        WRITE (*,20)
20      FORMAT ('0',' 1.....RIGHT COMPARTMENT CHANNEL 1')
        WRITE (*,30)
30      FORMAT (' ',' 2.....RIGHT COMPARTMENT CHANNEL 2')
```

```
        WRITE (*,40)
40      FORMAT ('0','ENTER YOUR SELECTION 1 OR 2')
        READ (*,50) SEL
50      FORMAT (1A1)
        IF (SEL .EQ. '1') THEN
                ACCESS = 'A R1'
        ELSEIF (SEL .EQ. '2') THEN
                ACCESS = 'A R2'
        ELSE
                WRITE (*,60)
60              FORMAT ('0','INVALID SELECTION, TRY AGAIN')
                GOTO 2
        ENDIF
C
        READ (ACCESS,70) ACCESM
70      FORMAT (4A1)
C
        CALL STRING (ACCESM,4,WRT)
        CALL IBWRT (DVM,WRT,4)
C
        CALL STRING (ASCII,2,WRT)
        CALL IBWRT (DVM,WRT,2)
C
C
        READ (REC,80) RECORM
80      FORMAT (13A1)
        CALL STRING (RECORM,13,WRT)
        CALL IBWRT (DVM,WRT,13)
C
        CALL IBRDF (DVM,'SCOPE ')
C
        WRITE (*,300)
300     FORMAT ('0','ENTER TIME/DIV AS X.XXEX')
        READ (*,310) TIMDIV
310     FORMAT (1BNF10.7)
        TIMDIV = TIMDIV * 10.0
        WRITE (*,320)
320     FORMAT ('0','ENTER VOLTS/DIV AS X.XXEX')
        READ (*,330) VOLDIV
330     FORMAT (1BNE12.6)
C
        CALL SCALER (TIMDIV,VOLDIV)
C
        READ (RELALL,110) RELALM
110     FORMAT (7A1)
        CALL STRING (RELALM,7,WRT)
        CALL IBWRT (DVM,WRT,7)
        CALL IBLOC (DVM)
        WRITE (*,900)
900     FORMAT ('0','YOU CAN NOW USE SLIDE WRITE PLUS TO PLOT YOUR')
        WRITE (*,910)
```

```
910     FORMAT (' ','DATA FILES. HIT ENTER TO CONTINUE')
        READ (*,920) SEL
920     FORMAT (1A1)
        RETURN
        END
C
C
        SUBROUTINE SCALER (TIMDIV, VOLDIV)
C***** THIS SCALES THE DATA RECEIVED FROM THE OSCOPE *****
        CHARACTER*6 CURVE
        INTEGER I,J,K,MINUS,L,COUNT,END,NOPTS
        CHARACTER*1 OSDATA(5000),COMMA
        REAL YDATA(1016),YTEMP(3),TIMDIV,VOLDIV,XDATA,OFFSET
        CHARACTER*12 FILNAM
C
        COMMA=','
C
        NOPTS = 508
        IF (TIMDIV .LT. 0.000075) NOPTS = 1016
C
        OPEN (1,FILE='SCOPE ',STATUS='OLD',FORM='BINARY')
C
        READ (1) CURVE
C
        I = 1
1       READ (1,ERR=30,END=40) OSDATA(I)
        I = I+1
        GOTO 1
30      WRITE (*,50)
50      FORMAT (' ',' ERROR IN READING FILE')
40      CLOSE (1)
C
        COUNT = I
        END = COUNT-3
C
        I = 1
        K = 1
        J = 1
        MINUS = 0
100     IF (OSDATA(I) .EQ. '-') THEN
                MINUS = 1
                I = I+1
        ENDIF
        YTEMP(K) = FLOAT(ICHAR (OSDATA(I)))
        YTEMP(K) = YTEMP(K) - 48.0
        I = I+1
        K = K+1
        IF (K .GT. 7) STOP
        IF (OSDATA(I) .NE. ',') THEN
                GOTO 100
        ELSE
```

```
                GOTO 118
        ENDIF
118     K = K-1
        IF (K .EQ. 3) THEN
                YDATA(J) = (100.0*YTEMP(K-2))+(10.0*YTEMP(K-1))+YTEMP(K)
        ELSEIF (K .EQ. 2) THEN
                YDATA(J) = (10.0*YTEMP(K-1))+YTEMP(K)
        ELSEIF (K .EQ. 1) THEN
                YDATA(J) = YTEMP(K)
        ENDIF
        IF (MINUS .EQ. 1) YDATA(J) = 0.0-YDATA(J)
        YDATA(J) = ((YDATA(J)+15.0)/100.)*VOLDIV
        J = J+1
        I = I+1
        MINUS = 0
        IF (J .EQ. NOPTS) GOTO 120
        K = 1
        GOTO 100
C
120     WRITE (*,400)
400     FORMAT ('0','ENTER DOS FILE NAME TO STORE WAVEFORM DATA IN')
        READ (*,410) FILNAM
410     FORMAT (1A12)
        OPEN(2,FILE=FILNAM,STATUS='NEW',FORM='FORMATTED')
        I=1
        K=1
        XDATA = 0.0
        IF (NOPTS .EQ. 508) THEN
420             WRITE (2,220) XDATA,COMMA,YDATA(I)
220             FORMAT (E10.5,A1,F10.3)
                XDATA = (TIMDIV*I)/508.0
                I=I+1
                K=K+1
                IF (K .EQ. 5) I=I+1
                IF (K .EQ. 5) K=1
                IF (I .EQ. 501) GOTO 300
                GOTO 420
        ELSE
                OFFSET = TIMDIV/1016
                DO 500 I=1,1016,3
                        XDATA = OFFSET * (I-1)
                        WRITE (2,220) XDATA,COMMA,YDATA(I)
500             CONTINUE
        ENDIF
300     CLOSE(2)
        RETURN
        END
C
C
        SUBROUTINE SETVOLT
C***** THIS SETS THE POWER SUPPLY OUTPUT VOLTAGE *****
```

```
        CHARACTER*1 SVOPT
C
1       CALL CLEAR
2       WRITE (*,10)
        WRITE (*,20)
        WRITE (*,30)
        WRITE (*,40)
        WRITE (*,50)
        WRITE (*,60)
        WRITE (*,65)
10      FORMAT ('0','*** VOLTAGE SETTING MENU ***')
20      FORMAT ('0','1........VPOS')
30      FORMAT ('0','2........VNEG')
40      FORMAT ('0','3........VLOGIC')
50      FORMAT ('0','RET......RETURN TO POWER SUPPLY MENU')
60      FORMAT ('0','X........EXIT PROGRAM')
65      FORMAT ('0',9X,'ENTER YOUR SELECTION.')
C
        READ (*,70) SVOPT
70      FORMAT (1A1)
        IF (SVOPT .EQ. '1') THEN
                CALL VPOS
        ELSEIF (SVOPT .EQ. '2') THEN
                CALL VNEG
        ELSEIF (SVOPT .EQ. '3') THEN
                CALL VLOG
        ELSEIF (SVOPT .EQ. ' ') THEN
                RETURN
        ELSEIF (SVOPT .EQ. 'X') THEN
                STOP
        ELSE
                WRITE (*,80)
80              FORMAT ('0',' INVALID INPUT, TRY AGAIN')
                GOTO 2
        ENDIF
        GOTO 1
        END
C
C
        SUBROUTINE SETCURRENT
C***** THIS SETS THE POWER SUPPLY CURRENT LIMITS *****
        CHARACTER*1 SCOPT
C
1       CALL CLEAR
2       WRITE (*,10)
        WRITE (*,20)
        WRITE (*,30)
        WRITE (*,40)
        WRITE (*,50)
        WRITE (*,60)
        WRITE (*,65)
```

```
C
10      FORMAT ('0','*** CURRENT SETTING MENU ***')
20      FORMAT ('0','1........IPOS')
30      FORMAT ('0','2........INEG')
40      FORMAT ('0','3........ILOGIC')
50      FORMAT ('0','RET......RETURN TO POWER SUPPLY MENU')
60      FORMAT ('0','X........EXIT PROGRAM')
65      FORMAT ('0',9X,'ENTER YOUR SELECTION.')
C
        READ (*,70) SCOPT
70      FORMAT (1A1)
        IF (SCOPT .EQ. '1') THEN
                CALL IPOS
        ELSEIF (SCOPT .EQ. '2') THEN
                CALL INEG
        ELSEIF (SCOPT .EQ. '3') THEN
                CALL ILOG
        ELSEIF (SCOPT .EQ. ' ') THEN
                RETURN
        ELSEIF (SCOPT .EQ. 'X') THEN
                STOP
        ELSE
                WRITE (*,80)
80              FORMAT ('0',' INVALID INPUT, TRY AGAIN')
                GOTO 2
        ENDIF
        GOTO 1
        END
C
C
        SUBROUTINE OUTONOFF
C***** THIS TOGGLES THE POWER SUPPLY OUTPUTS *****
        CHARACTER*1 ANSWER, MSGYES(6), MSGNO(7)
        INTEGER I,J,K,WRT(512),DVM
C
        MSGYES(1) = 'O'
        MSGYES(2) = 'U'
        MSGYES(3) = 'T'
        MSGYES(4) = ' '
        MSGYES(5) = 'O'
        MSGYES(6) = 'N'
C
        MSGNO(1) = 'O'
        MSGNO(2) = 'U'
        MSGNO(3) = 'T'
        MSGNO(4) = ' '
        MSGNO(5) = 'O'
        MSGNO(6) = 'F'
        MSGNO(7) = 'F'
C
        DVM = IBFIND ('PS ')
```

```
C
1       WRITE (*,10)
10      FORMAT ('0','POWER SUPPLY OUTPUTS ON? ENTER Y OR N.')
        READ (*,20) ANSWER
20      FORMAT (1A1)
        IF (ANSWER .EQ. 'Y') THEN
                CALL STRING (MSGYES,6,WRT)
                CALL IBWRT (DVM,WRT,6)
        ELSEIF (ANSWER .EQ. 'N') THEN
                CALL STRING (MSGNO,7,WRT)
                CALL IBWRT (DVM,WRT,7)
        ELSE
                WRITE (*,30)
30              FORMAT ('0','INCORRECT INPUT, TRY AGAIN')
                GOTO 1
        ENDIF
        RETURN
        END
C
C
        SUBROUTINE VPOS
C***** THIS MAKES THE POWER SUPPLY OUTPUT A SPECIFIED POSITIVE VOLTAGE
        CHARACTER*1 VPOS(9)
        CHARACTER*1 INPUT(4)
        INTEGER I,J,K,WRT(512),DVM
C
        VPOS(1) = 'V'
        VPOS(2) = 'P'
        VPOS(3) = 'O'
        VPOS(4) = 'S'
        VPOS(5) = ' '
C
        WRITE (*,10)
10      FORMAT ('0',' ENTER DESIRED POSITIVE VOLTAGE AS',
     +' X.XX (0-32.0Vdc)')
        READ (*,20) INPUT
20      FORMAT (4A1)
        DO 30 I= 6,9
                K= I-5
                VPOS(I) = INPUT(K)
30      CONTINUE
        CALL STRING (VPOS,9,WRT)
        DVM = IBFIND ('PS ')
        CALL IBWRT (DVM,WRT,9)
        RETURN
        END
C
C
        SUBROUTINE VNEG
C***** THIS MAKES THE PS OUTPUT A SPECIFIED NEG VOLTAGE *****
        CHARACTER*1 VNEG(9)
```

```
      CHARACTER*1 INPUT(4)
      INTEGER I,J,K,WRT(512),DVM
C
      VNEG(1) = 'V'
      VNEG(2) = 'N'
      VNEG(3) = 'E'
      VNEG(4) = 'G'
      VNEG(5) = ' '
C
      WRITE (*,10)
10    FORMAT ('0',' ENTER DESIRED NEGATIVE VOLTAGE AS',
     +' X.XX (0-32.0Vdc')
      READ (*,20) INPUT
20    FORMAT (4A1)
      DO 30 I= 6,9
            K= I-5
            VNEG(I) = INPUT(K)
30    CONTINUE
      CALL STRING (VNEG,9,WRT)
      DVM = IBFIND ('PS ')
      CALL IBWRT (DVM,WRT,9)
      RETURN
      END
C
C
      SUBROUTINE VLOG
C***** THIS MAKES THE PS OUTPUT A SPECFIED LOGIC VOLTAGE *****
      CHARACTER*1 VLOG(9)
      CHARACTER*1 INPUT(4)
      INTEGER I,J,K,WRT(512),DVM
C
      VLOG(1) = 'V'
      VLOG(2) = 'L'
      VLOG(3) = 'O'
      VLOG(4) = 'G'
      VLOG(5) = ' '
C
      WRITE (*,10)
10    FORMAT ('0',' ENTER DESIRED LOGIC VOLTAGE AS',
     +' X.XX (-4.50 TO 5.50 Vdc')
      READ (*,20) INPUT
20    FORMAT (4A1)
      DO 30 I= 6,9
            K= I-5
            VLOG(I) = INPUT(K)
30    CONTINUE
      CALL STRING (VLOG,9,WRT)
      DVM = IBFIND ('PS ')
      CALL IBWRT (DVM,WRT,9)
      RETURN
      END
```

```
C
C
        SUBROUTINE IPOS
C***** THIS SET THE PS POS CURRENT MAXIMUM *****
        CHARACTER*1 IPOS(9)
        CHARACTER*1 INPUT(4)
        INTEGER I,J,K,WRT(512),DVM
C
        IPOS(1) = 'I'
        IPOS(2) = 'P'
        IPOS(3) = 'O'
        IPOS(4) = 'S'
        IPOS(5) = ' '
C
        WRITE (*,10)
10      FORMAT ('0',' ENTER DESIRED POSITIVE CURRENT MAXIMUM',
     +' AS X.XX (1.60Amax')
        READ (*,20) INPUT
20      FORMAT (4A1)
        DO 30 I= 6,9
                K= I-5
                IPOS(I) = INPUT(K)
30      CONTINUE
        CALL STRING (IPOS,9,WRT)
        DVM = IBFIND ('PS ')
        CALL IBWRT (DVM,WRT,9)
        RETURN
        END
C
C
        SUBROUTINE INEG
C***** THIS SETS THE PS NEG CURRENT MAXIMUM *****
        CHARACTER*1 INEG(9)
        CHARACTER*1 INPUT(4)
        INTEGER I,J,K,WRT(512),DVM
C
        INEG(1) = 'I'
        INEG(2) = 'N'
        INEG(3) = 'E'
        INEG(4) = 'G'
        INEG(5) = ' '
C
        WRITE (*,10)
10      FORMAT ('0',' ENTER DESIRED NEGATIVE CURRENT MAXIMUM',
     +' AS X.XX (1.60Amax)')
        READ (*,20) INPUT
20      FORMAT (4A1)
        DO 30 I= 6,9
                K= I-5
                INEG(I) = INPUT(K)
30      CONTINUE
```

```
        CALL STRING (INEG,9,WRT)
        DVM = IBFIND ('PS ')
        CALL IBWRT (DVM,WRT,9)
        RETURN
        END
C
C
        SUBROUTINE ILOG
C***** THIS SETS THE PS LOGIC CURRENT MAXIMUM *****
        CHARACTER*1 ILOG(9)
        CHARACTER*1 INPUT(4)
        INTEGER I,J,K,WRT(512),DVM
C
        ILOG(1) = 'I'
        ILOG(2) = 'L'
        ILOG(3) = 'O'
        ILOG(4) = 'G'
        ILOG(5) = ' '
C
        WRITE (*,10)
10      FORMAT ('0',' ENTER DESIRED LOGIC CURRENT',
     +' MAXIMUM AS X.XX (1.60Amax)')
        READ (*,20) INPUT
20      FORMAT (4A1)
        DO 30 I= 6,9
                K= I-5
                ILOG(I) = INPUT(K)
30      CONTINUE
        CALL STRING (ILOG,9,WRT)
        DVM = IBFIND ('PS ')
        CALL IBWRT (DVM,WRT,9)
        RETURN
        END
C
C
        SUBROUTINE STRING (INPUT,LENGTH,WRT)
C***** THIS CONVERTS CHARACTER STRINGS INTO REQUIRED FORM FOR IBWRT **
        CHARACTER*1 INPUT(30)
        INTEGER LENGTH,I,J,K,WRT(512)
        J= 1
        DO 10 I=1,LENGTH,2
                K= I+1
                WRT(J)= ICHAR(INPUT(I)) + (ICHAR(INPUT(K))*256)
                J= J+1
10      CONTINUE
        RETURN
        END
C
C
        SUBROUTINE FINDER
C***** THIS HELPS TO FIND GPIB ERRORS *****
```

```
        COMMON /IBGLOB/ IBSTA, IBERR, IBCNT
        WRITE (*,10)
10      FORMAT (' FIND ERROR')
        RETURN
        END
C
C
        SUBROUTINE ERROR
C***** THIS WRITES THE STATUS, ERROR CODE, AND BYTE COUNT *****
        COMMON /IBGLOB/ IBSTA, IBERR, IBCNT
        WRITE (*,10) IBSTA, IBERR, IBCNT
10      FORMAT (' ERROR',I6,I6,I6)
        RETURN
        END
C
C
        SUBROUTINE CHKSTATUS
C***** THIS SUBROUTINE CHECKS IBSTA AND WRITES IT TO THE SCREEN. IF AN
C*****  ERRROR IS FOUND IBSTA, IBERR, AND IBCNT ARE WRITTEN TO THE
C*****  SCREEN.
C
        WRITE (*,10) IBSTA
10      FORMAT ('0','IBSTA IS ',I6)
        IF (IBSTA .LT. 0) CALL ERROR
        RETURN
        END
C
C
        SUBROUTINE FUNCMU
C***** THIS IS THE SPECIAL FUNCTION MENU DRIVER ******
C       ---SOME DECLARATIONS...
C
        CHARACTER*1 SELECT
C
C
1       CALL CLEAR
2       WRITE (*,10)
        WRITE (*,20)
C       WRITE (*,30)
C       WRITE (*,40)
        WRITE (*,50)
        WRITE (*,60)
        WRITE (*,65)
C
10      FORMAT ('0','*** SPECIAL FUNCTION MENU ***')
20      FORMAT ('0','1........BODE PLOT')
30      FORMAT ('0','2........BLANK')
40      FORMAT ('0','3........BLANK')
50      FORMAT ('0','RET......RETURN TO MAIN MENU')
60      FORMAT ('0','X........EXIT PROGRAM')
65      FORMAT ('0',9X,'ENTER YOUR SELECTION.')
```

```
C
      READ (*,70) SELECT
70    FORMAT (1A1)
      IF (SELECT .EQ. '1') THEN
            CALL BODE
C     ELSEIF (SELECT .EQ. '2') THEN
C           CALL BLANK
C     ELSEIF (SELECT .EQ. '3') THEN
C           CALL BLANK
      ELSEIF (SELECT .EQ. ' ') THEN
            RETURN
      ELSEIF (SELECT .EQ. 'X') THEN
            STOP
      ELSE
            WRITE (*,80)
80          FORMAT ('0',' INVALID INPUT, TRY AGAIN')
            GOTO 2
      ENDIF
      GOTO 2
      END
C
C
      SUBROUTINE BODE
C***** THIS IS THE BODE PLOT DRIVER *****
      COMMON /IBGLOB/ IBSTA, IBERR, IBCNT
      CHARACTER*12 RDDAT(400)
      INTEGER DVM,COUNT(400),LENGTH
      CHARACTER*16 FMT
      CHARACTER*14 FLNAME
C
      CHARACTER*8 INPUTA,INPUTB
      CHARACTER*1 FREQ(10),TEMP(8),INPUTC,ACV(3)
      INTEGER I,J,K,L,STEPS,WRT(512),TEMPA
      REAL STARTF,STEPF,VALUE,Y(400),LASTF,VOLTIN,VIN,DB(400),VOUT
C
      I=1
C
      FREQ(1) = 'F'
      FREQ(10) = 'I'
C
C
      CALL CLEAR
      WRITE (*,10)
10    FORMAT ('0','BODE PLOT DATA GENERATOR')
      WRITE (*,20)
20    FORMAT ('0','ENTER STARTING FREQUENCY AS XXX.XEX (.01Hz-12MHz)')
      READ (*,30) INPUTA
30    FORMAT (1A8)
      WRITE (*,40)
40    FORMAT ('0','ENTER STOPPING FREQUENCY AS XXX.XEX (.01Hz-12MHz)')
      READ (*,30) INPUTB
```

```
        WRITE (*,50)
50      FORMAT ('0','ENTER NUMBER OF DATA POINTS TO TAKE (1-400)')
        READ (*,51) STEPS
51      FORMAT (1I3)
        WRITE (*,52)
52      FORMAT ('0','ENTER PEAK AMPLITUDE OF INPUT SIGNAL AS X.XX')
        READ (*,53) VOLTIN
53      FORMAT (1F10.3)
        VIN = .707 * VOLTIN
C
        WRITE (*,55)
55      FORMAT ('0','EXECUTING...')
        READ (INPUTA,65) STARTF
65      FORMAT (BNF8.0)
        READ (INPUTB,65) LASTF
        STEPF = (LASTF-STARTF)/STEPS
C
        READ (INPUTA,80) TEMP
80      FORMAT (8A1)
C
        DVM = IBFIND ('DMM ')
C
        ACV(1)= 'A'
        ACV(2)= 'C'
        ACV(3)= 'V'
C
        CALL STRING (ACV,3,WRT)
        CALL IBWRT (DVM,WRT,3)
C
        CALL IBRD (DVM,RDDAT(I),12)
        DO 70 I=1,STEPS
                DO 90 J=1,8
                        FREQ(J+1)=TEMP(J)
90              CONTINUE
                DVM = IBFIND ('SIGGEN ')
                CALL STRING (FREQ,10,WRT)
                CALL IBWRT (DVM,WRT,10)
                Y(I) = STARTF
C
                DVM = IBFIND ('DMM ')
                CALL IBRD (DVM,RDDAT(I),12)
                COUNT(I) = IBCNT
C
                STARTF = STARTF + STEPF
                VALUE = STARTF
                DO 120 K=1,8
                        TEMPA = INT(STARTF/(10**(8-K)))
                        TEMP(K) = CHAR (TEMPA + 48)
                        STARTF = STARTF - (TEMPA*10**(8-K))
120             CONTINUE
                STARTF = VALUE
```

```
70      CONTINUE
        CALL CLEAR
        WRITE (*,71)
71      FORMAT ('0','ENTER NAME OF FILE YOU WANT DATA STORED IN,')
        WRITE (*,72)
72      FORMAT (' ','USE A DOS NAME AS <c:xxxxxxx.yyy>')
        READ (*,73) FLNAME
73      FORMAT (1A14)
        OPEN (1,FILE=FLNAME,STATUS='NEW',FORM='FORMATTED')
        DO 300 I=1,STEPS
                LENGTH = COUNT(I)-3
                IF (LENGTH .EQ. 9) FMT = '(F10.2,'','',1A9)'
                IF (LENGTH .EQ. 8) FMT = '(F10.2,'','',1A8)'
                IF (LENGTH .EQ. 7) FMT = '(F10.2,'','',1A7)'
                IF (LENGTH .EQ. 6) FMT = '(F10.2,'','',1A6)'
                IF (LENGTH .EQ. 5) FMT = '(F10.2,'','',1A5)'
                IF (LENGTH .EQ. 4) FMT = '(F10.2,'','',1A4)'
                IF (LENGTH .LE. 3) FMT = '(F10.2,'','',1A3)'
                WRITE (1,FMT) Y(I),RDDAT(I)
300     CONTINUE
        REWIND 1
        DO 400 I=1,STEPS
                READ (1,410) VOUT
410             FORMAT (11X,BNE9.3)
                DB(I) = 20 * LOG10(VOUT/VIN)
400     CONTINUE
        REWIND 1
        DO 420 I=1,STEPS
                WRITE (1,430) Y(I),DB(I)
430             FORMAT (F10.2,',',E12.5)
420     CONTINUE
        CLOSE (1)
        WRITE (*,500) FLNAME
500     FORMAT ('0',1A16,' NOW CONTAINS YOUR BODE PLOT DATA.')
        WRITE (*,501)
501     FORMAT (' ','COLUMN 1 IS THE FREQUENCY DATA, COLUMN 2 IS THE')
        WRITE (*,502)
502     FORMAT (' ','GAIN IN DECIBELS. THE INPUT VOLTAGE IS ASSUMED')
        WRITE (*,503)
503     FORMAT (' ','TO BE CONSTANT OVER THE FREQUENCIES SWEPT.')
        RETURN
        END
```

LIST OF REFERENCES

1. Taylor, T., Use of the GPIB for Data Collection and Display, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1985.

2. Beasley, H. A. Electronic Circuit Testing Via the GPIB, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1985.

3. National Instruments, GPIB-PC User Manual, Austin, Texas, 1984.

4. Etter, D. M., Structured FORTRAN 77 For Engineers and Scientists, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1983.

BIBLIOGRAPHY

Advanced Graphics Software, Inc., SlideWrite Plus, Sunnyvale, California, 1986.

Fairley, Richard, Software Engineering Concepts, McGraw-Hill, Inc., 1985.

International Business Machines Corp., Disk Operating System Version 3.10, Boca Raton, Florida, 1985.

Kreitzberg, Charles B. and Shneiderman, Ben, FORTRAN Programming: A Spiral Approach, Harcourt Brace Jovanovich, Inc., New York, New York, 1982.

Microsoft Corporation, Microsoft FORTRAN Compiler, Redmond, Washington, 1984

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	2
4.	Professor J.P. Powers, Code 62Po Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	2
5.	Professor S. Michael, Code 62Mi Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
6.	Commandant (G-PTE) U.S. Coast Guard Headquarters 2100 2nd Street SW Washington, DC 20593	2
7.	Lieutenant George H. Self Jr., USCG Commandant (G-NRN) U.S. Coast Guard Headquarters 2100 2nd Street SW Washington DC 20593	2

221168

Thesis
S41255 Self
c.1 Implementation of an IBM-PC/AT as a GPIB controller.