KEITHLEY DAC

SERIES 500 GPIB IEEE Controller

Includes D500GPIB Software Driver

Publication Date: January 1990
Document Number: 501-904-01 Rev. C

Table of Contents

Introduction

1.1 Description

The Series 500 GPIB Controller consists of the D500GPIB software, the GPIB interface module, the installation program provide with your system (INSTALL), and other utility programs. Together, they provide a full implementation of the IEEE 488-1978 bus including advanced capabilities such as Parallel Poll, Serial Poll, Secondary Addressing, and automatic error detection.

D500GPIB is the software interface between DOS and the GPIB controller module. It can be accessed by virtually any language that can communicate with DOS files. The examples in this manual are primarily for BASICA, or other similar BASICs (such as GW BASIC), but D500GPIB is compatible with most languages for the PC. Some examples are Aztec C, Microsoft C and Fortran, Turbo Pascal, as well as Microsoft QuickBASIC.

D500GPIB receives simple, high-level commands from the program and carries them out using the necessary IEEE bus control and handshaking.

1.2 Specifications

IEEE 488-1978 Interface

SH1, AH1, T6, TE0, L4, LE0, SR1, PP0, RL0, DC1, DT1, C0, E1/2
Controller subsets: C1, C2, C3, C4 and C10
Terminator: Software selectable characters and/or EOI
Connector: Standard Amphenol 57-20240 with metric studs

GPIB Interface Module

IEEE Controller Device: TI 9914
Speed: Software and IEEE device dependent
Environment: 0 to 35 Celsius, 0 to 70% RH
Fits in slot 10 of Series 500 or expansion slot of System 570

D500GPIB Device Driver Software

Memory Size: 16 Kbytes
Command Set: More than 20 commands providing complete IEEE bus control
Operating Modes: Bus Controller modes only
DOS Compatibility: MS-DOS or PC-DOS 3.1 or higher

Specifications subject to change without notice

1.3 Abbreviations

The following IEEE 488 abbreviations are used throughout this manual.

addr n	IEEE bus address "n"
ATN	Attention line
CA	Controller Active
CR	Carriage Return
data	Data String
DCL	Device Clear
GET	Group Execute Trigger
GTL	Go To Local
LA	Listener Active
LAG	Listen Address Group
LF	Line Feed
LLO	Local Lock Out
MLA	My Listen Address
MTA	My Talk Address
PPC	Parallel Poll Configure
PPU	Parallel Poll Unconfigure
SC	System Controller
SDC	Selected Device Clear
SPD	Serial Poll Disable
SPE	Serial Poll Enable
SRQ	Service Request
TA	Talker Active
TAD	Talker Address
TCT	Take Control
term	Terminator
UNL	Unlisten
UNT	Untalk
*	Unasserted

Installation

2.1 Inspection

The Keithley Series 500 GPIB system, including the GPIB interface module and the D500GPIB software, are carefully inspected, both mechanically and electrically, prior to shipment. When you receive the product, carefully unpack all items from the shipping carton and check for any obvious signs of physical damage which may have occurred during shipment. Immediately report any such damage found to the shipping agent. Remember to retain all shipping materials in the event that shipment back to the factory becomes necessary.

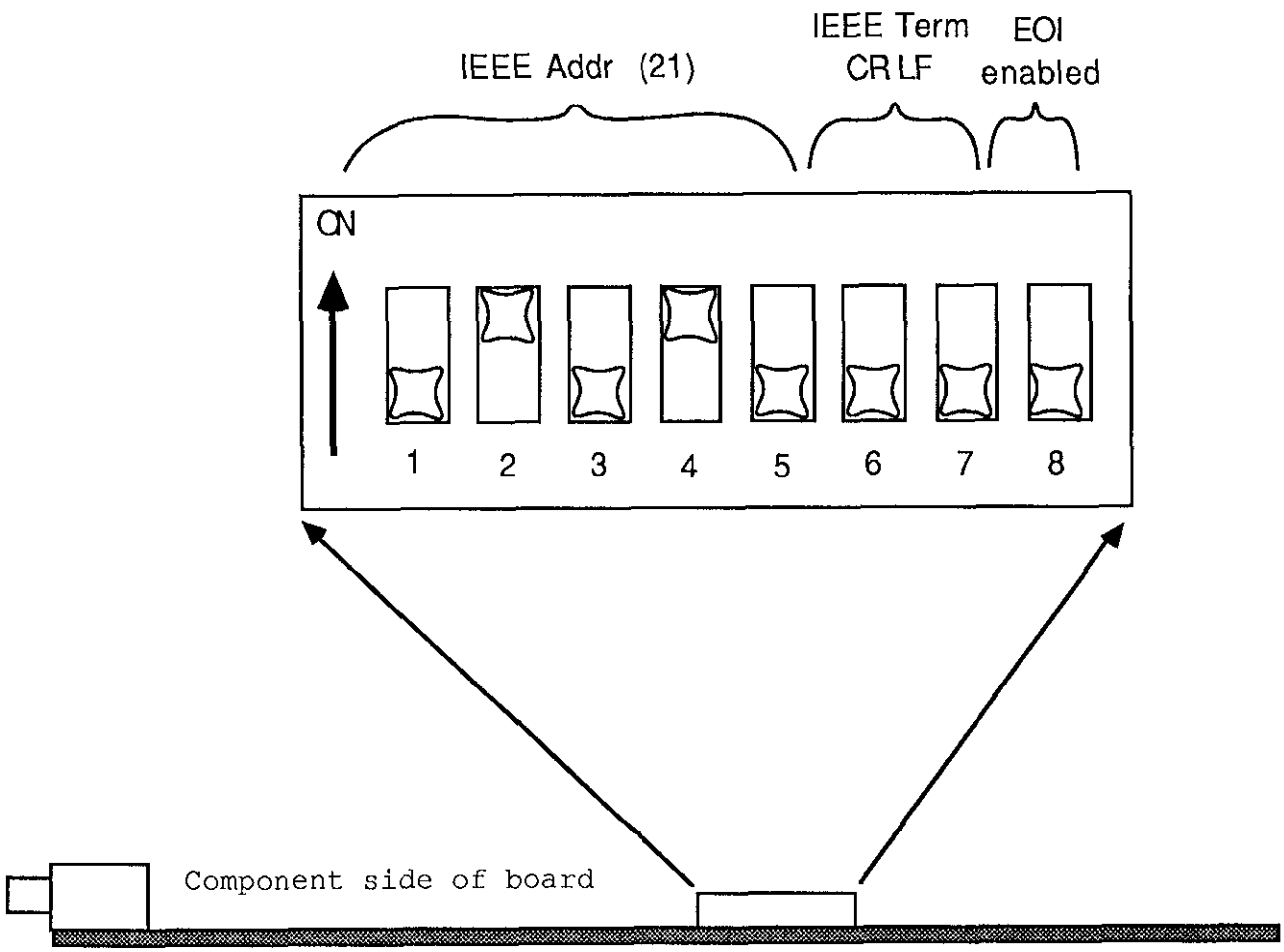
2.2 Hardware Installation

To use the D500GPIB software, you must have a GPIB interface module installed in your Series 500 or System 570. The module has a bracket that must be installed and switches that must be positioned correctly before installing it into your system. The following sections will discuss those things pertaining to either the Series 500 or System 570. This section details those things needed for either one.

The dip switches on the GPIB module set the configuration of the interface. Most of the selectable functions are read only at power up and should be set prior to applying power. The figure on the next page illustrates the factory default settings. To modify any of these defaults, follow this simple procedure. Turn the power off on the Series 500 or System 570 and modify the switch settings as specified on the following pages. When the switch setting modifications are complete, turn the system power back on, It will usually be necessary to reinitialize any software that is being used to control the module as well. This can easily be done if the software uses an IOCTL command near the beginning of the program. This command will be explained in detail later, but make a mental note of it for now.

As you can see from the illustration, there are two parameters that can be controlled via these switch settings. They are the IEEE bus address and the type of terminator used at the end of a bus transmission (including EOI if required). The factory defaults are IEEE address 21 (a good starting point since it is infrequently used by bus devices), and a terminator of CR LF with EOI set with the linefeed. This is practically an industry standard and is accepted by most instruments supplied by Keithley. If any of these settings are not suitable, then refer to the following sections for detailed information on how to modify them.

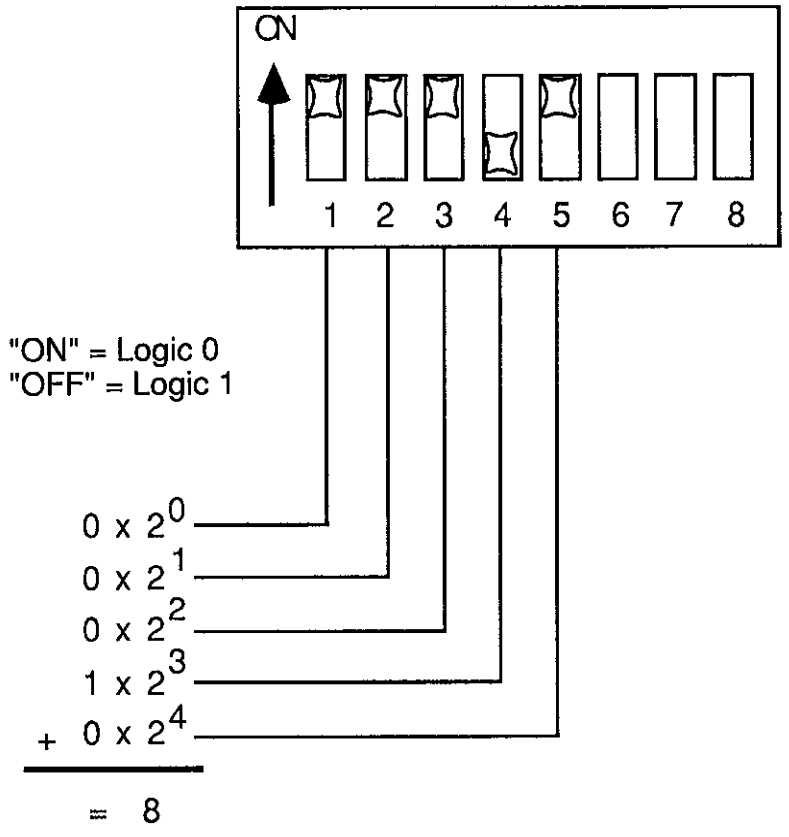
Factory Default Switch Settings



2.2.1 IEEE Address Selection

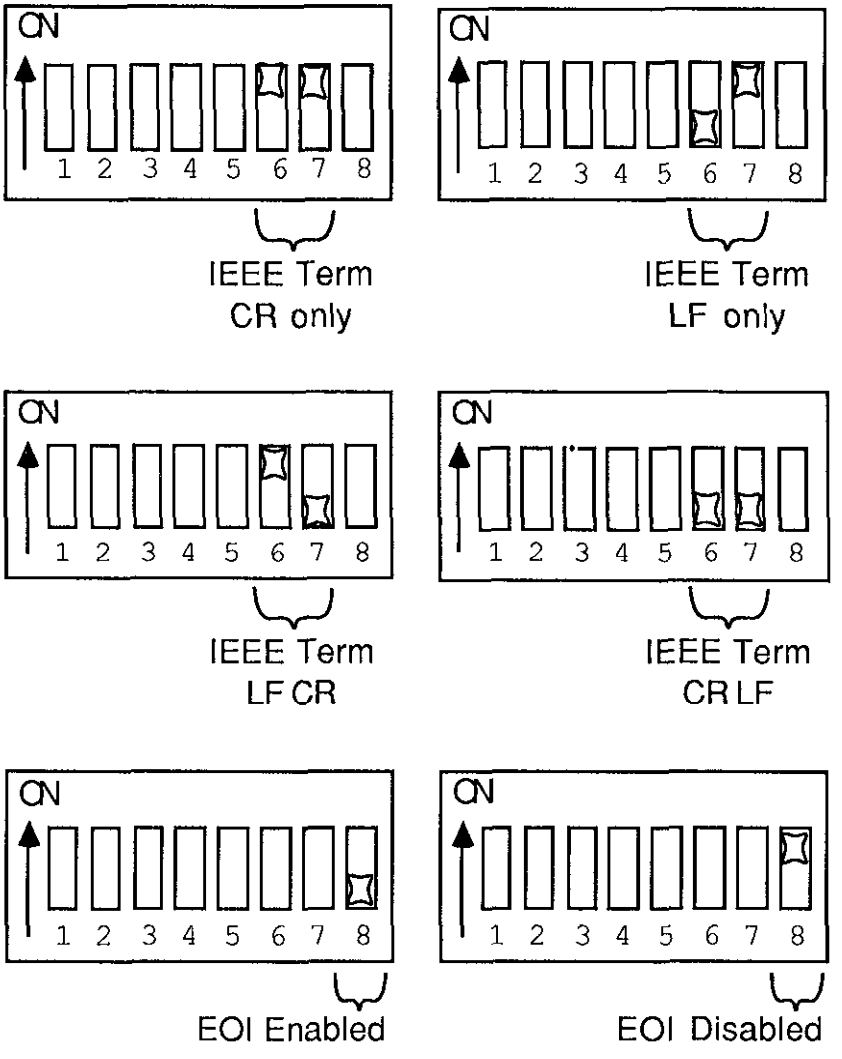
Switches 1 through 5 select the IEEE bus address as the SYSTEM CONTROLLER. The address is selected by simple binary weighting with switch 1 being the least significant bit and switch 5 being the most significant bit. The factory default is 21. The following is an example of how you might change the bus address to 08.

NOTE: That the switch logic is "low-true", i.e.



2.2.2 IEEE Bus Terminator Selection

Switches 6 through 8 set the IEEE bus terminator used for data which is sent to an instrument on the bus. These are the only switch selectable parameters that can be modified under software control (using the TERM command). As mentioned earlier, the factory default terminators are CR LF (carriage return line feed) with EOI set on the line feed. Refer to the next page for all the various switch configurations and their meanings.


ON
1 2 3 4 5 6 7 8
IEEE Term
CR only
ON
1 2 3 4 5 6 7 8
IEEE Term
LF only
ON
1 2 3 4 5 6 7 8
IEEE Term
LF CR
ON
1 2 3 4 5 6 7 8
IEEE Term
CR LF
ON
1 2 3 4 5 6 7 8
EOI Enabled
ON
1 2 3 4 5 6 7 8
EOI Disabled

2.2.3 Series 500 Installation Instructions

The GPIB interface module is installed into expansion slots inside the Series 500 mainframe. The GPIB interface module may be installed into any vacant slot of a Series 500 but slot 10 is the only one we recommend. The reason for this is that the mounting bracket supplied with the module can only be used with slot 10. This bracket provides additional stability to the module which is needed due to the bulky, stiff IEEE cables used to connect instruments to the module.

Install the long L shaped mounting bracket supplied with the GPIB module using the screws provided. The bracket should be attached to the soldered (non component) side of the module with the short tab facing away from the card. The hole in this tab is designed to line up with an existing screw in the base of your Series 500 mainframe near slot 10.

After attaching the bracket you are ready to install the module in the Series 500. Insert the GPIB interface module into the expansion slot as follows: Make sure the Series 500 is turned off, and unplug the power cord. Unscrew the cover mounting screws on the rear of the unit. Remove the system unit cover by sliding it backward and tilting it up.

First look at the base of the Series 500 near slot 10. You should see the screw towards the back of the unit that is used to secure the metal cover plate. Remove this screw and set it aside. Carefully insert the GPIB interface module into expansion slot 10. With the board firmly in place, fix its mounting bracket to the metal panel in the base of the unit using the longer screw provided with the module. The original screw that was in the base unit can not be used because it is too short.

Finally, slide the system unit cover back on, reattach it with its screws, plug it back in, and turn it on.

2.2.4 System 570 Installation Instructions

The GPIB interface module is installed into the single expansion slot in the System 570. A special mounting bracket is supplied with the module that replaces the existing bracket in the rear of the 570. This bracket provides additional stability to the module which is needed due to the bulky, stiff IEEE cables used to connect instruments to the module.

First remove the existing stabilizing bracket from the rear panel of the System 570. Save the screws since they will be used to mount the GPIB module mounting bracket. Install the short angled bracket supplied with the GPIB module using the screws provided. The bracket should be attached to the soldered (non component) side of the card. The holes in this bracket are designed to line up with the holes on the module and the existing holes for the original 570 bracket.

After attaching the bracket you are ready to install the module in the System 570. Insert the GPIB interface module into the expansion slot as follows: Make sure the System 570 is turned off. Carefully insert the GPIB interface module into the expansion slot. With the board firmly in place, fix its mounting bracket to the metal panel in the rear of the unit using the screws provided with the original 570 bracket.

Finally, turn the power back on on the System 570 and close the cover.

2.3 Software Installation

Once the GPIB interface board has been configured and installed, the software must be installed. The D500GPIB software is included on the Software disk that comes with the GPIB Board. Before installing the software driver for the GPIB card you must know the following:

1. The address of the Keithley IBIN interface card used with the data acquisition system which holds the GPIB card.
2. The slot number (1 through 10) that the GPIB card occupies in the data acquisition system.

To complete the installation you must edit your DOS AUTOEXEC.BAT file to include the following line:

GPIB_PATH -I IBIN_ADDR -S SLOT_NUM

where:

GPIB_PATH	is a complete DOS file specification to the Series 500 GPIB driver (DGPIB500.COM).
IBIN_ADDR	is the address of the IBIN card specified in hexadecimal format (e.g. 0xCFF8).
SLOT_NUM	is the slot number of the GPIB card (1 through 10).

Note: A space character must separate both the IBIN address and Slot Number from the command line flags of -I and -S respectively. For example:

c:\S500\D500GPIB.COM -I 0xcff8 -S 10

c:\D500GPIB.COM -I 0XDFF8 -S 0xA

Optionally, you can create a batch file with the above information in it which you run prior to using the Series 500 GPIB device. In this way yhe driver is only resident in memory when needed freeing up system RAM for your utilities.

When the driver is installed properly, it will display a signon message with copyright notice as follows:

D500GPIB Rev n.xx (C) Copyright 1987 Keithley DAC

After it is installed you can try the following program ...

```
10 OPEN "\dev\GPIBOUT" FOR OUTPUT AS 1
20 OPEN "\dev\GPIBIN" FOR INPUT AS 2
30 PRINT #1,"HELLO"
40 LINE INPUT #2,A$ : PRINT A$
```

The GPIB will respond with (and the host computer will display):

```
SERIES 500/488 Rev n.xx (C) Copyright IOtech Inc
```

If you obtain the above response, (or something very similar) then your GPIB module and the D500GPIB software are working well together. If you did not receive the above message, then you should contact our applications staff for assistance.

Now that the hardware and software are working properly, you can begin to use the rest of the commands to control your IEEE bus peripherals. The following command descriptions each have examples showing their use. You may also like to familiarize yourself with the various commands using the "dumb" terminal program provided on the disc (TERMGPIB.BAS).

Getting Started

Introduction

Once D500GPIB has been installed in your system, it is ready to begin controlling IEEE bus devices. To show how this is done, we will develop a short program, in BASIC, to control a Keithley Instruments Model 195 digital multimeter. The techniques used in this program are quite general, and will apply to the control of most instruments.

3.1 Using D500GPIB with BASIC

3.1.1 Initializing the System

Any program using D500GPIB must first establish communications with the software driver. In BASIC and most other languages this is accomplished using an OPEN statement. Communications both to and from D500GPIB are required. In BASIC, this means that two files must be opened, one for input, and one for output. Other languages may allow the same file to be opened for both input and output. Three file names are allowed: "\DEV\GPIBOUT", "\DEV\GPIBIN", and "\DEV\GPIB". By convention, they are used for output, input, and both input and output, respectively, but actually, they are all really the same and any one of them can be used for input, output, or both, depending on the programming language.

In BASIC, the files are opened with the following commands:

```
110 OPEN "\DEV\GPIBOUT" FOR OUTPUT AS #1
200 OPEN "\DEV\GPIBIN" FOR INPUT AS #2
```

Of course, the line numbers and the file numbers may change as desired, but throughout this manual, file #1 will be assumed to output to D500GPIB, and file #2 will be assumed to input from D500GPIB.

Once these files are opened, we can send commands and receive responses from D500GPIB. While D500GPIB should normally be in a reset, quiescent state, it is possible that is was left in some unknown state by a previous program failure error or from a power up. In order to force D500GPIB into its quiescent state we can use the IOCTL statement:

```
160 IOCTL#1,"BREAK"
```

IOCTL is a BASIC statement that sends commands through a "back door" to D500GPIB. D500GPIB recognizes this "back door" command regardless of what else it might be doing and resets itself so that it is ready to accept a normal command.

The IOCTL BREAK command guarantees that D500GPIB is ready for action. Note that the IOCTL BREAK command is executed before file #2 is opened for input from D500GPIB. This guarantees compatibility with the various versions of BASIC.

With the initialization commands and some comments, the program now appears as follows:

```
100 'Establish communications with D500GPIB
110 OPEN "\DEV\GPIBOUT" FOR OUTPUT AS #1
150 'Reset D500GPIB
160 IOCTL#1,"BREAK"
190 'Open file to read responses from D500GPIB
200 OPEN "\DEV\GPIBIN" FOR INPUT AS #2
```

Once everything is reset, we can test the communications and read the Driver488 revision number with the HELLO command:

```
310 PRINT#1,"HELLO"
320 INPUT#2,A$
330 PRINT A$
```

First we PRINT the HELLO command to file #1, then we INPUT the response from file #2 into the character string variable A$ ("A-string"). Finally we display the response with a PRINT to the screen. Because BASIC cannot both PRINT and INPUT from the same file, we use two OPEN statements, and two different file numbers to communicate with D500GPIB. PRINT must reference the file opened for output (in these examples, file #1) and INPUT must reference the file opened for input (file #2). Attempting to communicate with the wrong file (e.g. INPUT#1) will result in an error.

It is not necessary to perform the HELLO command, but is included here as a simple example of normal communication with D500GPIB. Its response is the revision identification of the Driver488 software ("D500GPIB Rev n.n (C) Copyright Keithley Instruments Inc.").

3.1.2 Configuring the 195 DMM

Once the system is initialized we are ready to start issuing bus commands. The IEEE bus has already been cleared by the Interface Clear (IFC) that was sent by the BREAK command, and so we know that all the bus devices are waiting for the controller to take some action. In the normal state, IEEE bus devices are controlled from their front panel or keyboard. They are not normally enabled to accept commands from the IEEE bus. To set them into the remote mode, so that they can accept bus commands, they must be addressed to listen while the Remote Enable (REN) line is asserted. The REMOTE command accomplishes this:

```
510 PRINT#1,"REMOTE 16"
```

where 16 is the bus device address of the bus device we wish to control (in this case a Keithley Instruments Model 195 digital multimeter).

Now that the 195 is ready to accept commands, we can control any of its features. For example, the command "F0R0X" command sets the 195 to read DC volts with automatic range selection:

```
610 PRINT#1,"OUTPUT 16;F0R0X"
```

The OUTPUT command takes a bus device address (16 in this case) and data ("F0R0X") and sends the data to the specified device. The address can be just a primary address, such as 12, or 05, or it can include a secondary address: 1201. Note that both the primary address and, if present, the secondary address are two-digit decimal numbers. A leading zero must be used, if needed to make the address two digits long.

3.1.3 Taking Readings

Once we have set the 195's operating mode, we can take a reading and display it:

```
710 PRINT#1,"ENTER 16"
720 INPUT#2,R$
730 PRINT R$
```

The ENTER command takes a bus address (with an optional secondary address) and configures that bus device so that it is able to send data (addressed to talk). No data is actually transferred, however, until the INPUT statement requests the result from D500GPIB at which time data is transferred to the program into the variable R$.

Once the result has been received, any BASIC functions or statements can be used to modify or interpret it. In this example the result will be in the form "NDCV+1.23456E-2" showing the range ("NDCV") and the numeric value of the reading ("+1.23456E-2"). The BASIC MID$ function can be used to strip off the range characters and keep only the numeric part (the fifth character and beyond), and the VAL function can be used to convert this string to a number:

```
740 N$=MID$(R$,5)
741 N=VAL(N$)
742 PRINT "The read value is";N
```

These may be combined for efficiency:

```
740 PRINT "The read value is";VAL(MID$(R$,5))
```

All the power of BASIC may be used to manipulate, print, store, and analyze the data read from the IEEE bus. For example, the following statements print the average of ten readings from the 195:

```
810 SUM=0
820 FOR I=1 TO 10
830 PRINT#1,"ENTER 16"
840 INPUT#2,R$
850 SUM=SUM+VAL(MID$(R$,5))
860 NEXT I
870 PRINT "The average of ten readings is";SUM/10
```

3.1.4 Polling for Status

The IEEE bus is designed to be able to attend to asynchronous (i.e. unpredictable) events or conditions. When such an event occurs, the bus device needing attention can assert the Service Request (SRQ) line to signal that condition to the controller. Once the controller notices the SRQ, it can interrogate the bus devices, using Parallel Poll and/or Serial Poll to determine the source and cause of the SRQ, and take the appropriate action.

Parallel Poll is the fastest method of determining which device requires service. Parallel Poll is a very short, simple IEEE bus transaction that quickly returns the status from many devices. Each of the eight IEEE bus data bits can holds the Parallel Poll response from one or more devices. So, if there are eight or fewer devices on the bus, then just the single Parallel Poll can determine which requires service. Even if the bus is occupied by the full complement of 15 devices, then Parallel Poll can narrow the possibilities down to a choice of at most two.

Unfortunately, the utility of Parallel Poll is limited when working with actual devices. Some have no Parallel Poll response capability. Others must be configured in hardware, usually with switches or jumpers, to set their Parallel Poll response. If Parallel Poll is not available, or several devices share the same Parallel Poll response bit, then Serial Polling will still be required to determine which device is requesting service.

Serial Poll, though it is not as fast as Parallel Poll, does offer two major advantages: It returns additional status information beyond the simple request/no-request for service, and it is implemented on virtually all bus devices.

The SRQ line can be monitored by using the SPOLL command.

The 195 can be set to request service on any of several different internal conditions. In particular, the M1 command will cause an SRQ when a reading has been completed by the 195:

```
540 PRINT#1,"OUTPUT 16;M1X"
```

This OUTPUT command is placed just after the 195 is placed in remote mode so that each reading taken by the 195 will cause an SRQ if it is valid.

At this point BASIC can check for an SRQ using a Serial Poll subroutine. We will also read the Serial Poll Status of the 195 to determine if there are any errors.

```
1999 '
2000 'Serial Poll routine used to check for reading
2001 'completed as well as any other errors !!!!!!!
2010 '
2050 'Check for an SRQ - wait until it occurs
2060 PRINT#1,"SPOLL"
2070 INPUT#2,SP
2080 IF SP=0 THEN GOTO 2060
```

Next we Serial Poll the 195 to determine its status. If there were other devices on the bus that could be generating the SRQ, then each of them would be have to be checked in turn.

```
2110 PRINT#1,"SPOLL 16"
2120 INPUT#2,ST195
2130 IF (ST195 AND 64) = 0 THEN PRINT "Non-195 SRQ!": STOP
```

Bit DIO7, with a value of 64, is returned as true (1) in the Serial Poll response of those devices requesting service. In our simple example we expect that the 195 is the only possible cause of an SRQ, and if it is not the cause then there must be some error.

Now that we have identified the device that is requesting service, we can further examine the Serial Poll status to classify the request. If DIO5 is set, then the 195 is signaling an error condition. If that bit is clear, then some non-error condition caused the SRQ:

```
2160 IF ST195 AND 32 THEN 2300 'Test ERROR Status Bit

2210 IF ST195 AND 1 THEN PRINT "Overflow"
2220 IF ST195 AND 2 THEN PRINT "Buffer Full"
2230 IF ST195 AND 4 THEN PRINT "Buffer 1/2 Full"
2240 IF ST195 AND 8 THEN PRINT "Reading Done"
2250 IF ST195 AND 16 THEN PRINT "Busy"
2260 GOTO 2400

2310 IF ST195 AND 1 THEN PRINT "Illegal Command Option"
2320 IF ST195 AND 2 THEN PRINT "Illegal Command"
2330 IF ST195 AND 4 THEN PRINT "No Remote"
2340 IF ST195 AND 8 THEN PRINT "Trigger Overrun"
2350 IF ST195 AND 16 THEN PRINT "Failed Selftest"
```

Finally, once we have diagnosed the error, we are ready to return to the main program:

```
2400 RETURN
```

3.1.5 The Complete BASIC Sample Program

This program is provided on the Supplemenatal disk as "DEMO195.BAS"

```
100 'Establish communications with D500GPIB
110 OPEN "\DEV\GPIBOUT" FOR OUTPUT AS #1
120 '
150 'Reset D500GPIB
160 IOCTL#1,"BREAK"
180 '
190 'Open file to read responses from D500GPIB
200 OPEN "\DEV\GPIBIN" FOR INPUT AS #2
290 '
300 'Read the signon and revision message
310 PRINT#1,"HELLO"
320 INPUT#2,A$
330 PRINT A$
340 '
500 'Put the 195 (assuming bus address 16) into REMOTE
510 PRINT#1,"REMOTE 16"
520 '
530 'Enable 195 SRQ on Data Aquired
540 PRINT#1,"OUTPUT 16;M1X"
550 '
600 'Set 195 to Autorange, DC Volts
610 PRINT#1,"OUTPUT 16;F0R0X"
620 '
700 'Display a reading here - wait for it!
705 GOSUB 1999
710 PRINT#1,"ENTER 16"
720 INPUT#2,R$
730 PRINT R$
740 PRINT "The read value is";VAL(MID$(R$,5))
790 '
800 'Find the average of 10 readings
810 SUM=0
820 FOR I=1 TO 10
830 GOSUB 1999
835 PRINT#1,"ENTER 16"
840 INPUT#2,R$
850 SUM=SUM+VAL(MID$(R$,5))
860 NEXT I
870 PRINT "The average of ten readings is";SUM/10
1900 END
1999 '
2000 'Serial Poll routine used to check for reading
2001 'completed as well as any other errors !!!!!!!
2010 '
2050 'Check for an SRQ - wait until it occurs
2060 PRINT#1,"SPOLL"
2070 INPUT#2,SP
```

```
2080 IF SP=0 THEN GOTO 2060
2090 '
2100 'Check the 195 Serial Poll status
2110 PRINT#1,"SPOLL 16"
2120 INPUT#2,ST195
2130 IF (ST195 AND 64) = 0 THEN PRINT "Non-195 SRQ!": STOP
2140 '
2150 'Test for 195 ERROR
2160 IF ST195 AND 32 THEN 2300 'ERROR STATUS BIT
2170 '
2200 'Interpret no-error status
2210 IF ST195 AND 1 THEN PRINT "Overflow"
2220 IF ST195 AND 2 THEN PRINT "Buffer Full"
2230 IF ST195 AND 4 THEN PRINT "Buffer 1/2 Full"
2240 IF ST195 AND 8 THEN PRINT "Reading Done"
2250 IF ST195 AND 16 THEN PRINT "Busy"
2260 GOTO 2400
2270 '
2300 'Interpret error status
2310 IF ST195 AND 1 THEN PRINT "Illegal Command Option"
2320 IF ST195 AND 2 THEN PRINT "Illegal Command"
2330 IF ST195 AND 4 THEN PRINT "No Remote"
2340 IF ST195 AND 8 THEN PRINT "Trigger Overrun"
2350 IF ST195 AND 16 THEN PRINT "Failed Selftest"
2360 '
2390 'Return to main program
2400 RETURN
```

Data Transfers

4.1 Terminators

The following section deals with the subject of terminators in a general sense. Under normal cicumstances you will have no need to know how terminators effect your programing. If you are interested please read on, however this information is not needed to get started.

Every transfer of data, between a program and D500GPIB, or between D500GPIB and a bus device, must have a definite end. This is a common requirement in most systems. For example, most printers will not print a line until they have received the carriage-return that ends that line. Similarly, a BASIC INPUT statement will wait for the Enter key to be pressed before returning the entered data to the program. The only time that some terminator is not required is when the number of characters that compose the data is known in advance. This is the case, for example, when reading fixed-length records from a random access disk file.

There are actually four terminators used by D500GPIB:

The end-of-line (EOL) terminator for output from the program to D500GPIB.

The end-of-line (EOL) terminator for input to the program from D500GPIB.

The data terminator (TERM) for output to bus devices from D500GPIB.

The data terminator (TERM) to input from bus devices into D500GPIB.

4.1.1 The End-Of-Line (EOL) Terminators

The EOL terminators mark the end of character strings transferred between the user's program and D500GPIB. The EOL output terminator marks the end of strings transferred from the user's program to D500GPIB, and the EOL input terminator marks the end of strings transferred into the user's program from D500GPIB.

The EOL terminators consist of two ASCII characters each. These characters default to carriage return and line feed and can not be modified under program control. It turns out that these terminators serve well for the vast majority of languages and are particularly usefull for BASIC.

The EOL output terminator is sensed by D500GPIB to detect the end of a command or, in the case of the `OUTPUT...;data` command, the end of the data. Most commands have many different variations. It is the EOL output terminator that lets D500GPIB known when the command has been completely received and is ready for execution. Without the EOL output terminator there would be no way of determining when one command ends and the next begins.

The EOL input terminator is provided by D500GPIB to the user's program so that the program will be able to detect the end of a response. BASIC needs to receive a carriage return, line feed combination when using the INPUT statement to receive a response from D500GPIB. D500GPIB automatically provides this EOL input terminator to the program.

As mentioned previously, the EOL output terminator is used to delimit the `data` portion of an

OUTPUT command. If, in the OUTPUT command, no character count is specified, then the EOL output terminator does delimit data, but if a character count is specified, then D500GPIB will accept exactly that number of characters from the program for output to the bus, even if the EOL output terminator is among those characters. Furthermore, if a character count is not specified, then the TERM output terminator will be sent to the bus devices after the data. If a character count is specified, then nothing will be sent to the bus except the exact characters that were sent from the program. For example `PRINT#1,"OUTPUT10;ABC"` sends ABC<TERM output terminator> to device 10, while `PRINT#1,"OUTPUT10#5;DEF"` sends DEF<CR><LF> to device 10 because BASIC will send carriage return, line feed (<CR><LF>) at the end of the command, and a character count of 5 was specified.

4.1.2 The TERM Terminators

Just as the EOL terminators delimit the end of strings transferred between the user's program and D500GPIB, the TERM terminators delimit the end of strings transferred between D500GPIB and bus devices. The TERM output terminator marks the end of strings transferred from D500GPIB to bus devices, and the TERM input terminator marks the end of strings transferred into D500GPIB from bus devices. One major difference between EOL and TERM is that TERM is programable while EOL is not. TERM input is, however, much like EOL in that it defaults to LF (linefeed) but can be modified on a "one shot" basis by appending the desired terminator to the end of the ENTER command. TERM output defaults to the values specied by the switch settings, but can be modified by the TERM command to D500GPIB.

The TERM terminators differ from the EOL terminators in one important aspect. While the EOL terminators are composed of one or two characters, the TERM terminators can include the IEEE bus signal EOI. The EOI signal, when asserted during a character transfer, marks that character as the last of the transfer. This allows the detection of the end of data regardless of what characters compose the data. This feature is very useful in binary data transfers which might very well contain any ASCII values from 0 to 255.

During normal OUTPUT, without a specified character count or buffer, the EOL output terminator received by D500GPIB is replaced by the TERM output terminator before sending the data to the bus devices. During normal ENTER the TERM input terminator received by D500GPIB is replaced with the EOL input terminator before being returned to the program. In this way, the program communicates with D500GPIB using the EOL terminators, and D500GPIB communicates with bus devices using the TERM terminators.

See the ENTER and OUTPUT command descriptions below and in Chapter 6 for more details.

4.2 Bus OUTPUT

The OUTPUT command sends data to bus devices. For example, the statement

```
PRINT#1,"OUTPUT 05;SP1;"
```

will send the characters "SP1;"<TERM output> to device 5. This is an example of direct I/O as the data is communicated directly to the D500GPIB through the PRINT statement. As discussed above, D500GPIB recognizes the EOL output terminator as the end of the data and sends the TERM output terminator in its place after sending the data. Binary direct output is also possible. For example, the following statements send all 256 ASCII characters:

```
PRINT#1,"OUTPUT 05 #256;";
FOR I=0 TO 255
PRINT#1,CHR$(I);
NEXT I
```

The first statement tells D500GPIB to expect 256 characters to follow that are to be sent to device 5. Note the semicolon just after the #256. This marks the end of the actual OUTPUT command and the start of the data. The semicolon at the end of the line tells BASIC not to send anything else, such as the normal carriage return, line feed combination, after sending the quoted characters. The next three lines send the 256 ASCII characters from 0 to 255 in order to D500GPIB for transfer to device 5. The semicolon at the end of the third line has the same function as the semicolon at the end of the first line: it prevents BASIC from sending any extra characters. In this example we are performing a binary transfer. D500GPIB knows how many characters are to be transferred and neither requires EOL output terminators to end the command, nor sends TERM output terminators to the bus device. The data is transferred to the bus device exactly as sent from the program.

4.3 Bus ENTER

The ENTER command is used to read data from bus devices. For example, the commands

```
PRINT#1,"ENTER 16"
INPUT#2,A$
```

read data from device 16 and store the returned data in the A$ variable. This is an example of direct ENTER input as the data received from the bus is read into the program via the INPUT statement that reads the result directly from D500GPIB. As discussed above, D500GPIB will accept data from device 16 until it detects the TERM input terminator. It returns the replaces the TERM input terminator with the EOL output terminator and returns the result to the program. BASIC accepts the data just as it accepts character data from any file. This allows us to use the varieties of BASIC input statements to control how the data is received. For example, if the data read form the device is in the form of a valid number then we can read it as a number:

```
PRINT#1,"ENTER 16"
INPUT#2,N
```

Or, if the data consists of several values separated by commas, then we can read it as several values:

```
PRINT#1,"ENTER 16"
INPUT#2,A$,N,B$,I
```

Or, if we want to read the entire input, even if it includes commas or other special characters, we can use LINE INPUT:

```
PRINT#1,"ENTER 16"
LINE INPUT#2,L$
```

Finally, just as we can perform direct binary OUTPUT, we can also perform binary direct ENTER:

```
PRINT#1,"ENTER 16#128"
A$=INPUT$(128,2)
```

When performing a binary ENTER, D500GPIB does not check for TERM input terminators when reading from the bus. The data is returned to the program just as it is received from the bus device. The INPUT$ function which is designed to read a specific number of characters from a file or device is ideal for reading the result from D500GPIB. A normal INPUT statement will also work, as D500GPIB even provides the EOL input terminators on binary ENTERs, however, the transfer may be terminated abnormally if the EOL sequence occurs within the data.

IEEE Operating Modes

5.1 Introduction

The following section deals with the subject of IEEE operating modes in a general sense. Under normal cicumstances you will have no need to know these modes since the GPIB module can only act as the system controller and all other devices must therfore be peripherals. If you are interested please read on, however this information is not needed to get started.

Although the GPIB module can only be the System Controller it is usefull to discuss the modes that can occur on the IEEE bus. There are actually four types of IEEE bus devices: Active Controllers, Peripherals, Talk-only devices, and Listen-always devices. Talk-only and Listen-always devices are usually used together, in simple systems, such as a Talk-only digitizer sending results to a Listen-always plotter. In these simple systems no controller is needed because the talker assumes that it is the only talker on the bus, and the listener(s) assume that they are all supposed to receive all the data send over the bus. This is a simple and effective method of transferring data from one device and another, but is not adequate for more complex systems where, for example, one computer is controlling many different bus devices.

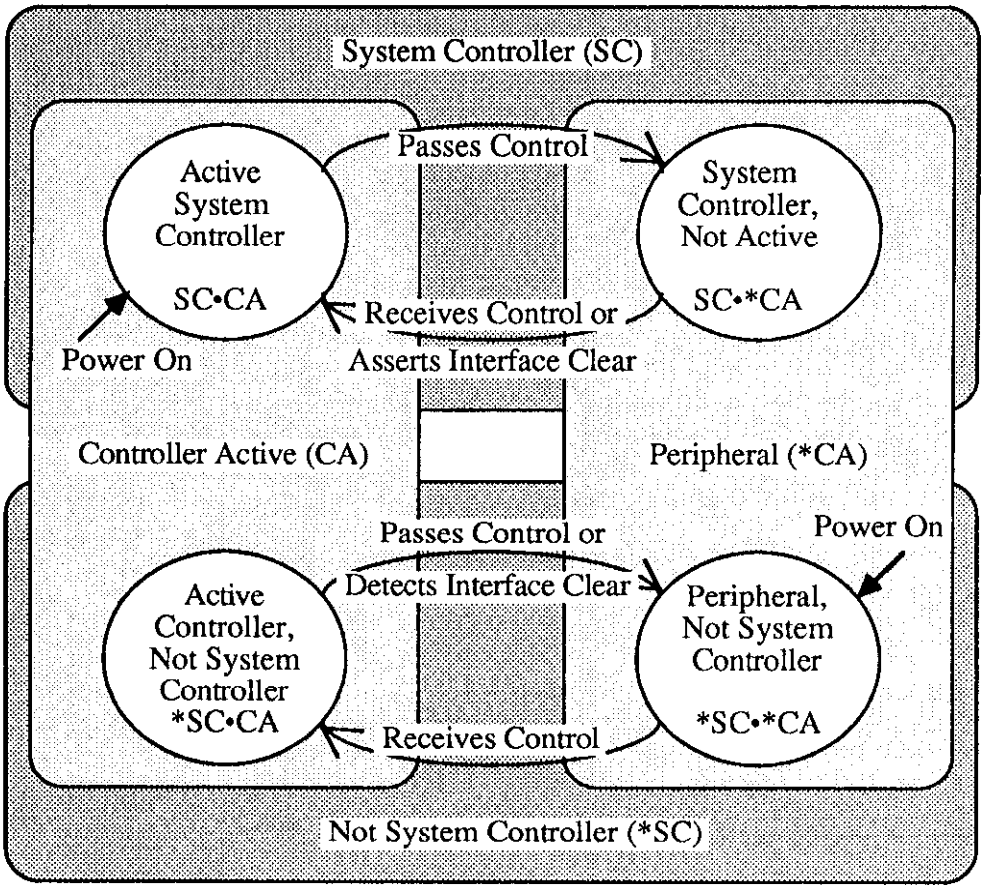
In more complex systems, the Active Controller sends commands to the various bus Peripherals telling them what to do. Commands such as Unlisten, Listen Address Group, Untalk, and Talk Address Group are sent by the controller to specify which device is to send data, and which are to receive it. For more details about the IEEE bus protocols see Chapter 7.

When an IEEE bus system is first turned on, some device must be the Active Controller. This device is the System Controller and always keeps some control of the bus. In particular, the System Controller controls the Interface Clear (IFC) and Remote Enable (REN) bus management lines. By asserting Interface Clear, the System Controller forces all the other bus devices to stop their bus operations, and regains control as the Active Controller.

5.2 Operating Mode Transitions

The System Controller is initially the Active Controller. It can, if desired, Pass Control to another device and thereby make that device the Active Controller. Note that the System Controller remains the System Controller, even when it is not the Active Controller. (NOTE: The GPIB module currently does not implement the Pass Control feature.) Of course, the device to which control is passed must be capable of taking on the role of Active Controller. It would make no sense to try to pass control to a printer. Control should only be passed to other computers that are capable, and ready, to become the Active Controller. Further, note that there must be exactly one System Controller on the IEEE bus. All other potential controller must be configured as Peripherals when they power up.

The state diagram below shows the relationships between the various operating modes. The top half of the state diagram shows the two operating states of a System Controller. At power on, it is the active controller. It directs the bus transfers by sending the bus commands mentioned previously. It also has control of the Interface Clear and Remote Enable bus lines. The System Controller can pulse Interface Clear to reset all of the other bus devices.



IEEE Bus Operating Modes State Diagram

As shown in the diagram, the System Controller can pass control to some other bus device and thereby become a Peripheral to the new Active Controller. If the System Controller receives control from the new Active Controller, then it will once again become the Active Controller. The System Controller can also force the Active Controller to relinquish control by asserting the Interface Clear signal.

The bottom half of the state diagram shows the two operating states of a Not System Controller device. At power on, it is a Peripheral to the System Controller which is the Active Controller. If it receives control from the Active Controller, then it becomes the new Active Controller. Even though it is the Active Controller, it is still not the System Controller. The System Controller can force the Active Controller to give up control by asserting Interface Clear. The Active Controller can also give up control by Passing Control to another device, which may or may not be the System Controller.

In summary, a bus device is set in hardware as either the sole System Controller in the system, or as a non-System Controller. At power on, the System Controller is the Active Controller, and the other devices are Peripherals. The System Controller can give up control by Passing Control, and can regain control by asserting Interface Clear, or by receiving control. A Peripheral can become the Active Controller by receiving control, and can give up control by Passing Control, or upon detecting Interface Clear.

5.3 The GPIB Module as System Controller

The most common D500GPIB configuration is as the System Controller, controlling several IEEE bus instruments. In this mode, D500GPIB can perform all of the various IEEE bus protocols necessary control and communicate with any IEEE 488 bus devices. As the System Controller in the Active Controller mode, D500GPIB can use all of the commands available for the Active Controller state, plus control the Interface Clear and Remote Enable lines. The allowed bus commands and their actions are as follows:

ABORT	Pulse Interface Clear.
LOCAL	Unassert Remote Enable, or send Go To Local to selected devices.
REMOTE	Assert Remote Enable, optionally setting devices to Remote.
LOCAL LOCKOUT	Prevent local (front-panel) control of bus devices.
CLEAR	Clear all or selected devices.
TRIGGER	Trigger selected devices.
ENTER	Receive data from a bus device.
OUTPUT	Send data to bus devices.
SPOLL	Serial Poll a bus device, or check the Service Request state.
PPOLL	Parallel Poll the bus.
PPOLL CONFIG	Configure Parallel Poll responses.
PPOLL DISABLE	Disable the Parallel Poll response of selected bus devices.
PPOLL UNCONFIG	Disable the Parallel Poll response of all bus devices.
SEND	Send low-level bus sequences.
RESUME	Unassert Attention. Used to allow Peripheral-to-Peripheral transfers.

Command Descriptions

6.1 Introduction

This chapter contains a detailed listing of each of the high-level commands available for D500GPIB. There are two types of commands: bus commands, and system commands. Bus commands communicate with the IEEE 488 bus. System commands configure or request information from D500GPIB.

Bus Commands:

ABORT
CLEAR
ENTER
LOCAL
LOCAL LOCKOUT
OUTPUT
SPOLL
PPOLL CONFIG
PPOLL DISABLE
PPOLL UNCONFIG
REMOTE
REQUEST
RESUME
SEND
PPOLL
TRIGGER

System Commands:

HELLO
TERM
IOCTL
TIME OUT

6.2 Command Description Format

Each command description is divided into several areas as follows:

6.2.1 Syntax

The syntax section of the command description describes the proper command syntax which must be sent to D500GPIB using the BASIC `PRINT#` command or its equivalent in other languages. The following conventions are used in the syntax descriptions:

No command may be more than 255 characters long. The `data` part of the `OUTPUT` command does not count in this length and so the `OUTPUT data` may be as long as necessary.

Items in capital letters, such as `ENTER` or `OUTPUT` must be used exactly as stated.

Items in lower case, such as `addr` or `count` represent parameters which must be substituted with an appropriate value.

Blank spaces in commands are generally ignored. Thus, LOCAL LOCK OUT is the same as LOCALLOCKOUT. Spaces are not ignored in three places: the data part of an OUTPUT command, within quoted strings in a SEND command, and after an apostrophe (') in a terminator specification (term).

The number sign character (#) and the semi-colon (;) must be present exactly as shown. A comma (,) represents an address separator. The oblique or slash character (/) may be used in its place as the address separator.

Items enclosed in square brackets ([item]) are optional. Multiple items enclosed in square brackets separated by vertical lines ([item1|item2|item3]) are optional, any one or none may be chosen. No more than one item may be selected.

Ellipses (...) within square brackets mean that the items in the brackets may be repeated as many times as desired. For example [,addr...] means that any number of address separator-address combinations may be used.

Braces, or curly brackets, ({item1|item2}) mean that exactly one of the enclosed items is required.

Combinations of brackets are possible. For example, {term[term] [EOI] |EOI} allows the choice of term, term EOI, term term, term term EOI, or just EOI, but does not allow the choice of "nothing."

Several of the commands require additional or optional parameters. These are further described with each command, but the more common ones are discussed below.

6.2.1.1 Bus Addressing

`pri-addr` A two-digit primary device address in the range of `00` to `30`.

`sec-addr` An optional two-digit secondary device address in the range of `00` to `31`.

`addr` An IEEE bus address. Bus addresses optionally include a secondary address. Thus they are of the form `pri-addr[sec-addr]` where `pri-addr` is a two-digit primary address in the range from `00` through `30` and `sec-addr` is a two-digit secondary address from `00` through `31`. Addresses must be given as two-digit numbers, e.g. `05` for address 5, and `1601` for primary address 16, secondary address 1

`[,addr...]` An optional list of bus addresses, each one preceded by an address separator; either a comma (`,`) or a slash (`/`).

No more than 15 bus addresses are allowed in any single command.

6.2.1.2 Character Count

`#count` The number of characters to be transferred. A pound sign (#) followed by an integer in the range of 1 to 65535 (2^{16}-1). A character count of zero is invalid.

6.2.1.3 ASCII Characters

`$char` A single character whose ASCII value is the number `char`, a decimal number in the range of 0 to 255 For example, $65 is the letter "A".

`CR` The carriage return character ($13).

`LF` The line feed character ($10).

`'X` Any (usually) printable character. The apostrophe is immediately followed, without any intervening spaces, by a single character which is taken to be the character specified.

6.2.1.4 ASCII Character Strings

`data`	An arbitrary string of characters. None of the special forms given above (`$char`, CR, LF, or 'X) are used. For example, CRLF as `data` is taken as the letters, "C", "R", "L", and "F", not as carriage return line feed.
`'data'`	An arbitrary string of characters enclosed in apostrophes.

6.2.1.5 Terminators

`term`	Any single character, specified as `CR`, `LF`, `'X`, or `$char` as described above (`{CR\|LF\|'X\|$char}`). Part of terminator sequence used to mark the end of lines of data and commands.
`[term]`	An optional term character. `term[term]` means that one or two terminators may be specified.
EOI	The IEEE bus End-Or-Identify signal. When asserted during the transfer of a character, EOI signals that that character is the last in the transfer. On input, EOI, if specified, causes the input to stop. On output, EOI causes the bus EOI signal to be asserted during transmission of the last character transferred.

6.2.2 Response

The response section of the command description describes the response that the user's program should read after sending the command. If a response is provided, it must be read. Errors will occur if it is not.

6.2.3 Bus States

This section describes the bus command and data transfers using IEEE bus mnemonics abbreviated as follows:

		DIO lines 8	7	6	5	4	3	2	1
ATN	Attention								
data	Data String								
DCL	Device Clear	x	0	0	1	0	1	0	0
GET	Group Execute Trigger	x	0	0	0	1	0	0	0
GTL	Go To Local	x	0	0	0	0	0	0	1
LAG	Listen Address Group	x	0	1	a	d	d	r	n
LLO	Local Lock Out	x	0	0	1	0	0	0	1
MLA	My Listen Address	x	0	1	a	d	d	r	n
MTA	My Talk Address	x	1	0	a	d	d	r	n
PPC	Parallel Poll Configure	x	0	0	0	0	1	0	1
PPD	Parallel Poll Disable	x	1	1	1	0	0	0	0
PPE	Parallel Poll Enable	x	1	1	0	S	P3	P2	P1
PPU	Parallel Poll Unconfigure	x	0	0	1	0	1	0	1
SDC	Selected Device Clear	x	0	0	0	0	1	0	0
SPD	Serial Poll Disable	x	0	0	1	1	0	0	1
SPE	Serial Poll Enable	x	0	0	1	1	0	0	0
SRQ	Service Request								
TAG	Talker Address Group	x	1	0	a	d	d	r	n
TCT	Take Control	x	0	0	0	1	0	0	1
term	Terminator								
UNL	Unlisten	x	0	1	1	1	1	1	1
UNT	Untalk	x	1	0	1	1	1	1	1

(x = "don't care")

If a command is preceded by an asterisk then that command is unasserted. For example, *REM states that the remote enable line is unasserted. Conversely, REM without the asterisk states that the line becomes asserted.

6.2.4 Examples

This section gives programming examples written in the BASIC language. For examples in other programming languages, refer to chapter 3.

6.3 The Commands

The commands, in alphabetical order, are described on the following pages.

ABORTIO

As the System Controller (SC), the ABORTIO command causes the Interface Clear (IFC) bus management line to be asserted for at least 500 microseconds. By asserting IFC, D500GPIB regains control of the bus even if one of the devices has locked it up during a data transfer. Asserting IFC also makes D500GPIB the Active Controller. If a Non System Controller was the Active Controller then it will be forced to relinquish control to D500GPIB. ABORT forces all IEEE bus device interfaces into a quiescent state.

SYNTAX	ABORTIO
RESPONSE	None
BUS STATES	IFC, *IFC
EXAMPLE	PRINT#1,"ABORTIO"

CLEAR

The CLEAR command causes the Device Clear (DCL) bus command to be issued by D500GPIB. If the optional addresses are included, the Selected Device Clear (SDC) command is issued to all specified devices. IEEE 488 bus devices which receive a Device Clear or Selected Device Clear command normally reset to their power-on state.

SYNTAX `CLEAR [addr[,addr...]]`

`addr` is a device address (primary with optional secondary).

`,` is the address separator, either a comma "," or a slash "/".

RESPONSE None

BUS STATES

ATN•DCL	(all devices)
ATN•UNL, MTA, LAG,SDC	(selected devices)

EXAMPLES

`PRINT #1,"CLEAR"`	issue a Device Clear to all devices
`PRINT #1,"CLEAR12,18"`	issue a Selected Device Clear to devices 12 and 18.

ENTER

The ENTER command reads data from the IEEE bus. If a device address (with optional secondary address) is specified, then that device will be addressed to talk. If no address is specified, then D500GPIB must already be configured to receive data, either as a result of an immediately preceding ENTER command, or as a result of a SEND command. If the character count, count, is specified, then exactly that number of characters will be read from the device. Otherwise, ENTER terminates reception on detection of the IEEE bus input terminator, which may be overridden by specifying the terminator in the ENTER command. The received terminator is then replaced with the EOL terminator before being returned to the user's program.

SYNTAX `ENTER[addr][#count|term[term][EOI]|[EOI]]`

`addr` is the IEEE bus device address.

`count` is the number of characters to ENTER.

`term` and `EOI` override the normal IEEE bus input terminator.

RESPONSE Device-dependent data. If `count` is specified, then exactly `count` characters will be returned. EOL is appended for convienience but need not be read back. Otherwise the response ends when the IEEE bus input terminator is detected and EOL is appended to the returned data.

BUS STATES ATN•UNL, MLA, TAG, *ATN, data..., ATN (With `addr`)

*ATN, data..., ATN (Without `addr`)

EXAMPLES

```
PRINT#1,"ENTER16"
INPUT#2,A$
```
Read data from device 16.

```
PRINT#1,"ENTER16"
LINE INPUT#2,A$
```
Read an entire line of data from device 16 even if it contains commas or other punctuation.

```
PRINT#1,"ENTER16 CR"
INPUT#2,A$
```
Read data from device 16 until CR is encountered.

```
PRINT#1,"ENTER16 $000"
INPUT#2,A$
```
Read data until a NULL is encountered.

```
PRINT#1,"ENTER16 LF EOI"
INPUT#2,A$
```
Read data until LF or EOI is detected.

```
PRINT#1,"ENTER0702"
INPUT#2,A$
```

Read data from device 7, secondary address 2.

```
PRINT#1,"ENTER12#5"
A$=INPUT$(5,#2)
PRINT#1,"ENTER#20"
A$=INPUT$(20,#2)
```

Read 5 bytes from device 12.
INPUT$ returns 5 bytes from file #2
Read 20 more bytes.

HELLO

The HELLO command is used to verify communication with D500GPIB, and to read the software revision number. When the command is sent, D500GPIB returns the following string:

SERIES 500/488 Rev n.xx (C) Copyright 1987 IOtech Inc

where n.xx is the software revision number.

SYNTAX HELLO

RESPONSE SERIES 500/488 Rev n.xx (C) Copyright 1987 IOtech Inc

BUS STATES None

EXAMPLE

```
PRINT#1,"HELLO"          Get the HELLO response
INPUT#2,A$
PRINT A$                 and display it.
```

IOCTL (BASIC statement)

IOCTL is a BASIC statement that can be used to unconditionally reset D500GPIB. When the message "BREAK" is sent to D500GPIB via the IOCTL DOS function (accessible via the IOCTL BASIC statement), D500GPIB stops any command currently executing, and prepares to accept a new command. This can be used even when D500GPIB is not expecting a command, but transferring data. Commands such as ABORT or RESET may then be used to reset the entire IEEE bus.

No IOCTL commands other than BREAK are supported by D500GPIB.

SYNTAX	`IOCTL#2,"BREAK"`	
RESPONSE	None	
BUS STATES	None	
EXAMPLE	`IOCTL#2,"BREAK"`	Send the IOCTL message "BREAK" to the D500GPIB.

IOCTL$ (BASIC function)

IOCTL$ is a BASIC function that can be used to determine the communication state of D500GPIB. IOCTL$ returns a character single ASCII character, either "0", "1", "2" , or "3". The meaning of these responses is as folows:

"0" A response of "0" indicates that D500GPIB is ready to receive a command. It has no data to read, nor is it expecting data for output to the IEEE bus.

"1" A response of "1" indicates that D500GPIB has a response ready to be read by the user's program. The program should read the response before sending a new command (except IOCTL "BREAK").

The IOCTL$ function has one primary use: in the Keyboard Controller Program (see Appendix A) it allows the program to know when D500GPIB has data available to read.

SYNTAX `A$=IOCTL$(#2)`

A$ is a string variable that is set to "0" if there is nothing to read, "1" if there is.

RESPONSE None

BUS STATES None

EXAMPLE
```
100 PRINT#1,"ENTER16"
110 A$=IOCTL$(#2)
120 IF A$="1" THEN PRINT INPUT$(1,#2); : GOTO 110
130 PRINT "NO INPUT READY"
```

LOCAL

In the System Controller mode, the LOCAL command without optional addresses causes D500GPIB to unassert the Remote Enable line. This causes devices on the bus to return to manual operation. As the Active Controller, with bus addresses specified, bus devices are placed in the local mode by the Go To Local (GTL) bus command. If addresses are specified, then the Remote Enable line is not unasserted.

SYNTAX LOCAL

BUS STATES *REM

SYNTAX LOCAL addr[,addr...]

addr is a bus device address.

BUS STATES ATN•UNL, MTA, LAG,GTL

EXAMPLES

```
PRINT#1,"LOCAL"          Unassert the Remote Enable Line
PRINT#1,"LOCAL12,16"     Send Go To Local to devices 12 and 16
```

LOCAL LOCKOUT

The LOCAL LOCKOUT command causes D500GPIB to issue a Local Lockout IEEE bus command. Bus devices that support this command are thereby inhibited from being controlled manually from their front panels.

SYNTAX {LOCAL LOCKOUT}

BUS STATES ATN•LLO

EXAMPLES

```
PRINT#1,"LOCAL LOCKOUT"
```

Send Local Lockout bus command.

OUTPUT

The OUTPUT command sends data to the IEEE bus. If device addresses (with optional secondary addresses) are specified then those devices will be addressed to listen. If no addresses are specified, then D500GPIB must already be configured to send data, either as a result of an immediately preceding OUTPUT command or as the result of a SEND command. If the character count, count, is specified then exactly that number of characters will be sent to the bus devices. Otherwise, OUTPUT terminates data transfer upon detection of the EOL output terminator from the user's program. The EOL output terminator is replaced with the bus output terminator before being sent to the bus devices.

SYNTAX `OUTPUT[addr[,addr...]][#count];data`

`addr` is a bus device address. Up to 15 addresses may be specified.

`count` is the number of characters to OUTPUT.

`data` is a string of characters to OUTPUT terminated by the EOL output terminator (unless `count` is specified in which case no terminator is needed).

RESPONSE None

BUS STATES

*ATN, data	(without addr)
ATN•MTA, UNL, LAG, *ATN, data	(with addr)

EXAMPLES

```
PRINT#1,"OUTPUT22;R0C0T1X"
```
Send "R0C0T1X" to device 22.

```
PRINT#1,"OUTPUT06,12;ABC"
PRINT#1,"OUTPUT;XYZ"
```
Send "ABC" to devices 6 and 12. And send them "XYZ".

```
PRINT#1,"OUTPUT0602;DEF"
```
Send "DEF" to device 6, secondary address 2.

```
PRINT#1,"OUTPUT06#26;abcdefghijklmnopqrstuvwxyz"
```
Send the 26 letters of the alphabet without terminators to device 6.

PPOLL

The Parallel Poll command, PPOLL, is used to request status information from many bus devices simultaneously. If a device requires service then it will respond to a Parallel Poll by asserting one of the eight IEEE bus data lines (DIO1 through DIO8, with DIO1 being the least significant). In this manner, up to eight devices may simultaneously be polled by the controller. More than one device can share any particular DIO line. In this case it is necessary to perform further Serial Polling to determine which device actually requires service.

Parallel polling is often used upon detection of a Service Request (SRQ), though it may also be performed periodically by the controller. In either case, PPOLL will respond with a number from 0 to 255 corresponding to the eight binary DIO lines.

Not every device supports parallel polling. Refer to the manufacturer's documentation for each bus device to determine if Parallel Poll capabilities are supported.

SYNTAX	PPOLL	
RESPONSE	Number in the range of 0 to 255	
BUS STATES	ATN•EOI,<parallel poll response>, *EOI	
EXAMPLE	PRINT#1"PPOLL"	Conduct a Parallel Poll
	INPUT#2,PPSTAT	Receive the PPOLL status

PPOLL CONFIG

PPOLL CONFIG (Parallel Poll Configure) configures the Parallel Poll response of a specified bus device. Not all devices support Parallel Polling and, among those that do, not all support software control of their Parallel Poll response. Some devices are configured by internal switches.

The Parallel Poll response is set by a four-bit binary number (S P2 P1 P0), `response`. The most significant bit of `response` is the Sense (S) bit. The Sense bit is used to determine when the device will assert its Parallel Poll response. Each bus device has an internal individual status (*ist*). The Parallel Poll response will be asserted when this *ist* equals the Sense bit value. *ist* is normally a logic "1" when the device requires attention, so the S bit should normally also be a logic "1". If the S bit is "0" then the device will assert its Parallel Poll response when its *ist* is a logic "0", i.e. it does not require attention. However, the meaning of *ist* can vary between devices, so refer to your IEEE bus device documentation.

The remaining 3 least significant bits of `response`, P2, P1, and P0, specify which DIO bus data line will be asserted by the device in response to a Parallel Poll. These bits form a binary number with a value from 0 through 7, specifying data lines DIO1 through DIO8, respectively.

SYNTAX `PPOLL CONFIG addr;response`

`addr` is a bus address.

`response` is the decimal equivalent of the four binary bits S, P2, P1, and P0 where S is then Sense bit, and P2, P1, and P0 assign the bus data line used for the response.

RESPONSE None

BUS STATES ATN•UNL, MTA, LAG, PPC, PPE

EXAMPLES

```
PRINT #1,"PPCONFIG23;13"
```

Configure device 23 to assert DIO6 when it desires service (*ist* = "1") and it is Parallel Polled (&H0D = 1101 binary; S = 1, P2P1P0 = 101 = 5 decimal = DIO6).

PPOLL DISABLE

PPOLL DISABLE disables the Parallel Poll response of selected bus devices.

SYNTAX PPOLL DISABLE addr[,addr...]

addr is a bus device address

RESPONSE None

BUS STATES ATN•UNL, MTA, LAG, PPC, PPD

EXAMPLE
```
PRINT#1,"PPOLL DISABLE18,06,13"
```
Disable Parallel Poll response of devices 18, 6, and 13.

PPOLL UNCONFIG

PPOLL UNCONFIG (Parallel Poll Unconfigure) disables the Parallel Poll response of all bus devices.

SYNTAX	{PPOLL UNCONFIG}	
RESPONSE	None	
BUS STATES	ATN•PPU	
EXAMPLE	`PRINT #1,"PPOLL UNCONFIG"`	Disable the Parallel Poll response of all bus devices.

REMOTE

The REMOTE command asserts the Remote Enable (REN) bus management line. If the optional bus addresses are specified, then REMOTE also address those devices to listen, placing them in the Remote state.

SYNTAX REMOTE

RESPONSE None

BUS STATES REN

SYNTAX REMOTE addr[,addr...]

addr is a bus device address

RESPONSE None

BUS STATES REN, ATN•UNL, MTA, LAG

EXAMPLES

```
PRINT #1,"REMOTE"
```
Assert Remote Enable

```
PRINT #1,"REMOTE16,28"
```
Assert Remote Enable and address devices 16 and 28 to listen.

RESUME

The RESUME command unasserts the Attention (ATN) bus signal . Attention is normally kept asserted by D500GPIB, but it must be unasserted to allow transfers to take place between two Peripheral devices. In this case, D500GPIB SENDs the appropriate talk and listen addresses, and the must unassert Attention with the RESUME command.

SYNTAX	RESUME	
RESPONSE	None	
BUS STATES	*ATN	
EXAMPLE	PRINT#1,"RESUME"	Unassert ATTENTION line.

SEND

The SEND command provides byte-by-byte control of data and control transfers on the bus and gives greater flexibility than the other commands. The command can specify exactly which operations will be executed by D500GPIB.

The following are available within the SEND command:

UNT	Send the multiline Untalk command. ATN is asserted.
UNL	Send the multiline Unlisten command. ATN is asserted.
MTA	Send My (D500GPIB) Talk Address. ATN is asserted.
MLA	Send My (D500GPIB) Listen Address. ATN is asserted.
TALKaddr	Send Talk Address addr device (TAG). ATN is asserted.
LISTENaddr[,addr...]	Send Listen Addresses (LAG). ATN is asserted.
DATA'data'	Send character string data with ATN unasserted.
DATA char[,char...]	Send characters with the given numeric ASCII values with ATN unasserted.
CMD'data'	Send character string data with ATN asserted.
CMD char[,char...]	Send characters with the given numeric ASCII values with ATN asserted.

The DATA and, CMD subcommands send data bytes or characters over the bus. The characters to be sent are specified either as a quoted string ('data') or as individual ASCII values (char[,char...]). For example, DATA'R0X' sends the characters R, 0, and X to the active listeners, and DATA 13,10 sends carriage-return and line-feed.

The CMD subcommand sends the data bytes with Attention (ATN) is asserted. This tells the bus devices that the characters are to be interpreted as IEEE bus commands, rather than as data. EOI is not asserted during CMD transfers. For example CMD 63 is the same as Unlisten (UNL).

Note that the maximum length of the SEND command, including any subcommands, is 255 characters. If large amounts of data must be transferred using the SEND command, then multiple SEND commands must be used so that they are each less than 255 characters long. For example

```
PRINT#1,"SEND; UNT UNL MTA LISTEN 16 DATA 1,2,3,4,5,6"
```

is equivalent to

```
PRINT#1,"SEND; UNT UNL MTA LISTEN 16"
PRINT#1,"SEND; DATA 1,2,3"
PRINT#1,"SEND; DATA 4,5,6"
```

In this way, a long SEND command can be broken up into shorter commands.

SYNTAX `SEND; subcommand[subcommand...]`

RESPONSE None

BUS STATES user defined

EXAMPLES
```
PRINT#1,"SEND; MTA UNL LISTEN16 DATA 'T1S0R2X'"
```
is the same as
```
PRINT#1,"OUTPUT16;T1S0R2X"
```

```
PRINT#1,"SEND; CMD 128,0,10 DATA 156,35 EOI 'ABC'"
```

sends the following byte sequence:

Data	ATN	EOI
10000000	ATN	*EOI
00000000	ATN	*EOI
00001010	ATN	*EOI
10011100	*ATN	*EOI
00100011	*ATN	*EOI
01000001	*ATN	*EOI
01000010	*ATN	*EOI
01000011	*ATN	EOI

SPOLL

The SPOLL command performs a Serial Poll of the bus device specified and responds with number from 0 to 255 representing the decimal equivalent of the eight-bit device response. If *rsv* (DIO7, decimal value 64) is set, then that device is signaling that it requires service. The meanings of the other bits are device-specific.

Serial Polls are normally performed in response to assertion of the Service Request (SRQ) bus signal by some bus device.

In Active Controller mode, with no bus address specified, the SPOLL command returns the in SRQ line status. If the SRQ status is set, which indicates that the SRQ line is asserted, then D500GPIB will return a "64", if it is not set, indicating that SRQ is not asserted, then D500GPIB will return a "0".

SYNTAX `SPOLL [addr]`

`addr` is the bus device to be Serial Polled

RESPONSE
0 or 64 (without `addr`)
Number in the range from 0 to 255 (with `addr`)

BUS STATES ATN•UNL, MLA, TAG, SPE, *ATN, data, ATN•SPD, UNT

EXAMPLES

```
PRINT#1,"SPOLL 16"          Serial Poll device 16
INPUT#2,SPSTAT              Receive the Spoll status
IF SPSTAT AND 64 THEN...    Test rsv...

PRINT#1,"SPOLL"             Check the SRQ status
INPUT#2,SRQ
IF SRQ<>0 THEN...           If SRQ is asserted then ...
```

TIME OUT

The TIME OUT command sets the number of seconds that D500GPIB will wait for a transfer before declaring a time out error.

Time out checking may be suppressed by specifying time out after zero seconds. The default time out is 10 seconds.

SYNTAX `TIME OUT; n`

n is the number of seconds to allow in the range of 0 to 65535. If n is zero, ignore time outs.

RESPONSE None

BUS STATES None

EXAMPLES

`PRINT#1,"TIME OUT;10"`	Reset to default (10 seconds).
`PRINT#1,"TIME OUT;3600"`	Wait an hour before time out error.
`PRINT#1,"TIME OUT;0"`	Ignore time outs.

TRIGGER

The TRIGGER command issues a Group Execute Trigger (GET) bus command to the specified devices. If no addresses are specified, then the GET will only affect those devices that are already in the listen state as a result of a previous OUTPUT or SEND command.

SYNTAX `TRIGGER [addr[,addr...]]`

`addr` is a bus device address to be triggered.

RESPONSE None

BUS STATES ATN•GET (without `addr`)
ATN•UNL, MTA, LAG, GET (with `addr`)

EXAMPLES

```
PRINT#1,"TRIGGER02,04,16"
```
Issue Group Execute Trigger to devices 2, 4, and 16.

```
PRINT#1,"TRIGGER"
```
Trigger all current listeners.

IEEE 488 Primer

7.1 History

The IEEE 488 bus is an instrumentation communication bus adopted by the Institute of Electrical and Electronic Engineers in 1975 and revised in 1978. The Personal488 conforms to this most recent revision designated IEEE 488-1978.

Prior to the adoption of this standard, most instrumentation manufacturers offered their own versions of computer interfaces. This placed the burden of system hardware design on the end user. If his application required the products of several different manufacturers, then he might need to design several different hardware and software interfaces. The popularity of the IEEE 488 interface (sometimes called the General Purpose Interface Bus or GPIB) is due to the total specification of the electrical and mechanical interface as well as the data transfer and control protocols. The use of the IEEE 488 standard has moved the responsibility of the user from design of the interface to design of the high level software that is specific to the measurement application.

7.2 General Structure

The main purpose of the GPIB is to transfer information between two or more devices. A device can either be an instrument or a computer. Before any information transfer can take place, it is first necessary to specify which will do the talking (send data) and which devices will be allowed to listen (receive data). The decision of who will talk and who will listen usually falls on the System Controller which is, at power on, the Active Controller.

The System Controller is similar to a committee chairman. On a well run committee, only one person may speak at a time and the chairman is responsible for recognizing members and allowing them to have their say. On the bus, the device which is recognized to speak is the Active Talker. There can only be one Talker at a time if the information transferred is to be clearly understood by all. The act of "giving the floor" to that device is called Addressing to Talk. If the committee chairman can not attend the meeting, or if other matters require his attention, he can appoint an acting chairman to take control of the proceedings. For the GPIB, this device becomes the Active Controller.

At a committee meeting, everyone present usually listens. This is not the case with the GPIB. The Active Controller selects which devices will listen and commands all other devices to ignore what is being transmitted. A device is instructed to listen by being Addressed to Listen. This device is then referred to as an Active Listener. Devices which are to ignore the data message are instructed to Unlisten.

The reason some devices are instructed to Unlisten is quite simple. Suppose a college instructor is presenting the day's lesson. Each student is told to raise their hand if the instructor has exceeded their ability to keep up while taking notes. If a hand is raised, the instructor stops his discussion to allow the slower students the time to catch up. In this way, the instructor is certain that each and every student receives all the information he is trying to present. Since there are a lot of students in the classroom, this exchange of information can be very slow. In fact, the rate of information transfer is no faster than the rate at which the slowest note-taker can keep up. The instructor, though, may have a message for one particular student. The instructor tells the rest of the class to ignore this message (Unlisten) and tells it to that one student at a rate which he can understand. This information transfer can then happen much quicker, because it need not wait for the

slowest student.

The GPIB transfers information in a similar way. This method of data transfer is called handshaking.

For data transfer on the IEEE 488, the Active Controller must...

a) Unlisten all devices to protect against eavesdroppers.
b) Designate who will talk by addressing a device to talk.
c) Designate all the devices who are to listen by addressing those devices to listen.
d) Indicate to all devices that the data transfer can take place.

7.3 Send It To My Address

In the previous discussion, the terms Addressed to Talk and Addressed to Listen were used. These terms require some clarification.

The IEEE 488 standard permits up to 15 devices to be configured within one system. Each of these devices must have a unique address to avoid confusion. In a similar fashion, every building in town has a unique address to prevent one home from receiving another home's mail. Exactly how each device's address is set is specific to the product's manufacturer. Some are set by DIP switches in hardware, others by software. Consult the manufacturer's instructions to determine how to set the address.

Addresses are sent with universal (multiline) commands from the Active Controller. These commands include My Listen Address (MLA), My Talk Address (MTA), Talk Address Group (TAG), and Listen Address Group (LAG).

7.4 Bus Management Lines

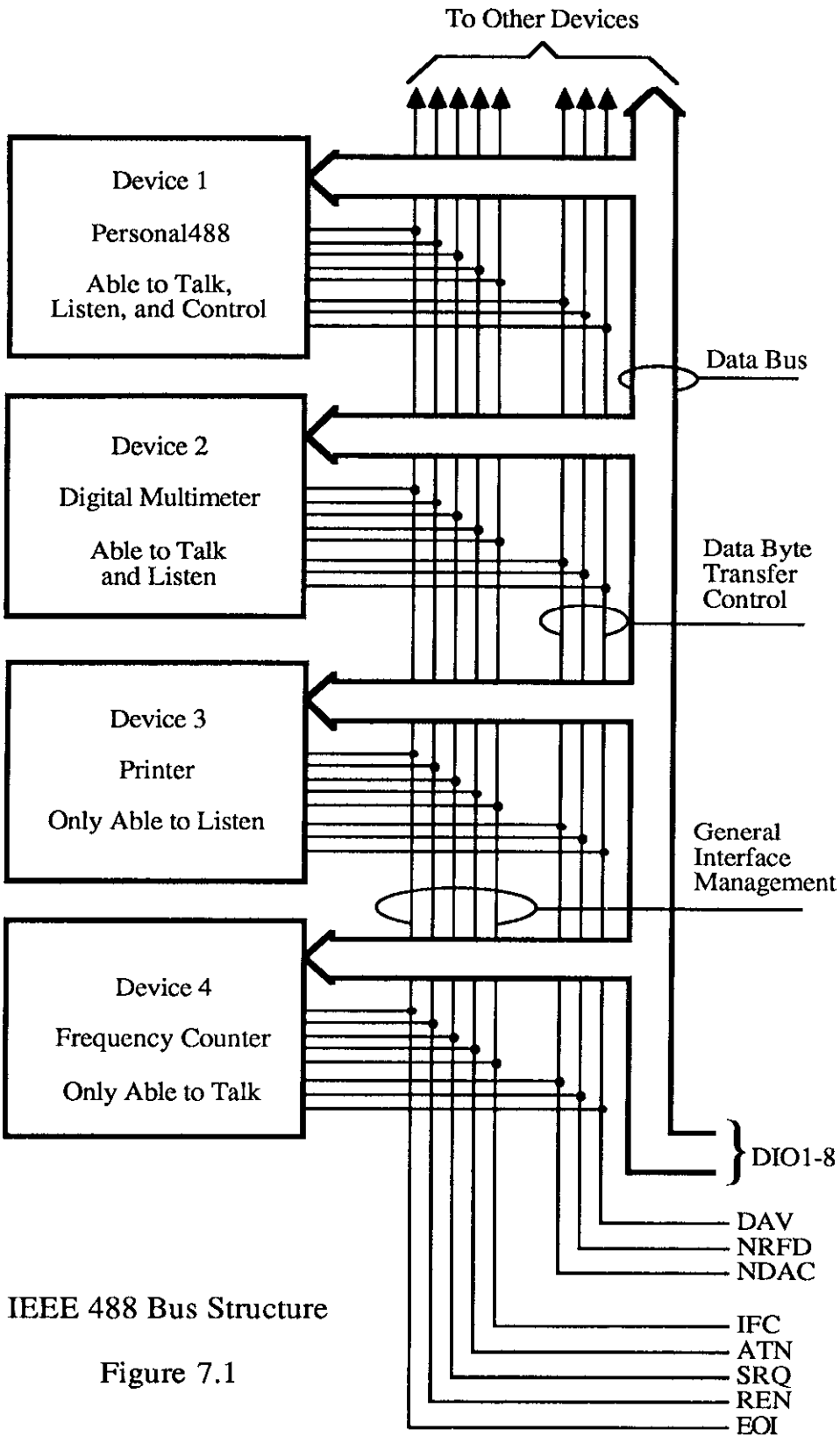
Five hardware lines on the GPIB are used for bus management. Signals on these lines are often referred to as uniline (single line) commands. The signals are active low, i.e. a low voltage represents a logic "1" (asserted), and a high voltage represents a logic "0" (unasserted).

7.4.1 Attention (ATN)

ATN is one of the most important lines for bus management. If Attention is asserted, then the information contained on the data lines is to be interpreted as a multiline command. If it is not, then that information is to be interpreted as data for the Active Listeners. The Active Controller is the only bus device that has control of this line.

7.4.2 Interface Clear (IFC)

The IFC line is used only by the System Controller. It is used to place all bus devices in a known state. Although device configurations vary, the IFC command usually places the devices in the Talk and Listen Idle states (neither Active Talker nor Active Listener).



IEEE 488 Bus Structure

Figure 7.1

7.4.3 Remote Enable (REN)

When the System Controller sends the REN command, bus devices will respond to remote operation. Generally, the REN command should be issued before any bus programming is attempted. Only the System Controller has control of the Remote Enable line.

7.4.4 End or Identify (EOI)

The EOI line is used to signal the last byte of a multibyte data transfer. The device that is sending the data asserts EOI during the transfer of the last data byte. The EOI signal is not always necessary as the end of the data may be indicated by some special character such as carriage return.

The Active Controller also uses EOI to perform a Parallel Poll by simultaneously asserting EOI and ATN.

7.4.5 Service Request (SRQ)

When a device desires the immediate attention of the Active Controller it asserts SRQ. It is then the Controller's responsibility to determine which device requested service. This is accomplished with a Serial Poll or a Parallel Poll.

7.5 Handshake Lines

The GPIB uses three handshake lines in an "I'm ready - Here's the data - I've got it" sequence. This handshake protocol assures reliable data transfer, at the rate determined by the slowest Listener. One line is controlled by the Talker, while the other two are shared by all Active Listeners. The handshake lines, like the other IEEE 488 lines, are active low.

7.5.1 Data Valid (DAV)

The DAV line is controlled by the Talker. The Talker outputs data on the bus and waits until NRFD is unasserted which indicates that all Addressed Listeners are ready to accept the information. At the same time, the Talker also verifies that NDAC is asserted which indicates that all Listeners have accepted the previous data byte transferred. If these conditions are not met, the Talker must wait until the NRFD and the NDAC line are in the proper state. Once in the proper state, the Talker asserts DAV to indicate that the data on the bus is valid.

7.5.2 Not Ready for Data (NRFD)

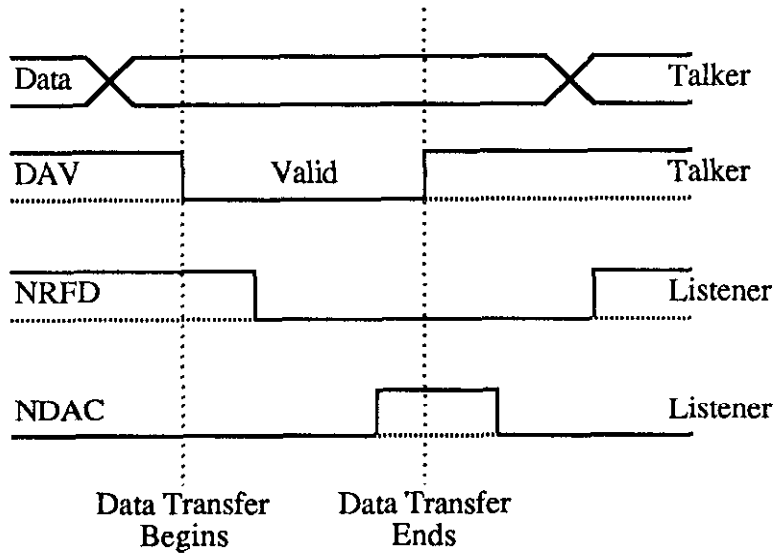
This line is used by the Listeners to inform the Talker when they are ready to accept new data. The Talker must wait for each Listener to unassert this line which it will do when it is ready for more data. This assures that all devices that are to accept the information are ready to receive it.

7.5.3 Not Data Accepted (NDAC)

The NDAC line is also controlled by the Listeners. This line indicates to the Talker that each device addressed to listen has accepted the information. Each device releases NDAC at its own rate, but the NDAC will not go high until the slowest Listener has accepted the data byte.

7.6 Data Lines

The GPIB provides eight data lines for a bit parallel/byte serial data transfer. These eight data lines use the convention of DIO1 through DIO8 instead of the binary designation of D0 to D7. The data lines are bidirectional and are active low.



IEEE Bus Handshaking

7.7 Multiline Commands

Multiline (bus) commands are sent by the Active Controller over the data bus with ATN asserted. These commands include addressing commands for talk, listen, Untalk and Unlisten.

7.7.1 Go To Local (GTL)

This command allows the selected devices to be manually controlled. ($01)

7.7.2 Listen Address Group (LAG)

There are 31 (0 to 30) listen addresses associated with this group. The 3 most significant bits of the data bus are set to 011 while the 5 least significant bits are the address of the device being told to listen.

7.7.3 Unlisten (UNL)

This command tells all bus devices to Unlisten. The same as Unaddressedto Listen. ($3F)

7.7.4 Talk Address Group (TAG)

There are 31 (0 to 30) talk addresses associated with this group. The 3 most significant bits of the data bus are set to 101 while the 5 least significant bits are the address of the device being told to talk.

7.7.5 Untalk (UNT)

This command tells bus devices to Untalk. The same as Unaddressed to Talk. ($5F)

7.7.6 Local Lockout (LLO)

Issuing the LLO command prevents manual control of the instrument's functions. ($11)

7.7.7 Device Clear (DCL)

This command causes all bus devices to be initialized to a predefined or power up state. ($14)

7.7.8 Selected Device Clear (SDC)

This causes a single device to be initialized to a predefined or power up state. ($04)

7.7.9 Serial Poll Disable (SPD)

The SPD command disables all devices from sending their Serial Poll status byte. ($19)

7.7.10 Serial Poll Enable (SPE)

A device which is Addressed to Talk will output its Serial Poll status byte after SPE is sent and ATN is unasserted. ($18)

7.7.11 Group Execute Trigger (GET)

This command usually signals a group of devices to begin executing a triggered action. This allows actions of different devices to begin simultaneously. ($08)

7.7.12 Take Control (TCT)

This command passes bus control responsibilities from the current Controller to another device which has the ability to control. ($09)

7.7.13 Secondary Command Group (SCG)

These are any one of the 32 possible commands (0 to 31) in this group. They must immediately follow a talk or listen address. ($60 to $7F)

7.7.14 Parallel Poll Configure (PPC)

This configures devices capable of performing a Parallel Poll as to which data bit they are to assert in response to a Parallel Poll. ($05)

7.7.15 Parallel Poll Unconfigure (PPU)

This disables all devices from responding to a Parallel Poll. ($15)

7.8 More on Service Requests

Most of the commands covered, both uniline and multiline, are the responsibility of the Active Controller to send and the bus devices to recognize. Most of these happen routinely by the interface and are totally transparent to the system programmer. Other commands are used directly by the user to provide optimum system control. Of the uniline commands, SRQ is very important to the test system and the software designer has easy access to this line by most devices. Service Request is the method by which a bus device can signal to the Controller that an event has occurred. It is similar to an interrupt in a microprocessor based system.

Most intelligent bus peripherals have the ability to assert SRQ. A DMM might assert it when its measurement is complete, if its input is overloaded or for any of an assortment of reasons. A power supply might SRQ if its output has current limited. This is a powerful bus feature that removes the burden from the System Controller to periodically inquire, "Are you done yet?". Instead, the Controller says, "Do what I told you to do and let me know when you're done" or "Tell me when something is wrong."

Since SRQ is a single line command, there is no way for the Controller to determine which device requested the service without additional information. This information is provided by the multiline commands for Serial Poll and Parallel Poll.

7.8.1 Serial Poll

Suppose the Controller receives a service request. For this example, let's assume there are several devices which could assert SRQ. The Controller issues an SPE (Serial Poll enable) command to each device sequentially. If any device responds with DIO7 asserted it indicates to the Controller that it was the device that asserted SRQ. Often times the other bits will indicate why the device wanted service. This Serial Polling sequence, and any resulting action, is under control of the software designer.

7.8.2 Parallel Poll

The Parallel Poll is another way the Controller can determine which device requested service. It provides the who but not necessarily the why. When bus devices are configured for Parallel Poll, they are assigned one bit on the data bus for their response. By using the Status bit, the logic level of the response can be programmed to allow logical OR/AND conditions on one data line by more than one device. When SRQ is asserted, the Controller (under user's software) conducts a Parallel Poll. The Controller must then analyze the eight bits of data received to determine the source of the request. Once the source is determined, a Serial Poll might be used to determine the why.

Of the two polling types, the Serial Poll is the most popular due to its ability to determine the who and why. In addition, most devices support Serial Poll only.

Keyboard Controller Program

The Keyboard Controller Program is a simple BASIC program that accepts commands from the keyboard and sends them to D500GPIB. It then displays any responses from D500GPIB on the screen. The Keyboard Controller Program is the most convenient method of exercising D500GPIB and becoming familiar with the commands and their actions. It also demonstrates the use of ON ERROR to trap I/O errors. This program is provided on the D500GPIB disk as "TERMGPIB.BAS"

```
10 ' D500GPIB Keyboard Controller Program
20 '
30 ' For use with the Keithley DAC D500GPIB
40 ' driver and GPIB module
50 '
60 OPEN "\DEV\GPIBOUT" FOR OUTPUT AS #1
70 IOCTL#1,"BREAK"
80 OPEN "\DEV\GPIBIN" FOR INPUT AS #2
90 '
100 ON ERROR GOTO 210
110 '
120 LINE INPUT "CMD> ",CMD$
130 PRINT#1,CMD$
140 '
150 IF IOCTL$(2)<>"1" THEN 120
160 PRINT INPUT$(1,2);
170 GOTO 150
180 '
190 ' Error Handler
200 '
210 IOCTL#1,"BREAK"
220 PRINT"-------- error --------> "
230 RESUME NEXT
```

$00 0 NUL	$10 16 DLE	$20 32 SP 00	$30 48 0 16	$40 64 @ 00	$50 80 P 16	$60 96 ` SCG	$70 112 p SCG
$01 1 SOH GTL	$11 17 DC1 LLO	$21 33 ! 01	$31 49 1 17	$41 65 A 01	$51 81 Q 17	$61 97 a SCG	$71 113 q SCG
$02 2 STX	$12 18 DC2	$22 34 " 02	$32 50 2 18	$42 66 B 02	$52 82 R 18	$62 98 b SCG	$72 114 r SCG
$03 3 ETX	$13 19 DC3	$23 35 # 03	$33 51 3 19	$43 67 C 03	$53 83 S 19	$63 99 c SCG	$73 115 s SCG
$04 4 EOT SDC	$14 20 DC4 DCL	$24 36 $ 04	$34 52 4 20	$44 68 D 04	$54 84 T 20	$64 100 d SCG	$74 116 t SCG
$05 5 ENQ PPC	$15 21 NAK PPU	$25 37 % 05	$35 53 5 21	$45 69 E 05	$55 85 U 21	$65 101 e SCG	$75 117 u SCG
$06 6 ACK	$16 22 SYN	$26 38 & 06	$36 54 6 22	$46 70 F 06	$56 86 V 22	$66 102 f SCG	$76 118 v SCG
$07 7 BEL	$17 23 ETB	$27 39 ' 07	$37 55 7 23	$47 71 G 07	$57 87 W 23	$67 103 g SCG	$77 119 w SCG
$08 8 BS GET	$18 24 CAN SPE	$28 40 (08	$38 56 8 24	$48 72 H 08	$58 88 X 24	$68 104 h SCG	$78 120 x SCG
$09 9 HT TCT	$19 25 EM SPD	$29 41) 09	$39 57 9 25	$49 73 I 09	$59 89 Y 25	$69 105 i SCG	$79 121 y SCG
$0A 10 LF	$1A 26 SUB	$2A 42 * 10	$3A 58 : 26	$4A 74 J 10	$5A 90 Z 26	$6A 106 j SCG	$7A 122 z SCG
$0B 11 VT	$1B 27 ESC	$2B 43 + 11	$3B 59 ; 27	$4B 75 K 11	$5B 91 [27	$6B 107 k SCG	$7B 123 { SCG
$0C 12 FF	$1C 28 FS	$2C 44 , 12	$3C 60 < 28	$4C 76 L 12	$5C 92 \ 28	$6C 108 l SCG	$7C 124 | SCG
$0D 13 CR	$1D 29 GS	$2D 45 - 13	$3D 61 = 29	$4D 77 M 13	$5D 93] 29	$6D 109 m SCG	$7D 125 } SCG
$0E 14 SO	$1E 30 RS	$2E 46 . 14	$3E 62 > 30	$4E 78 N 14	$5E 94 ^ 30	$6E 110 n SCG	$7E 126 ~ SCG
$0F 15 SI	$1F 31 US	$2F 47 / 15	$3F 63 ? UNL	$4F 79 O 15	$5F 95 _ UNT	$6F 111 o SCG	$7F 127 DEL SCG
ACG	UCG	LAG		TAG		SCG	

ACG = Addressed Command Group
UCG = Universal Command Group
LAG = Listen Address Group
TAG = Talk Address Group
SCG = Secondary Command Group

Command Summary

ABORTIO

```
ABORTIO
```

Send IFC Stops bus activity.

CLEAR

```
CLEAR [addr[,addr...]]
```

Clear all or selected devices.

ENTER

```
ENTER [addr][;][#count|term[term][EOI]|[EOI]]
```

Read data from bus device.

HELLO

```
HELLO
```

Read the Personal488 revision identification.

IOCTL

```
IOCTL#2,"BREAK"
```

BASIC statement. Force Personal488 to accept commands from the user's program.

IOCTL$

```
A$=IOCTL$(#2)
```

BASIC statement. Test if there is any response to be read from Personal488 (A$="1"), A$="0" if not, "2" if waiting for `data`, "3" if waiting for the remainder of a command.

LOCAL

```
LOCAL
```

Unassert Remote Enable line (SC) or send Go To Local commands (CA)

LOCAL LOCKOUT

```
LOCAL LOCKOUT
```

Prevent bus devices from acting upon manual (front-panel) control.

OUTPUT

```
OUTPUT [addr[,addr...]][#count];data
```

Send `data` to bus device(s).

POLL	PPOLL Read the Parallel Poll response from all bus devices.
PPOLL CONFIG	PPOLL CONFIG addr;response or Set the Parallel Poll response of a bus device. response is the decimal equivalent of four bits: S P2 P1 P0.
PPOLL DISABLE	PPOLL DISABLE addr[,addr...] or Prevent bus devices from responding to Parallel Poll.
PPOLL UNCONFIG	PPOLL UNCONFIG Prevent all bus devices from responding to Parallel Poll.
REMOTE	REMOTE [addr[,addr...]] Assert the Remote Enable line. Optionally address devices to listen, putting them in the Remote state.
RESUME	RESUME Unassert Attention. Used to allow Peripheral-to-Peripheral transfers.
SEND	SEND; subcommand[subcommand...] Send low-level bus sequences. Subcommands are: UNT, UNL, MTA, MLA, TALK addr, LISTEN addr[,addr...], DATA 'data', DATA char[,char...], EOI 'data', EOI char[,char...], CMD 'data', and CMD char[,char...].
SPOLL	SPOLL [addr] Read device's Serial Poll response. Get internal SRQ state (CA), or Serial Poll status (*CA).

TERM

```
TERM; [term[term]][EOI]
```

Set the terminators for IEEE bus transfers.

TIME OUT

```
TIME OUT; n
```

Set the number of seconds (1 to 65535) to wait for a bus transfer before a Time Out error will be declared. Checking for Time Out is suppressed by specifying 0 seconds.

TRIGGER

```
TRIGGER [addr[,addr...]]
```

Trigger bus devices by sending optional listen addresses followed by Group Execute Trigger.

KEITHLEY DAC

Data Acquisition and Control Division Keithley Instruments, Inc. • 28775 Aurora Road • Cleveland, Ohio 44139 • (216) 248-0400 • Fax: 248-6168

WEST GERMANY:	**Keithley Instruments GmbH** • Heiglhofstr. 5 • Munchen 70 • 089-71002-0 • Telex: 52-12160 • Fax: 089-7100259
GREAT BRITAIN:	**Keithley Instruments, Ltd.** • 1 Boulton Road • Reading, Berkshire RG 2 ONL • 0734-861287 • Telex: 847 047 • Fax: 0734-863665
FRANCE:	**Keithley Instruments SARL** • 3 Allee des Garays • B.P. 60 • 91124 Palaiseau • Z.I. • 1-6-0115 155 • Telex: 600 933 • Fax: 1-6-0117726
NETHERLANDS:	**Keithley Instruments BV**•Avelingen West 49 • 4202 MS Gorinchem • P.O. Box 559 • 4200 AN Gorinchem • 01830-35333 • Telex: 24 684 • Fax: 01830-30821
SWITZERLAND:	**Keithley Instruments SA** • Kriesbachstr. 4 • 8600 Dubendorf • 01-821-9444 • Telex: 828 472 • Fax: 0222-315366
AUSTRIA:	**Keithley Instruments GesmbH** • Rosenhugelstrasse 12 • A-1120 Vienna • (0222) 84 65 48 • Telex: 131677 • Fax: (0222) 84 35 97
ITALY:	**Keithley Instruments SRL** • Viale S. Gimignano 4/A • 20146 Milano • 02-4120360 or 02-4156540 • Fax: 02-4121249