

**ICS
ELECTRONICS**

division of Systems West Inc.

MODEL 488-PC2

IEEE 488.2 Bus Controller

User's Manual

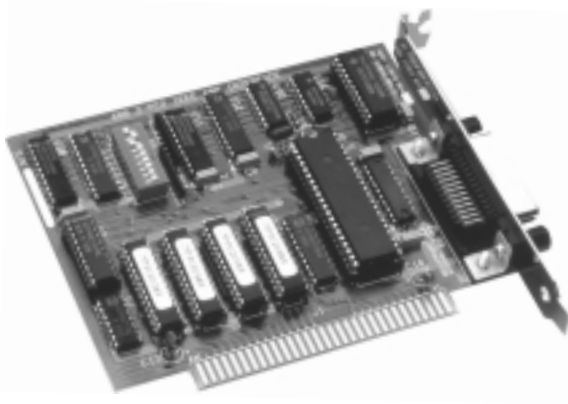


IEEE-488.2

Model 488-PC2

IEEE 488.2 Bus Controller

User's Manual



7034 Commerce Circle, Pleasanton, CA 94588
(925) 416-1000, FAX (925) 416-0105
WEB <http://www.icselect.com>

Publication Number 123077
July 2000 Edition Rev 6

LIMITED WARRANTY

Within 60 months of delivery, ICS Electronics will repair or replace this product, at our option, if any part is found to be defective in materials or workmanship (labor is included). Return this product to ICS Electronics, or other designated repair station, freight prepaid, for prompt repair or replacement. Contact ICS for a return material authorization (RMA) number prior to returning the product for repair. Any software delivered with this product is only warranted for 90 days from the date of delivery.

CERTIFICATION

ICS Electronics certifies that this instrument was carefully inspected and tested at the factory prior to shipment and was found to meet all requirements of the specification under which it was furnished.

EMI/RFI WARNING

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause interference to radio communications. The 488-PC2 Card has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of the FCC Rules and to comply with the EEC Standards EN 55022 and EN 50082-1, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.



Certificate of Compliance reproduced in Figure 2-2

TRADEMARKS

The following trademarks referred to in this manual are the property of the following companies:

HP and HPIB are registered trademarks of Hewlett-Packard Corp., Palo Alto, CA

IBM and IBM PC are registered trademarks of International Business Machines Corporation, Boca Raton, FL

ICS is a registered trademark of ICS Electronics, Systems West, Inc, Milpitas, CA

APPLICABILITY and CHANGES

This manual applies to the 488-PC2 Revision 1 Card and Windows Drivers Version 5.0 or later.

Contents

Getting Started

Shipment verification, Software License, Installation, Keyboard Controller Programs and Testing the Hardware.

1

General Information

Controller Card Descriptions, Software Capabilities, Supported Languages , Hardware Specifications, Approvals and Accessories.

2

488.2 Drivers

Driver Description, WIN and DOS Components, Command Set Quick Reference List

3

GPIB Programming

GPIB Programming Concepts, Device Addressing, Command Conventions, Basic Programming, IEEE-488.2 Programming, Advanced Programming, Visual Basic and C Program Examples.

4

DOS Language References

Installation Instructions, Supplied Files, Unique Program Instructions and Using Interrupts for: Interpreted BASIC, Quick Basic, Pascal and C/C++ Languages.

5

488-PC2 Command References

Command conventions, Constants, Errors and Command definitions

6

Appendix

A1 Background information on IEEE 488.1 Bus, IEEE 488.2 Status Structure, Protocols, Common Commands and SCPI Commands
A-2 Trouble shooting, Testing and Repair.

A

Index

I

Getting Started

1.1 INTRODUCTION

This section provides you with all the information you need to get started with the Model 488-PC2 IEEE 488.2 Controller Card. This section covers shipment verification, compatibility with earlier products, switch setting and installation, software license, backing up your software and the use of ICS's interactive keyboard command program. Follow the instructions in this section to install the card in your computer.

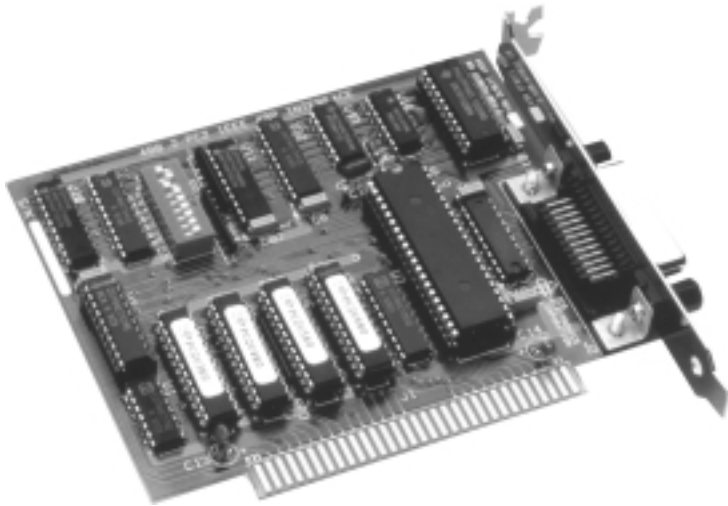


Figure 1-1 488-PC2 GPIB Controller Card

1.2 488-PC2 SHIPMENT VERIFICATION

Take a moment to verify you have everything you need. If you ordered the 488-PC2 Card and Software we should have sent you:

- (1) Model 488-PC2 IEEE 488.2 Controller Card, Rev 1
- (1) 488.2 DOS Driver Disk
- (1) 488.2 Windows Driver Disk
- (1) Model 488-PC2 User's Manual

The Driver and Utility programs may be on 3.5 inch disks or on a CD ROM disk. If anything is missing or damaged, save the shipping carton and contact ICS immediately. ICS will arrange for a replacement shipment without waiting for the shipper to process any claim.

1.3 COMPATIBILITY WITH EARLIER PRODUCTS

The 488-PC2 Revision 1 Card is compatible with the Revision 0 Card and may be used with the earlier drivers in any PC/XT or PC/AT computer.

The 488.2 Driver shipped with the 488-PC2 Card supports DOS programs written in Interpretive BASIC, Quick Basic, Visual Basic for DOS, Microsoft Professional Basic 7.1, Pascal, C, Turbo C, or C++. The 488.2 Driver's Dynamic Linked Libraries provide 16 and 32-bit support for C, C++, Visual Basic for Windows or any other programming language that follows the Microsoft Windows calling conventions. They may be used with most existing programs by changing the include file and recompiling the program. The 488.2 Drivers are only compatible with the Revision 1 or later 488-PC2 Cards.

Any programs written with earlier software drivers will have to be checked for command syntax before linking them to the new 488.2 Driver. Quick Basic programs will have to modify the ieEnterB and ieOutputB commands for the new command syntax.

1.4 ICS SOFTWARE LICENSE AGREEMENT

Note - Please carefully read this License agreement prior to opening the media envelope or using the software. By opening the media envelope and/or using the software, the customer agrees to all provisions of this license. If you do not agree with the license, you may return this product for a full refund.

1

1.4.1 License

In exchange for payment of his invoice, ICS Electronics (ICS) grants the customer a license in the software subject to the following conditions.

Customer may use the software with any ICS GPIB Controller product and may not reverse engineer or reverse-compile the software. Customer agrees the software is copyrighted and may only make archival copies of it. Customer shall not sublicense or distribute copies of the software without the written permission of ICS. A transfer or sale of the software to a third party is permitted, if the third party agrees to this license and the original purchaser ceases use of the software.

Customer may use ICS's software to make executable programs and distribute them freely without permission of ICS.

ICS may terminate this license and seek damages if the customer fails to comply with the license after being notified in writing to cure the failure. Customer agrees that the software does not include updates and ICS is not responsible for any damage to the customer's computer or other equipment.

1.4.2 Warranty

ICS warrants for a 90 day period, that the software will execute its instructions when properly installed on a computer as specified herein. ICS further warrants the media to be free of defects and workmanship for a like 90 day period. The customer's remedy for a failure in either case is to return the media to ICS for replacement.

ICS makes no other warranty with respect to the software. In no event shall ICS be liable for any other incidental or consequential damages from the use of this software.

1.5 BACK UP YOUR DISK

1 If the software is on 3.5 inch disks, it is strongly recommended that you make a backup copy of the Disks supplied with the 488-PC2 Card. Use the backups for installation and put the original disks in a safe location. The storage location should be in a temperature stable location and away from all magnetic fields.

1.6 INSTALLATION

Check ICS's web site at www.icselect.com for the latest information about installing the 488.2 Drivers and periodically for the latest version of the 488.2 Drivers. or for any updates.

1.6.1 Hardware Installation Procedure

To install the 488-PC2 Card, perform the following steps:

1. Turn off the computer by clicking the Window's 95 or 98 Start button and then Shutdown. On the Shutdown Window, check the Shutdown box and click Yes to start the shutdown process. Turn power off when told it is safe to do so.
On computers with Windows 3.1, Select EXIT and YES when asked if you wish to Exit Windows. Turn power off when safely at the DOS prompt.
On computers with DOS, exit any running program and turn power off.
2. Remove the cover from the computer. Leave the power cord connected to the computer so that it stays grounded.
3. Locate a unused ISA bus slot and remove its connector cover plate and save the retaining screw.
4. Cut the antistatic bag containing the 488-PC2 card but do not remove it from the bag.
5. Touch the computer frame with both hands to be sure that your body is free of any static electricity.
6. Remove the 488-PC2 card from its antistatic bag and set the rockers on Switch S1 by following the instructions in Section 1.7. The factory setting of all switches in the '0' position should work unless you are installing multiple 488-PC2 cards, want to use interrupts or DMA.
7. Insert the card firmly into the empty ISA bus slot and fasten it down with the saved retaining screw. (Note-the 488-PC2 card must be fastened with the retaining screw for it work correctly.)
8. Replace the cover and turn the computer back on.

1.6.2 Windows 95/98 Software Installation Procedure

Perform the following steps to install the 488.2 Drivers in a PC with Windows 95 or Windows 98. Note - Close all applications before installing the software.

1. Make a backup copy of the Windows Driver Disk if on a 3.5 inch disk.
2. Insert the Windows Driver Disk in the computer's floppy disk drive or put the CD in the computer's CD ROM drive.
3. If the installation program does not start automatically, click the Windows' Start menu, point to Run and execute a:\setup
4. Follow the instructions on the screen to complete the software installation. Setup will install all of the WIN32 files on your computer.

1.6.3 Windows 3.1 Software Installation Procedure

Perform the following steps to install the 488.2 Drivers in a PC with Windows 3.1. Note - Close all applications before installing the software.

1. Make a backup copy of the Windows Driver Disk if on a 3.5 inch disk.
2. Insert the Windows Driver Disk in the computer's floppy disk drive or put the CD in the computer's CD ROM drive.
2. Click the Windows' Start menu, point to Settings and then execute a:\setup.
4. Follow the instructions on the screen to complete the software installation. Setup will install all of he WIN16 files on your computer.

1.6.4 DOS Software Installation Procedure

The software installation instructions for DOS drivers are included in Section 5 for each supported DOS language. If you are planning on doing DOS programming, follow the instructions for your DOS programming language.

1.7 488-PC2 CONFIGURATION SWITCH SETTINGS

The 488-PC2 contains a nine position rocker switch assembly, S1, that sets the card's I/O port base address, the interrupt request level, the DMA level, and the I/O READY wait time. The rocker switch is located in the upper left of the 488-PC2 Card as shown in Figure 1-2.

The rockers are factory set to the off or '0' position which is fine for most single card applications. This setting specifies Card I/O address 02E1, no interrupt level, no DMA Channel and no I/O READY wait states. The following sections provide specific instructions for setting the switch rockers

1.7.1 I/O Address Selection

When defining the PC/AT Computer's I/O addresses, IBM set the address for the GPIB Adapter 0 Card at 02E1 hex. Subsequent GPIB Adapter card addresses are located in 2000 hex increments above Adapter 0's address as shown in Table 1-1. If, like most 488-PC2 users, your system has only one card it should be set to address 02E1. Any additional 488-PC2 cards should use the next available GPIB Adapter address.

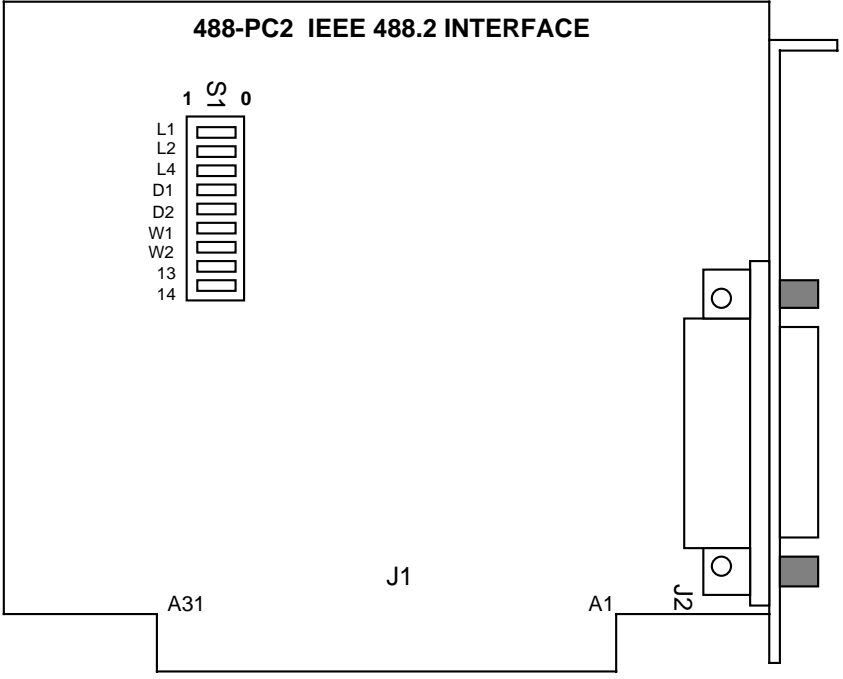
Rockers 13 and 14 are used to set the 488-PC2's I/O starting address. The rocker's correspond to the PC address lines A13 and A14. Use Table 1-1 to convert the two most significant address digits into rocker switch positions. The CardID parameter identifies the card for the Windows DLL.

When setting the rocker switches, follow the 1 and 0 silk screened on the card. Do not use the switch's "on" and "off" labels to set the rockers.

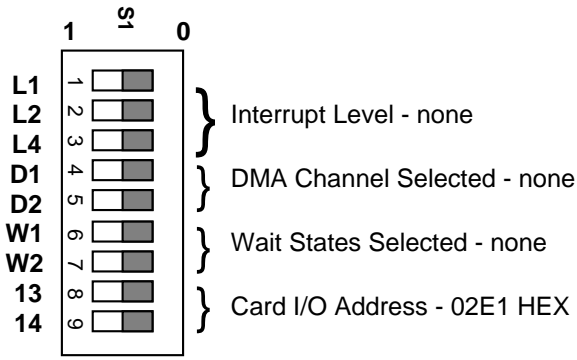
TABLE 1-1 I/O ADDRESS ROCKER SETTINGS

I/O Address (Hex)	CardID	Switch Rocker	
		14	13
02E1*	7	0	0
22E1	6	0	1
42E1	5	1	0
62E1	4	1	1

*Factory setting



(a) 488-PC2 Card Layout



(b) Factory Settings for the 488-PC2 Configuration Switch

Figure 1-2 488-PC2 Switch Location and Rocker Assignments

1.7.2 Wait State Switch Settings

Rockers W1 and W2 control the insertion of a wait state delay onto the computer's I/O READY signal line. This delay, while not required on most computers, will allow the 488-PC2 to operate in computers with high bus clock speeds. The 488-PC2's on card oscillator provides the clock signal to the card's logic so timing is normally not a problem. Bus clock speed is **NOT** the same as the CPU clock frequency and is normally a submultiple of the CPU clock. Most new PC/ATs are designed with a 8 ± 1 MHz bus clock and do not require any wait states for correct operation. Check your computer's manual for your computer's bus clock frequency and set the W1 and W2 rockers per Table 1-2.

TABLE 1-2 WAIT DELAY SETTINGS

CLOCK FREQUENCY (MHz)	WAIT SWITCHES	
	W1	W2
0 to 9*	0	0*
9 to 20	1	0

* Factory Setting

1.7.3 DMA Switch Setting

Rockers D1 and D2 select the PC's DMA channel that the 488-PC2 would use for transferring data. DMA data transfers can be faster than programmed data transfer on older, slow computers especially for large blocks of data. Using DMA data transfers is not advised if the processor clock speed is greater than 100 MHz. (See Table 1-2) Using DMA is purely a user's option; the 488-PC2 will function just as well without using DMA.

Before setting the DMA switches, check your computer's manual or the Device Manager for an available DMA channel. IBM has made the following DMA channel assignments for the PC/AT:

DMA Channel	Function
CH 0	Memory refresh
CH 1	SDLC I/O
CH 2	Floppy disks
CH 3	Hard disk

Most stand alone computers don't have an SDLC interface so channel 1 is normally the safe channel to use. Only use channels 2 or 3 if your computer does not have floppy disk drives or a hard disk. Use Table 1-3 to convert the selected DMA channel into D1 and D2 switch settings.

TABLE 1-3 DMA CHANNEL SETTINGS

DMA Channel	SWITCHES	
	D2	D1
None*	0	0
1#	0	1
2	1	0
3	1	1

*Factory setting

#Recommended DMA Channel

1.7.4 Interrupt Level Settings

Rockers L1, L2 and L4 select the PC's Interrupt Channel that the 488-PC2 card can use to interrupt the current program. Note that using interrupts is purely a user's option; the 488-PC2 will function just as well without interrupts.

Before setting the Interrupt Level switches, check your computer's manual or the Device Manager for available IRQs. In the PC/AT, IBM established a shared interrupt concept that lets several adapters share the same channel. The 488-PC2 complies with IBM's shared interrupt concept and can be set to interrupt levels 2, 3, 5 and 7. The following are IBM's interrupt level assignments for the PC/AT:

Interrupt Level	Users
0	Timer
1	Keyboard
2	Clocks, PC networks, Fixed disk ctrl
3	Serial port #2, PC network, SDLC
4	Serial port #1, BSC, SDLC
5	Parallel port #2
6	Floppy disk controller
7	Parallel port #1, Data Acquisition, GPIB

Use Table 1-4 to convert the selected Interrupt Level into switch settings.

TABLE 1-4 INTERRUPT LEVEL SETTINGS

IRQ Level	L4	L2	L1
None *	0	0	0
2	0	1	0
3	0	1	1
5#	1	0	1
7	1	1	1#

*Factory setting

#Recommended GPIB adapter interrupt level

1.8 SOFTWARE ORGANIZATION

The 488.2 Windows Driver software includes DLLs for controlling the GPIB hardware, files for supporting various languages utility files and demonstration programs. During installation, the DLLs are placed in the Windows System directory. The language support files, demos and utility files are placed in the ICS_GPIB directory. The files are organized into directories that correspond to the operating system and the supported language. Software use instructions vary with the computer's operating system and are described in the section for the appropriate operating system. ICS_GPIB directory contains a Readme.doc file and ICS's GPIB Keyboard Controller Program. Refer to Section 3 for more information about the 488.2 Driver.

The 488-PC2 DOS Driver software is organized into directories that correspond to the languages they support. Software installation instructions vary with each language group and are described in the particular section for that language. The root directory on the DOS Driver contains a Readme.doc file and the PC2_KYBD Interactive Command Line Program.

1.9 KEYBOARD CONTROLLER PROGRAMS

The 488.2 Driver includes two Keyboard Controller Programs. GPIBKybd is a Windows 32-bit program that runs on Windows 95 and Windows 98. PC2_Kybd is a DOS program and runs on Windows 3.1. Use the program that matches your computer's operating system.

The Keyboard Controller Program lets a user interactively control GPIB devices directly from the computer's keyboard and is the recommended way to test the 488-PC2 Card after its installation. The Keyboard Controller Program is also useful for testing GPIB (HP-IB or IEEE-488) devices without writing a program or for trying out device commands on a new instrument before incorporating them into a program.

1.9.1 GPIB Keyboard Controller Operation

The Keyboard Controller program is installed in the Util directory with other ICS files. To run the Keyboard Controller program, click the Window's Start menu, point to Programs and then click ICS Electronics GPIB. Select GPIBkybd from the submenu.

The program opens with the panel shown in Figure 1-3. The Keyboard Controller panel is best viewed with monitor screen settings of 1024 x 768 or 800 x 600. Other screen settings may result in a distorted view or lost switch legends.

The Keyboard Controller launches it automatically runs the 488.2 FindLstn to learn what devices are connected to the GPIB bus and displays the found device addresses in the Device Response message box.. If there is more than one device, enter the correct address in the Device Address box and click the Set Address button.

The user can also change the Controllers address by entering a new value into the Controller Address box and clicking Set Address. The Keyboard Controller defaults to a Controller address of 21 which it uses as the address of the GPIB controller card in the computer. You only need to change this address if it conflicts with the address of your device.

The Keyboard Controller enables the user to send or execute the most popular 488.1 and 488.2 functions by clicking the buttons on the panel and

Figure 1-3 GPIB Keyboard Controller Panel

to send and receive device commands. Once the device address has been set, clicking on any 488.1 or 488.2 Command button will cause that command to be executed. The 488.1 buttons generate IFCs, Device Clear, Serial Poll the device and send the Trigger command. The 488.2 buttons perform the FindLstn protocol, the AllSerialPoll and the FindRQS protocols. Any device response appears in the Device Response message box. The Help button, in the upper right, provides a description of each control and what it does to the device.

To send data or a command to a device, simply type the command in the Device Command box and click Send. Click Enter to read data from a device or responses to a query. Device Responses appear in the large Device Response message box. If Auto Query is checked, then any Device command that ends with a question mark or includes a question mark (such as *idn? or A?1) will have its response automatically appear in the Response message box.

The buttons along the bottom of the Keyboard Controller window set the bus timeout. One or three seconds is recommended for manual operation. Use Off when working with a Bus Analyzer. Clicking Exit returns the user to the Windows operating system.

1.9.2 PC2 Keyboard Controller Program

The PC2_KYBD Interactive Command Line Program is a live keyboard program for DOS that lets a user control GPIB devices directly from the computer's keyboard. The PC2_KYBD program is a good way to test the 488-PC2 Card after its installation or to test a GPIB devices's response to a command before writing your program.

Before running the program, copy the PC2_KYBD.EXE program from the Util directory to the computer's root directory. To run the program, type PC2_KYBD. The program will first ask if you want to change the default address of the GPIB device. The program uses GPIB address 21 as its own address. Enter a new address or press the RETURN key to accept the default address of 04 for the GPIB device.

The program then sends an Abort (IFC) to gain control of the bus and an escape sequence for ICS's minibox modules. The program then uses the *IDN? query to read an IDN message from the device at the GPIB address set above. The idn message or a timeout message is displayed to the user.

To send a device command to a device, simply type the command on the keyboard and press RETURN. The program will address the device as to listen and output your command. If the command ends with or contains a question mark, i.e. *IDN? or A?1, then the program will automatically address the device as a talker and input the response. To input a response manually, type a question mark and press RETURN.

e.g

```
?  
device response
```

PC2_Kybd supports a number of 488.1 commands and 488.2 protocols. To see a list of the supported bus commands or for assistance at any time, type **!help** and press RETURN. To send a bus command to the device, start the command with an exclamation mark. i.e. **!spoll** serial polls the device.

To exit the interactive command program, type **!exit** and press RETURN. Answer N when asked if you want to restart the program.

1.10 Testing the GPIB Card Installation

To test the 488-PC2 card installation, connect a device to the GPIB bus. Use a 488.2 device if possible. Start the appropriate Keyboard Controller program as described in Section 1.9.

In the GPIB Keyboard program, the device's address should appear in the Device Address window. If not, enter the device's primary address from 700 to 730. (7 identifies the first 488-PC2 card in your computer) If the device uses secondary addresses, enter the devices's address as 70000 to 73030. (Note: Using address 21 will conflict with the 488-PC2 Card's address setting.)

In the PC2_kybd Program, the device address defaults to 4. Change the device address to match the GPIB device's address when asked or at any time with the **!addr** command. (Note: Using address 21 will conflict with the 488-PC2 Card's address setting.)

What you do next depends upon the device. If the device is an 488.2 compatible device, send it the `*idn?` query and read back its `idn` response string. Figure 1-3 shows an example of the `*idn?` query and a typical response. Next, query the Event Status Register with the `*esr?` query. Repeat until the response becomes 0.

If the device is an older 488.1 device, use a device particular command to make the device do something and/or send back data. i.e. if the device is a DVM, set it up to make internally triggered readings. Use the Read Device Response button to read the measured value.

General Information

2.1 INTRODUCTION

This section provides general information and specifications for the Model 488-PC2 IEEE 488.2 Controller Card and ICS's 488.2 Driver.

2.2 MODEL 488-PC2 IEEE 488.2 CONTROLLER CARD

The 488-PC2 IEEE 488.2 Controller Card provides the GPIB physical interface (electrical and mechanical components) for any ISA/EISA PC on a small half size card. The 488-PC2 Card fits inside any PC with ISA expansion slots. When used with ICS's 488.2 Driver, the 488-PC2 supports the 488.2 Driver languages and Microsoft DOS and Windows 3.1/95 and 98 operating systems. The 488-PC2 Card may be used as a GPIB controller or as a device interface for transferring data to or from the PC.

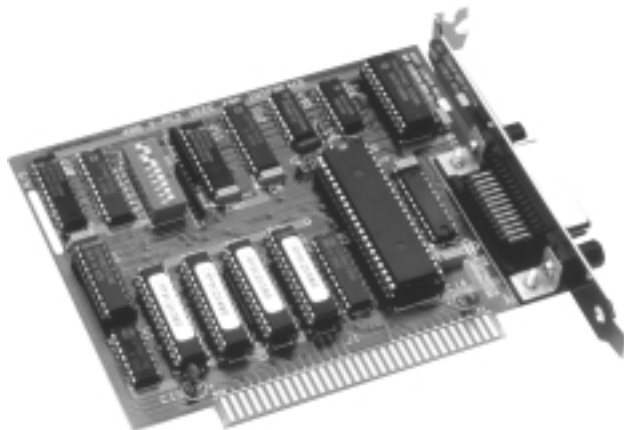


Figure 2-1 488-PC2 IEEE-488.2 Controller Card

2.3 488.2 DRIVER SOFTWARE CAPABILITIES

ICS's 488.2 Driver provides the functional portion of the GPIB interface with versatile, easy-to-use GPIB language extensions for your current programming language. For interpretive BASIC¹ programming, the driver software is contained in a device driver which is loaded when the PC is powered up. For compiled programs such as C, Pascal or Quick Basic, the driver software is contained in linkable libraries. For Windows programs, these libraries are Dynamically Linked Libraries that are loaded as needed by the operating system.

2.3.1 488.2 Driver Features

The 488.2 Driver software has the following features:

- All required IEEE-488.2 controller protocols and functions for Quick Basic, Visual Basic for DOS, Professional Basic 7.1, C and C++ DOS applications. A DLL provides IEEE-488.2 support for MS Windows based applications running on Windows 3.1, Windows 95 and Windows 98.
- 488-PC2 Driver provides 488.1 support for most interpretive BASICs and libraries for Pascal.
- Standardized commands across all languages provide high level easy to use command extensions for your program language plus the ability to generate low level bus mnemonics for custom commands.
- Automatic initialization of default values at power on.
- Include files for each language to minimize programming effort.
- Timer functions for program delays or status checking.
- Interactive Keyboard Controller Programs for testing GPIB devices or the 488-PC2 hardware.

¹ Throughout this manual, the term BASIC refers to Interpretive BASIC.

2.3.2 Supported Languages

Included 488-PC2 drivers support:

GW BASIC (Ver 3.11 and 3.22)

BASICA (Ver 3.11 and 3.22)

Vectra BASIC (Ver 3.11 and 3.22)

Included libraries support:

DOS Languages:

Quick Basic (Ver 4.5)

Microsoft Professional BASIC (Ver 7.1)

Microsoft Visual Basic for DOS (Ver 1.0)

Borland Turbo C for DOS (Ver 3.0)

Microsoft Quick C (Ver 2.5)

Microsoft Pascal (Ver 4.0)

Borland Turbo PASCAL (Ver 4.0)

Windows 3.1 Languages:

Borland Turbo C++ for Windows 3.1

Microsoft Visual C++ (Ver 1.0 and 1.5) (16-bit programs)

Windows 95 and Windows 98 Languages:

Borland C/C++ (Ver 2.0, 3.0 and 4.0)

Microsoft Visual C++ (Ver 6.0)

Microsoft Visual Basic for Windows (Ver 4.0, 16-bit DLL)

Microsoft Visual Basic for Windows (Ver 6.0, 32-bit DLL)

Note: QBASIC from Microsoft is the same as Quick Basic and it is not supported.

2.4 IEEE-488 BUS SPECIFICATIONS

The GPIB Interface on the 488-PC2 conforms to the IEEE STD 488.1 specification. Drivers are tristate devices, capable of driving up to 14 other Bus compatible devices over 20 meters of cable. The connector is the standard IEEE-488 24-pin connector with metric studs. Signal pin assignments and a description of the GPIB bus's operation are provided in Appendix A1. The data transceivers in the 488-PC2 will not cause latchup in older GPIB devices.

2.4.1 Controller Functions

As an IEEE 488.2 Bus controller, the 488-PC2 performs AH1 and SH1 handshakes plus controller subsets C1 through C4, and C9 depending upon the user's program. These subsets include: single and extended addresses, service requests, remote enable, device trigger, device clear, interface clear, serial poll, parallel poll, pass control, and take control. This enables messages to be transferred from

- a) Computer to device(s)
- b) Device to computer
- c) Device to device(s)
- d) PC to PC, or PC to other controllers

The Controllers address 31 talk/listen primary addressable devices and 930 secondary addressable devices. The 488-PC2 responds to its primary address when acting as a device or when taking control.

The 488-PC2 functions as IEEE-488.2 controller and performs the following protocols:

ALLSPOLL
FINDLSTN
FINDRQS
RESET

2.4.2 Bus Device Functions

As a device, the 488-PC2 becomes a transparent IEEE 488 Bus-to-PC interface with AH1, SH1, T2, TE0 L2, LE0, SR1, RL2, PP1, DC1, DT1, C0 and E1 capabilities depending upon the user's program. When addressed as a listener, the 488-PC2 inputs data bytes from the Bus and passes them to the computer. When addressed as a talker, the 488-PC2 accepts data bytes from the computer and handshakes them onto the 488 Bus. Data transfer can either be under program control, or implemented as a block transfer under DMA control.

The 488-PC2 will respond to a serial poll by outputting the data stored in its serial poll response register. With the exception of the DIO7 bit, the bit pattern and meanings are user defined.

2.5 488-PC2 CARD SPECIFICATIONS

2.5.1 Architecture

The 488-PC2 card contains an NEC7210 IEEE 488 Bus Controller chip, address decoding logic, DMA/Interrupt selection logic and a buffer for sensing GPIB signal status. The 488-PC2 Bus controller chip is controlled by sending it data and commands via the PC's I/O Bus. The 488-PC2's address control switches lets you set the 488-PC2's I/O address from 02E1 hex to 62E1 hex in 2000 hex increments (Default address is 02E1. The 488-PC2 card uses 9 separate I/O addresses to address and access its control registers and the GPIB signal buffer.

BASIC programs that control the IEEE 488 bus use a 16K byte memory resident device driver to interface with the 488-PC2 card. Programs written in other languages use loadable libraries that link to the program at run or compile time.

2.5.2 PC Interrupt

The 488-PC2 has a single, switch selectable interrupt level. Assignable interrupt levels are 2, 3, 5, and 7. Setting all switches to off disables the hardware interrupts.

2.5.3 DMA

The 488-PC2 supports DMA data transfer in system controller, active controller or device operating modes. Switch selectable DMA channels are 1,2, and 3 (default is 1). Setting all switches off disables DMA transfers.

2.5.4 Data Transfer Capability

Data transfer rate under program control is proportional to the CPU speed and processor type. This rate varies from a 44 Kbytes/sec with the original IBM PC to over 200 Kbytes/second with a 486 processor as shown in Table 2.1. Data transfer rate using DMA is relatively constant at 240 Kbytes/sec. Maximum data block size for DMA transfers is 64 kbytes.

TABLE 2-1 488-PC2 TRANSFER RATES (BYTES/SEC)

PC MODEL	CLOCK RATE	TRANSFER RATES	
		NO DMA	W/ DMA
IBM XT	4.77 MHz	44.5 K	238 K
386AT	20.0 MHz	160 K	204 K
486AT (Dx2)	66/33 MHz	210 K	240 K

2.5.5 Physical

Size	One-half size PC card, 5 inches long
PC Bus	Compatible with PC/XT/AT and PS/2 models 25 and 30, ISA and EISA buses.
Slot	Fits in any available PC slot
Power	+5 \pm 0.2 Vdc at 500 mA max.
Connector	IEEE 488 Bus Interface: Amphenol 57-20240 style with metric lock studs

2.5.6 Certifications and Approvals

Meets limits for Part 15, Class A of US FCC Docket 20780 and complies with EEC Standards EN 55022 and 50082-1. CE Certificate of Compliance reproduced in Figure 2-2.

CE Approved

2

<i>Declaration of Conformity</i>	
Application of Council Directive(s).....	89/336/EEC
Standard(s) to which Conformity is Declared.....	EN 55022, EN 50082-1
Manufacturer's Name	ICS ELECTRONICS CORPORATION
Manufacturer's Address	473 LOS COCHES STREET MILPITAS, CA 95035-5422
Importer's Name	
Importer's Address	
Type of equipment	GPIB CONTROLLER
Model No.	488-PC2
Serial No. _____ Thru _____	Year of Manufacture <u>1995-1996</u>
<i>I, the undersigned, hereby declare that the equipment specified above conforms to the above Directive(s) and Standard(s)</i>	
Place <u>Milpitas California USA</u>	<u><i>G. K. Mercola</i></u> (Signature)
Date <u>11-15-95</u>	<u>G. K. MERCOLA</u> (Full Name)
	<u>President</u> (Position)

Figure 2-2 488-PC2 CE Certificate of Compliance

2.6 ACCESSORIES

ICS provides the following optional accessory items for use with the 488-PC2 Controller Card.

Part No.	Accessory
104705	Double shielded Bus Cable with stackable bus connectors, 1/2 M long
104710	Double shielded Bus Cable with stackable bus connectors, 1 M long
104720	Double shielded Bus Cable with stackable bus connectors, 2 M long
104740	Double shielded Bus Cable with stackable bus connectors, 4 M long
105705	Double shielded Bus Cable with a straight-in bus connector, 0.5 M long
105710	Double shielded Bus Cable with a straight-in bus connector, 1.0 M long
105720	Double shielded Bus Cable with a straight-in bus connector 2.0 M long
123077	Spare User's Manual

488.2 Driver

3.1 INTRODUCTION

This section describes ICS's 488.2 Driver's support for Windows 95 and Windows 98, Windows 3.1, and support for DOS. This section also includes a description of the Command Set characteristics, Command Set Quick Reference lists and selection criteria.

3.2 488.2 DRIVER WIN COMPONENTS

ICS's 488.2 Driver provides Win32 support for C/C++ and Visual Basic applications using the 488-PC2 Command Sets. The following paragraphs describe the Driver's software components. Figures 3-1 and 3-2 show how they interact to control the GPIB hardware.

3.2.1 Components

1. PC2W32.DLL - A 32-bit thunk DLL that implements the 488-PC2 Command Set for Windows 95 and 98.
2. PC2W16.DLL - A 16-bit intermediate DLL that implements the 488-PC2 Command Set for Windows 95 and 98.
3. PC2W.DLL - A 16-bit DLL that implements the 488-PC2 Command Set and controls the 488-PC2.
4. PC2W32.lib - A 32-bit library file with the 488.2 Driver functions. Used for linking C/C++ programs with the PC2W32.dll.

- 5. PC2W.h - C/C++ Language header file for accessing the 488.2 Driver functions
- 6. PC2W32.bas - A Visual Basic file with program constants and Command Set declarations for linking 32-bit applications to the 488-PC2 Command Set. Combines PC2W.bas and Global.bas files for 32-bit applications.
- 7. PC2W.bas - A Visual Basic file with program constants and Command Set declarations for linking 16-bit applications to the 488-PC2 Command Set.
- 8. Global.bas - A Visual Basic file with program variables for 16-bit Applications.
- 9. Readme.doc - A test files with file descriptions and late information about the 488.2 Driver.
Readme.txt -

3

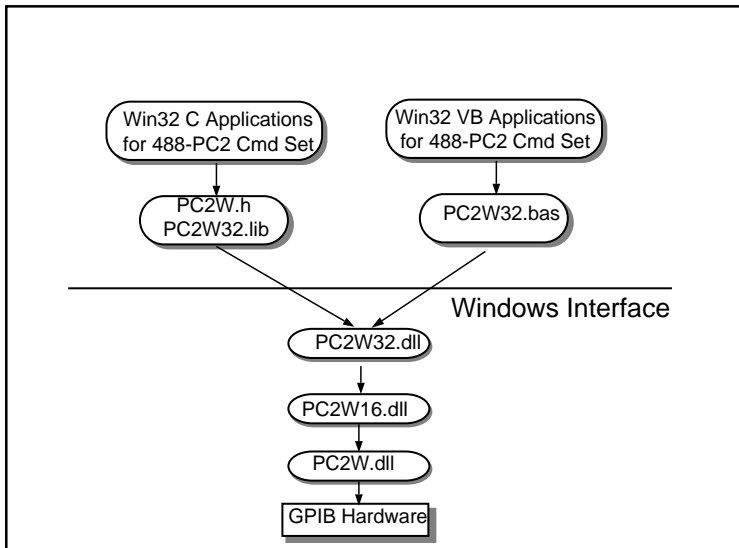


Figure 3-1 Win 32 Components Organization for Windows 95/98

3.2.2 32-Bit Applications

Figure 3-1 shows how the files work together to control the GPIB bus from 32-bit Windows applications. The PC2W.h or PC2W32.bas header files are included or linked with the application to specify the 488.2 Driver constants and define command syntax and variables. The PC2W32.lib files allow C/C++ applications to make calls to the PC2W32.dll. Visual Basic applications make direct calls to the PC2W32.dll. PC2W32.dll and PC2W16.dll are think layer DLLs that convert the 32-bit calls into 16-bit calls. The PC2W.dll interprets the 488-PC2 commands to operate the GPIB hardware.

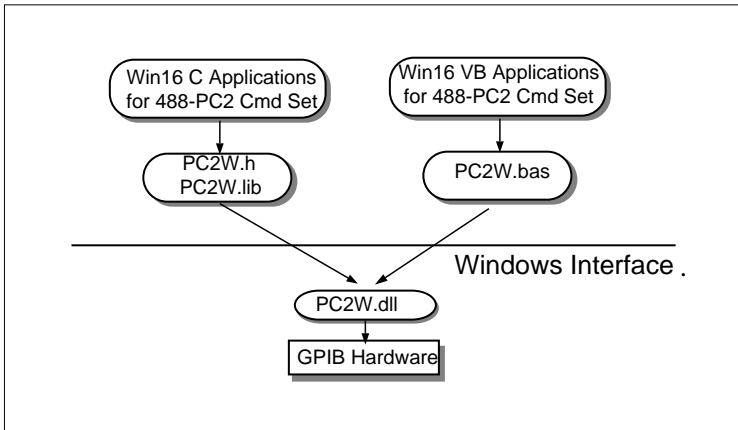


Figure 3-2 Win 16 Component Organization for Windows 3.1 or Windows 95

3.2.3 16-Bit Applications

Figure 3-1 shows how the files work together to control the GPIB bus from 16-bit Windows applications. The PC2W.h or PC2W32.bas header files are included or linked with the application to specify the 488.2 Driver constants and define command syntax and variables. The PC2W.lib files allow the C/C++ applications to make calls to the PC2W.dll. Visual Basic applications directly call the PC2W.dll. The PC2W.dll interprets the 488-PC2 commands and operates the GPIB hardware.

3.3 DOS Language Support

The 488.2 Driver has the following components for DOS Languages.

3.3.1 C/C++ Language Support

Libraries and header files that are linked to the C/C++ program.

1. PC2INC.H Header file for accessing the 488.2 Driver functions
2. LMSCPC2.LIB Library for MS large memory model.
3. MMSCPC2.LIB Library for MS medium memory model.
4. SMSCPC2.LIB Library for MS small memory model.
5. LMSCPC2.LIB Library for Borland large memory model.
6. MMSCPC2.LIB Library for Borland medium memory model.
7. SMSCPC2.LIB Library for Borland small memory model.

3.3.2 DOS Quick Basic/Professional Basic Support

Libraries and header files that are linked to the Quick Basic program.

1. PC2INCL.BAS An include file for the 488.2 Driver functions
2. PC2COM.BAS An include file with the 488.2 Driver constants, settings and variables.
3. PC2QBDOS.LIB 488.2 Driver library for making Quick Basic .exe programs.
3. PC2QBDOS.QLB 488.2 Driver library for running programs in the Quick Basic environment.
4. PC2QBDOS.OBJ 488.2 Driver library for making other libraries.

3.3.3 DOS Pascal Language Support

1. GPIB.TPU Library file with Turbo Pascal 488.1 function calls.
2. GPIB.PAS GPIB.TPU source file.
3. PC2TPDVR.OBJ Object file with 488.1 functions for Turbo Pascal programs.
4. MSPC2.INC Library file with MS Pascal 488.1 function calls.
5. PC2MSPDV.LIB Library file with MS Pascal 488.1 functions.

3.3.4 DOS BASIC Language Support

Following are loaded into memory at the start of the program.

1. ICSDECL.BAS Sample BASIC program header file.
2. PC2-DRVR.BIN 488-PC2BASIC Driver
3. LOADER.BIN Routine to generate pointers for BASIC driver commands.

3.3.5 Utilities for DOS and Windows

1. GPIBkybd.exe - An interactive graphical control utility for Windows 95/98 that lets a user communicate directly with GPIB devices without having to write a program.
2. PC2wKybd.exe - A console mode control utility for Windows 3.1 that lets a user communicate directly with GPIB devices without having to write a program.
3. PC2_kybd.exe - An interactive control utility for DOS that lets a user communicate directly with GPIB devices without having to write a program.

3.4 DRIVER COMMANDS

Table 3-1 provides a brief listing of the 488.2 Driver commands and their compatibility across languages and operating environments. Table 3-1 should only be used as a quick reference guide. Refer to the Command Reference Section for a complete description of each command, its required parameters, parameter types, and examples. The commands are listed with mixed case letters as they would be used in Compiled Basic, C or C++ programming for better legibility. For Interpretive BASIC programs, the commands would be written as all capitals.

The Compatibility columns indicate which languages can use the command: **P** = PASCAL, **B** = BASIC, **Q** = Quick, Professional or Visual Basic, **C** = C/C++ and **W** = Windows 3.1 or Windows 95 and 98.

TABLE 3-1 488.2 DRIVER COMMANDS

MNEMONIC	Compatibility					DESCRIPTION
	P	B	Q	C	W	
ieAbort	•	•	•	•	•	Outputs an IFC pulse to clear all devices off the GPIB bus and asserts ATN to take control of the bus.
ieAllSpoll (AddrList, RespList)			•	•	•	Performs IEEE 488.2 Serial Poll All Devices protocol. Serial polls all listed bus devices and returns a list of the results.
ieClose			•	•	•	Cleans up drivers and restores system environment. Must be called at end of program and before all ieInits except for the first ieInit.
ieDevClr (DevAddr)	•	•	•	•	•	Sends the selected device clear command (SDC) to device(s) or the universal device clear (DCL) command to all devices.
ieDevice (DevAddr, DOSPort)	•	•				Installs a Bus device driver in place of LPTn or COMn.

TABLE 3-1 488.2 DRIVER COMMANDS (CONT.)

MNEMONIC	Compatibility					DESCRIPTION
	P	B	Q	C	W	
ieEnter (DevAddr, InString)	•	•	•	•	•	Inputs data from bus device into a string variable. String terminator is specified by the EOL command.
ieEnterA (DevAddr, InString)	•	•				Inputs data from bus device into an array variable. Data terminator is specified by the EOL command.
ieEnterB (DevAddr, InString)	•	•	•	•	•	Inputs data from bus device into a string variable. Input is terminated when the EOI line is detected or InString is full.
ieEol (DevAddr, OutEOL, OutEOLStr, InEOL, InEOSByte)	•	•	•	•	•	Sets input and output terminators for the specified bus device. Default output termination is CR LF with EOI asserted during the LF. Default input terminator is LF or EOI line true.
ieErrPtr (IOErrCode, IOBytes)		•	•			Assigns variables to keep track of bus transfer errors and count of input/output string bytes. Execute before other commands
ieEventEnable (Addr, hWnd, wEvMask)					•	Enables notification messages of hardware events to be sent to a particular window.
ieEventStat			•	•		Function returns an integer that specifies what event occurred: 1 = SRQ, 2 = Time-out, 4 = Error.
ieFindLstn (AddList, ResultList)			•	•	•	Performs the IEEE 488.2 Find Listener protocol and returns a list of found listeners
ieFindRQS (AddrList,			•	•	•	Performs the IEEE 488.2 Find Device Requesting Service protocol and returns the address of the first

TABLE 3-1 488.2 DRIVER COMMANDS (CONT.)

MNEMONIC	Compatibility					DESCRIPTION
	P	B	Q	C	W	
ieFindRQS cont'd						device found requesting service and its status byte response. Returns 3131 if no device found that needs service.
ieGotoStby			•	•		Makes the 488-PC2 a Standby Controller and de-asserts ATN if the 488-PC2 was an active controller.
ieGPIBStat			•	•		Function reads the status of the GPIB control and handshake lines. Returns an integer value between 0 and 255.
ieInit (IOPort, MyAddr, Setting)	•	•	•	•	•	Initializes the 488-PC2's hardware, sets card's I/O port address, Bus address and DMA/Interrupt/System Controller parameters.
ieLlo	•	•	•	•	•	Sends local lockout message (LLO) to all devices.
ieLocal (DevAddr)	•	•	•	•	•	Sends a go-to-local message (GTL) to DevAddr. If Addr ≤ -1 or ≥ 31, the card clears the remote message (turns the REN Bus line off)
ieOutput (DevAddr, OutString)	•	•	•	•	•	Outputs a string to DevAddr. String terminator is specified by the EOL command.
ieOutputA (DevAddr, DataSeg, ByteCnt)	•	•				Outputs data in an array to DevAddr. Data terminator is specified by the EOL command.
ieOutputB (DevAddr, OutString)			•	•	•	Outputs a string or binary data to DevAddr. String or data is terminated with EOI asserted on last byte.
iePassCtl (DevAddr)			•	•		Passes control to an inactive bus controller at address DevAddr.

3

TABLE 3-1 488.2 DRIVER COMMANDS (CONT.)

MNEMONIC	Compatibility					DESCRIPTION
	P	B	Q	C	W	
iePPoll (PResp)	•	•	•	•	•	Conducts a parallel poll of all configured devices. Returns a response byte value (0-255)
iePPollC (DevAddr, ConfigBit)	•	•	•	•	•	Configures DevAddr to respond to a parallel poll (true or false) on line 0-7 as specified in ConfigBit. 0-7 = False response on lines 0-7 8-15 = True response on lines 0-7
iePPollU (DevAddr)	•	•	•	•	•	Sends DevAddr the parallel poll disable command. If Addr ≤ -1 or ≥ 31, all devices are sent the unconfigure command.
ieRemote (DevAddr)	•	•	•	•	•	Addresses DevAddr to listen and sends it the remote enable message by setting REN true. If Addr ≤ -1 or ≥ 31, then just the REN line is set true.
ieReset (AddrList)			•	•	•	Performs the IEEE 488.2 Reset protocol on all devices and sends *RST command to all listed devices.
ieSend (CmdString)	•	•	•	•	•	Sends user specified Bus commands or data to the Bus. CmdString is a string of English-like Bus commands and data. e.g. (CMD="UNT LISTEN 4 SEC 0 DATA 'T1'"). ieSend mnemonics and commands are listed in the Command Reference section under the ieSend command and described in Appendix A1.
ieServiceDisable (ServCode)				•		Function disables driver call to user's program when specified service event occurs.

TABLE 3-1 488.2 DRIVER COMMANDS (CONT.)

MNEMONIC	Compatibility					DESCRIPTION
	P	B	Q	C	W	
ieServiceEnable (ServCode, SRQServ [,TOutServ[,ErrServ]])					•	Function enables driver to call user programs when specified service event occurs. Serv Codes are: 1 = SRQ, 2 = Time-out, 4 = Error. User programs are: SRQServ, TOutServ and ErrServ.
ieSPoll (DevAddr, SResp)	•	•	•	•	•	Serial polls DevAddr. Returns the response byte as value (0-255) .
ieSRQStat			•	•		Function returns an integer 0 or 1 that specified the logical state of the SRQ line.
ieStatus (StatusCode, Status)	•	•	•	•		Reads status information about the interface and the last called command. StatusCode specifies the information type. Returns value (0-255) in the Status parameter.
ieTakeCtl (Mode)			•	•		Commands the 488-PC2 change from a standby controller to an active controller, take control of the bus and assert ATN. Mode selects control method: Mode = ASYNC, SYNC or EOI.
ieTimeOut (Time)	•	•	•	•	•	Sets the handshake timeout in milli seconds (Time) Time = 0, infinite time-out Time = 1 to 32767, time-out in ms. Time = -1 to -32767, time-out in (Time+65,536)ms.
ieTrigger (DevAddr)	•	•	•	•	•	Sends the bus trigger command (GET) to DevAddr. If Addr≤-1 or ≥31, sends the GET command to all listeners.

3

TABLE 3-1 488.2 DRIVER COMMANDS (CONT.)

MNEMONIC	Compatibility					DESCRIPTION
	P	B	Q	C	W	
ieWaitSRQ (SleepTime)			•	•		Function suspends program until SRQ is asserted or sleep time expires. SleepTime is in milliseconds. Function returns SRQ state when returning: 1 = SRQ asserted, 0 = not asserted
TIME UTILITIES						
msDelay(ms)			•	•		Suspends program execution for ms milliseconds.
icsTimer			•	•		Reads high resolution system timer. Returns number of 215 μsec periods since midnight.
isTimeOut(msMark)			•	•		Sets background time mark when first called. Later calls check to see if system time is greater than the time mark. msMark is in milliseconds.

GPIB Programming

4.1 INTRODUCTION

This section describes how to use the 488.2 Drivers to write programs to control GPIB devices, describes Visual Basic and C++ program examples, and supporting Test and Measurement Application Programs.

4.2 BASIC GPIB PROGRAMMING TECHNIQUES

This section describes some basic GPIB programming concepts and is intended as a guide for anyone doing GPIB programmer. A new GPIB user should read the description of the GPIB bus operation in Appendix A1 before proceeding. The concepts described below are general in nature. Always refer to the selected Command Set Reference for the exact parameter definitions.

Also note that variable names used in the command examples are placeholders and can be changed to make them more descriptive for your program. Outstring\$ can become CmdStr\$, DVMSsetup\$ etc. Similarly, InString\$ can become Rdg\$, OvenTemp\$ etc. What is important is the order of the variables and their definitions.

4.2.1 GPIB Program Outline

A typical GPIB program has the following steps:

1. Initialize the GPIB Controller Card and the bus.
2. Verify that the device(s) is (are) present.
3. Initialize the device(s) by sending it setup commands etc.
4. Read data or a response back from the device.

The good news is that most GPIB programs only use six to eight of the 42 commands available in the 488-PC2 command set. Once you have mastered these few commands, you can make programs of any degree of complexity and rarely ever need another command. Figure 4-1 shows a typical test setup with two instruments. It uses just six 488-PC2 commands to initialize the instruments, generate analog outputs and to take readings.

In some cases, it may be necessary to check a device's status to see if it has completed a task or has data ready before reading data from a device. It may also be necessary to send a device some special GPIB bus commands to put it in a configure mode. These actions are also covered in the following paragraphs.

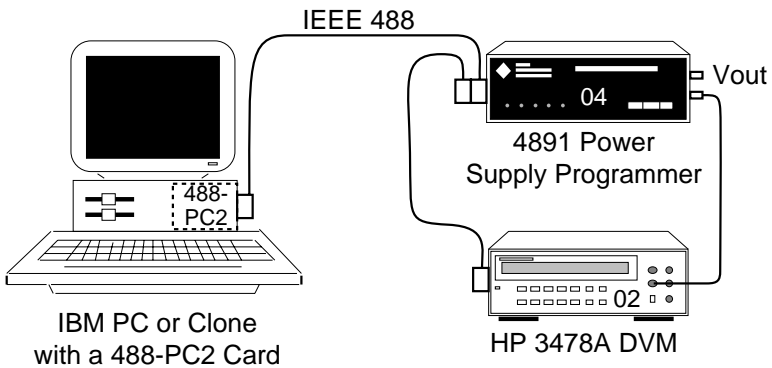


Figure 4-1 A Small GPIB Test System

4.2.2 Device Addressing

The GPIB bus uses 32 addresses from 0 to 31. All GPIB devices are addressed by a primary address between 0 to 30 so each device is set to a unique address. Address 31 is the Untalk or Unlisten address and is not a device address. The GPIB Controller card is considered a bus device and has its own bus address. ICS and Hewlett-Packard Controllers default to address 21; National Instruments and other controllers default to address 0.

Some devices use secondary addresses to address a channel or subfunction in the device or to escape to a setup mode. An example is ICS's 4896 Quad Serial Interface which uses secondary addresses to select a serial channel. There are 31 possible secondary addresses, 0 to 30. Again, secondary address 31 is not used by a device.

A Windows operating system can support multiple GPIB Controller Cards. The GPIB Controller Card address is the CardID. In the 488-PC2 Command Set, the first CardID is 7, the second card is 6 etc.

The 488-PC2 Command Set combines the card, device primary address and device secondary address into a single DevAddr variable. The overall form of DevAddr for Windows environments is:

$$\text{DevAddr} = \text{CardID} + \text{pp} + [\text{ss}]$$

where [ss] is the optional secondary address. To address a device at primary address 4,

$$\text{DevAddr} = 704$$

To address a 2nd channel at primary address 4,

$$\text{DevAddr} = 70402$$

For DOS programs, DevAddr reduces to:

$$\text{DevAddr} = \text{pp} + [\text{ss}]$$

where [ss] is the optional secondary address. To address a device at primary address 4,

$$\text{DevAddr} = 4$$

To address a 2nd channel at primary address 4,

$$\text{DevAddr} = 402$$

The No Address concept is used when a command applies to all of the devices on the bus. A NoAddr constant has been predefined to simplify the programming. An example is

$$\text{ieDevClr}(\text{DevAddr}\%)$$

sends the SDC command to the specified device.

$$\text{ieDevClr}(\text{NoAddr}\%)$$

sends the Clear command to all devices on the bus who are addressed as listeners.

4.2.3 Command Calling Conventions

For Visual Basic, C and Pascal, the 488-PC2 commands return a variable that indicates if the command was correctly executed. In these languages, the 488-PC2 commands are called by equating them with an error variable i.e.

ioerr% = ieInit(IOPort, MyAddr, Setting)

The error variable can be checked in a standard routine to display any error conditions to the program operator.

In Quick Basic, Professional Basic and other interpreted languages, the commands do not return a status variable. An example is:

CALL ieInit(IOPort, MyAddr, Setting)

In Windows programs, the command syntax often includes an additional variable for the command length or number of parameters in a list. This extra variable is simply due to differences between the DOS and Windows calling conventions. An example is the ieOutput command:

For Windows:

ioerr% = ieOutput(DevAddr, OutString\$, L)

For DOS:

CALL ieOutput(DevAddr, OutString\$)

The following paragraphs show how to initialize, command and control GPIB devices. Refer to the Command Reference for the correct syntax and for command examples in your programming language.

4.2.4 Initializing the Controller and GPIB Bus

The 488-PC2 Card is initialized to be sure that it is the System Controller and is the Controller-in-charge of the bus. The bus is initialized to be sure that all of the devices are in a non-addressed state after their power turn-on. This is done by having the GPIB Controller issue an Interface Clear command (IFC pulse) and assert the REN line. It is also a good idea to check or set the bus timeout. The bus timeout is the amount of time that the program will wait for a device to respond to a command before proceeding. In the 488-PC2 Command Set, this is done with the following commands:

```
ioerr% = ieInit(IOPort, MyAddr, Setting)      'initializes the Driver
ioerr% = ieAbort                             'sends IFC, sets REN on
ioerr% = ieTimeOut (Time)                   'sets bus timeout
```

The 488-PC2 Command Set returns an error status value in the `ioerr%` variable when it is finished. If the value is zero, the command was successfully executed. A nonzero value means the command was not executed correctly and that the device probably was not there and/or did not receive the Output String.

4

4.2.5 Sending Data to a Device

Data or device commands are normally sent to a device as strings of ASCII characters. This could be the setup commands to a DVM so it knows the type of reading to take, to changing the baud rate in a GPIB-to-Serial converter, or to output a value or data to a device.

In the 488-PC2 Command Set, ASCII data is sent by first specifying the string and then calling the `ieOutput` command:

Windows example:

```
Outstring$ = "Command to be sent"
L = Len(Outstring$)
ioerr% = ieOutput(DevAddr, OutString$, L)
```

Quick Basic example:

```
Outstring$ = "Command to be sent"
Call = ieOutput(DevAddr, OutString$)
```

The IEEE 488.2 Standard states that all command strings need to be terminated with a linefeed character or by asserting the EOI line on the last character. ICS's 488.2 Driver defaults so that ieOutput automatically terminates the output string by appending a LF and asserting EOI. The termination can be changed for a specified device with the ieEOL command. If the output data is binary data, use the ieOutputB command which terminates the output by only asserting EOI on the last character.

4.2.6 Reading Data from a Device

ASCII data strings are read from a device by first specifying an empty string and then reading the data into the string. Data is read until a terminator is found or the defined Input string is full. Typical GPIB message terminators are linefeed or EOI asserted. An example in the 488-PC2 Command Set is:

Instring\$ = String\$(Lin, 32) 'fills the string with spaces
ioerr% = ieEnter(DevAddr, Instring\$, Lin)

where Lin is the length of the input buffer. The 488-PC2 ieInput command defaults to terminating when a linefeed or EOI is sensed. For binary data, use the ieInputB command. The ieEOL command can be used to change the terminator for a specific device.

4.2.7 Clearing a Device

Some devices have buffers that accumulate unwanted data and it occasionally becomes necessary to clear out the old data or return a device to a known condition. This is done by sending the device the Device Clear Command. In the 488-PC2 Command Set this is done by:

ioerr% = ieDevClr(DevAddr)

4.2.8 Reading Device Status

Some times it is desirable to read the device's Status Register to see if it has data, has a problem or has completed some task. IEEE-488 devices report their status, Status Register contents, in response to Serial polls. Serial polls provide up to 8 bits of information from a single device and can also identify a device that has requested service. Consult the device's instruction manual for the meaning of the bits in its Status Register. IEEE-488.2 compliant devices also report their status in response to the *STB? query. (See section

A1.2 in the Appendix for information about the 488.2 mandated Status Reporting Structure). In the 488-PC2 Command Set, the status register is serial polled and the value placed in the DevStatus variable by:

```
DevStatus = ieSPoll(DevAddr)
```

To use the *STB? query do:

```
Outstring$ = "*STB?"  
L = Len(Outstring$)  
ioerr% = ieOutput(DevAddr, OutString$, L)  
Instring$ = String$(Lin, 32) 'fills the string with spaces  
ioerr% = ieEnter(DevAddr, Instring$, Lin)
```

4.2.9 Sending Custom Bus Commands to a Device

Sometimes it is necessary to send custom Bus Commands to a device to address a device as a talker or as a listener or to enter a special configuration mode. Bus commands are single character commands that are sent to a device with ATN on. Refer to Table A-1 in the Appendix for a list of these commands.

In the 488-PC2 Command Set, this is done by specifying a command string and then calling the ieSend command. The ieSend command interprets the mnemonics and converts them into GPIB bus bytes. In the following example, CmdStr\$ is set to the escape sequence used with some of ICS's miniboxes to put them in their command mode. Consult the Command Reference for the Send command mnemonics.

```
CmdStr$ = "UNL LISTEN 4 UNL LISTEN 4 UNL "  
'device address = 4  
L = Len (CmdStr$)  
ioerr% = ieSend(DevAddr, CmdStr$, L)
```

4.2.10 Setting Timeouts

Due to the nature of the Windows environment, if a user sets the bus handshake timeouts too short, another application could use up the allotted time, thereby creating false errors. If the user sets the handshake timeouts too long or to infinity, then a bus problem could hang the system up in the GPIB application. The recommendation is to use a moderate timeout of 1 to 3 seconds. Press the ESC key to exit the suspending loop.

4.2.11 Using Multiple 488-PC2 Cards

Even though the 488.2 Driver supports multiple cards (and GPIB buses), there is no hardware or software locking. Users should take care to not run multiple applications that access the same 488-PC2 card at the same time.

4.2.12 Error Reporting and Handling

Visual Basic, C/C++ and Pascal programs normally call the 488.2 Driver commands by equating them to an error variable. The 488.2 Driver returns a zero in the error variable if there is no error. The recommended procedure is for the programmer to call a standard routine to test the error variable and to take some corrective action for a nonzero value. The example function beeps and displays an error message.

```
ioerr% = ieEnter(DVM, Rdg$, Len(Rdg$))  
ProcessError (ioerr%)
```

ProcessError is a function that has been added to the (General) Object in the Visual Basic form. See the example Visual Basic programs.

```
Private Sub ProcessError(ErrCode%)  
  Select Case ErrCode%  
    Case 0  
      txtError.Text = ""  
    Case 1  
      txtError.Text = "Device Timeout!"  
    Case 2  
      txtError.Text = "7210 Error!"  
    Case 3  
      txtError.Text = "Non-System Controller Error!"  
    Case 4  
      txtError.Text = "Error: Wrong parameter!"  
    Case 5  
      txtError.Text = "Ctrl-Brk pressed?"  
    Case 6  
      txtError.Text = "Can't allocate DMA buffer"  
    Case 7  
      txtError.Text = "Can't find RQS device"  
  End Select  
  If txtError.Text <> "" Then  
    Beep  
    Tdelay (0.5)  
  End If  
End Sub
```

4.3 IEEE 488.2 PROGRAMMING AND PROTOCOLS

The IEEE-488.2 Standard standardized the data transfer protocol between the GPIB Controller and devices, added an expanded Status Reporting Structure to devices, added a set of Common Commands that a device must respond to and added Controller protocols for handling multiple devices. These changes are described in paragraph A1.2 of the Appendix.

The expanded reporting structure in IEEE-488.2 devices supplies additional information about the device's status. This additional information is contained in extra status registers whose conditions are summarized in bits in the existing Status Register. The Common Commands set bits in enable registers so a condition of a bit will be reported in the Status Register and can generate a SRQ. Review your device's manual for specific information on its capabilities before programming the device.

The Common Command set simplifies the GPIB programmer's job since you can expect any IEEE-488.2 compatible device to respond to the Common Commands and to behave in a defined manner.

4.3.1 Confirming a Device's Presence

IEEE 488.2 compatible devices respond to the *IDN? query with a response (up to 72 characters long) that identifies the device, its serial number and revision. This query-response is a good way to verify that the device is available to your system when starting a program.

A Windows example is:

```
CmdStr$ = "*IDN?"  
L = Len$(CmdStr$)  
ioerr% = ieOutput(DevAddr, CmdStr$, L)  
Rdg$ = String(100, 32)  
ioerr% = ieEnter(DevAddr, Rdg$, 100)
```

4.3.2 Finding Devices

The IEEE-488.2 FindLstn Protocol returns a list of all active devices on the bus that can be addressed as listeners. FindLstn creates a list that then can be used by the other protocols to perform serial polls or to reset all of the devices on the bus.

In the 488-PC2 Command Set, the user defines a list of addresses to be checked and FindLstn returns a list of found addresses.

```
FOR I = 0 to 30           'generates an AddrList  
  AddrList(I) = (Card# * 100 + I)  
NEXT I  
numDevices = 31       'sets number of devices  
ioerr% = ieFindLstn(AddrList, numDevices, ResultList)
```

CAUTION

Presence of a device on the bus that handshakes all data bytes such as a bus analyzer, bus extender, bus expanders or a listen-only device will cause the ieFindLstn command to find all possible device addresses. In this case, the user should supply a list of the actual devices on the bus, not all possible devices, before executing the ieAllSpoll, ieFindRQS or ieReset commands.

RECOMMENDATION

Set the timeout to ≤ 500 milliseconds when using the 488.2 protocols with large device lists to prevent unnecessary program slowdowns.

The user should read the address list after executing the ieFindLstn command to verify presence of all known devices and the absence of any phantom device addresses. Refer to the Command Reference Section for detailed information on the address list format and for directions on passing the address list to the 488.2 Driver.

4.4 ADVANCED PROGRAMMING NOTES

This section covers advanced programming concepts that do not apply to

most GPIB bus applications.

4.4.1 Passing Control

Passing control from one controller to another controller, lets the second controller take charge of all of the devices on the Bus. However, only the system controller (the controller in charge at power turn-on) can control the IFC and REN lines. Passing control is performed by addressing the new controller as a talker and sending it the take control (TCT) command. The following Basic example shows how to pass control to a controller at address 22.

```
REM PASSES CONTROL TO ADDR 22  
C$ = "UNL TALK 22 TCT" 'COMMAND STRING  
CALL ieSend (C$)
```

Compiled Basic and C language programs can use the `iePassCtl` command to pass control to another controller. An example of its use for C is:

```
DevAddr = 22;  
ioerr%=iePassCtl(DevAddr);
```

4.4.2 Direct Memory Access Data Transfer

Direct memory access (DMA) improves system performance in slow computers by allowing external devices to directly transfer information to or from the system memory without processing or program intervention. Newer high-speed processors have eliminated the DMA speed advantage. See Table 2-1.

With DMA transfers, any memory location can source data for the DMA transfer and any read-write memory location can receive data. The 488-PC2 data transfer can be programmed to proceed with or without DMA. When you use DMA, the 488-PC2 provides a number of unique features.

- 1) The ability to run application programs and DMA simultaneously.
- 2) Selection of two DMA operating modes.

- 3) The ability to run IEEE-488 DMA and disk DMA simultaneously.
- 4) The ability to continuously transmit or receive data blocks of up to 64K bytes without processor overhead.

4.4.3 488-PC2 DMA Operations

The 488-PC2 DMA settings allows you to choose from three of the four possible DMA channels. Bits 0 and 1 in the ieInit Setting parameter enable DMA data transfer and select the DMA channel. Refer to Section 1.7.3 to select a DMA channel.

The 488.2 Drivers support single byte and block transfer operating modes. In single-byte-transfer mode, the DMA chip and the system processor share control of the system bus. This allows both DMA and CPU processing to continue simultaneously. In block-transfer mode, transfers are activated by a request for service and continue until the number of bytes specified by the programmer are transferred. In block-transfer mode, control of the system bus is returned to the microprocessor only after all data is transferred.

Demand-transfer mode and Cascade mode are not supported by the 488-PC2. The 488.2 Drivers disable the auto-initialization function and use only the address register increment direction of the DMA controller because of the nature of GPIB data transfer.

Bit 12 of the Setting parameter in the ieInit command determines the handshake speed of the 488-PC2's GPIB interface chip. Setting the bit to "0" selects the slower IEEE 488 Bus handshake times for devices with open collector drivers (Subset E1). Setting the bit to "1" shortens these times for devices with tristate drivers (Subset E2) and speeds up the data transfer.

CAUTION

Using high speed data transfer with open collector drivers will result in data errors.

Bit 14 of the ieInit command Setting parameter determines the DMA operating mode. Setting the bit to "0" lets the DMA operate in single-byte-transfer mode for low DMA rate and high CPU throughput. Setting the bit

to “1” lets the DMA operate in block-transfer mode for high DMA rate and low CPU throughput.

Bit 15 of the ieInit command Setting parameter determines the sequence of driver to handle a DMA. When bit 15 is set to “0”, the driver waits for the completion of DMA before it goes further. When bit 15 is set to “1”, the driver initializes the DMA and then goes away. It is also called BACKGROUND operation. When data transfer is related to other subsequent commands, BACKGROUND operation cannot be used.

The 488-PC2’s DMA operation is quite transparent. Once you select the DMA mode by calling the ieInit routine, all further data transfer proceeds in this mode. However, when you select block-transfer mode, the DMA channel 0 memory refresh may be blocked long enough to corrupt RAM memory content. Do not use block-transfer mode unless you have confidence that the data transfer will be completed within the time limits of memory refresh.

When using background operation, the CPU does not monitor the data transfer. The data string can be terminated by byte count only. You must then handle the data format problem. It is safe to use DMA in single-byte-transfer and non-background operation mode (the default condition). Use of the other modes is not recommended.

Program examples showing how to use DMA data transfers are provided in the language directories on the 488-PC2 DOS Driver Disk.

4.4.4 Using the PC as a Device

There are times when it is desirable to connect a PC to a GPIB Bus and yet not be a bus controller. Examples of this are:

1. Using a PC as a stand alone data acquisition unit where it reports to another computer. Typical applications are engineering work stations and factory test stands.
2. Using the GPIB interface as a data transfer network between computers. One computer is programmed as the bus controller, the other ones as bus devices.

4.4.5 Device Program Considerations

The major program changes to using the 488-PC2 card as a device interface are modifying the card's initialization and then making provisions in the program to input data or commands from the Bus controller. The `ieInit` statement uses three parameters: `IOPort%`, `MyAddr%` and `Setting%`.

The `IOPort%` address is the same regardless of whether the card will be a controller or a device.

The `MyAddr%` parameter should be set to the address assigned to the PC as a Bus device. Bus system controllers normally use addresses 0 or 21 (HP and ICS default to 21). Alternate or additional Bus controllers normally take the next higher addresses i.e. 1 or 22. Bus devices are assigned unused Bus addresses between 0 and 30, i.e., address 4.

The `Setting%` parameter needs to be changed to reflect the non-system controller use of the 488-PC2 card. Bit 5 must be set on for operation in the non-controller mode. Bit 12 may be set on for high speed handshakes or left off for lower speed handshakes. If the Bus controller uses tristate drivers, bit 12 should be set on.

An `ieErrPtr` command should be executed immediately after the `ieInit` command. The `ieErrPtr` command assigns two variables for error number and I/O byte count. An `ieEOL` command should be executed next to set up the input and output terminators that will be used for data transfers with the Bus Controller. You may also want to lengthen the 488-PC2's default time-out period to avoid timing out in case the Bus controller or some other device slows down the Bus.

Sample Device programs are given in the language directories.

4.5 PROGRAM EXAMPLES

4.5.1 Visual Basic Examples

Three Microsoft Visual Basic 4.0 example programs are included in the ICS-win directory. Two of the programs are PC2Cmd and PC2Cmd32. Both command programs are simple interactive programs that give the user control of any GPIB device and demonstrate a wide selection of GPIB commands. The third program, VBdemo, is a simple test program that operates two instruments. Visual Basic 4.0 allows both 16 and 32-bit program examples. Visual Basic 5.0 and later only support 32-bit programs. Contact ICS if you have a problem running the examples on later versions of Visual Basic.

Each program includes an executable version and the source files for the Visual Basic forms as well as the included .bas files. This way you can try the programs out and then examine their contents for example usage of a particular command.

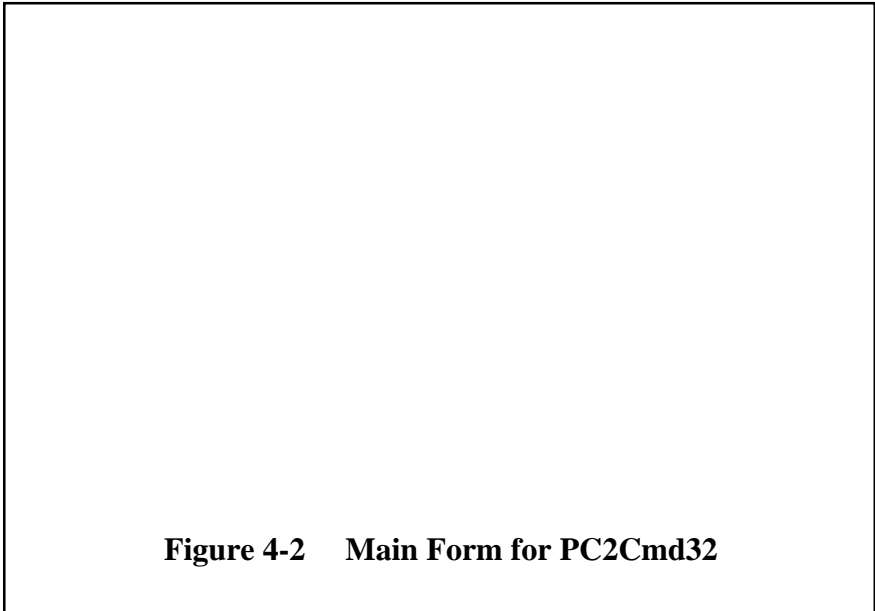


Figure 4-2 Main Form for PC2Cmd32

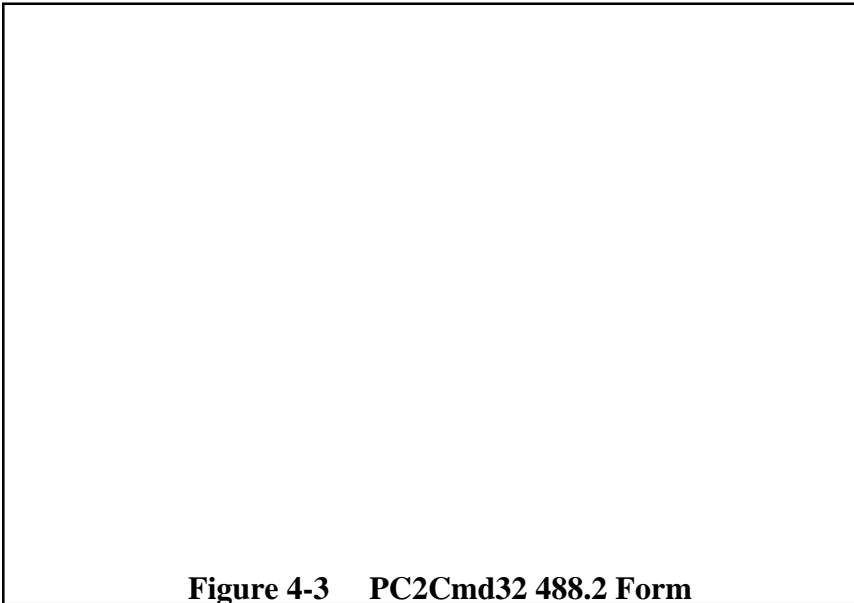
~~PC2Cmd32 is a 32-bit, Visual Basic 4.0 example using the 488-PC2 command set. PC2Cmd is a 16-bit, Visual Basic example using the 488-PC2 command set. Both programs demonstrate how to use a large number~~

of the 488-PC2 commands in Visual Basic. The examples can also be extrapolated into Quick Basic, Professional Basic or C.

Figure 4.2 shows the main form for the programs with all buttons enabled. When the program starts, only the Initialize and Exit buttons are enabled. When the user clicks Initialize, the GPIB REN line is asserted and IFC is pulsed. If the GPIB hardware responds correctly, the remaining program buttons are enabled. To change the address, enter a new value in the Address box (in the upper lefthand corner) and click Set. The remaining buttons invoke a unique command each time they are pressed.

To output a message to a device, the user clicks on the Output button to open a text message box. Type the output message in the text message box and click Okay to send the message. Figure 4-2 shows how the form appears with '*idn?' in the Output Message box, just before clicking Okay. Click Enter to read the response back from the device.

4



The 488.2 button causes 488.2 Form to appear as shown in Figure 4-3. The 488.2 Form has a selection of IEEE-488.2 commands. Each button in the upper row of the form causes an IEEE-488.2 command to occur. Each

button in the lower row executes an IEEE-488.2 controller protocol.

The user can examine or modify the source files to make his own Visual Basic project. The recommended method is to copy all of the files but the project files (.vbp and .exe) files to another directory. Create a new Visual Basic project and use Visual Basic to add the copied files to the new project. There they can be edited, copied or later discarded.

The third Visual Basic example, VBdemo, operates an ICS Model 4861 to generate analog voltages and inputs readings from a Hewlett-Packard DVM. Figure 4-4 shows the equipment setup.

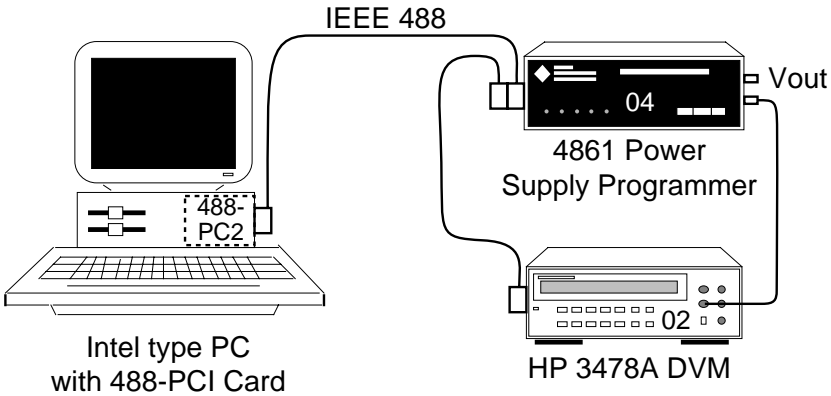


Figure 4-4 VBdemo Program Setup

The program has only two simple forms. The Initialize button on the main forms initializes the 488-PC2 card, the GPIB bus and the instruments. The Run button is an eleven step loop that commands the 4861 to set and output voltage and then enters a reading from the DVM. The program calculates the differences between the commanded value and the reading. The results are displayed on the main form. The addr form is used to change the device address settings.

4.5.2 Visual C++ Example

A Microsoft Visual C++ 5 example program is included in the ICS_GPIB

directory. The program is called VC5_demo and includes an .exe file, the mak and all of the source files. This lets the C programmer run the program and later recompile it on his computer.

VX5_demo is a simple application that writes messages to and reads data back from a device at address 4. It was built as a Microsoft Foundation Class program following the instructions from "Inside Visual C++" by David J. Kruglinski at Microsoft Press. Chapters 1 through 6 and 22 provide the basic instructions for preparing Visual C++ programs. The user should not spend too much time studying the source code files as the Foundation Wizard added most of the code. The important parts are the GPIB calls and the handling of the responses.

Check ICS's web site at <http://www.icselect.com> for a future application note on building a Visual C++ programs.

DOS Language Reference

5.1 INTRODUCTION

This section explains how to install and use the 488.2 Driver routines with DOS programming languages. This section builds on Section 4 which covered GPIB Programming concepts by adding unique instructions for the DOS applications.

5.2 INTERPRETIVE BASIC PROGRAMMING

This section describes how to use the 488.2 Drivers for Interpretive BASIC programming with the 488-PC2 Card. The word BASIC when used in this manual refers to Interpretive BASIC languages or programs. The 488.2 Driver supports Microsoft's GW BASIC, IBM's BASICA and certain versions of HP's Vectra BASIC. Refer to section 2.3.3 for the supported language versions.

5.2.1 BASIC Files and Programs

All of the 488.2 Driver BASIC Driver files and demo programs are located in the BASIC directory on the 488-PC2 DOS Driver Disk. Table 4-1 lists the files supplied to install and run the BASIC Drivers.

5.2.2 Installing the BASIC Drivers

Before programming in BASIC, you have to copy the BASIC Driver to your work disk. Normally this is a directory on your hard disk. The 488-PC2 DOS Driver Disk contains an INSTALL program that copies the BASIC drivers to your system for you.

TABLE 4-1 BASIC DIRECTORY CONTENTS

FILE	DESCRIPTION
PC2INS.COM	488-PC2 Card installation program.
PC2TEST.COM	488-PC2 Card test program.
INSTALL.BAT	BASIC Driver and files installation routine.
DIRGEN.BAT	Creates new directory during BASIC driver installation routine.
PC2-DRVR.BIN	488-PC2 BASIC Driver.
LOADER.BIN	Routine that provides BASIC programs with pointers to BASIC Driver commands.
ICSDECL.BAS	Sample BASIC program header.
PC2BDEMO.BAS	Demo program shows output and enter commands.
PC23478A.BAS	Demo program executes various commands from a menu. Works with HP 3478A DVM or any bus device.
TEST94.BAS	Demo Program that sends strings to the serial and GPIB ports.

To use INSTALL:

1. Be sure that you have made a backup copy of ICS's original disk as described in paragraph 1.5 and are working from the copy.
2. Boot up your computer and type **CD** to get to the root directory. If you are in DOSSHELL, select the root directory.
2. Insert the copy disk into your floppy disk drive.
3. To use INSTALL from drive A, type

> a:\ BASIC\INSTALL C:

where 'a' is the letter for your floppy drive.

4. The INSTALL routine will ask you for the name of the directory to create for the 488 Drivers. Enter PC2 as the default directory name.

The following files are copied to your system:

PC2INS.COM
PC2TEST.COM
PC2-DRVR.BIN
LOADER.BIN
ICSDECL.BAS
README.DOC

5. At the end of the installation, the INSTALL Routine will ask if you have a 488-PC2 card installed. If you answer **y** (yes), the program will verify the PC2 Card switch settings.
6. Remove the PC2 Driver disk.

5.2.3 Programming in Interpreted BASIC

For BASIC programming, the driver is implemented as a series of assembly language subroutine calls. Each subroutine performs a single GPIB function such as outputting data, inputting data, conducting a serial poll, etc. Although these routines are scattered throughout the device driver, their starting addresses are all located in a short table at the beginning of the driver. To access the subroutines, your program needs to include the setup instructions from the ICSDECL.BAS file at the beginning of your program.

The set up instructions use the BLOAD statement to load the assembly routine LOADER.BIN into interpretive BASIC's low memory space. This routine, when called, will return the pointers to GPIB interface routines located within the GPIB device driver. The next statement is a call to the loaded assembly routine with the pointers to the interface functions. Refer to your BASIC's User Manual for in-depth explanation of the BLOAD command and its usage.

```
e.g.  10 CLEAR ,59747! : INIT1=59747! : BLOAD
      "LOADER.BIN", INIT1
      20 CALL INIT1 (IEINIT, IEOUTPUT, IEENTER,
        IEABORT, IEEOL, IEDEVCLR, IELLO, IELOCAL,
        IEPOLL, IEPOLLC, IEPOLLU, IEREMOTE, ISEND,
        IESPOLL, IESTATUS, IETIMEOUT, IETRIGGER,
        IEENTERA, IEOUTPUTA, IEDEVICE, IEERRPTR)
      30 INIT2=INIT1+3
      40 CALL INIT2(IREV)
```

There are several ways to combine your application program with ICSDECL.BAS:

- Write your program, starting at line 100 or higher and then merge ICSDECL.BAS into it. When ready to merge type

> merge "ICSDECL"

You can save the combined program under your application program name by typing:

> save "new program name"

- Or start with a copy of ICSDECL.BAS and add your application on to it. To start type:

> load "ICSDECL"

To avoid overwriting ICSDECL.BAS, immediately execute a save to the new file name by typing:

> save "new program name"

Enter your new program statements at line 100 or higher. When done save as the file as the "new program file" name. The BASIC program should have an Initialization section to initialize the 488-PC2 Card and the GPIB bus and a Main section with the main body of code. Follow the instructions in Section 3 and the examples include in the BASIC directory when writing your program.

5.2.4 Using Interrupts in BASIC

The 488-PC2 has the capability to interrupt the PC's processor when certain events happen. However, Microsoft BASIC in MS-DOS only provides for interrupts from the keyboard, the light pen, the communication port, the game port and on BASIC errors. To use the interrupt capability in BASIC, the 488 Drivers replace Function Key 19 and 20 interrupts and add an error code to inform BASIC that there is an interrupt from the GPIB interface. When these interrupts are enabled, Function Keys 19 and 20 cannot be used in their normal mode or the program will be confused. The three 488-PC2 interrupts are:

TABLE 5-2 PC2 BASIC INTERRUPTS

Type	Interrupt												
ERROR	<p>Error. 488-PC2 firmware detects an error. The error number is ERR-128.</p> <p>The error codes are:</p> <table data-bbox="330 771 1004 990"> <tr> <td>Code</td> <td>Error</td> </tr> <tr> <td>0</td> <td>None</td> </tr> <tr> <td>1</td> <td>Handshake Timeout</td> </tr> <tr> <td>2</td> <td>Interface Error</td> </tr> <tr> <td>3</td> <td>Called IEABORT when the 488-PC2 is not the system controller</td> </tr> <tr> <td>4</td> <td>Invalid command parameters.</td> </tr> </table> <p>F19 Timeout. The GPIB handshake is hung or exceeds the timeout value.</p> <p>F20 SRQ. The GPIB SRQ line is being pulled low by a device.</p>	Code	Error	0	None	1	Handshake Timeout	2	Interface Error	3	Called IEABORT when the 488-PC2 is not the system controller	4	Invalid command parameters.
Code	Error												
0	None												
1	Handshake Timeout												
2	Interface Error												
3	Called IEABORT when the 488-PC2 is not the system controller												
4	Invalid command parameters.												

The Setting% default value of 0 disables all three interrupts. To enable the interrupts, you must set the appropriate Setting% bits to 1 when calling the IEINIT routine.

Setting Bit	PC2 Interrupt
9	SRQ
10	Timeout
11	Error

The BASIC syntax for the interrupt traps are:

ON KEY (20) GOTO 100	'Handle SRQ
ON KEY (19) GOTO 200	'Handle Timeout
ON ERROR GOTO 300	'Handle Error

Note - The interrupt handling of the 488-PC2's software is written for BASICA Version 3.00. Since other BASIC versions may have different traps for the Function Keys and different address of error number, the 488-PC2 GPIB interrupt handling routines may not work for different versions of BASIC. Check your BASIC manual for the trapping and error address compatibility before using the interrupts

5.3 QUICK BASIC PROGRAMMING

This section describes how to use the 488.2 Driver libraries for Quick Basic programming with the 488-PC2 Card. The same instructions also apply to the other compiled Basic languages - Professional Basic and Visual Basic for DOS.

5.3.1 Files and Programs

All of the Quick Basic files and Demo programs are located in the QB directory on the 488 Driver Disk. Visual Basic for DOS files are located in the VB DOS directory. Professional Basic files are located in the PDS7 directory. The directories include libraries which are linked to the user's Basic program at the compile time. The libraries contain all of the IEEE-488.2 commands and protocols.

The QB directory also contains two 'include' files which reference all of the library file commands, global variables and default values. The QB directory also includes three demo programs which show the user how to control instruments, how to use interrupts and how to program the PC as a device. Table 5-3 lists all of the files in the QB directory.

5.3.2 Installing the Quick Basic Libraries

Before programming in Quick Basic, you have to copy the Quick Basic library files to your working directory. Normally this is a directory on your hard disk. The 488-PC2 DOS Driver Disk contains an INSTALL program that copies the necessary library files to your system for you.

To use INSTALL:

1. Make a backup copy of ICS's original disk as described in paragraph 1.7 and you are working from the copy.
2. Boot up your computer and type **CD** to get the root directory. Using the name of your existing Quick Basic directory, type **CD **directoryname**** to get to the Quick Basic directory. If you are in DOSHELL, select the Quick Basic directory.
2. Insert the copy disk into your floppy disk drive.

TABLE 5-3 QB DIRECTORY CONTENTS

FILE	DESCRIPTION
PC2INCL.BAS	An include file that contains all of the 488 Driver declarations.
PC2COM.BAS	An include file that contains the Interface default contents, settings and common variables.
PC2QBDOS.LIB	488 Driver library for making Quick Basic *.EXE programs.
PC2QBDOS.QLB	488 Driver library for running programs in the Quick Basic environment.
PC2QBDOS.OBJ	488 Drivers for creating other libraries.
PC2QBDEM.BAS*	Demo program that shows output and enter commands.
TST4891.BAS*	Advanced version of PC2QBDEMO.
PC2QBINT.BAS	Demo program that shows who to use SRQs to read data.
EVENTTST.BAS*	Demo program that shows how to write an interrupt handler and test for SRQ, Error or Timeout interrupts.
README.DOC	QB Directory readme file.

* .EXE and .OBJ program versions are included on the disk.

3. To install the files from drive A type

>a:\QB\INSTALL C: where 'a' is the floppy disk drive.

4. The INSTALL routine will ask you for the name of the directory it will create for the library files. Enter PC2QB as the default directory name.

The following files are copied to your system:

PC2QBDOS.LIB	}	Quick Basic library files.
PC2QBDOS.QLB		
PC2QBDOS.OBJ		
PC2INCL.BAS	}	Include file modules that contains all of the subroutines, and global vari- ables shared between the main mod- ule and the libraries.
PC2COM.BAS		
README.DOC		Documentation file.

5. If you want to use or run any of the same programs on your hard disk, they can be copied at this time. To copy a file to your new directory type

```
>copy a:\QB\filename c:\QB\PC2QB\
```

where PC2QB is your new directory name.

6. When done, remove the 488.2 Driver Disk.

5.3.3 Programming in Quick BASIC

The easiest way to program in Quick Basic is to use Microsoft's Quick Basic editor. Start the editor by typing **QB**. DOSSHELL users double click on the QB.EXE file. A typical Quick Basic program has the following organization:

```
COMMENTS AND TITLE  
SUBROUTINE DECLARATIONS  
INCLUDE STATEMENTS  
VARIABLE DEFINITIONS AND VALUES  
PROGRAM STATEMENTS AND CALLS  
SUBROUTINES (IF ANY)  
END
```

The easiest way to do a Quick Basic program is to open or copy one of the supplied demo programs (such as TST4891.BAS), rename it by doing a 'save-as' and then modify it. This gives you the structure of an existing program and will takes care of the include files and declarations.

5.3.4 Declarations and Includes

Quick Basic uses declarations at the beginning of a program to reference subroutines. A typical declaration is

```
DECLARE SUB ieEnter (DEVADDR%, INSTRING$)
```

There must be a separate declaration statement for each subroutine or library command used in the program. To save you having to enter individual declarations in each program, the 488 Driver provides two include files which contains the declarations for all of the PC2 IEEE-488 commands and global variables. The Quick Basic commands for linking these include files with your program are:

```
'$INCLUDE: 'PC2INCL.BAS'  
'$INCLUDE: 'PC2COM.BAS'
```

The include statement starts with a single quote mark or REM and should be the first statement in your program after declarations and variable definitions. Because PC2COM.BAS defines global variables, it is best to place declaration statements for other subroutines before the above include statements.

5 The program initialization, body and subroutines depend upon the nature of your program. Refer to the examples in the QB directory and to Section 4 for programming ideas. At the end of the program, use the ieClose command to cleanup the GPIB interface and restore the host environment.

```
CALL ieClose ( )  
END
```

When you have finished writing your program, save it. This step is very important as you could loose the program if the computer hangs up while running the program and you have to reboot the computer.

To run the demo program from the command line type

```
> QB PC2QBDEM /L PC2QBDOS
```

To run your program, substitute your program's name for PC2QBDEM in the above line

5.3.5 Parameter Passing

In Quick Basic, all parameters used in CALL statements may be passed as a variable name or as a numeric or literal value. For example the commands

```
DVM% = 4  
CALL ieTrigger (DVM%)
```

and

```
CALL ieTrigger (04)
```

are equivalent. Quick Basic variables and constants are defined in the Command Reference Section.

5.3.6 Using Interrupts in Quick Basic

The 488.2 Drivers recognize three events in Quick Basic - Error; SRQ and Timeout. The event meanings are listed in Table 5-4.

TABLE 5-4 QUICK BASIC INTERRUPTS

Type	Interrupt
ERROR	488 Driver detected error. The error type is determined by calling the STATUS command.
SRQ	GPIB bus SRQ signal asserted by a device who needs service.
TIMEOUT	GPIB handshake is hung or a device response exceeded the set timeout value.

To use the PC2 interrupt capability in Quick Basic, the user inserts the following commands into his program:

```
ON UEVENT GOSUB <user subroutine>  
UEVENT ON
```

The default SETTING% value of 100 HEX disables all interrupts. To enable the interrupts, you must set the appropriate SETTING% bits to 1 before calling IEINIT. The bits are:

SETTING% BIT	PC2 INTERRUPT
9	SRQ
10	Timeout
11	Error

Note - An SRQ interrupt also requires that hardware interrupts be enabled. Refer to section 1.7.4 for enabling hardware interrupts.

The following example enables the timeout interrupt by setting bit 10 on.

```
SETTING% = 500 HEX      'Enables Timeout  
CALL ieInit (IOPORT%, MYADDR%, SETTING%)
```

When an interrupt occurs in Quick Basic, the program will go to the service routine specified in the ON UEVENT GOSUB () statement. If more than one event is enabled, the user will have to test the Event Status and branch to the correct service routine. A suggested interrupt routine is

```
'EVENT ROUTINE  
CAUSE% = ieEventStat( )  
IF CAUSE% ≥ 8 THEN  
    PRINT 'UNEXPECTED EVENT STATUS'  
    STOP  
ELSE  
IF CAUSE% AND 4 THEN  
    GOSUB SERVICE_ERR  
END IF  
IF CAUSE% AND 2 THEN  
    GOSUB SERVICE_TIMEOUT  
END IF  
IF CAUSE% AND 1 THEN  
    GOSUB SERVICE_SRQ  
END IF  
END IF  
RETURN
```

Figure 5-2 Quick Basic Interrupt Handling Routine

5.4 PASCAL PROGRAMMING

This section explains how to use the 488 Drivers for Pascal Language programming with the 488-PC2 Card. Use the 488 Driver Borland libraries with Borland Turbo Pascal versions 4.0. Use the 488 Driver Microsoft libraries with Microsoft Pascal version 4.0. Both Pascal libraries provide all of the IEEE 488.1 functions necessary to program a GPIB system in the Pascal language.

5.4.1 Pascal Libraries and Programs

The Pascal libraries and demo programs are located in two directories on the 488 Driver Disk. One directory (BTPASCAL) supports Borland Turbo Pascal programs and the other directory (MSPASCAL) supports Microsoft Pascal programs. The BTPASCAL directory contains a library file, an object file for IEEE functions, demo programs and a readme file. The MSPASCAL directory contains a Pascal library file, an include file, demo programs and a readme file. Tables 5-5 and 5-6 list the contents of each directory.

5.4.2 Installing the Pascal Libraries

Before programming in Pascal, you have to copy the appropriate Pascal library files to your working directory. Normally this is a directory on your hard disk.

The Pascal directories on the 488 Driver Disk contain all of the Pascal callable IEEE 488 library functions for application programs written in Microsoft Pascal or Borland Turbo Pascal. Copy the complete contents of the appropriate directory to the working directory on your hard disk. You can use the demo programs as a starting point for your program and delete them when done.

To INSTALL the libraries:

1. Make a backup copy of ICS's original disk as described in paragraph 1.7 and you are working from the copy.

TABLE 5-5 BTPASCAL DIRECTORY CONTENTS

FILE	DESCRIPTION
GPIB.TPU	A library file that has all of the IEEE- 488.1 functions required for Borland Turbo Pascal programs.
GPIB.PAS	Source file for GPIB.TPU
PC2TPDRV.OBJ	An object file that contains all of the IEEE-488.1 functions for Turbo Pascal programs.
PC2_34.PAS	Demo program that transfers data to and from a ICS Model 4834 Serial Interface. Program also demonstrates serial and parallel polling. Disk contains an .exe version.
PC2INT.PAS	Uses SRQs to input data. Disk contains an .exe version.
PC2DEV.PAS	Shows how to use the PC2 card as a bus device. Disk also contains an .exe version.
DEMOLIB.TPU	Library file used by the demo programs.
DEMOLIB.PAS	Source file for DEMOLIB.TPU
README.DOC	This file contains latest information about the Borland libraries and commands.
PC2FUN.PAS	Demo program
EX1_TB.PAS	
EX4_TB.PAS	Demo programs.

2. Boot up your computer and type **CD** to get the root directory. Using the name of your existing Pascal or project directory, type **CD directoryname** to get to the Pascal directory. If you are in DOSHELL, select the appropriate Pascal or project directory.
3. Insert the copy disk into your floppy disk drive.
4. Copy the desired library files onto your directory. Directory **BTPASCAL** contains the Borland Turbo Pascal GPIB library. Directory **MSPASCAL** contains the Microsoft Pascal GPIB library and an include file.

TABLE 5-6 MSPASCAL DIRECTORY CONTENTS

FILE	DESCRIPTION
MSPPC2.INC	An include file that defines the IEEE 488.1 functions.
PC2MSPDV.LIB	Library file that contains all of the GPIB functions for Microsoft Pascal programs.
PC2_34.PAS	Demo program that transfers data to and from a ICS Model 4834 Serial Interface. Program also demonstrates serial and parallel polling. Disk contains an .exe version.
PC2INT.PAS	Uses SRQs to input data. Disk contains an .exe version.
PC2DEV.PAS	Shows the PC2 card as a bus device. Disk also contains an .exe file.
DEMOLIB.PAS	A Pascal unit file used by the above demo programs.
DEMOLIB.INT	The interface portion of the DEOMLIB
DEMOLIB.IMP	The implementation portion of the DEMOLIB
README.DOC	This file contains latest information about the Microsoft libraries and commands.

5.4.3 Programming In Pascal

Each Pascal language library contains routines that provide IEEE-488 language extensions for your Pascal language programs. Each routine includes bus error checking. The routines also check parameter values to insure that appropriate bus protocol is followed.

Each 488 Driver Pascal function specifies parameters that it must receive from the program or parameters that it passes back to the program. These parameters are shown in parentheses following the function name.

1. The order, number and type of variables passed to the functions must be exactly as shown for each function.
2. Passed parameters can be variables or constants. However, the passed parameter must be a pointer of a character string when string I/O is involved.

Most of the Pascal commands return an integer that equals zero if there is no error. Otherwise it equals the error number. The error number and meanings are listed under `ieErrPtr` in the Command Reference section. To use the commands without checking for errors just put them in the program as stand alone functions.

i.e., `ieOutput (DVM, SetupString);`

To check for errors, create an integer and set it equal to the command.

i.e., `ioerr : integer`
`ioerr := ieOutput (DVM, SetupString);`

5.4.4 Turbo Pascal Programming

The 488 Driver functions for Turbo Pascal are contained in the file `PC2TPDVR.OBJ`.

The file `GPIB.TPU` is a precompiled unit file of the source file `GPIB.PAS`. Units are the basis of modular programming in Turbo Pascal. They are used to create libraries that can be included in Turbo Pascal programs without having to include the source code.

5 The file `GPIB.PAS` has the definitions and declarations needed for all of the GPIB functions. The user can copy the definitions into the program or use the statement `'uses GPIB'` to include the definitions in the program. The definitions will be included as if they were on a single line. The functions contained in the `GPIB.TPU` file are indicated by the P Compatibility column in Table 3-1.

To use the 488.2 Driver functions include the statement `'uses GPIB'` at the beginning of your Turbo Pascal program. Files `GPIB.TPU` and `PC2TPDRV.OBJ` must be in the same directory as your Turbo Pascal program. Develop the program with the desired 488 Driver functions and run the program.

5.4.5 Microsoft Pascal Programming

The 488 Driver library procedures and functions for Microsoft Pascal (PC2MSPDV.LIB) can be thought of as an extension to the Pascal libraries that Microsoft offered. The extension consists of those procedures and functions indicated by the P Compatibility column in Table 3-1. These functions and procedures are fully compatible with the IEEE 488.1 Standard.

Using the 488 Driver procedures and functions requires the following steps:

1. Define the 488 Driver functions and procedures that will be called in the program by including the following include statement in your program. Type the include statement exactly as it is shown.
(\$INCLUDE:'MSPPC2.INC')
2. Develop the program with the 488 Driver functions that you need.
3. Compile the source code with the Microsoft Pascal compiler.
4. Link the OBJ code with the 488 Driver library PC2MSPDV.LIB and any other necessary libraries.
5. Execute the program.

5.5 C/C++ LANGUAGE PROGRAMMING

This section explains how to use the 488 Drivers for C/C++ Language programming with the 488-PC2 Card. Use the 488 Driver Microsoft libraries with Microsoft Quick C version 2.5, Microsoft C versions 5.0 and 6.0, Microsoft C/C++ version 7.0 and Visual C++ programs. Use the 488 Driver Borland libraries with Borland Turbo C++ and Borland C/C++ versions 2.0, 3.0 and 4.0 language programs. Refer to paragraph 2.3 for a complete list of the supported C languages and versions.

5.5.1 C Library Files and Demo Programs

The C libraries and demo programs are located in two directories on the 488 Driver Disk. One directory (MSC_CPP) supports Microsoft C programs and the other directory (BORC_CCP) supports Borland C programs. Both directories contain C libraries for various memory sizes, an include files and some demo programs. The include file PC2INC.H references the Driver commands, sets global variables, and establishes the default values. Tables 5-7 and 5-8 list the contents of each directory.

5.5.2 Installing the C Libraries

Before programming in C, you have to copy the appropriate C library files to your working directory. Normally this is a directory on your hard disk.

The C directories on the 488 Driver Disk contain all of the C callable 488 Driver library functions for application programs written in Microsoft C or Borland C. All library files have the .LIB extension. The first letter of the file name specifies the memory model size. The next three letters specify Microsoft or Borland compilers.

<u>Prefix</u>	<u>Size</u>	<u>Code</u>	<u>Company</u>
S	Small	MSC	Microsoft
M	Medium	TBC	Borland
L	Large		

To INSTALL the libraries:

1. Make a backup copy of ICS's original disk as described in paragraph 1.7 and you are working from the copy.

TABLE 5-7 MSC_CPP DIRECTORY CONTENTS

FILE	DESCRIPTION
PC2INC.H	A header file for accessing the commands in the C libraries. Required for calling 488 Driver functions.
SMSCPC2.LIB	Library for small memory model applications.
MMSCPC2.LIB	Library for medium model applications.
LMSCPC2.LIB	Library for large memory model applications.
PC2CINT.C	This is a demo program for MS C ver 7.0 that uses a 488-PC2 card to send commands to an ICS 4891 Power Supply Programmer to generate DC voltages and uses an HP 3478 DVM to read the voltages. Shows use of interrupts to read in data from the DVM. Disk also contains an .EXE file.
PC2_34.C	This is a demo program for MS C ver 5.1 that uses as 488-PC2 Card to write and read back test strings from ICS 4834 GPIB-Serial Interface. Shows how to use secondary addresses. Disk also contains .EXE file.
PC2DEV.C	This is a demo program for MS C ver 5.1 that shows how to use the 488-PC2 card as a device. Disk also contains .EXE file.
README.DOC	This file contains latest information about the Microsoft libraries and commands.

2. Boot up your computer and type **CD** to get the root directory. Using the name of your existing C or project directory, type **CD **directoryname**** to get to the C directory. If you are in DOSHELL, select the appropriate C or project directory.
- 3.. Insert the copy disk into your floppy disk drive.
4. Copy the desired library files onto your directory.
Directory MSC_CPP contains the Microsoft C libraries.
Directory BORC_CPP contains the Borland C libraries.

TABLE 5-8 BORC_CPP DIRECTORY CONTENTS

FILE	DESCRIPTION
PC2INC.H	A header file for accessing the commands in the C libraries. Required for calling 488 Driver functions.
STBCPC2.LIB	Library for small memory model applications.
MTBCPC2.LIB	Library for medium memory model applications.
LTBCPC2.LIB	Library for large memory model applications.
PC2CINT.C	This is a demo program that sends commands to an ICS 4891 Power Supply Programmer to generate DC voltages and uses an HP 3478 DVM to read the voltages. Shows use of interrupts to read in data from the DVM. Disk also contains an .EXE file.
PC2_34.C	This is a demo program for Borland C ver 2.0 that writes and reads back test strings from ICS 4834 GPIB-Serial Interface. Shows how to use secondary addresses. Disk also contains .EXE file.
PC2_DEV.C	This is a demo program for Borland C ver 2.0 that shows how to use the 488-PC2 card as a device. Disk also contains an .EXE file.
BOR_CPP	A sub-directory containing sample C++ programs and files.
README.DOC	This file contains latest information about the Borland libraries and commands.

5**5.5.3 Programming in C/C++**

Each 488 Driver C function specifies parameters that it must receive from the program or parameters that it passes back to the program. These parameters are shown in parentheses following the function name. The order, number, and type of variables passed to the functions must be **exactly** the same as the syntax shown for each functions.

Most of the C commands return an integer that equals zero if there is no error. Otherwise it equals the error number. The error number and meanings are listed under ieErrPtr in the Command Reference section. To use the commands without checking for errors just put them in the program as stand alone functions.

ieOutput (DVM, SetupString);

To check for errors, create an integer and set it equal to the command.

```
int CmdErr;  
CmdErr = ieOutput (DVM, SetupString);
```

5.5.4 Using the 488.2 Driver Library Functions

The 488 Driver Libraries for Microsoft C/C++ (SMSCPC2.LIB, MMSCPC2.LIB, LMSCPC2.LIB) can be thought of as an extension to the SLIBC.LIB, EM.LIB and LIBH.LIB libraries that Microsoft Corp. offers.

The 488 Driver Libraries for Borland C/C++ (STBCPC2.LIB, MTBCPC2.LIB, LTBCPC2.LIB) can be thought of as an extension to the libraries that Borland International offers.

Each C library contains all of the functions listed in Table 3-1. These functions allow the PC to operate as an IEEE 488.2 Bus Controller with both interfaces or as a GPIB device when used with ICS's 488-PC2 card. Both C and C++ languages use the same C library. The PC2INC.H header file will automatically handle the interface for the two languages.

Using 488 Driver library functions requires the following steps.

1. Include PC2INC.H in all programs which use PC2 Driver.
2. Develop the program with the 488 Driver functions that you need.
3. Compile the source code with the appropriate memory model, e.g. for small Model C programs use the following commands:

Microsoft C:	cl	/c	/AS	prog.c
Borland C:	bcc	- c	- ms	prog.c

4. Link the OBJ code with the necessary libraries. The 488 Driver functions are stored in the following libraries according to different memory models.

Microsoft C:

SMSCPC2.LIB 488 Driver library for S model
MMSCPC2.LIB 488 Driver library for M model
LMSCPIC2.LIB 488 Driver library for L model

e.g.: Link prog SMSCPC2.LIB for small model

Borland C:

STBCPC2.LIB 488 Driver library for S model
MTBCPC2.LIB 488 Driver library for M model
LTBCPC2.LIB 488 Driver library for L model

5. Prepare and Execute the program.

Note: The PC2INC.H header file contains ANSI function prototypes for the library functions. If the header file is included then the compiler will detect most parameter errors.

Compile and run the program.

5.5.5 Using Interrupts in C/C++

The 488.2 Driver adds the capability of handling of SRQ, time out and error conditions. With these functions, the C programmer may design his own handler and install or remove it seamlessly. When installed, the user defined handler will become part of the PC2 driver interrupt processing sequence and will be activated when an abnormal condition occurs. To use interrupts, do the following:

- 1) Design your own handler as a separate C function which has type far, return void and takes no argument. Follow all recommended notes of the ieServiceEnable function in the Command Reference section.
- 2) In the main function, do all necessary initialization for your program.

- 3) Call `ieInit` with the selected event and IRQ channel. Note - an SRQ interrupt also requires that the hardware interrupts be enabled. Refer to section 1.7.4 for setting hardware interrupts.
- 4) Call `ieServiceEnable` to install your handler.
- 5) Start your main processing section. From this point, your handler will be triggered by the intended event until you explicitly call `ieServiceDisable` or `ieClose`.

Example interrupt source code is provided in both C directories.

5.5.6 Using the PC as a Device

There are times when it is desirable to connect a PC to a GPIB Bus and yet not be a bus controller. The major changes to use the 488-PC2 card as a device interface require modifying the card's initialization and input/output string terminators. The initialization function, `ieInit()`, has three parameters: '`ioport`', '`myaddr`', and '`setting`'. The `ioport` address is the same regardless of whether the card will be a controller or device. The only reason that the `ioport` address would change is if the address switch setting is changed.

The `myaddr` parameter is the address assigned to the PC as a Bus device. Bus controllers normally use addresses 0 or 21 (HP preference). Alternate Bus controllers normally take the next higher addresses, i.e. 1 or 22. Bus devices are assigned unused Bus address 1 to 20 and 22 to 30. Address 31 is an invalid address and cannot be used.

The `setting` parameter must be changed to reflect the non-system controller use of the 488-PC2 card. Bit 5 must be set to '1' for operation in the device mode. Bit 12 may be set on for high speed handshakes or left off for lower speed handshakes. If the Bus controller uses tristate drivers, bit 12 should be set on.

The '`outeol`' and '`ineol`' parameters in the `ieEol()` function that set the output string and the input string terminators need to match the Bus Controller's format for output and input string terminators. They must be altered to properly handle the `ieEol` parameter setting of string terminators of the device. That is '`outeol`' must match the bus controller's expected input string terminator. '`ineol`' must be set to match the bus controller's output string terminators.

The demo program PC2dev.C makes 488-PC2 card function as a device that sends out and receives data from the bus controller. As a device, the 488-PC2 will not be able to issue 'REN', 'IFC' or 'ATN.' The program polls the address status register of the 7210 controller IC on the 488-PC2 card to see whether MLA, my listen address or MTA, my talk address has been sent to the IEEE bus by the bus controller, and then the PC does a data transfer.

The demo program uses a FIFO ring buffer data structure to do the data transfer. The head of the ring buffer points to the last string that has been read from the buffer and output to the bus controller. The tail of the ring buffer points to the string that the PC user has most recently entered from the keyboard. The user can press the RETURN key to enter a string at any time after the device address of the 488-PC2 card has been entered. If the 488-PC2 receives its talk address and the ring buffer is empty, the program will prompt the user to enter a string and then send it to the bus. If return is pressed and a string is entered while the 488-PC2 is not addressed as a talker, it will output an SRQ to the bus and respond to a serial poll with a hex 41 character.

CAUTION

The 488-PC2 can be switched from listener to talker directly, but it cannot be switched from a talker to listener. The bus controller has to send a UNT message to the 488-PC2 before it makes the 488-PC2 a listener.

An alternate approach to data transfer is using the address status change interrupt instead of polling the 7210's address status register. Use the ieInit function to select a desired IRQ level, and enable the ADSC interrupt by writing a 1 to bit position 0 of 7210 Interrupt Mask 2 Register. An interrupt service routine needs to be created that will either do data input upon sensing its listen address or a data output upon sensing its talk address. See paragraph 5.5.5.

5.5.7 C++ Demo Program

The BOR_CPP directory contains a C++ demo program labeled PC2_34.cpp. This program demonstrates how to control and pass data to a GPIB device which is an ICS Model 4834 GPIB to Serial Interface. In the program, the 4834 configuration is set to 'T1' which puts the 4834 in local loopback test mode. The program sends out data strings to the 4834 and then reads the data back from the same serial channel.

Command Reference

6.1 INTRODUCTION

This section describes the 488.2 Driver commands and functions. The description for each command or function includes the command and function syntax, parameter types, responses where appropriate, and usage examples for Pascal, Interpretive BASIC, Quick Basic, C and C++. Bus activity is described where appropriate to explain the command.

The commands are organized on individual pages and labeled in mixed capital and lower case letters for legibility. The command syntax for each language is also shown in mixed letters except for Interpretive BASIC which is shown in all capitals. When Interpretive BASIC and Quick Basic (Compiled Basic) have the same syntax, they are shown together on the same line and labeled BASIC/QB

6.2 COMMAND CONVENTIONS

Table 6-1 lists the common conventions used in the 488.2 Driver commands. Constants are shown all upper case.

6.3 CONSTANTS AND DEFAULT VALUES

Some of the command parameters remain constant throughout most programs. To simplify the users programming tasks these constants have been placed in include files in each language's directory. Table 6-2 lists the constants and their default settings for each language.

TABLE 6-1 COMMON COMMAND PARAMETERS

Parameter	Description
< >	a required parameter.
[]	an optional parameter.
DevAddr	GPIB device address which may be one of three forms: pp = primary addresses 0 to 30 for Interpretive BASIC and Pascal pp [ss] = primary addresses 0 to 30 and optional secondary address of 00 to 31 for Quick Basic, C and C++. This form does not support secondary addresses for primary address 0 CardID [pp [ss]] = CardID value of 7 to 4, primary address of 00 to 30, and optional secondary address of 00 to 31 for MS Windows.
PC2ADDR	type definition for PC2 DLL device address. The format is CardID [pp [ss]].
CardID	an integer which identifies the logical 488-PC2 card.
PrimAddr	an integer which represents primary address field (pp).
AddrList	One dimensional array whose entries are device addresses and is terminated by END_ADDR . Format is pp [ss].
PrimList	an address list composed of only primary addresses.
END_ADDR	a constant (3131) used to terminate an address list.
n	a numeric integer.
string	a series of ASCII characters.
data	a series of binary bytes or ASCII characters.
<nl>	a required command terminator sequence. For IEEE-488.2 commands, <nl> is a line feed (LF), EOI asserted on the last character or a combination of both events.

6

TABLE 6-2 DEFAULT CONSTANT VALUES

Language	BASIC	Quick Basic	C/C++
File name	ICSDECL.BAS	PC2COM.BAS	PC2COM.H
488.2 DRIVER Constants			
CardID	-	-	-
MyAddr%	21	21	21
NoAddr%	-1	-1	-1
END_ADDR%	—	3131	3131
IOPort%	&H2E1	&H2E1	0x02E1
Setting%	&H0	&H100	0x1100
Time%	10000	10000	10000
OutEOL%	0	0	0
OutEOLStr\$	CHR\$10	CHR\$(10)	CHR\$(10)
InEOL%	0	0	0
InEOSByte%	10	10	10
SRQServ	—	1	1
TimeOutServ	—	2	2
ErrorServ	—	4	4
TC_ASYNC	—	0	0
TC_SYNC	—	1	1
TC_END	—	2	2

64 GPIB COMMAND TERMINATORS

IEEE-488.2 commands can be terminated by a line feed (LF), EOI asserted on the last character or a combination of both events. Older IEEE-488.1 devices may recognize a carriage return (CR) as the command terminator. The 488.2 Drivers default output terminator is line feed (LF) with EOI

TABLE 6-2 DEFAULT CONSTANT VALUES (CONT.)

	BASIC	Quick Basic	C/C++
Other Constants			
LF		10	
CR		13	
YKEY		89	
NKEY		78	
ESCKEY		27	
RTNKEY		13	
Strings			
TALKS		"UNL UNT MLA TALK"	
LISTENS		"UNL UNT MTA LISTEN"	
UNTS		"UNT"	
UNLS		"UNL"	
SEC0DATAS		"SEC 0 DATA"	
SEC0S		"SEC 0"	
SPACE80\$		STRING\$(80)	

6 asserted on the last byte. The output terminator is changed by changing the '**OutEOLStr**' parameter in the **ieEol** command.

Input GPIB messages can be terminated by a carriage return (CR), line feed (LF), by EOI asserted on the last character or by a combination of all two or three events. An IEEE 488.2 device always appends the line feed (LF) character to an ASCII message and asserts EOI on the last byte of the message. The 488.2 Drivers default input termination is line feed (LF) and /or EOI asserted. Pure binary data with EOI asserted can be terminated by a byte count with the **ieEnterB** command. The input terminator is changed by changing the '**InEOSByte**' parameter in the **ieEOL** command.

6.5 RETURN VALUES AND ERROR CODES

The C/C++, Visual Basic and Pascal language commands or functions return a value which contains the function result and/or an error code. The error code is zero if there were no errors, non-zero otherwise. The error codes and definitions are listed below in Table 6-3. See paragraph 4.2.12 for how to handle command errors.

Interpetive BASIC and Quick Basic commands and functions that do not directly return a value can have their error codes read by calling the ieStatus command. The ieStatus command returns the error code for the last called command. Refert to the ieStatus page for the correctsyntax and an example of how to use the command with the Basic languages.

TABLE 6-3 COMMAND ERROR CODES

Code	Error Description
0	none
1	Handshake Timeout
2	Interface Error
3	Called a controller related function when the PC2 is not the system controller
4	Invalid passed parameter(s)
5	Keyboard break key used to quit handshake timeout
6	Failed to allocate DMA buffer
7	Could not find a device requesting service
8	No acceptor present on the bus
9	Incompatible hardware. Could not find 488-PC2 Revision 1 Card or 4818 Module.
10-255	Reserved

6.6 COMMAND REFERENCE

The remainder of this section contains a detailed description of each 488.2 Driver command or function. The commands and functions are listed alphabetically. Immediately following the command references are three Time Utility functions that the user will find convenient for creating time related programs.

Command or function examples are shown for Interpretive BASIC, Quick Basic (Compiled Basic) and C languages where appropriate. The command syntax for each language is also shown in mixed letters except for Interpretive BASIC which is shown in all capitals. When Interpretive BASIC and Quick Basic (Compiled Basic) have the same syntax, they are shown together on the same line and labeled BASIC/QB

Pascal programmers may use the C example as a guide and should refer to the example programs in the Pascal language directory.

Purpose:

This command clears all device interfaces and causes the Interface to take control of the bus.

Syntax:

BASIC/QB - CALL IEABORT

Pascal - ieAbort: *integer*;

C, C++ - *int* ieAbort()

Win DLL - ieAbort(*PC2ADDR* CardID)

Return Value:

Error Code for Pascal, C/C++ and Win DLL commands

Bus Activity:

IFC is pulsed for a minimum of 100 microseconds

REN is set

ATN is cleared

Comments:

ieAbort can only be called when in system controller mode. An error will occur otherwise. Refer to ieInit for the system controller setting.

Examples:

For BASIC/QB CALL IEABORT

For C/C++ if(ieAbort()==ERR_NO4882) {...};

For Win DLL ieAbort(7)

ieAllSpoll

Purpose:

This command performs 488.2 Serial Poll All Devices protocol. Polls the devices in the AddrList and returns the responses in a separate list.

Syntax:

Quick Basic - CALL ieAllSpoll (ADDRLIST%(), RESPLIST%())

C, C++ - void ieAllSpoll(int far *AddrList, int far *RespList)

Win DLL - ieAllSPoll(int CardID, PC2ADDR* DevList,
int numDevices, WORD* RespList);

Return Value:

Error Code for Win DLL commands

Comments:

This command causes the 488-PC2 to serial poll all devices listed in the **AddrList**. Responses are placed in a one dimensional integer array **RespList** on a one-to-one match with the **AddrList**. **RespList** length must be greater than or equal to **AddrList**. Response values are 0 to 255.

If the device fails to respond to the serial poll, the response value is a 16 bit negative value with bit 15 true. Bits 14-8 contain the error code and bits 7-0 contain the devices response if any.

Examples:

For Quick Basic

```
DIM ADDRLIST% (3)
DIM RESPLIST% (2)
ADDRLIST% (1) = 04
ADDRLIST% (2) = 0501
ADDRLIST% (3) = 3131      'marks end of list
CALL ieAllSpoll (ADDRLIST%(), RESPLIST%())
```

ForC/C++

```
int AddrList[] = {4,506,3131};  
int RespList[3];  
ieAllSpoll(AddrList, RespList);
```

ieClose

Purpose:

This command cleans up the 488.2 Drivers and restores system environment.

Syntax:

Quick Basic - `CALL ieClose()`

C, C++ - `void ieClose(void);`

Return Value:

none

Bus Activity:

None

Comments:

ieClose should be called as the last command in the program and before calling ieInit a second time. If ieInit is called more than once in a program then ieClose must be used before the second and subsequent ieInits to clean up the prior ieInit.

Examples:

For Quick Basic

```
CALL ieClose()
```

For C/C++

```
ieClose();
```

Purpose:

Returns a bus device's internal logic to a known device-dependent state. It can be sent to all addressed listeners or to a specific device.

Syntax:

BASIC/QB - CALL IEDEVCLR (DEVADDR%)

Pascal - ieDevClr (DevAddr: *integer*) : *integer*;

C, C++ - *int* ieDevClr (*int* DevAddr)

Win DLL - ieDevClr (PC2ADDR DevAddr)

Return Value:

Error code for Pascal,C/C++ and Win DLL commands.

Comments:

If **PrimAddr** is 0 to 30, then the driver sends a Selective Device Clear command to the device. If **DevAddr** is -1 (**NoAddr**) or 31, then the driver outputs a universal Device Clear command to the bus.

Examples:

For BASIC/Quick Basic

```
DEVADDR% = 4
```

```
CALL IEDEVCLR (DEVADDR%) 'clears device 4
```

```
CALL IEDEVCLR (NOADDR%) 'outputs DCL to the bus
```

For C/C++

```
ieDevClr (4); /* clears device 4 */
```

```
ieDevClr (NoAddr); /* outputs DCL to the bus */
```

ieDevice

Purpose:

This command maps the PC printer or COM port to the GPIB port so that data directed to the printer or com port is outputted to a device on the bus.

Syntax:

BASIC/QB -CALL IEDEVICE (DEVADDR%, DOSPORT%)

Pascal - ieDevice (DevAddr, DOSPort:*integer*) : *integer* ;

C, C++ - *int* ieDevice (*int* DevAddr, *int* DOSPort)

Return Value:

Error code for Pascal and C/C++ commands.

Comments:

If $0 \leq \mathbf{PrimAddr} \leq 30$ then the 488.2 Driver will address the device to listen before outputting data. If $\mathbf{PrimAddr} < 0$ or > 30 , the replacement of LPT n or COM is disabled.

DOSPort Specifies the PC printer or COM port that is replaced according to the following table.

<u>DOSPort Value</u>	<u>Replaced PC Port</u>
1	LPT1
2	LPT2
3	LPT3
4	COM1
5	COM2

Warning:

Using this command to replace COM1 or COM2 could hang the system if another application attempts to initialize the replaced COM port.

Purpose:

This command reads device data. Reading continues until one of the following events occurs:

- EOI line is sensed
- EOS character is sensed if enabled
- the maximum number of characters is received

Syntax:

BASIC /QB - CALL IEENTER (DEVADDR%, INSTRING\$)

Pascal - ieEnter (DevAddr: *integer*; var InString: *string*):
integer;

C, C++ - *int* ieEnter (*int* DevAddr, *char* **const* InStringBuf,
int MaxLen)

Win DLL - ieEnter (PC2ADDR DevAddr, LPSTR InString,
int MaxLen)

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Comments:

If $0 \leq \text{PrimAddr} \leq 30$ then the specified device is addressed as a talker and the data is entered into **InString** from the device. Otherwise, data is entered from the current addressed talker and the EOL used will be for the PC2's own address. **MaxLen** is an integer used by C language commands to specify the maximum number of characters that may be inputted; $0 \leq \text{MaxLen} \leq 65535$. Always set **MaxLen** to one more than the number of characters to be input. Caution - use the ieEnterB command instead of the ieEnter command to input binary data.

Examples:

For BASIC/Quick Basic

DEVADDR% = 04

INSTRING\$ = SPACE\$ (80)

CALL IEENTER (DEVADDR%, INSTRING\$)

ieEnter continued

For C/C++

```
char InString [1024];  
ieEnter (4, Instring, 1024);
```

Purpose:

This command enters data from the device into an array. Reading continues until the message terminator is sensed or the maximum number of characters is received.

Syntax:

BASIC - CALL IEENTERA(DEVADDR%, DATASEG%,
MAXLEN%)

Pascal - ieEnterA (DevAddr, DataSeg: *integer*): *integer*;

Return Value:

Error code for Pascal commands.

Comments:

If $0 \leq \text{DevAddr} \leq 30$, then the specified device is addressed as a talker and data is inputted from the device. Otherwise data is inputted from the current addressed talker. The data is put into a memory area segment specified **DATASEG%** with the starting address offset = 0 **MaxLen** specifies the maximum number of characters that may be inputted; $0 \leq \text{MaxLen} \leq 65535$.

Examples:

For BASIC

```
DEVADDR% = 04  
DATASEG% = &H8000  
MAXLEN% = 100  
CALL IEENTERB (DEVADDR%, DATASEG$,  
MAXLEN%)  
'enters up to 100 characters in into memory segment
```

ieEnterB

Purpose:

This command enters binary or ASCII data from the device. Reading continues until the EOI line is sensed or the maximum number of characters is received.

Syntax:

- Quick Basic - CALL ieEnterB (DEVADDR%, INSTRING\$)
- C, C++ - *int* ieEnterB (*int* DevAddr, *char** *const* InStringBuf, *unsigned int* MaxLen)
- Win DLL - ieEnterB (*PC2ADDR* DevAddr, *LPSTR* InStringBuf, *int* MaxLen)

Return Value:

Error code for C/C++ and Win DLL commands.

Comments:

If $0 \leq \text{PrimAddr} \leq 30$, then the specified device is addressed as a talker and data is inputted from the device. Otherwise data is inputted from the current addressed talker. **InString** is the string variable that contains the received data. **MaxLen** specifies the maximum number of characters that may be inputted; $0 \leq \text{MaxLen} \leq 65535$. Use ieEnterB to input binary data from devices such as waveform recorders and digital oscilloscopes.

Examples:

The following examples will terminate when EOI is sensed or when the device has outputted 100 characters. MaxLen should not be set larger than the expected character count if the device **does not** assert EOI on the last character.

For Quick Basic

```
DEVADDR% = 04
INSTRING$ = SPACE80$      'see Table 6-2
MAXLEN% = 100
CALL IEENTERB (DEVADDR%, INSTRING$,
MAXLEN%)
'enters up to 100 characters in INSTRING$
```

For C/C++

```
ieEnterB (4, Instring, 100);
/* enters up to 100 characters in InString */
```

ieEol

Purpose:

This command sets the input and output data terminators for the specified device. Default values for ieEnter and ieOutput are LF with EOI asserted on the output and sensed on the input. The terminators for all devices remain at the default value if this command is not called for those particular devices.

Syntax:

- BASIC/QB - CALL IEEOL (DEVADDR%, OUTEOL%, OUTEOLSTR\$, INEOL%, INEOSBYTE%)
- Pascal - ieEol (DevAddr, OutEOL: *integer*; var OutEOLStr: *string*; InEOL, InEOSByte: *integer*): *integer*;
- C, C++ - *int* ieEol (*int* DevAddr, *int* OutEOL, *const char* *OutEOLStr, *int* InEOL, *int* InEOSByte)
- Win DLL - ieEOL (PC2ADDR DevAddr, *int* OutEOL, *const char* *OutEOLStr, *int* OutEOLlen, *int* InEOL, *int* InEOSByte)

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Comments:

If $0 \leq \text{PrimAddr} \leq 30$ then the terminators and specifiers are stored for the specified device.

OutEOL selects the terminator that is appended to the output string. Choices are:

- OutEOL = 0 for both OutEOLStr and EOI (default)
- OutEOL = 1 for EOI only
- OutEOL = 2 for OutEOLStr only

OutEOLStr specifies the output terminator string. Maximum string length is 8 characters. Default is "\n" (line feed)

ieEol continued

InEOL specifies the condition that terminates the input string.

- InEOL = 0 for terminate input when EOS character received, EOI sensed or input string full. (default)
- InEOL = 1 for terminate input when EOI sensed or input string full.

InEOSByte specifies the byte code that terminates the input. The default is decimal 10 (Line feed).

Notes:

When data transfers to or from a secondary addressed device, the terminators of its primary address are used.

Examples:

For BASIC/Quick Basic

```
DEVADDR% = 4
'OUTEOLSTR$ = CHAR$ (13) + CHAR$ (10)
CALL IEEOL (DEVADDR%, OUTEOL%,
OUTEOLSTR$, INEOL%, INEOSBYTE%)
'changes EOL string for device 4 to CR LF.
'keeps default EOS Byte
```

For C/C++

```
ieEol (4, 0, "\\15\\n", 0, 10);
/* changes EOL string for device 4 to CR LF
* keeps default EOS byte
*/
```

ieErrPtr

Purpose:

This command assigns variables for error number and byte transfer count. This function must be executed before calling any other command except `ieInit` or it will not work. This command is not required in Pascal and C/C++ language programs as the commands return an error value when called.

Syntax:

BASIC/QB - `CALL IEERRPTR (IOERR%, IOBYTES%)`

Pascal - n/a

C, C++ - n/a

Comments:

IOERR is an integer which contains the error number of the last called command. **IOBYTES** is an integer that contains the number of bytes entered or outputted by the last called command.

The assigned **IOERR** and **IOBYTES** integer variables can be read at any time by the `ieStatus` function. The error codes are listed in Table 6-3.

Examples:

For BASIC/Quick Basic

```
CALL IEERRPTR (IOERR%, IOBYTES%)
```

For C/C++

```
/* every C command returns an error value */  
int CmdErr;  
CmdErr = ieAny Command();  
/* error = 0 if the command was successful */
```

Purpose:

This function enables a particular event to send a notification to a particular window application.

Syntax:

Win DLL - ieEventEnable(*PC2ADDR* CardID, *HWND* hWnd, *WORD* wEvMask);

Return Value:

Error code for Win DLL commands.

Comments:

CardID specifies the card which is to generate a hardware interrupt. **EvMask** is a combination of the flags listed below. The user must select and enable a 488-PC2 IRQ line for this function to work.

<u>Flag</u>	<u>Window Msg</u>	<u>Meaning</u>
PC2EV_SRQ1	WM-PC2_SRQ1	SRQdetected
PC2EV_CO	WM-PC2_CO	Commandoutrequest
PC2EV_LOKC	WM-PC2_LOKC	Lockoutstatuschange
PC2EV_REMC	WM-PC2_REMC	Remotestatuschange
PC2EV_ADSC	WM-PC2_ADSC	Addressstatuschange
PC2EV_CPT	WM-PC2_CPT	Undefinedcommandor secaddressreceived
PC2EV_APT	WM-PC2_APT	Secaddressdetected
PC2EV_DET	WM-PC2_DET	Device trigger detected
PC2EV_END	WM-PC2_END	Data transfer complete EOI or EOS sensed
PC2EV_DEC	WM-PC2_DEC	Device clear detected
PC2EV_ERR	WM-PC2_ERR	GPIB Bus Error
PC2EV_DO	WM-PC2_DO	Data byte out
PC2EV_DI	WM-PC2_DI	Data byte in

For additional information about the event flags, refer to NEC 7210 data book.

ieEventStat

Purpose:

This function returns an integer that indicates the type of event detected for Quick Basic and C/C++ language programs.

Syntax:

Quick Basic - IEVENTSTAT

C, C++ - *unsigned* ieEventStat()

Return Value:

Error code for C/C++ commands.

Comments:

The function `ieEventStat` may be called at the beginning of an interrupt service routine to determine which event(s) were detected. The function `ieEventStat` returns an integer value with the following meanings:

<u>Value</u>	<u>Meaning</u>	<u>Include file constants</u>
1	SRQ detected	SRQSERV
2	Timeout occurred	TIMEOUTSERV
4	Error detected	ERRORSERV

For Quick Basic, this function will read-then-clear the driver's `EventStat` Register. Only call this function once after an interrupt.

For C, the event flag is set when the driver activates the user's service routine and is maintained throughout the routine. The driver will automatically clear the flag when exiting the service routine.

continued ieEventStat

For both C and Quick Basic, the bus SRQ event is treated differently than the other two events; it is triggered and latched when SRQ is asserted on the bus. The user has to ensure his service routine will clear the device's SRQ before exiting, otherwise, the subsequent SRQ event will most likely be lost.

Examples:

For Quick Basic

```
EVENTFLAG% = IEEVENTSTAT
'AND EVENTFLAG% with SRQSERV, TIMEOUTSERV
and ERRORSERV to determine event cause(s)
'See section 5.7 for a complete example
```

For C/C++

```
EventFlag = ieEventStat();
/* AND EventFlag with SRQServ, TimeOutServ and ErrorServ
to determine event cause(s) */
```

ieFindLstn

Purpose:

This command performs the IEEE 488.2 Find Listeners protocol and returns a list of all found listeners in the given address list.

Syntax:

- Quick Basic - `CALL IEFINDLSTN (ADDRLIST%(),
RESULTLIST%())`
- C, C++ - `void ieFindLstn(int far*AddrList, int far *ResultList)`
- Win DLL - `WINAPI ieFindLstn(int CardID, PC2ADDR*
AddrList, int numDev, PC2ADDR *ResultList,
int*numResults)`

Return Value:

Result List as described below

Comments:

This command causes the 488-PC2 tests for presence of a device at each primary address in the **AddrList**. If there is no response to the primary address, the 488-PC2 tests for secondary address responses with the same primary address. The address of each found device is placed in the **ResultList**. The protocol searches for listeners until all are found or until the **ResultList** is full. The **ResultList** is terminated with **END_ADDR** (3131) by the user. The **AddrList** and **ResultList** are defined in Table 6-3. The Quick Basic and C/C++ protocols do not check primary address 0 for secondary addresses. Note that the Win DLL only returns the primary addresses of any found secondary addresses.

The user should examine the **ResultList** to see if all expected devices were found. The **ResultList** can be edited and/or used as the address list for the other 488.2 protocols which require an address list.

Caution:

Be careful when performing this command on systems with Bus Analyzers, Bus Extenders, Bus Expanders or listen-only devices as they may respond to all possible addresses.

Examples:

The following examples are looking for devices at primary addresses 4 and 5.

For Quick Basic

```
DIM ADDRLIST%(1 TO 3)
DIM RESULTLIST%(1 TO 10)
ADDRLIST%(1)=04
ADDRLIST%(2)=05
ADDRLIST%(3)=3131 'end of list
RESULTLIST%(10)=3131 'mark end of list
CALL ieFindLstn(ADDRLIST%(1), RESULTLIST%(1))
```

For C/C++

```
int AddrList[]={04,05,3131};
int ResultList[10];
ResultList[9]=3131;
ieFindLstn(AddrList, ResultList);
```

ieFindRQS

Purpose:

This function performs the IEEE 488.2 FindRQS protocol and returns the address and response of the first device that it finds which is requesting service.

Syntax:

Quick Basic - IEFINDRQS (ADDRLIST%(0), RESPONSE%)

C, C++ - *int* ieFindRQS(*int* *AddrList, *int* Response)

Win DLL - ieFindRQS(PC2ADDR CardID, PC2ADDR* AddrList, *int* numDev, PC2ADDR* respDev, *word** Response)

Return Value:

Address of requesting device for Quick Basic and C/C++ commands.
Error code for Win DLL command.

Comments:

This function causes the 488-PC2 to serial poll all devices in the **AddrList** and returns the address of the first device found requesting service. The found device's status byte is returned in the **Response** integer. The returned address is 3131 if no devices are found that requested service and **Response** is -1 if there is no status byte. If SRQ is not asserted, Win DLL returns the STB and address of the first device on the bus.

Examples:

For Quick Basic

```
DIM ADDRLIST%(1 TO 4)
ADDRLIST%(1)=2
ADDRLIST%(2)=10
ADDRLIST%(3)=1701
ADDRLIST%(4)=3131 'mark end of list
FOUNDRQS=ieFindRQS% (ADDRLIST%(1), RESPONSE%)
```

continued ieFindRQS

For C/C++

```
int AddrList[]={04,1701,3131};  
int* Response;  
int FoundRQS;  
FoundRQS=ieFindRQS(int far*AddrList, int far*Response);
```

ieGotoStby

Purpose:

This command changes the Interface from an active Controller to a Standby Controller and unasserts ATN if it was on.

Syntax:

Quick Basic - `CALL ieGotoStby()`

C, C++ - `void ieGotoStby(void);`

Return Value:

none

Bus Activity:

If ATN was on it is unasserted.

Comments:

The `ieGotoStby` command makes the Interface go to the Standby Controller state and unassert ATN if it was the Active Controller. Control is recovered by the `ieTakeCtl` command. When in standby, the Interface can participate in a shadow handshake and let two external devices exchange data.

Examples:

For Quick Basic
`CALL ieGotoStby`

For C/C++
`ieGotoStby();`

Purpose:

This function returns an integer that represents the state of all eight GPIB interface signals.

Syntax:

Quick Basic - `ieGPIBStat()`

C, C++ - `unsigned char ieGPIBStat(void)`

Return Value:

An 8-bit value representing the state of the GPIB Control lines. A logical '1' indicates the signal is asserted.

Bus Activity:

None

Comments:

The function `ieGPIBStat` returns the logical state of the GPIB interface signals as an integer with a value of 0 to 255. The return value equals the binary sum of the asserted signals. Signals are viewed inside the Interface's transceivers so the results only reflect the external GPIB bus signals as indicated in the following chart:

Bit	Bit Weight	Signal	Interface Mode			
			Ctlr	Dev	Talker	Listener
0	1	ATN	x	x		
1	2	EOI	x			x
2	4	SRQ	x			
3	8	IFC		x		
4	16	REN		x		
5	32	DAV				x
6	64	NRFD			x	
7	128	NDAC			x	

ieGPIBStat continued

Examples:

For Quick Basic

```
SIGNALS% = ieGPIBStat
```

For C/C++

```
GPIB_Signals = ieGPIBStat();
```

Purpose:

This command initializes the drivers and sets the Interface parameters. This command must be called at least once at the beginning of the program.

Syntax:

BASIC/QB -CALL IEINIT (IOPORT%, MYADDR%,
 SETTING%)

Pascal - ieInit(IOPort, MyAddr, Setting: *integer*): *integer*;

C, C++ - void ieInit (*unsigned int* IOPort, *int* MyAddr,
 unsigned int Setting)

Win DLL - ieInit(*WORD* IOPort, *DWORD* CardID, *int* MyAddr,
 WORD Setting)

Return Value:

none

Comments:

IOPort, **CardID**, **MyAddr** and **Setting** have their default values set by an 'include' file. See Table 6-2.

IOPort specifies the Interface's PC internal IO Address in HEX. Must be the same value as the card's address switch setting (see section 1.4.1). Default value is 02E1H for the 488-PC2 Card and 0378H for the 4818 Module. **CardID** specifies the card based on the switch setting. Default value is 7

MyAddr specifies the IEEE 488 bus address of the Interface from 0 to 30. Typical bus controller addresses are 0 and 21. Default is 21.

Setting is a 16 bit integer, expressed as 4 HEX characters, that sets the card's operational parameters as shown in the following figure. Default value varies according to the language used. The setting bit meanings are:

ieInit continued

HEX BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Wait for DMA	0		0							0					0	0		
Background DMA	1														1	0		
Single byte DMA xfr		0										0	0	0	0	No interrupt		
Block DMA xfr		1										1	0	0	0	IRQ2		
												1	0	1	0	IRQ3		
												1	1	0	0	IRQ5		
												1	1	1	0	IRQ7		
Slow Bus handshake				0												0	System Controller	
Fast Bus handshake				1												1	Nonsystem Controller or Device	
Enable error interrupt					1											0	0	BASIC interpreter, Pascal
Enable Timeout interrupt					1											0	1	Reserved - do not use
Enable SRQ interrupt					1											1	0	Quick Basic, C Language
					1											1	1	Visual Basic for DOS, MS PDS 7.0

Examples:

Note - **These examples are not meant to show default settings.** Rather, they are used to redefine some of the default values given in Table 6-2. The user does not have to redefine non-changed parameters before calling ieInit.

For BASIC

```

IOPORT% = &H2E1
MYADDR% = 21
SETTING% = &H0000
'BASIC language with no DMA or interrupts
CALL IEINIT (IOPORT%, MYADDR%, SETTING%)
'Default value example - Card is at IO address 2E1, PC is
'device 21 and drivers execute interpretive BASIC.
    
```

For Quick Basic

```

IOPORT% = &H22E1 'some other card is using 02E1
MYADDR% = 0 'alternate bus ctrl address
SETTING% = &H0700
CALL IEINIT (IOPORT%, MYADDR%, SETTING%)
'Changed values example - card is at IO address 22E1, PC is
'device 0, timeout and SRQ interrupts are enabled
'Drivers execute compiled Basic.
    
```

6

For Quick Basicwith 4818 Module

```
IOPORT% = &H0378 '4818 Module on ptr port#1
MYADDR% = 0 'alternate bus ctr address
SETTING% = &H0500
CALL IEINIT (IOPORT%, MYADDR%, SETTING%)
'Changed values example - 4818 module is at IO address 0378,
PC is 'device 0, interrupt for a time-out is enabled
'Drivers execute compiled Basic.
```

For C/C++

```
IOPort = 0x02E1; /* default IO value */
MyAddr = 21; /* default GPIB address */
Setting = 0x9701;
ieInit (IOPort, MyAddr, Setting);
/*This examples shows some non-default values
* Card is at IO address 02E1, PC is GPIB device 21
* Background DMA and fast handshakes are enabled
* Timeout and SRQ interrupts are enabled
* DMA channel #1 is enabled
*/
```

ieLlo

Purpose:

This command executes a Local Lockout (LLO) to disable a device's front panel. It is received by all of the devices on the bus.

Syntax:

BASIC /QB - CALL IELLO

Pascal - ieLlo: *integer*;

C, C++ - *int* ieLlo(void)

Win DLL - ieLLO(*PC2ADDR* CardID)

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Bus Activity:

ATN is set
LLO is sent

Comments:

If a device is in Local mode when the LLO is received, LLO does not take affect until the device is addressed to listen.

Examples:

For BASIC/Quick BASIC
 Call IELLO

For C/C++
 ieLlo();

Purpose:

This command executes a Go-to-Local (GTL) or clears the REN line to enable a device's front panel controls. It de-asserts the REN line.

Syntax:

BASIC/QB - CALL IELOCAL (DEVADDR%)

Pascal - ieLocal(DevAddr: *integer*): *integer*;

C, C++ - *int* ieLocal (*int* DevAddr)

Win DLL - ieLocal (PC2ADDR DevAddr)

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Comments:

If $0 \leq \text{PrimAddr} \leq 30$ then the driver addresses the specified device to listen and sends it the GTL command and REN is set false (high). The device will remain in local mode until next addressed to listen. If local lockout is in affect, the device will return to the local lockout state when next addressed to listen.

If $\text{PrimAddr} \leq -1$ or ≥ 31 then the driver sets REN false. All devices are placed in local mode and local lockout mode is cancelled. Win DLL does **not** clear the REN line or cancel the local lockout mode and returns Error code 4 for an invalid parameter.

Examples:

For BASIC/Quick Basic

```
DEVADDR% = 4
```

```
CALL IELOCAL (DEVADDR%)
```

```
'Sets device 4 to local state
```

```
CALL IELOCAL (NOADDR%)
```

```
'Sets all devices to local state, NOADDR is defined in the  
'include' file
```

ieLocal continued

ForC/C++

ieLocal (4); /* sets device 4 to local state */

ieLocal (NoAddr); /* sets all devices to local state */

Purpose:

This command outputs a string to a device. After the string is sent, the terminator specified by the `ieEol` function is sent. Used to send ASCII strings and numbers to devices.

Syntax:

BASIC/QB -CALL IEOUTPUT (DEVADDR%, OUTSTRING\$)

Pascal - `ieOutput(DevAddr: integer,var OutString: string): integer;`

C, C++ - `int ieOutput (int DevAddr, char *OutString)`

Win DLL - `ieOutput (PC2ADDR DevAddr, char *OutString, int length)`

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Comments:

If $0 \leq \text{PrimAddr} \leq 30$ then the driver addresses the specified device to listen and sends it the **OutString** followed by the `ieEol` termination specified in `OutEOLStr`. Otherwise **OutString** is sent to the current addressed listener(s).

Examples:

```
For BASIC/Quick BASIC
DEVADDR% = 4
OUTSTRING$ = "ASCII String"
CALL IEOUTPUT (DEVADDR%, OUTSTRING$)
'Sends "ASCII String" to device 4
```

```
For C/C++
ieOutput (4, "ASCII string");
/* sends "ASCII String" to device 4
```

ieOutputA

Purpose:

This command outputs a long string from an array to a device and asserts EOI on the last character as enabled by the ieEOL command. This command only applies to Interpretive BASIC and Pascal languages.

Syntax:

BASIC - CALL IEOUTPUTA (DEVADDR%, DATASEG%,
BYTECNT%)

Pascal - ieOutputA (DevAddr, Datasег, ByteCnt: *integer*):
integer;

Return Value:

Error code for Pascal commands.

Comments:

If $0 \leq \text{PrimAddr} \leq 30$ then the driver addresses the specified device to listen and sends it the output with EOI asserted on the last byte if enabled by the ieEol command. Otherwise the output is sent to the currently addressed listener(s). **ByteCnt** specifies the number of bytes to be output by the command, 0 to 65,535 bytes.

The output is taken from a segment in memory specified by **DATASEG**. The starting address of the output data has an offset of 0 in that segment.

Examples:

For BASIC

```
DEVADDR% = 4
DATASEG%=&H4000
BYTECNT% = 300
CALL OUTPUTB (DEVADDR%, VARSEG
(OUTSTRING$), BYTECNT%)
'sends 300 bytes to device 4 from memory segment at 4000
HEX.
```

Purpose:

This command outputs a string or a series of bytes to a device and asserts EOI on the last character. This command is used to output binary data to a device.

Syntax:

Quick Basic - CALL ieOutputB (DEVADDR%, OUTSTRING\$)

C, C++ - *int* ieOutputB (*int* DevAddr, *char* *OutString,
unsigned int ByteCnt)

Win DLL - ieOutputB (PC2ADDR DevAddr, *char* *OutString,
unsigned int ByteCnt)

Return Value:

Error code for C/C++ and Win DLL commands.

Comments:

If $0 \leq \text{PrimAddr} \leq 30$ then the driver addresses the specified device to listen and sends it the **OutString** with EOI asserted on the last byte. Otherwise the **OutString** is sent to the currently addressed listener(s). **ByteCnt** specifies the number of bytes to be output by the command.

Examples:

For Quick Basic

```
DEVADDR% = 4
OUTSTRING$ = "D" + CHAR$(129)
CALL ieOutputB (DEVADDR%, OUTSTRING$)
'sends ASCII 'D' plus 81 hex to device 4.
```

For C/C++

```
ieOutputB (4, "\65\31\n", 3);
/* sends 3 binary bytes to device 4 */
```

iePassCtl

Purpose:

This command passes control to a specified device.

Syntax:

Quick Basic - iePassCtl (DEVADDR%)

C/C++ - *int* iePassCtl (*int* DevAddr)

Return Value:

Error Code for C/C++ commands

Comments:

If $0 \leq \mathbf{PrimAddr} \leq 30$ then the driver addresses the device to talk, sends it the TCT command and deasserts ATN. Otherwise the command is not executed and an error value is returned.

DevAddr specifies the address of the device being passed control.

Note - This command is not to be confused with the 488.2 protocol. It is merely a convenient way to pass control without having to use the ieSend command.

Example:

For Quick Basic

```
DEVADDR% = 22
```

```
CALL iePassCtl (DEVADDR%)
```

For C/C++

```
iePassCtl (22);
```

Purpose:

This command performs a parallel poll on the bus. The eight bit parallel poll response is returned as an integer with a value of 0 to 255.

Syntax:

BASIC/QB - CALL IEPPOLL (PRESP%)

Pascal - iePPoll: *integer*;

C, C++ - *unsigned char* iePPoll(void)

Win DLL - iePPoll(PC2ADDR CardID)

Return Value:

Parallel poll response for Pascal, C/C++ and Win DLL commands.

Bus Activity:

ATN and EOI are asserted for 25 microseconds. Data lines are pulled low (true) by devices' affirmative responses.

Comments:

For BASIC and Quick Basic commands, the parallel poll response is returned in the integer **PRESP**.

The parallel poll response is an integer with a value of 0 to 255 that is the sum of the data lines that were true at the end of the parallel poll time. i.e. response = 129 if lines DIO1 and DIO8 were true.

Examples:

For BASIC/Quick Basic

```
CALL IEPPOLL (PRESP%)
```

```
'PRESP% contains the response byte value
```

For C/C++

```
PResp = iePPoll();
```

```
/* PResp contains the response byte value */
```

iePPollC

Purpose:

This command configures a device to respond to a parallel poll.

Syntax:

BASIC/QB - CALL IEPPOLL (DEVADDR%, CONFIGBIT%)

Pascal - iePPollC (DevAddr, ConfigBit: *integer*): *integer*;

C, C++ - *int* iePPollC (*int* DevAddr, *unsigned char* ConfigBit)

Win DLL - iePPollC (*PC2ADDR* DevAddr, *int* ConfigBit)

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Comments:

If $0 \leq \text{PrimAddr} < 30$ then the driver outputs the parallel poll enable (PPE) command to the specified device, otherwise the PPE command is sent to the current addressed listeners.

ConfigBit is an integer which specifies the data bit (line) for the parallel poll response and the polarity of the response.

ConfigBit Values

0 - 7

8 - 15

Meaning

Specifies data lines 0 - 7, false response

Specifies data lines 0 - 7, true response

Examples:

For BASIC/Quick Basic

DEVADDR% = 4

CONFIGBIT% = 9

CALL IEPPOLL (DEVADDR%, CONFIGBIT%)

'sets device 4 to respond positively on data line DIO2 (bit 1)

For C/C++

iePPollC (4, 15);

/* sets device 4 to respond positively on data line DIO8 (bit 7) */

Purpose:

This command performs a parallel poll unconfigure. It directs one or all devices not to respond to a parallel poll.

Syntax:

BASIC /QB - CALL IEPPOLLU (DEVADDR%)

Pascal - iePPollU(DevAddr: *integer*): *integer*;

C, C++ - *int* iePPollU (*int* DevAddr)

Win DLL - iePPollU (*PC2ADDR* DevAddr)

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Comments:

If $0 \leq \mathbf{DevAddr} \leq 30$ then the driver unconfigures a specific device, otherwise the driver unconfigures all devices on the bus.

Examples:

For BASIC/Quick Basic

```
DEVADDR% = 4
```

```
CALL IEPPOLLU (DEVADDR%)
```

```
'removes device 4 from responding to a parallel poll
```

```
CALL IEPPOLLU (NOADDR%)
```

```
'all devices are unconfigured and will no longer respond to the  
parallel poll
```

For C/C++

```
iePPollU (4);
```

```
/* removes device 4 from responding to a parallel poll */
```

```
iePPollU (NoAddr);
```

```
/* all devices are unconfigured and will no longer respond to  
a parallel poll */
```

ieRemote

Purpose:

This command sets the REN line true and places any specified device in the remote mode.

Syntax:

BASIC/QB - CALL IEREMOTE (DEVADDR%)

Pascal - ieRemote(DevAddr: *integer*): *integer*;

C, C++ - *int* ieRemote (*int* DevAddr)

Win DLL - ieRemote (PC2ADDR DevAddr)

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Comment:

If $0 \leq \text{PrimAddr} \leq 30$ then the driver sets REN on and addresses the specified device to listen, otherwise no device is addressed. Note that a device will not switch to remote mode until it is addressed to listen.

Examples:

For BASIC/Quick Basic

```
DEVADDR% - 4
```

```
CALL IEREMOTE (DEVADDR%)
```

```
'Sets REN true and device 4 to remote mode
```

```
CALL IEREMOTE (NOADDR%)
```

```
'Sets REN line true
```

For C/C++

```
ieRemote (4);
```

```
/* Sets REN true and device 4 to remote mode */
```

```
ieRemote (NoAddr);
```

```
/* Sets REN true */
```

Purpose:

This command performs the IEEE 488.2 Reset protocol on the entire system and sends the *RST command to the listed devices.

Syntax:

Quick Basic - CALL ieReset (ADDRLIST%())

C, C++ - void ieReset(int far* AddrList)

Win DLL - ieReset(PC2AADDR CardID, PC2AADDR AddrList, int numDev)

Return Value:

none

Comments:

This command causes the 488-PC2 to assert REN, pulse IFC for over 100 microseconds, output the universal Device Clear command and sends the *RST command to all listed devices in the **AddrList**. For the PC2 Win DLL, the **numDev** must equal the number of listed addresses.

Caution - Check all non-IEEE 488.2 devices' response to the *RST command before putting them in the **AddrList**.

Examples:

For Quick Basic

```
DIM ADDRLIST%(1 TO 3)
ADDRLIST%(1)=2
ADDRLIST%(2)=5
ADDRLIST%(3)=3131
CALL ieReset (ADDRLIST%(1))
```

For C/C++

```
AddrList[3]={2,5,3131};
ieReset(AddrList);
```

ieSend

Purpose:

This command outputs user specified GPIB commands and data to the bus to generate custom command sequences. The ieSend command is used to address devices with secondary addresses, to establish multiple listeners, to pass control or to establish another talker.

Syntax:

BASIC/QB - CALL IESEND (CMDSTRING\$)

Pascal - ieSend(*var* CmdString: *string*): *integer*;

C, C++ - *int* ieSend (*const char** CmdString)

Win DLL - ieSend (*PC2ADDR* CardID, *LPSTR* CmdString, *int* ByteCnt)

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Comments:

CmdString is a string of standard IEEE 488 bus interface command mnemonics separated by a space. The mnemonics are:

MLA	My listen address (MyAddr)
MTA	My talk address (MyAddr)
UNL, UNT	Unlisten and Untalk
LISTEN n	Listen device n
TALK n	Talk device n
SEC n	Secondary Address n
DCL	Device Clear
SDC	Selected Device Clear
GET	Trigger
GTL	Go to Local
LLO	Local Lockout
PPC	Parallel Poll Configure
PPD	Parallel Poll Unconfigure
PPE	Parallel Poll Enable
PPU	Parallel Poll Disable

SPE	Serial Poll Enable
SPD	Serial Poll Disable
TCT	Take Control
DATA	Unasserts ATN to send any subsequent characters as data.

See Appendix A1 for more information about the above bus commands. Recommendation - Always start the command sequences with UNL to unlisten any unwanted listeners.

Examples:

For BASIC/Quick Basic

```
CMDSTRING$ = "UNL MTA LISTEN 4 SEC 01 DATA 'T1"  
CALL IESEND (CMDSTRING$)  
'sends string "T1" to device 4 secondary 1
```

```
CMDSTRING$ = "UNL MTA LISTEN 4 5 GTL"  
CALL IESEND (CMDSTRING$)  
'addresses devices 4 and 5 to both listen, and triggers both  
'devices
```

For C/C++

```
CmdString = "UNL MTA LISTEN 4 SEC 01 DATA 'T1"  
ieSend (CmdString);  
/* sends 'T1' instruction to device 4 secondary 1 */
```

```
CmdStr = "UNL MTA LISTEN 4 5 GTL"  
ieSend (cmdStr);  
/* addresses devices 4 and 5 to listen and  
* triggers both devices  
*/
```

ieServiceDisable

Purpose:

This command disables user event service routines.

Syntax:

C, C++ - *void* ieServiceDisable (*unsigned int* EventCode)

Return Value:

None.

Bus Activity:

None

Comments:

EventCode is an unsigned integer representing one or more of the following service routines:

<u>Value</u>	<u>Disabled Service Routine</u>
4	if command intended to disable error service
2	if command intended to disable time out service
1	if command intended to disable SRQ service

This command allows a C program to disable from one to three service routines previously enabled by the **ieServiceEnable** command. If **EventCode** has a bit set for a service that is not enabled, the bit is ignored.

Examples:

```
For C/C++
    ieServiceDisable (1);
    /* SRQ service disabled */
```

ieServiceEnable

Purpose:

Installs user defined event-service routine(s). Use `ieServiceDisable` to disable the service routine.

Syntax:

```
C, C++    - int ieServiceEnable (unsigned EventCode,  
                               pGPIBService SRQServ [, pGPIBServiceTOutServ  
                               [, pGPIBServiceErrServ]]);
```

Comments:

EventCode is an unsigned integer representing one or more of the following routines:

<u>Value</u>	<u>Service Routine</u>
4	if command intended to install error service
2	if command intended to install time out service
1	if command intended to install SRQ service

Parameters - **SRQServ**, **TOutServ** & **ErrServ**: point to user defined service routines.

For Interpretive BASIC, Quick Basic or Pascal language programs, refer to the interrupt handling section in their language description chapters.

ieServiceEnable continued

For C this command allows the user to install from one to three service routines in a single call. User defined service routine must be declared as *a far routine, take no argument and return void*. When designing a service routine, the C user must pay attention to the following:

Rule of thumb for user' installed event-handler:

1. The user's handler may safely call PC2 routines for servicing.
2. The user's handler may safely call other routines.
Provided those called routines are stack-and-local base, they do not have any side-affect and they are recursivable and do not rely on any static and/or global variables for correct operation.
3. The provision in rule #2 also applies to chained routines as well. That is a called called called ... called routine.
4. Provision rules #2 and #3 applies to user designed routines as well as runtime libraries.
5. Violation of rules #2 and #3 is considered unsafe unless the user can guarantee he doesn't call those called routines in the event-waiting-loop.

Examples:

For C/C++

```
ieServiceEnable (1, UserSRQService);  
/*to install SRQ Service Routine */
```

Purpose:

This command performs a serial poll on the specified device. The eight bit serial poll response is returned as an integer with a value of 0 to 255.

Syntax:

BASIC/QB - CALL IESPOLL (DEVADDR%, SRESP%)

Pascal - ieSPoll (DevAddr: *integer*): *integer*;

C, C++ - *int* ieSPoll (*int* DevAddr)

Win DLL - ieSPoll (*PC2ADDR* DevAddr)

Return Value:

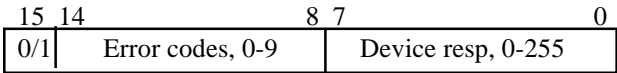
Serial poll response for Pascal, C/C++ and Win DLL commands.

Comments:

If $0 \leq \mathbf{PrimAddr} \leq 30$ then the driver addresses the device as a talker, sends it the serial poll enable command and reads back its status byte. After the status byte has been read, the device is sent the serial poll disable command and untalked. Otherwise the **DevAddr** is out of range and an error is created.

For Interpretive BASIC, Pascal and Win DLL, the response is a 8 bits in the lower order byte with a value of 0 to 255.

For Quick Basic and C/C++, the response is 16 bits. Bit 15 is the Error flag, bits 14-8 are the Error codes (see ieStatus) and bits 7-0 are the device's serial poll response.



ieSPoll continued

Examples:

For BASIC/Quick Basic

```
DEVADDR% = 04
```

```
CALL IESPOLL (DEVADDR%, SRESP%)
```

```
'SRESP% contains device 4's response to the serial poll
```

For C/C++

```
Sresp = ieSPoll (4);
```

```
/* SResp contains device 4's response to the serial poll */
```

Purpose:

This function returns an integer that indicates the logic level of the GPIB SRQ line.

Syntax:

Quick Basic - `ieSRQStat`

C, C++ - `bool ieSRQStat(void)`

Return Value:

Logic level of GPIB SRQ line.

Bus Activity:

None

Comments:

This function returns the logic level for the GPIB interface signal SRQ. 1 = asserted, 0 = not asserted.

Examples:

For Quick Basic
`SRQ% = ieSRQStat`

For C/C++
`SRQ = ieSRQStat();`

ieStatus

Purpose:

This command returns status information about the interface and the last called command.

Syntax:

BASIC/QB - CALL IESTATUS (STATUSCODE%, STATUS%)

Pascal - ieStatus (StatusCode: *integer*): *integer*;

C, C++ - *unsigned int* ieStatus (*int* StatusCode)

Win DLL - ieStatus (PC2ADDR CardID,
int StatusCode)

Return Value:

Status response for Pascal, C/C++ and Win DLL commands.

Bus Activity:

None

Comments:

The **StatusCode** selects whether the command reads the registers of the NEC7210 GPIB controller chip on the Interface, returns driver settings or status information about the last called command. For BASIC and Quick Basic, **STATUS** contains the command status.

The Status Codes are:

<u>Status Code</u>	<u>Status Contents</u>
0 - 7	GPIB chip (NEC7210) registers 0 - 7.
8	Error Number of last called command.
9	Count of string bytes that are output or entered.
10	Timeout interval in milliseconds.
11	I/O port address of NEC7210.
12	Returns the Setting parameter of the last ieInit command.

Status Code 8 returns the following error codes:

<u>Status</u>	<u>Error</u>
0	No error
1	Handshake timeout
2	Interface error
3	Called a command when not system controller or controller-in-charge
4	Invalid passed parameter(s)
5	Break key used to quit handshake
6	Failed to allocate DMA buffer
7	Could not find a device requesting service
8	No acceptor present on the bus
9	Could not find 488-PC2 Card or 4818 Module

Refer to the ieErrPtr command for additional information on reading error codes and byte transfer counts.

Examples:

For BASIC/Quick Basic

```
STATUS CODE% = 9 'Reads byte count
CALL IESTATUS (STATUSCODE%, STATUS%)
'STATUS% value is the number of bytes sent or received by
the last called command.
```

For C/C++

```
Status = ieStatus (9);
/* Status value is the number of bytes sent or received by the
last called command */
```

ieTakeCtl

Purpose:

The `ieTakeCtl` command changes the Interface from a standby Controller to an active Controller. The Interface takes control asynchronously, synchronously or synchronously after sensing EOI. This command is not available for BASIC or Pascal language programs.

Syntax:

Quick Basic - `CALL ieTakeCtl(MODE%)`

C, C++ - `int ieTakeCtl(int Mode)`

Return Value:

Error code for C/C++ commands

Bus Activity:

INTERFACEtakes_control and asserts ATN.

Comments:

The command `ieTakeCtl` causes the Interface to take control of the bus. If **Mode** = `TC_ASYNC` then the Interface takes control immediately without regard for any existing data transfer operations. If **Mode** = `TC_SYNC` then the Interface takes synchronously by waiting for any existing data handshake to be completed before asserting ATN. If **Mode** = `TC_END` then the Interface takes control synchronously at the end of the next data handshake that has EOI asserted or handshakes the EOS byte. Use `TC_ASYNC` to take control when no handshakes exist such as after a timeout. Use `TC_END` to take control after two devices have exchanged a data message.

Include file

<u>Value</u>	<u>constant</u>	<u>Method</u>
0	<code>TC_ASYNC</code>	TakeControlAsynchronously
1	<code>TC_SYNC</code>	TakeControlSynchronously
2	<code>TC_END</code>	TakeControlSynchronously after EOIassertedorEOSbyte sensed.

Caution:

The IEEE-488 Standard recommends against taking control asynchronously since there is a possibility that bus devices can erroneously interpret a data character as a bus command when ATN is asserted asynchronously.

Examples:

For Quick Basic

```
CALL ieTakeCtl (TC_END%)  
' takes control after handshaking EOS or a byte with EOI  
' asserted
```

For C/C++

```
ieTakeCtl(TC_SYNC);  
/*takes control after the next handshake */
```

ieTimeOut

Purpose:

This command sets the GPIB handshake Timeout value in milliseconds. This command should be called before executing any data transfers on the bus.

Syntax:

BASIC/QB - CALL IETIMEOUT (TIME%)

Pascal - ieTimeOut (Time: *integer*): *integer*;

C, C++ - *int* ieTimeOut (*unsigned int* Time)

Win DLL - ieTimeOut (PC2ADDR CardID, *unsigned int* Time)

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Bus Activity:

None

Comments:

The **Time** sets the maximum length of time in milliseconds that a bus handshake can take before the handshake is aborted. An aborted handshake produces error code 1 (see ieStatus command). Set **Time** to 0 to disable the timeout function when single stepping the system.

<u>Time</u>	<u>Timeout</u>
0	Disables the timeout function
1 to 32,767	1 to 32,767 milliseconds
-1 to -32,767	(65,536 + Time) milliseconds

Note that the Time period for Interpretive BASIC, Pascal and Win DLL commands is dependent upon the CPU clock and will vary from PC to PC. The Time period for Quick Basic and C/C++ commands is independent of the CPU clock and is therefore as accurate as the hardware permits.

Examples:

For BASIC/Quick Basic

```
NOTIMEOUT% = 0
CALL IETIMEOUT (NOTIMEOUT%)
' disables timeout function
```

```
TIME% = 10000
CALL IETIMEOUT (TIME%)
' sets timeout to 10 seconds
```

For C/C++

```
ieTimeout(0);           /* disables timeout */
ieTimeout (10000);     /* sets timeout to 10 seconds */
```

ieTrigger

Purpose:

This command triggers a specific device or sends a trigger command (GET) to the bus.

Syntax:

BASIC/QB - CALL IETRIGGER (DEVADDR%)

Pascal - ieTrigger (DevAddr: *integer*): *integer*;

C, C++ - *int* ieTrigger (*int* DevAddr)

Win DLL - ieTrigger (*PC2ADDR* DevAddr)

Return Value:

Error code for Pascal, C/C++ and Win DLL commands.

Comments:

If $0 \leq \text{PrimAddr} \leq 30$ then the driver addresses the specified device to listen and sends it the GET command, otherwise the driver outputs the GET command to all currently addressed listeners.

Examples:

For BASIC/Quick Basic

```
DEVADDR% = 4
```

```
CALL IETRIGGER (DEVADDR%) ' Triggers device 4
```

```
CALL IETRIGGER (NOADDR%)
```

```
' Sends GET command to the bus
```

For C/C++

```
ieTrigger (4); /* Triggers device 4 */
```

```
ieTrigger (NoAddr); /* sends GET command to the bus */
```

Purpose:

This function suspends program activities and waits for SRQ to be asserted or until timeout occurs. The function returns the logical state of the SRQ line at return time.

Syntax:

Quick Basic - `ieWaitSRQ(SLEEPTIME%)`

C/C++ - `bool ieWaitSRQ (int SleepTime)`

Return Value:

One if SRQ is set, zero otherwise.

Bus Activity:

None

Comments:

This function suspends the program for a specified time period specified by **SleepTime** in milliseconds or until SRQ is asserted, whichever comes first. Returned value indicates state of the SRQ bus signal when the function ended; 1 = asserted. If **SleepTime** is set to 0, the function returns immediately.

Examples:

For Quick Basic:

```
SLEEPTIME%=2      ' sleep for 2 seconds
SRQSTATE=ieWaitSRQ (SLEEPTIME%)
```

For C/C++

```
SRQState = ieWaitSRQ(0);
/* SRQState is immediately returned */
```

msDelay - Utility Function

Purpose:

This Utility function suspends program execution for the specified time in milliseconds.

Syntax:

Quick Basic - `CALL msDelay CDECL(BYVAL ms%)`

C/C++ - `void_far msDelay (unsigned ms)`

Return Value:

None

Bus Activity:

None

Comments:

This function uses the system real time clock and is independent of the CPU clock frequency.

Examples:

For Quick Basic:

```
DelayTime% = 100
CALL msDelay(DelayTime%)
'delays program for 100 milliseconds
```

For C/C++

```
msDelay(100)
/* delays program for 100 ms */
```

Utility Function - icsTimer

Purpose:

This Utility function provides the ability to read system time with a high resolution.

Syntax:

Quick Basic - icsTimer& CDECL()

C/C++ - *unsigned long_far* icsTimer (void)

Return Value:

Number of 215 microsecond periods since last midnight.

Bus Activity:

None

Comments:

Returned value is the number of 215 microsecond periods since midnight. Since Quick Basic does not support unsigned numbers, if the returned value is less than zero, it is actually:

4292967295-(absolute of the returned value)

Examples:

For Quick Basic:

```
Time = icsTimer&
```

For C/C++

```
Time = icsTimer();
```

```
/* Time since midnight is immediately returned */
```

isTimeout - Utility Function

Purpose:

This Utility function provides the ability to set a time mark and then test to see if the real time has gone past the time mark. This function should not be confused with the `ieTimeout` command which sets the GPIB bus handshake timeout value.

Syntax:

Quick Basic - `isTimeout% CDECL(BYVAL msMark%)`

C/C++ - `bool_far isTimeout (unsigned msMark)`

Return Value:

TRUE if system time has passed the previously set time mark, FALSE otherwise.

Comments:

A call of the **isTimeout** function with **msMark** \neq zero sets a time mark at system time plus **msMark**. All subsequent calls with **msMark** = zero test to see if the system time has passed the set time mark value. **msMark** is in milliseconds

Examples:

For Quick Basic:

```
msMark% = 16000           'establishes a 16 second period
Status% = isTimeout(msMark%) 'set the mark
                        'execute some program statements here
Status% = isTimeout(0)
IF Status% = 1 THEN PRINT "Time elapsed"
```

For C/C++

```
Time = isTimeout(16000); /*sets a 16 second time mark*/
                        /* execute some program statements here */
if ( isTimeout(0))
    {print "Time Elapsed" ;
    exit ;
```

Appendix

Appendix	Page
A1 IEEE 488 Bus Description	A-2
A1.1 IEEE 488.1 Bus	A-2
A1.2 IEEE 488.2 Standard	A-8
A1.3 SCPI Commands	A-12
A2 Troubleshooting & Repair	A-15
A2.2 Troubleshooting Procedure	A-15
A2.3 Board Tests	A-18
A2.4 Repair	A-20

A1 IEEE 488 BUS DESCRIPTION (IEEE 488.1, IEEE 488.2, SCPI)

The IEEE 488 Bus, also known as the GPIB bus (General Purpose Instrument Bus), is a convenient means of connecting instruments and computers together to form a test system or to transfer data. The IEEE STD 488.1 covers the electrical and mechanical bus specifications and the state diagrams for each bus function. The IEEE STD 488.2-1987 expanded on the original specification and established data formats, common commands for 488.2 compatible devices, a basic status reporting structure, and controller protocols. Later, the SCPI standard developed a tree like series of standard commands for programmable instruments and a way of expanding the status structure so that similar instruments by different manufacturers can be controlled by the same program.

A1.1 IEEE 488.1 Bus

The IEEE STD 488 Bus, or GPIB as it is commonly referred to, provides a means of transferring data and commands between devices. The physical portion of the bus is governed by IEEE-STD 488.1 - 1978. The interface functions for each device are contained within that device itself, so only passive cabling is needed to interconnect the devices. The cables connect all instruments, controllers and other components of the system in parallel to the signal line as shown in Figure A-1. Eight of the lines (DIO1-DIO8) are reserved for the transfer of data and other messages in a byte-serial, bit-parallel manner. Data and message transfer is asynchronous, coordinated by the three handshake lines (DAV, NRFD, NDAC). The other five lines (ATN, REN, IFC, EOI and SRQ) control Bus activity.

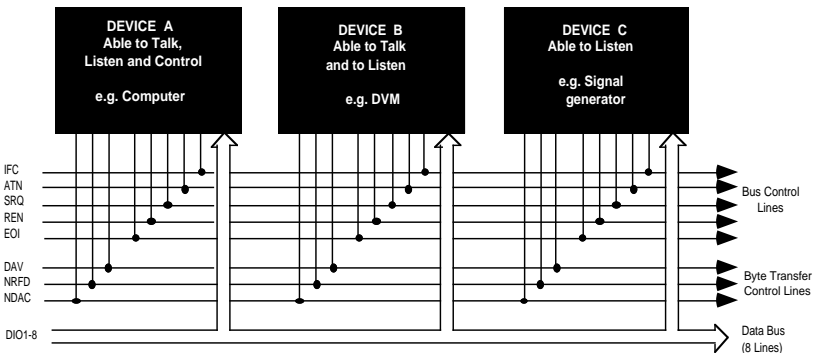


Figure A-1 IEEE 488 Bus

Devices connected to the bus may act as talkers, listeners, controllers, or combinations of the three functions, depending upon their internal capability.

A **talker** sends device dependent messages, i.e., data, status.

A **listener** accepts interface messages, bus commands and device-dependent messages, i.e., setup commands, data.

A **controller** can send interface messages to manage the other devices, address devices to talk or listen and command specific actions within devices.

A talker might be a dumb measuring device like a sensor. A listener could be an output device like a printer. However, most devices have both talker-listener capability such as a DVM which accepts instructions and returns data. Any 488.2 compatible device has to be both a talker and a listener to be able to respond to the 488.2 Common Commands.

A1.1.2 Bus Controllers

Controllers are referred to as the Active Controller, the Controller-in-charge (CIC) and the System Controller. The System Controller is the controller that becomes active at power turn-on. The System Controller is the only controller in the system that can drive the REN and IFC lines and is the initial Controller-in-charge. Systems may have multiple controllers but only one controller can be the Active Controller or CIC at a time. In a multiple controller system, control is 'passed' from the System Controller, who was the initial controller, to an inactive controller by the Take Control command. The Controller receiving this command becomes the Active Controller (CIC) and the passing controller becomes an inactive controller.

Most GPIB systems have just one controller and there is never a need to pass control to another controller.

A1.1.3 Device Addressing

The GPIB bus has both primary and optional secondary address capability. Each GPIB device must have a unique primary address so it can be addressed as a listener or as a talker. The secondary addresses provide a way to address multichannel hardware or special modes inside a device.

The GPIB address fields in Table A-1 are 5 bits wide and have 32 unique values. Values 0 to 30 can be assigned to any device as its primary address. Value 31 is used as the untalk or unlisten address and cannot be used by a device. Controllers have an address so they can address themselves to talk, to listen and to pass control. Traditionally, controllers have used addresses 0 (National Instruments) or 21 (Hewlett-Packard and ICS Electronics).

Devices traditionally have their primary address set by a small rocker switch on their rear panel. Instruments with front panel displays often have a way of setting their address from the front panel. Some new instruments whose firmware includes a SCPI command parser, can save their GPIB address in EEPROM and have it changed by a command from the GPIB Controller.

A1.1.4 Data Transmission and Handshake Lines

Information is transmitted on the data lines under sequential control of the three handshake lines (DAV, NFRD, NDAC). No step in the sequence can be initiated until the previous step is completed. Information transfer proceeds as fast as the devices respond (up to 1 Mbs) [Note 1], but no faster than that allowed by the slowest addressed device. This permits several devices to receive the same message byte at the same time. Although several devices can be addressed to listen simultaneously, only one device at a time can be addressed as a talker. When a talk address is put on the data lines, all other talkers are normally unaddressed.

When a device is addressed as a talker, it is allowed to send device-dependent messages (e.g., data) when the controller-in-charge sets the ATN line false. The data messages are typically a series of ASCII characters ending with a LF. The data messages may also be eight-bit binary characters terminated with EOI asserted when the last byte is talked onto the GPIB bus. The controller-in-charge must be programmed to correctly respond to each device's message termination sequence to avoid hanging-up the system or leaving characters that will be output when the device is next addressed as a talker.

1. National Instruments has advocated a higher hadshake rate which has been voted down by Hewlett-Packard, ICS and others as having potential problems with older divices.

A1.1.5 Interface Management Lines

ATN (attention) is set true by the controller-in-charge while it is sending interface messages or device addresses. The messages are transmitted on the seven least significant data lines and are listed in the MSG columns in Table A-1. The Interface Message bytes and Address codes are the same as the 128 ASCII character set. The ATN line is asserted when sending Interface Messages or Addresses and off when sending data.

IFC (interface clear) is sent by the system controller and places the interface system in a known quiescent state with all devices unaddressed.

REN (remote enable) is sent by the system controller and is used with other interface messages or device addresses to select either local or remote control of each device.

SRQ (service request) is sent by any device on the bus that wants service, such as counter that has just completed a time-interval measurement. EOI (end or identify) is used by a device to indicate the end of a multiple-byte transfer sequence. When a controller-in-charge sets both the ATN and EOI lines true, each device configured to respond to a parallel poll indicates its current status on the DIO line assigned to it.

A1.1.6 Interface Messages and Bus Commands

Table A-1 shows the Interface Messages and Device Addresses in the MSG columns of the Table and their definitions below the table. Interface Messages in the Universal Command Group and will be accepted by all devices at any time. Interface Messages in the Addressed Command Group and will only be accepted by devices that have been addressed as a listener and are normally directed to a specific device.

A1.1.7 System Configuration Guidelines

The designers of the IEEE 488 Bus made it virtually foolproof to assemble a trouble free system as long as the user follows the Standard. The rules that the user must observe are:

1. Limit the number of devices on the GPIB bus to 15 devices including the bus controller.

ASCII -- IEEE 488 BUS MESSAGES (COMMANDS AND ADDRESS) HEX CODES

LSD \ MSD	0		1		2		3		4		5		6		7	
	ASCII	MSG	ASCII	MSG	ASCII	MSG1	ASCII	MSG1	ASCII	MSG1	ASCII	MSG1	ASCII	MSG	ASCII	MSG
0	NUL		DLE		SP	00	0	16	@	00	P	16	,	▲	p	▲
1	SOH	GTL	DC1	LLO	!	01	1	17	A	01	Q	17	a	MEANING DEFINED BY PCG CODE	q	MEANING DEFINED BY PCG CODE
2	STX		DC2		"	02	2	18	B	02	R	18	b		r	
3	ETX		DC3		#	03	3	19	C	03	S	19	c		s	
4	EOT	SDC	DC4	DCL	\$	04	4	20	D	04	T	20	d		t	
5	ENQ	PPC	NAK	PPU	%	05	5	21	E	05	U	21	e		u	
6	ACK		SYN		&	06	6	22	F	06	V	22	f		v	
7	BEL		ETB		'	07	7	23	G	07	W	23	g		w	
8	BS	GET	CAN	SPE	(08	8	24	H	08	X	24	h		x	
9	HT	TCT	EM	SPD)	09	9	25	I	09	Y	25	i		y	
A	LF		SUB		*	10	:	26	J	10	Z	26	j		z	
B	VT		ESC		+	11	;	27	K	11	[27	k	(
C	FF		FS		.	12	<	28	L	12	\	28	l			
D	CR		GS		-	13	=	29	M	13]	29	m)		
E	SO		RS		.	14	>	30	N	14	^	30	n	~		
F	SI		US		/	15	?	UNL	O	15	-	UNT	o	▼	DEL	▼

ADDRESSED
COMMAND
GROUP

UNIVERSAL
COMMAND
GROUP

LISTEN ADDRESS GROUP

TALK ADDRESS GROUP

SECONDARY COMMAND
GROUP

PRIMARY COMMAND GROUP (PCG)

TABLE A-1 IEEE 488 COMMAND AND ADDRESS MESSAGES

A-6

- Notes:
1. Device Address messages shown in decimal
 2. Message codes are:

DCL -- Devices Clear	LLO -- Local Lockout	SDC -- Selected Device Clear
GET -- Device Trigger	PPC -- Parallel Poll Configure	SPD -- Serial Poll Disable
GTL -- Go to Local	PPU -- Parallel Poll Unconfigure	SPE -- Serial Poll Enable
 3. ATN off, Bus data is ASCII; ATN on, Bus data is an IEEE MSG.

2. Maximum total cable length of 20 meters.
3. Cable length between devices and should be 2 meters or less to obtain the full 1 Mbyte/second data transfer rate.
4. More than 50% of the devices must be powered on. (Power on all devices for full 1 Mbyte/second data transfer)

Use Bus Extenders or Expanders to increase the cable length of number of devices on the bus. Extenders and Expanders are available from ICS.

A1.1.8 Mechanical and Electrical Interface

Devices on the bus are normally interconnected by cables with dual male/female connectors at each end to allow easy cable stacking. The 24 conductor cable pinouts are shown in Figure A-2. Signal levels are 0 and 3.3 Vdc with 0 being the logic true level. Cable connectors are modified Amphenol 24 pin Blue ribbon style connectors (57-30240) with metric jack screws.

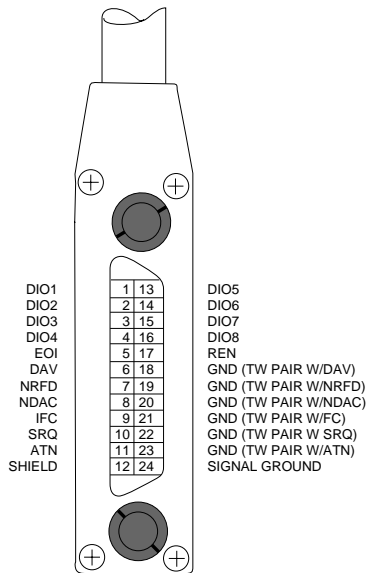


Figure A-2 GPIB Signal-Pin Assignments

A1.2 IEEE 488.2 STANDARD

A1.2.1 IEEE 488.2 Message Formats

The IEEE 488.2 Standard was established in 1987 to standardize message protocols, status reporting and define a set of common commands for use on the IEEE 488 bus. IEEE 488.2 devices are supposed to receive messages in a more flexible manner than they send. A message sent from GPIB controller to GPIB device is called: PROGRAM MESSAGE. A message sent from device to controller is called: RESPONSE MESSAGE. As part of the protocol standardization the following rules were generated:

- (;) Semicolons are used to separate messages.
- (:) Colons are used to separate command words.
- (,) Commas are used to separate data fields.
- <nl> Line feed and/or EOI on last character terminates a 'program message'. Line feed (ASCII 10) and EOI terminates a RESPONSE MESSAGE.
- (*) Asterisk defines a 488.2 common command.
- (?) Ends a query where a reply is expected.

A1.2.2 IEEE 488.2 Reporting Structure

With IEEE 488.2, status reporting was enhanced from the simple serial poll response byte in IEEE 488.1 to the multiple register concept shown in Figure A-3. The IEEE 488.2 Standard standardized the bit assignments in the Status Byte Register, added eight more bits of information in the Event Status Register and introduced the concept of summary bits reporting to the Status Byte Register. The Status and Event registers have enabling registers that can control the generation of their summary reporting bits and ultimately SRQ generation. Each 488.2 device must implement a Status Byte Register, a Standard Event Status Register and an Output Message Queue as shown in Figure A-3. A 488.2 compliant device must have this minimum status reporting structure and may include any number of additional condition registers, event registers and enabling registers providing they follow the model shown in Figure A-3.

A1

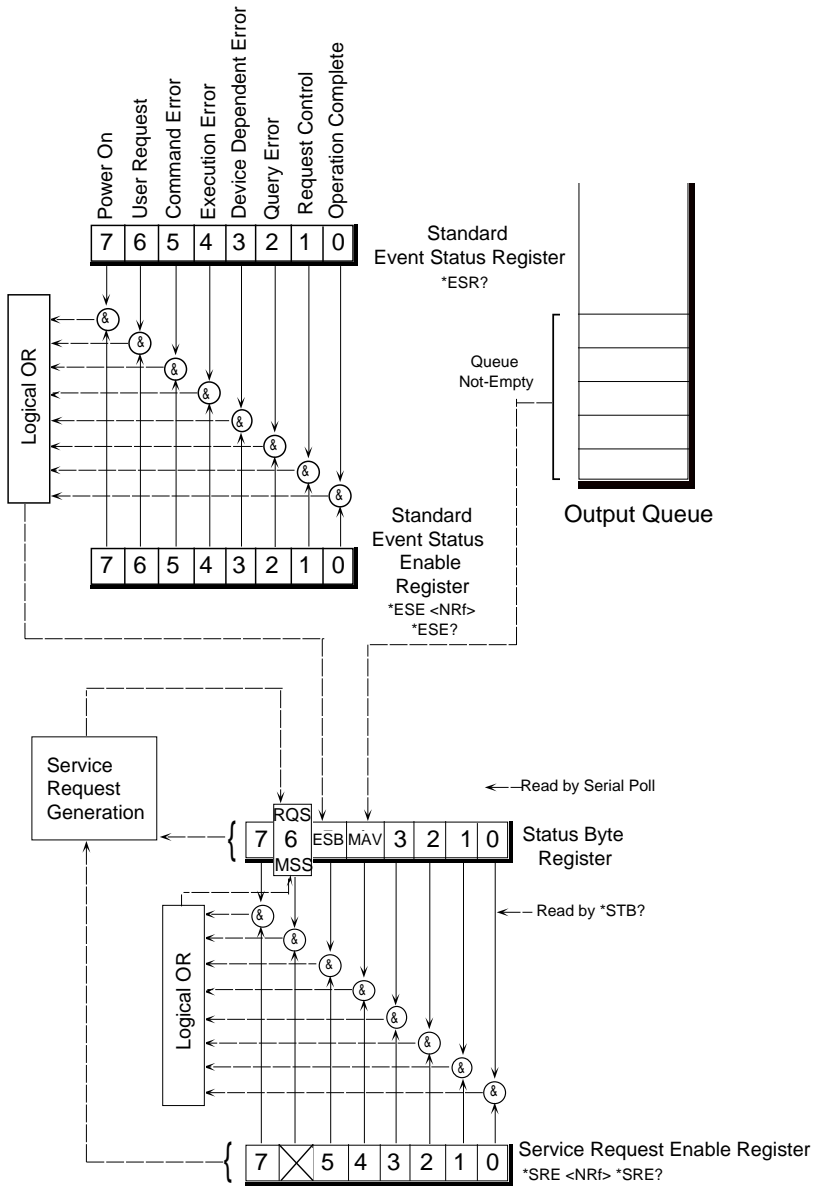


Figure A-3 488.2 Required Status Reporting Capabilities

A1.2.3 IEEE 488.2 Common Commands

The 488.2 specification also mandated a list of common commands that all devices will support. All of the Common Commands start with an asterisk. Commands that end with a question mark are queries. Query responses can be an ASCII number or an ASCII string. Other numerical formats such as HEX or Binary are legal as long as the device supports the required ASCII format. These commands are:

- | | | |
|----|--------------|--------------------------------------|
| 1 | *CLS | Clear Status Command |
| 2 | *ESE | Standard Event Status Enable Command |
| 3 | *ESE? | Standard Event Status Enable Query |
| 4 | *ESR? | Standard Event Status Register Query |
| 5 | *IDN? | Identification Query |
| 6 | *OPC | Operation Complete Command |
| 7 | *OPC? | Operation Complete Query |
| 8 | *RST | Reset Command |
| 9 | *SRE | Service Request Enable Command |
| 10 | *SRE? | Service Request Enable Query |
| 11 | *STB? | Status Byte Query |
| 12 | *TST? | Self-Test Query |
| 13 | *WAI | Wait-to-Continue Command |

In addition to the above common commands, devices that support parallel polls must support the following three commands

- | | |
|--------------|---------------------------------------|
| *IST? | Individual Status Query? |
| *PRE | Parallel Poll Register Enable Command |
| *PRE? | Parallel Poll Register Enable Query |

Devices that support Device Trigger must support the following commands:

- *TRG Trigger Command**

Controllers must support the following command:

- | | |
|-------------|---------------------------|
| *PCB | Pass Control Back Command |
|-------------|---------------------------|

Devices that save and restore settings support the following commands:

- | | |
|-------------|----------------------|
| *RCL | Recall configuration |
| *SAV | Save configuration |

A1.2.4 IEEE 488.2 Differences From IEEE 488.1

The user who is familiar with the older 488.1 devices should take the following differences into account when programming a 488.2 device.

A 488.2 device outputs the Status Byte Register contents plus the RQS bit in response to a serial poll. The RQS bit is reset by the serial poll. The same 488.2 device outputs the Status Byte Register contents plus the MSS bit in response to a *STB? query. The MSS bit is cleared when the condition is cleared.

488.2 restricts the Device Clear to only clearing the device's buffers and pending operations. It does not clear the Status Reporting Structure or the output lines. Use *CLS to clear the Status Structure and *RST or *RCL to reset the outputs.

488.2 commands are really special data messages and are executed by the device's parser. Always allow sufficient time for the parser to execute the commands before sending the device a 488.1 command. i.e. a Device Clear sent too soon will erase any pending commands and reset the parser.

Enable Register values are only saved and restored if the *PSC command is 0. A *PSC command of 1 causes zeros to be loaded into the enable registers when the unit is next reset or powered on.

A1.3 SCPI COMMANDS

A1.3.1 Introduction

SCPI (Standard Commands for Programmable Instruments) builds on the programming syntax of 488.2 to give the programmer the capability handling a wide variety of instrument functions in a common manner. This gives all instruments a common "look and feel".

SCPI commands use common command words defined in the SCPI specification. Control of any instrument capability that is described in SCPI shall be implemented exactly as specified. Guidelines are included for adding new defined commands in the future as new instruments are introduced without causing programming problems.

SCPI is designed to be laid on top of the hardware - independent portion of the IEEE 488.2 and operates with any language or graphic instrument program generators. The obvious benefits of SCPI for the ATE programmer is in reducing the learning time on how to program multiple SCPI instruments since they all use a common command language and syntax.

A second benefit of SCPI is that its English like structure and words are self documenting, eliminating the needs for comments explaining cryptic instrument commands. A third benefit is the reduction in programming effort to replace one manufacturer's instrument with one from another manufacturer, where both instruments have the same capabilities.

This consistent programming environment is achieved by the use of defined program messages, instrument responses and data formats for all SCPI devices, regardless of the manufacturer.

A1.3.2 Command Structure and Examples

SCPI commands are based on a hierarchical structure that eliminates the need for most multi-word mnemonics. Each key word in the command steps the device parser out along the decision branch - similar to a squirrel hopping from the tree trunk out on the branches to the leaves. Subsequent keywords are considered to be at the same branch level until a new complete command is sent to the device. SCPI commands may be abbreviated as shown by the capital letters in Figure A-4 or the whole key word may be

A1

used when entering a command. Figure A-4 shows some single SCPI commands for setting up and querying a serial interface.

SYSTem:COMMunicate:SERial:BAUD 9600 <nl>

Sets the baud rate to 9600 baud

SYST:COMM:SER:BAUD? <nl>

Queries the current baud setting

SYST:COMM:SER:BITS 8 <nl>

Sets character format to 8 data bits

Figure A-4 SCPI Command Examples

Multiple SCPI commands may be concatenated together as a compound command using semi colons as command separators. The first command is always referenced to the root node. Subsequent commands are referenced to the same tree level as the previous command. Starting the subsequent command with a colon puts it back at the root node. IEEE 488.2 common commands and queries can be freely mixed with SCPI messages in the same program message without affecting the above rules. Figure A-5 shows some compound command examples.

SYST:COMM:SER:BAUD 9600; BAUD? <nl>

SYST:COMM:SER:BAUD 9600; :SYST:COMM:SER:BITS 8 <nl>

SYST:COMM:SER:BAUD 9600; BAUD?; *ESR?; BIT 6; BIT?; PACE XON; PACE?; *ESR? <nl>

Figure A-5 Compound Command Examples

A typical response would be: **9600; 0; 8; XON; 32 <nl>**

The response includes five items because the command contains 5 queries. The first item is **9600** which is the baud rate, the second item is **ESR=0** which means no errors (so far). The third item is **8** (bit/word) which is the current setting. The BIT 6 command was not accepted because only 7 or 8 are valid for this command. The fourth item **XON** means that XON is

A1

active. The last item is **32** (ESR register bit 5) which means execution error - caused by the BIT 6 command.

A1.3.3 Variables and Channel Lists

SCPI variables are separated by a space from the last keyword in the SCPI command. The variables can be numeric values, boolean values or ASCII strings. Numeric values are typically decimal numbers unless otherwise stated. When setting or querying register values, the decimal variable represents the sum of the binary bit weights for the bits with a logic '1' value. e.g. a decimal value of 23 represents $16 + 4 + 2 + 1$ or 0001 0111 in binary. Boolean values can be either 0 or 1 or else OFF or ON. ASCII strings can be any legal ASCII character between 0 and 255 decimal except for 10 which is the Linefeed character.

Channel lists are used as a way of listing multiple values. Channel lists are enclosed in parenthesis and start with the ASCII '@' character. The values are separated with commas. The length of the channel list is determined by the unit. A range of values can be indicated by the two end values separated by a colon. e.g.

- (@1,2,3,4)** lists sequential values
- (@ 1:4)** shows a range of sequential values
- (@ 1,5,7,34)** lists random values

Figure A-6 Channel List Examples

A1.3.4 Error Reporting

SCPI provides a means of reporting errors by responses to the **SYST:ERR?** query. If the SCPI error queue is empty, the unit responds with 0, "No error" message. The error queue is cleared at power turn-on, by a ***CLS** command or by reading all current error messages. The error messages and numbers are defined by the SCPI specification and are the same for all SCPI devices.

A1.3.5 Additional Information about SCPI

For more information about SCPI refer to the SCPI Standard or to the SCPI Consortium in care of Bodie Enterprises, La Mesa, CA, (619) 697-8790

A2 TROUBLESHOOTING AND REPAIR

A2.1 INTRODUCTION

This section provides trouble shooting and repair information for the Model 488-PC2 Card.

A2.2 TROUBLESHOOTING PROCEDURE

If the installation was successful, the majority of problems are due to a misunderstanding of the GPIB commands, the format the different devices use for data messages (especially the terminators), and incorrect programming. Some system faults are caused by poor cabling contacts, overly long cables and mixing fast and slow handshake devices on the same bus. The remedy for these last three problems is to keep the Bus connectors clean and tight, follow the IEEE-488 Standard for cable lengths and to check the devices' specifications to avoid mixing devices with different handshake speeds on the same bus. Older devices with open collector data drivers should not be used with a high speed Bus Controller.

The best solution for the first three problems is to read each device's manual before starting the program to gain an understanding of the device's unique setup requirements and command/data formats. Use a Keyboard Controller program (see Getting Started) to test a device's response to unfamiliar commands before using it in a program. New GPIB users should read Appendix A1 and Section 4 of this manual. The Troubleshooting Guide in Table A-2 lists the symptom, probable cause and suggested corrective action for some of the more common bus problems.

TABLE A-2 TROUBLESHOOTING GUIDE

Symptom	Possible Fault	Action or Check
Computer will not boot up after board installed	Board not correctly placed in connector	Check card to be sure it is firmly seated in the connector
	Other boards loose	Check other boards to see that they were not disturbed.
	I/O Address Conflict	Check computer for I/O address conflict and reassign the 488-PC2 to another address
Computer hangs up when calling driver routines	IO address incorrect or does not match program setting	Check device manager settings for conflicts Check value in Init command Setting parameter
Card unable to control instruments	Installation fault	Check Device Manager to be sure the computer recognizes the card.
	Open Bus Cable	Check Bus Cable connections
	Instrument Power	Check Instrument
	Instrument Address	Check instrument address setting.
Instrument does not respond	Wrong instrument address	Address setting on the device Address setting used in the program. Device address conflicts with controller's address

A2

**TABLE A-2 TROUBLESHOOTING GUIDE
(CONTINUED)**

Symptom	Possible Fault	Action or Check
	<p data-bbox="425 228 649 253">Bad bus connections</p> <p data-bbox="425 358 678 415">Wrong instrument programming sequence</p> <p data-bbox="425 578 592 634">Missing escape sequence</p>	<p data-bbox="712 228 982 318">Check bus with bus analyzer or substitute a known good instrument</p> <p data-bbox="712 358 959 415">Verify with instrument manual</p> <p data-bbox="712 448 982 537">Test with Keyboard Controller. Start with simple commands like *idn?</p> <p data-bbox="712 578 982 699">Instrument needs an escape sequence before recognizing commands</p>
<p data-bbox="161 737 345 794">Bad or corrupted data</p>	<p data-bbox="425 737 626 794">Bad or dirty GPIB Bus Cable</p>	<p data-bbox="712 737 902 761">Check Bus Cable</p> <p data-bbox="712 834 976 859">Clean Cable Connectors</p>
<p data-bbox="161 899 385 956">Instrument hangs up when sends data out</p>	<p data-bbox="425 899 621 956">Instrument output control not reset</p> <p data-bbox="425 997 649 1086">Instrument output terminates on wrong characters</p>	<p data-bbox="712 899 959 956">Send instrument a Selective Device Clear</p> <p data-bbox="712 997 982 1118">Verify instrument output terminator matches what the program expects. See the ieEOL command.</p>

A2.3 BOARD TESTS

A2.3.1 General tests

If the GPIB Controller appears to have suddenly failed, check the Device Manager to be sure that it is still installed. Be sure the card is physically plugged into the computer. Cable stresses can cause the card to become loose.

Use the Keyboard Controller program and a known good device to test the GPIB controller. Read back the device's IDN message and verify that all of the characters are correct and that there are no dropped bits.

A2.3.2 BASIC Test Program

The 488.2 DOS Driver Disk contains a board test routine in the BASIC directory that verifies most of the 488-PC2's circuits. This program checks the card's internal data bus, GPIB controller chip, the interrupt and the DMA logic. Of necessity, this program is limited to only being able to check functions that are not being used by the other devices, i.e. it cannot check DMA level2 with a floppy disk controller in the system.

Table A-3 lists the board test errors and the corrective action. When the test is run, any errors are displayed on the monitor. In case of a conflict between the test program and Table A-3, the board test program error messages take precedence over those in Table A-3. Check the BASIC readme file for any changes to the Board Test Program.

A2.3.3 PC2TEST Program

The Utility Directory contains a test program that checks the 488-PC2's data and signal lines for boards at I/O address 02E1. This program tests some of the 488.2 logic that was added after the BASIC test was written. To run the PC2TEST Program, disconnect any GPIB bus cables from the card. Select the Utility directory and type PC2TEST at the DOS prompt. The program will display any errors it finds on the PC monitor.

TABLE A-3 BOARD TEST ERROR CODES

Error Code	Error	Cause	Correction
1	Unable to access the 488-PC2	Hardware failure Data bus fault Bus transceiver failure Decoder failure	Replace card Check internal bus for shorts Check data transceiver Replace PAL
2	Unable to access card's I/O address	Bad I/O address selection Hardware failure Defective GPIB chip	Check other cards I/O addresses Check CMD latch setting Replace GPIB chip
40	Interrupt level failure	IRQ Switch value wrong Interrupt level not available for use DCDR Pal failure GPIB chip failure	Verify IRQ switch setting Other devices in the PC using that level Replace PAL Replace GPIB controller
50	DMA channel failure	DMA switch value DMA channel not available for use DCDR Pal failure GPIB chip failure	Verify DMA switch Other devices in the PC using the channel Replace PAL Replace GPIB controller

A2

A2.4 REPAIR AND RETURNING FOR REPAIR

Repair of the 488-PC2 Card is done by returning the unit to the factory or to your local distributor. Units in warranty should **always** be returned to the factory or else repaired only after receiving permission from an ICS customer service representative.

When returning a unit, a board assembly, or other products to ICS for repair, it is necessary to go through the following steps:

1. Contact the ICS customer service department and ask for a return material authorization (RMA) number. An ICS applications engineer will want to discuss the problem at this time to verify that the unit needs to be returned, or to assist in correcting the problem. We have discovered that more than one-third of the difficulties customers call about can be resolved over the phone as opposed to returning a unit for repair.
2. Write a description of the problem and attach it to the material being returned. Describe the installation, system failure symptoms, and how it was being used. If the item being returned is a board assembly, describe how you isolated the fault to it. Include your name and phone number so we can call you if we have any questions. Remember, we need to locate the problem in order to fix it.
3. Pack the item with the fault description in a box large enough to accommodate a minimum of two inches of packing material on all four sides, the top, and the bottom of the box. Securely seal the box.
4. Mark the shipping label to the attention of RMA #_____. The RMA number is very important since it is our way of identifying your unit in order to return it to you.
5. Ship the box to ICS freight prepaid. ICS does **not** pay freight to return the unit to ICS, but will prepay the freight to return the repaired item to you.

Index

Symbols

- 488-PC2
 - Compatibility 1-2
 - Description 2-1
 - DMA Operations 4-12
 - Features 2-1
 - Shipment Verification 1-2
 - Specifications 2-6
 - Switch Settings 1-7
- 488.2 Driver
 - Description 2-2
 - DMA data transfers 4-11
 - DOS Support 3-4
 - Features 2-2
 - Quick Reference table 3-6
 - Supported Languages 2-3
 - Utilities 3-5
 - WIN16 Components 3-3
 - WIN32 Components 3-1
- 488.2 Drivers
 - Short Form Command List 3-6
- 488.2 Programming 4-9
- 488.2 Protocols 4-9

A

- ABORT command 9-7
- Accessories 2-9
- Address
 - Switch setting 1-7
- ALLSPOLL command 9-8, 9-9
- Approvals 2-8
- Architecture
 - 488-PC2 2-6

B

- Backups 1-4
- BASIC
 - Directory contents 5-2
 - PC2 interrupts 5-5
- BASIC Drivers 5-1
 - Installation 5-1
 - Programming 5-3
 - Using interrupts 5-5
- Board Test Routines
 - BASIC Test Program
 - Error Codes A-19
- Bodie Enterprises A-14
- BORC_CPP directory contents, table 5-20
- Bus device functions 2-5



C

C/C++

- Demo program 5-24
- Demo programs 5-18
- Installation 5-18
- Interrupts 5-22
- Library files 5-18
- Programming 5-18, 5-20
- Using 488-PC2 as a device 5-23
- Using the IEEE-488 library functions 5-21

CardID 4-2

Certifications 2-8

CLOSE command 9-10

Command

- ABORT 9-7
- ALLSPOLL 9-8, 9-9
- Calling Conventions 4-4
- CLOSE 9-10
- Common Parameters 9-2
- Constants and default values 9-1
- Conventions 9-1
- Default constant values, table 9-4, 9-5
- DEVCLR 9-11
- DEVICE 9-12
- ENTER 9-13, 9-14
- ENTERB 9-15, 9-16, 9-17
- EOL 9-18, 9-19
- Error Codes 9-5
- ERRPTR 9-20
- FINDLSTN 9-24, 9-25
- FINDRQS 9-26, 9-27
- GOTOSTBY 9-28
- INIT 9-31, 9-32, 9-33
- LLO 9-34
- LOCAL 9-35, 9-36
- OUTPUT 9-37
- OUTPUTB 9-38, 9-39
- PASSCTL 9-40

PPOLL 9-41

PPOLLC 9-42

PPOLLU 9-43

Reference 9-1, 9-6

REMOTE 9-44

RESET 9-45

SEND 9-46, 9-47

SERVICEDISABLE 9-48

SERVICEENABLE 9-49, 9-50

S POLL 9-51, 9-52

STATUS 9-54, 9-55

Error Codes 9-55

Status Codes 9-54

TAKECTL 9-56, 9-57

Terminators 9-3

TIMEOUT 9-58, 9-59

TRIGGER 9-60

Usage, advanced

488-PC2 DMA operations 4-12

Passing control 4-11

Utility Functions 9-62

DELAY 9-62

TIMEOUT 9-64

TIMER 9-63

Command and address messages, IEEE 488 A-6

Commands

488-PC2 Table 3-6, 3-7, 3-8, 3-9, 3-10, 3-11

Default Value Table 9-3

Driver 3-6

Return Values 9-5

SCPI A-12

SCPI, example A-13

Common Command Parameters 9-2

Compatibility with earlier 488-PC2 Products 1-2

Control

Passing 4-11

Controller

Functions 2-4

D

- DEVCLR command 9-11
- Device
 - Program considerations 4-14
 - Using a PC 4-13
- DEVICE command 9-12
- Direct Memory Access. *See* DMA
- DLL. *See* PC2 Windows DLL
- DMA 2-6
 - 488-PC2 operations 4-12
 - Channel settings 1-10
 - Data Transfer Rates 2-7
 - Features 4-11
 - Switch setting 1-9
 - Transfer rates 2-7

DOS

- C Language Programming 5-18
- Interpretive Basic 5-1
- Pascal Programming 5-13
- Quick Basic Programming 5-7

E

- ENTER command 9-13, 9-14
- ENTERA command 9-15
- ENTERB command 9-16, 9-17
- EOL command 9-18, 9-19
- ERRPTR command 9-20
- EVENTSTAT function 9-21, 9-22, 9-23

F

- FINDLSTN command 9-24, 9-25
- FINDRQS command 9-26, 9-27
- Function
 - EVENTSTAT 9-21, 9-22, 9-23
 - GPIBSTAT 9-29, 9-30
 - SRQSTAT 9-53
 - WAITSRQ 9-61
- Functions
 - Bus Device 2-5
 - Controller 2-4

G

- General information 2-1
- GOTOSTBY command 9-28
- GPIB
 - Command terminators 9-3
 - Pinouts A-7
 - System Configuration
 - Guidelines A-5
- GPIB Bus
 - Signal-Pin Assignments A-7
- GPIB Keyboard
 - Operation 1-12
- GPIB Programming 4-1
- GPIBSTAT function 9-29, 9-30

H

- Hardware Installation 1-5

I

- I/O Address
 - Selection 1-7
 - Switch settings 1-7
 - Wait State Switch Setting 1-9
- IEEE 488
 - Command and address messages A-6
 - Message formats (IEEE 488.2) A-11
- IEEE 488 Bus
 - 488.2 required status reporting capabilities A-9
 - Commands A-5
 - Controller capabilities 2-4
 - Controllers A-3
 - Data Transmission A-4
 - Description A-2
 - Device Addressing A-3
 - Devices A-3
 - Electrical Pinouts A-7
 - Interface Messages A-5
 - Management Lines A-5
 - Mechanical Interface A-7
 - PC2 Device Capabilities 2-5

- Specifications 2-4
- System Guidelines A-5
- IEEE 488 Interface
 - Bus Figure A-2
 - SCPI command structure and examples A-12
 - SCPI commands A-12
 - SCPI error reporting A-14
- IEEE 488.1 Bus Description A-2
- IEEE 488.2 STANDARD A-8
 - Common Commands A-10
 - Differences from 488.1 A-11
 - Message Formats A-8
 - Reporting Structure A-8
- Information, general 2-1
- INIT command 9-31, 9-32, 9-33
- Installation 1-5
 - Hardware 1-5
 - Software 1-6
 - DOS 1-6
 - Testing GPIB Interface 1-15
- Interactive Command Line Program
 - PC2-KYBD 1-14
- Interrupts
 - Level settings
 - 488-PC2 1-10
 - PC2 BASIC 5-5
 - PC2 Quick Basic 5-11
 - PC2 settings 2-6
 - Quick Basic 5-11
 - Using in BASIC 5-5
 - using in C 5-22

K

- Keyboard Controller Program 1-12

L

- Languages
 - Supported 2-3
- LLO command 9-34
- LOCAL command 9-35, 9-36

M

- MSC_CPP directory contents, table 5-19
- Multiline Messages A-5

O

- OUTPUT command 9-37
- OUTPUTA command 9-38
- OUTPUTB command 9-39

P

- Pascal
 - BTPASCAL Directory 5-14
 - Installation 5-13
 - Microsoft Programming 5-17
 - MSPASCAL Directory 5-15
 - Programming 5-13, 5-15
 - Turbo Programming 5-16
- PASSCTL command 9-40
- PC
 - Interrupt
 - PC2 2-6
 - Using as a device 4-13
 - PC2_Kybd 1-14
 - Operation 1-14
- Physical
 - PC2 2-7
- PPOLL command 9-41
- PPOLLC command 9-42
- PPOLLU command 9-43
- Primary Address 4-2
- Program
 - C/C++
 - Demo program 5-24
 - Device considerations 4-14
 - Program Examples 4-15
 - Visual Basic 4-15
 - Visual C++ 4-18

- Programming 4-1
 - 488.2 Protocols 4-9
 - Basic GPIB Techniques 4-1
 - Clearing a device 4-6
 - Confirming a device's presence 4-9
 - Device Addressing 4-2
 - Examples 4-15
 - Finding devices 4-9
 - Initializing the Bus 4-5
 - Multiple 488-PC2 Cards 4-8
 - Notes & Cautions 4-11
 - Program Outline 4-1
 - Reading Data 4-6
 - Reading device status 4-6
 - Sending Data 4-5
 - Sending GPIB Bus commands 4-7
 - Timeouts 4-8

Q

- Quick Basic
 - Directory contents 5-8
 - Includes 5-10
 - Installation 5-7
 - Interrupts 5-11
 - Interrupts, PC2 5-11
 - Parameters 5-11
 - Programming 5-7, 5-9

R

- REMOTE command 9-44
- Repair A-20. *See also* Returning for Repair
- RESET command 9-45
- Returning for repair A-20
- Rocker
 - Assignments and switch location 1-8

S

- SCPI
 - Additional Information A-14
 - Channel list A-14
 - Examples A-14
 - Command structure and example A-12
 - Commands
 - Example A-13
 - Commands and queries A-12
 - Compound commands
 - examples A-13, A-14
 - Error Reporting A-14
 - Error reporting A-14
 - Variables A-14
- Secondary Address 4-2
- SEND command 9-46, 9-47
- SERVICEDISABLE command 9-48
- SERVICEENABLE command 9-49, 9-50
- Settings
 - Address Switch 1-7
 - DMA channel 1-10
 - DMA Switch 1-9
 - Interrupt level 1-10, 1-11
 - Wait delay 1-9
 - Wait State Switch 1-9
- Shipment verification 1-2
- Signal-Pin assignments
 - GPIB bus A-7
- Software
 - Organization 1-11
- Software License 1-3
- Specifications
 - Architecture
 - PC2 2-6
 - Bus device functions 2-5
 - DMA transfer rates 2-7
 - PC interrupt 2-6
 - Physical
 - PC2 2-7
- SPOLL command 9-51, 9-52
- SRQSTAT function 9-53

- Starting 1-1
- Status Code 9-54
- STATUS command 9-54, 9-55
- Switch
 - Address Settings 1-7
 - DMA setting 1-9
 - Location and rocker assignments 1-8
 - Wait State Setting 1-9

T

- TAKECTL command 9-56, 9-57
- Terminators
 - GPIO command 9-3
- Testing
 - 488-PC2 Installation 1-15
- TIMEOUT command 9-58, 9-59
- TRIGGER command 9-60
- Troubleshooting
 - BASIC Board Test ProgramA-18
 - Board Test Routines
 - Basic Test Error CodesA-19
 - Board Tests A-18
 - Guide A-16, A-17
 - PC2TEST Program A-18
 - Procedure A-15
- Troubleshooting and Repair A-15
- Troubleshooting Guide A-16, A-17

V

- Visual Basic examples 4-15

W

- Wait
 - Delay settings 1-9
 - State switch setting 1-9
- WAITSRQ function 9-61
- Warranty 1-3