ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *National Instruments VXIcpu-030* at our website: *Click HERE*

ESP-488 Software Reference Manual for the VXIcpu-030®

National Instruments IEEE-488 Engineering Software Package for the VxWorks Operating System



July 1994 Edition

Part Number 320345-01

© Copyright 1991, 1994 National Instruments Corporation.
All Rights Reserved.

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices:
Australia (03) 879 9422, Austria (0662) 435986, Belgium 02/757.00.20, Canada (Ontario) (519) 622-9310,
Canada (Québec) (514) 694-8521, Denmark 45 76 26 00, Finland (90) 527 2321, France (1) 48 14 24 24,
Germany 089/741 31 30, Italy 02/48301892, Japan (03) 3788-1921, Netherlands 03480-33466, Norway 32-848400,
Spain (91) 640 0085, Sweden 08-730 49 70, Switzerland 056/20 51 51, U.K. 0635 523545

Limited Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

NI-488® and VXIcpu-030® are trademarks of National Instruments Corporation.

Product names and company names listed are trademarks or trade names of their respective companies.

Warning Regarding Medical and Clinical Use of National Instruments Products

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual vii
Organization of This Manual vii
Conventions Used in This Manual viii
Related Documentation viii
Customer Communication viii

Chapter 1
Introduction 1-1
Important Considerations 1-1

Chapter 2
The C Language Library 2-1
Global Variables 2-1
Status Word – ibsta 2-1
Error Variable – iberr 2-2
Count Variable – ibcnt 2-3
Read and Write Termination 2-3
Compiling C Programs 2-4
GPIB Function Descriptions 2-4
Device-Level Functions 2-4
Low-Level Functions 2-5

Chapter 3
IBIC 3-1
Running IBIC 3-1
Syntax Translation Guidelines 3-1
Sample Session 3-2
Auxiliary Functions 3-3

Chapter 4
ESP-488 Functions and Utilities Reference 4-1
IBIC(1) 4-2
IBTEST(1) 4-6
DVCLR(3) 4-7
DVRD(3) 4-8
DVRSP(3) 4-10
DVTRG(3) 4-12
DVWRT(3) 4-13
IBCAC(3) 4-15
IBCMD(3) 4-16
IBEOS(3) 4-18
IBEOT(3) 4-20
IBGTS(3) 4-21

IBLINES(3)....4-22
IBONL(3)....4-23
IBPAD(3)....4-24
IBRD(3)....4-25
IBRPP(3)....4-27
IBRSV(3)....4-28
IBSAD(3)....4-29
IBSIC(3)....4-30
IBSRE(3)....4-31
IBTMO(3)....4-32
IBWAIT(3)....4-34
IBWRT(3)....4-36

Appendix A
Multiline Interface Command Messages....A-1

Appendix B
Software Configuration and Installation....B-1

Appendix C
GPIB Programming Example....C-1

Appendix D
Customer Communication....B-1

Glossary....Glossary-1

Tables

Table 2-1. Status Word Layout....2-1
Table 2-2. GPIB Error Codes....2-2

Table 3-1. Auxiliary Functions That IBIC Supports....3-3

Table 4-1. Syntax of ESP-488 Functions in IBIC....4-2
Table 4-2. Status Word Layout....4-4
Table 4-3. GPIB Error Codes....4-5
Table 4-4. Auxiliary Functions That IBIC Supports....4-5
Table 4-5. Data Transfer Termination Method....4-18
Table 4-6. Timeout Settings....4-32
Table 4-7. Wait Mask Layout....4-34

Table B-1. Software Distribution Files....B-1

About This Manual

This manual describes the IEEE 488 Engineering Software Package (ESP-488) for the VxWorks operating system (Version 5.0 and higher) from Wind River Systems, Inc. This package is intended to be used with the VXIcpu-030 controller.

Organization of This Manual

This manual is organized as follows:

Chapter 1, *Introduction*, contains an overview of the ESP-488 VxWorks software and describes important considerations for using the software.

Chapter 2, *The C Language Library*, contains a general description of the C language programming interface to the ESP-488 VxWorks device driver, including the GPIB device-level and low-level functions.

Chapter 3, *IBIC*, introduces you to the Interface Bus Interactive Control (`ibic`) program. This chapter also contains instructions for running `ibic`, contains guidelines for translating `ibic` syntax, contains a sample session, and summarizes the auxiliary functions that `ibic` supports.

Chapter 4, *ESP-488 Functions and Utilities Reference*, contains detailed information for using the functions and utilities contained in the ESP-488 software package. For ease of use, this material is presented in a format familiar to most users of the UNIX and VxWorks operating systems.

Appendix A, *Multiline Interface Command Messages*, is a listing of multiline interface messages.

Appendix B, *Software Configuration and Installation*, describes how to configure and install the software.

Appendix C, *GPIB Programming Example*, illustrates the steps involved in programming a representative IEEE 488 instrument from a terminal using the ESP-488 functions in C language. This appendix is designed to help you learn how to use the ESP-488 driver software to execute certain programming and control sequences.

Appendix D, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.

Glossary, contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, and symbols.

Conventions Used in This Manual

The following conventions are used to distinguish elements of text throughout this manual:

italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
bold italic	Bold italic text denotes a note.
`monospace`	Text in this font denotes text or characters that are to be literally input from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, variables, filenames and extensions, and for statements and comments taken from program code.

Note: References in this manual to IEEE 488 and IEEE 488.2 are referring to the ANSI/IEEE Standard 488.1-1987 and the ANSI/IEEE Standard 488.2-1987, respectively, which define the GPIB specification.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- ANSI/IEEE Standard 488-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*
- ANSI/IEEE Standard 488.2-1987, *IEEE Standard Codes, Formats, Protocols, and Common Commands.*
- *ESP-488 for VxWorks and the GPIB-1014/1014D Software Reference Manual* (part number 320429-01)
- *Getting Started with Your VXIcpu-030 and the NI-VXI Software for the VxWorks Operating System*
- *VxWorks Programmer's Guide*, Wind River Systems
- *VXIbus System Specification*, Revision 1.4, VXIbus Consortium (available from National Instruments, part number 350083-01)

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix D, *Customer Communication*, at the end of this manual.

Chapter 1
Introduction

This chapter contains an overview of the ESP-488 VxWorks software and describes important considerations for using the software.

ESP-488 is a functional subset of the industry standard NI-488 GPIB driver software. Standard ESP-488 implements an optimized set of 10 fundamental GPIB functions for low-level communication and control through a single GPIB interface. In addition to this core set of 10 functions, ESP-488 for VxWorks includes several functions for interface configuration and high-level device communication. Other features include timeout support, error reporting, and an interactive control utility (`ibic`) similar to the `ibic` included with NI-488 software packages.

Important Considerations

Before using the ESP-488 VxWorks software, you must install the software from tape. Refer to your Getting Started manual and also to Appendix B, *Software Configuration and Installation*, of this manual for instructions about installing the software.

Consider also the following points when using the ESP-488 software:

- The ESP-488 functions support synchronous I/O transfers through a single GPIB interface. The functions are intended to be accessed by only one program task at a time.

- All functions return a subset of the standard NI-488 status bit vector as described later in this manual. The result of the last call is also available in the global variable, `ibsta`. Additional information on the result of the last call is sometimes contained in the global variables `ibcnt` and `iberr`. Refer to Chapter 2, *The C Language Library*, for more information on the global variables.

- The GPIB interface is normally designated to be the System Controller. Most ESP-488 functions are optimized to assume the GPIB interface is also the Controller-In-Charge (CIC).

- Call the `ibonl` function to initialize the GPIB interface before any other call is made.

- Prior to calling `ibrd` or `ibwrt`, you must address the appropriate devices, including the GPIB interface, by calling `ibcmd` with the proper addressing commands.

- Five device-level calls are included with this package. All of these calls need the primary address (PAD) and secondary address (SAD) of the device you want to communicate with. If the device does not have a secondary address, pass a zero for the SAD portion of the address argument.

- Include the header file `ugpib.h` in any application program that uses the ESP-488 functions.

- Refer to the `Readme` file on the software distribution media for additional information on a specific ESP-488 VxWorks kit.

Chapter 2
The C Language Library

This chapter contains a general description of the C language programming interface to the ESP-488 VxWorks device driver, including the GPIB device-level and low-level functions.

Global Variables

The following sections explain the status word (`ibsta`), the error variable (`iberr`), and the count variable (`ibcnt`). These variables are updated each time a driver call is made, to reflect the status of the GPIB interface.

Status Word – ibsta

All functions return a status word which reports the success of the function call and information about the state of the GPIB interface. The status word is also available as the external variable `ibsta`.

The status word contains 16 bits, nine of which are meaningful. A bit value of one indicates that the corresponding condition is in effect; a bit value of zero indicates that the condition is not in effect. Table 2-1 lists each condition and the corresponding bit position to be tested for that condition.

Table 2-1. Status Word Layout

Mnemonic	Bit Position	Hex Value	Description
ERR	15	8000	GPIB error
TIMO	14	4000	Time limit exceeded
END	13	2000	END detected
SRQI	12	1000	SRQ is asserted
CMPL	8	100	I/O completed
CIC	5	20	Controller-In-Charge
ATN	4	10	Attention is asserted
TACS	3	8	Talker
LACS	2	4	Listener

A description of each status bit and its condition follows.

ERR The ERR bit is set in the status word following any call that results in an error; the particular error can be determined by examining the `iberr` variable. The ERR bit is cleared following any call that does not result in an error.

TIMO The TIMO bit indicates whether the time limit for I/O completion has been exceeded.

END The END bit indicates whether the END message has occurred during a read operation.

SRQI The SRQI bit indicates whether the GPIB line SRQ is asserted.

CMPL The CMPL bit indicates that the previous I/O operation is complete. Because I/O is synchronous, CMPL is always set.

CIC The CIC bit indicates whether the GPIB interface is the Controller-In-Charge.

ATN The ATN bit indicates whether the GPIB line ATN is asserted.

TACS The TACS bit indicates whether the GPIB interface is addressed to talk.

LACS The LACS bit indicates whether the GPIB interface is addressed to listen.

Error Variable – iberr

When the ERR bit is set in the status word, a GPIB error has occurred. One of the following error codes is returned in the external variable `iberr.`

Table 2-2. GPIB Error Codes

Suggested Mnemonic	Decimal Value	Explanation
ECIC	1	Function requires GPIB interface to be CIC
ENOL	2	Write handshake error (e.g., no listener)
EADR	3	GPIB interface not addressed correctly
EARG	4	Invalid argument to function call
EABO	6	I/O operation aborted
ENEB	7	GPIB interface is offline
EDMA	8	DMA hardware error
EBUS	14	GPIB bus error

A description of each error and some conditions under which it may occur follow:

ECIC (1) This code is returned when a call requiring the GPIB interface to be Controller-In-Charge (CIC) is made, but the interface is not CIC. This could have happened because the interface was never made CIC, or it passed control to another controller.

ENOL (2) The most common cause of this error code is when a write operation is attempted with no listeners addressed. For a device write, this indicates that the GPIB address passed in to the driver does not match the GPIB address of any device connected to the bus. For a low-level write, the appropriate addressing commands were not previously sent.

This error may also occur in situations in which the GPIB interface is not the Controller-In-Charge and the controller asserts ATN before the write call in progress has ended.

EADR (3) This error results from the GPIB interface not addressing itself before read and write calls when it is the Controller-In-Charge.

EARG (4) This error results when an invalid argument is passed to a function call.

EABO (6) This error indicates that I/O has been cancelled. This error usually results from a timeout condition.

ENEB (7) This error, which literally means *non-existent board*, occurs when the GPIB interface is offline.

EDMA (8) This error indicates that a DMA hardware error occurred during an I/O operation.

EBUS (14) This error indicates a GPIB bus error during a device call. This is usually the result of the internal time limit being exceeded.

Count Variable – ibcnt

The `ibcnt` variable is updated after each read, write, or command function call with the number of bytes actually transferred by the operation.

Read and Write Termination

The IEEE 488 specification defines two methods of identifying the last byte of device-dependent (data) messages. The two methods permit a talker to send data messages of any length without the listener(s) knowing in advance the number of bytes in the transmission. The two methods are as follows:

- END message. In this method, the talker asserts the End Or Identify (EOI) signal simultaneously with transmission of the last data byte. By design, the listener stops reading when it detects a data message accompanied by EOI, regardless of the value of the byte.
- End-Of-String (EOS) character. In this method, the talker uses a special character at the end of its data string. By prior arrangement, the listener stops receiving data when it detects that character. Either a 7-bit ASCII character or a full 8-bit binary byte can be used.

These two methods can be used individually or in combination. However, it is important that the listener be properly configured to unambiguously detect the end of a transmission.

The GPIB interface always terminates `ibrd` operations on the END message. For `ibwrt` operations, the GPIB interface always sends the END message with the last byte in the transfer. Use the `ibeos` and `ibeot` functions to select other modes of operation.

Compiling C Programs

In addition to any VxWorks or other required include files, always include the file `ugpib.h` in every GPIB program. This file defines all status bits, error codes, and externals needed.

Compile the application program on a suitable UNIX host system using the native C compiler. For example, to compile the program `prog.c`, enter the following command:

```
cc -c -I/usr/vw/h -O prog.c
```

The resulting object module, `prog.o`, can be linked directly with the GPIB driver library `esp488.o` and then loaded into the VxWorks system, or `prog.o` can be loaded separately into a VxWorks system that already contains `esp488.o`. In the latter case, all ESP function references in `prog.o` are resolved dynamically by the VxWorks loader. Dynamic linking is the method used by the ESP-488 utility programs `ibic.o`, `ibtsta.o`, and `ibtstb.o`.

For more information on creating and running VxWorks applications, refer to *`cc(1)`* or the equivalent in your UNIX documentation, and to the *Cross-Development* section in the *VxWorks Programmer's Guide*.

GPIB Function Descriptions

The remainder of this chapter is intended as a quick reference to the GPIB device-level and GPIB low-level functions. Refer to Chapter 4, *ESP-488 Functions and Utilities Reference*, in this manual for more thorough information and specific examples.

Device-Level Functions

The following functions can be performed on a GPIB device at the specified address. All controller sequences conform to the IEEE 488.2 specification.

`dvclr(a)`	Sends the message Selected Device Clear (SDC) to a device at address `a`.
`dvrd(a,buf,cnt)`	Reads from the device at address `a` into a buffer.
`dvrsp(a,buf)`	Performs a serial poll of a device at address `a`.
`dvtrg(a)`	Triggers the device at address `a` by sending the message Group Execute Trigger (GET).
`dvwrt(a,buf,cnt)`	Writes from a buffer to the device at address `a`.

Low-Level Functions

The following functions can be performed directly on or through the GPIB interface.

`ibcac(v)`	Takes the interface from Controller Standby to Active Controller state (asserts ATN). `v` equal to 1 takes control synchronously, if possible. `v` equal to 0 takes control asynchronously. The interface must be CIC.
`ibcmd(buf,cnt)`	Sends a buffer of command messages. The interface must be CIC, but need not be Active Controller.
`ibeos(v)`	Changes the end-of-string (EOS) mode. The low byte contains the eos character and the high byte is any of `REOS`, `XEOS`, or `BIN`. `v` equal to 0 disables EOS checking.
`ibeot(v)`	Enables sending END with the last byte of every GPIB write. A value of 0 disables.
`ibgts()`	Puts the interface in standby state (deasserts ATN).
`iblines(clines)`	Returns the state of the GPIB control lines in `clines`.
`ibonl(v)`	Reinitializes the GPIB software and hardware. `v` equal to 1 places the interface online. `v` equal to 0 places the interface offline.
`ibpad(v)`	Changes the value of the primary GPIB address.
`ibrd(buf,cnt)`	Reads from the GPIB into a buffer. The interface must have been previously addressed to listen.
`ibrpp(buf)`	Executes a parallel poll. The interface must be CIC.
`ibrsv(v)`	Sets the serial poll response byte of the board. If bit 0x40 is set, the board asserts SRQ. If the board is CIC, it will not assert SRQ.
`ibsad(v)`	Changes the secondary GPIB address. `v` equal to 0 disables secondary address recognition.
`ibsic()`	Pulses Interface Clear (IFC).
`ibsre(v)`	Asserts Remote Enable (REN) if `v` equal to 1. `v` equal to 0 clears REN.
`ibtmo(v)`	Changes the timeout value. `v` equal to 0 disables timeouts. Timeout values are given in `ugpib.h`.
`ibwait(mask)`	Waits for events to occur. Valid mask bits are: TIMO, SRQI, CIC, TACS, and LACS.
`ibwrt(buf,cnt)`	Writes from a buffer to the GPIB. The interface must have been previously addressed to talk.

Chapter 3
IBIC

This chapter introduces you to the Interface Bus Interactive Control (`ibic`) program. This chapter also contains instructions for running `ibic`, contains guidelines for translating `ibic` syntax, contains a sample session, and summarizes the auxiliary functions that `ibic` supports.

Refer to Chapter 4, *ESP-488 Functions and Utilities Reference*, for detailed descriptions of the C language functions.

Running IBIC

From the VxWorks shell, load `ibic.o` using the `ld` command. For example,

```
ld < host:ibic.o
```

One or more ESP-488 driver modules must also be loaded into the system before running `ibic`.

Once the required modules are loaded, run `ibic` by entering the following command at the shell prompt:

```
ibic
```

If more than one driver module is loaded, `ibic` will initially direct all calls to the first module it can find in memory. Other modules can be activated using the `set` command (refer to the *Auxiliary Functions* section later in this chapter).

Syntax Translation Guidelines

To translate between C syntax and `ibic` syntax, use the following guidelines:

- Omit the parentheses around the function argument list.
- Regardless of which driver modules are loaded, all functions are called using the default naming syntax.

  `ib2wrt` becomes: `ibwrt`

- Functions with a single numeric argument are followed by a number.

  `ibsre(1)` becomes: `ibsre 1`

- Functions that write a buffer are followed by a string, but no count.

  `ibwrt("text",4)` becomes: `ibwrt "text"`

- Functions that read a buffer are followed by a count only.

  `ibrd(buf,50)` becomes: `ibrd 50`

- Functions that perform a poll take no buffer argument.

  `ibrpp(buf)` becomes: `ibrpp`

- Functions that take a mask argument are followed by a list of mask bits in parentheses.

  `ibwait(TIMO|SRQI)` becomes: `ibwait (timo srqi)`

Sample Session

The following is a sample session of `ibic` that triggers a digital voltmeter at address 3, waits for a service request, and reads in a buffer of data. User inputs are underlined.

```
ESP: ibonl 1
[0100]  ( cmpl )

ESP: dvclr 3
[0100]  ( cmpl )

ESP: dvwrt 3 "F3R7T3"
[0100] (  cmpl  )
count: 6

ESP: ibwait (srqi timo)
[0900]  ( srqi  cmpl  )

ESP: dvrsp 3
[0100]    (  cmpl  )
Poll: 0xC0

ESP: dvrd 3 10000
[2100]    (  end  cmpl  )
count: 10

01  02  03  04  05  06  25  07      . . . . . % .
62 03                               a .
```

Auxiliary Functions

Table 3-1 summarizes the auxiliary functions that `ibic` supports.

Table 3-1. Auxiliary Functions That IBIC Supports

Function Syntax	**Description**
`set ESP[x]`	Direct all subsequent calls to driver module `x`.
`help [option]`	Display help information. All available functions are briefly described.
`!`	Repeat previous command.
`-`	Turn printing *off*. This is most often used with the `$` command.
`+`	Turn printing *on*.
`n* function`	Execute command `n` times.
`n* !`	Execute previous command `n` times.
`$ filename`	Execute indirect file.
`print string`	Display string on screen.
`e`, `q`, or `^d`	Exit or quit `ibic`.

Chapter 4
ESP-488 Functions and Utilities Reference

This chapter contains detailed information for using the functions and utilities contained in the ESP-488 software package. For ease of use, this material is presented in a format familiar to most users of the UNIX and VxWorks operating systems.

IBIC(1) GPIB IBIC(1)

Name

`ibic` - interface bus interactive control program

Synopsis

```
ld < host:ibic.o
ibic
```

Description

`ibic` is a command language for controlling the National Instruments GPIB interface. It executes commands read from `stdin` or a file and returns detailed status information. All commands from the GPIB library `esp488.o` are supported.

Commands

Table 4-1 summarizes the ESP-488 functions and syntax when called from `ibic`.

Table 4-1. Syntax of ESP-488 Functions in IBIC

Function Syntax	Description	Function Type	Note
`dvclr a`	Clear specified device	device-level	1
`dvrd a v`	Read data from a device	device-level	1,5
`dvrsp a`	Return serial poll byte	device-level	1
`dvtrg a`	Trigger selected device	device-level	1
`dvwrt a string`	Write data to a device	device-level	1,4
`ibcac [v]`	Become Active Controller	low-level	2,3
`ibcmd string`	Send commands from string	low-level	4
`ibeos v`	Change/disable EOS message	low-level	3
`ibeot [v]`	Enable/disable END message	low-level	2,3
`ibgts`	Go from Active Controller to Standby	low-level	
`iblines`	Get state of GPIB control lines	low-level	
`ibonl [v]`	Place GPIB interface online or offline	low-level	2,3
`ibpad v`	Change primary address	low-level	3
`ibrd v`	Read data	low-level	5

(continues)

Table 4-1. Syntax of ESP-488 Functions in IBIC (continued)

Function Syntax	Description	Function Type	Note
`ibrpp`	Conduct a parallel poll	low-level	
`ibrsv v`	Request service	low-level	3
`ibsad v`	Change secondary address	low-level	3
`ibsic`	Send interface clear	low-level	
`ibsre [v]`	Set/clear remote enable line	low-level	2,3
`ibtmo v`	Change/disable time limit	low-level	3
`ibwait [mask]`	Wait for selected event	low-level	2,6
`ibwrt string`	Write data	low-level	4

Notes

1. `a` is the hex, octal, or decimal integer (see note 3) specifying the GPIB address of the device. The least significant byte (bits 0 through 7) contains the primary address and the next least significant byte (bits 8 through 15) contains the secondary address. If the device has no secondary address, pass a zero in bits 8 through 15.

2. Values enclosed in square brackets (`[]`) are optional. The default value is zero for `ibwait` and one for all other functions.

3. `v` is a hex, octal, or decimal integer. Hex numbers must be preceded by zero and x (for example, `0xD`). Octal numbers must be preceded by zero only (for example, `015`). Other numbers are assumed to be decimal.

4. `string` consists of a list of ASCII characters, octal or hex bytes, or special symbols. The entire sequence of characters must be enclosed in quotation marks. An octal byte consists of a backslash character followed by the octal value. For example, octal 40 would be represented by `\40`. A hex byte consists of a backslash character and a character `x` followed by the hex value. For example, hex 40 would be represented by `\x40`. Two special symbols are `\r` for a carriage return character and `\n` for a linefeed character. These symbols are a convenient method for inserting the carriage return and linefeed characters into a string, as shown in the following string: `"F3R5T1\r\n"`. Because the carriage return can be represented equally well in hex, `\xD` and `\r` are equivalent strings.

5. `v` is the number of bytes to read.

6. `mask` is a hex, octal, or decimal integer (see note 3) or a mask bit mnemonic.

Return Values

All `ibic` functions return a status word in both hex and bit mnemonic form. Table 4-2 lists the mnemonics of the status word. (This is the same information that is given in Table 2-1.)

Table 4-2. Status Word Layout

Mnemonic	Bit Position	Hex Value	Description
ERR	15	8000	GPIB error
TIMO	14	4000	Time limit exceeded
END	13	2000	END detected
SRQI	12	1000	SRQ is asserted
CMPL	8	100	I/O completed
CIC	5	20	Controller-In-Charge
ATN	4	10	Attention is asserted
TACS	3	8	Talker
LACS	2	4	Listener

If the ERR bit is set, an error mnemonic will be displayed as shown in Table 4-3. (This is the same information that is given in Table 2-2.)

Table 4-3. GPIB Error Codes

Suggested Mnemonic	Decimal Value	Explanation
ECIC	1	Function requires GPIB interface to be CIC
ENOL	2	Write handshake error (e.g., no listener)
EADR	3	GPIB interface not addressed correctly
EARG	4	Invalid argument to function call
EABO	6	I/O operation aborted
ENEB	7	GPIB interface is offline
EDMA	8	DMA hardware error
EBUS	14	GPIB bus error

Auxiliary Functions

Table 4-4 summarizes the auxiliary functions that `ibic` supports. (This is the same information that is given in Table 3-1.)

Table 4-4. Auxiliary Functions That IBIC Supports

Function Syntax	Description
`set ESP[x]`	Direct all subsequent calls to driver module `x`.
`help [option]`	Display help information. All available functions are briefly described.
`!`	Repeat previous command.
`-`	Turn printing *off*. This is most often used with the `$` command.
`+`	Turn printing *on*.
`n* function`	Execute command `n` times.
`n* !`	Execute previous command `n` times.
`$ filename`	Execute indirect file.
`print string`	Display string on screen.
`e`, `q`, or `^d`	Exit or quit `ibic`.

See Also

Chapter 2, *The C Language Library*
Chapter 3, *ibic*

IBTEST(1) GPIB IBTEST(1)

Name

`ibtsta`, `ibtstb` - installation tests (parts A and B) for ESP-488

Synopsis

```
ld < host:ibtsta.o
ibtsta [x]
ld < host:ibtstb.o
ibtstb [x]
```

Description

`ibtsta` and `ibtstb` are test programs for verifying the correct installation and operation of an ESP-488 library. If the optional argument `x` is specified, the test is run on the indicated driver module. For example,

```
ibtsta 1
```

will run installation test part A on `esp488_1.o`. If the `x` argument is omitted, the test is run on the default module, `esp488.o`, or on the first module found in memory.

`ibtsta` checks for basic driver functionality, takes only a few seconds to complete, and requires no interaction from the user. `ibtstb` performs a more thorough check of I/O and interrupt operation and requires the use of a GPIB analyzer. Both tests give on-screen instructions at program startup for the user to set up and run the test.

`ibtsta` should be run first. If `ibtsta` completes with no errors and a GPIB analyzer is available, `ibtstb` should then be run. `ibtstb` may be omitted if an analyzer is not available.

See Also

ibic (1)
Chapter 2, *The C Language Library*

DVCLR(3) device-level DVCLR(3)

Name

`dvclr` - send Selected Device Clear (SDC) to a GPIB device

Synopsis

```
#include "ugpib.h"
dvclr (a)
int a;
```

Description

`a` is the GPIB address of the device. The least significant byte (bits 0 through 7) contains the primary address and the next least significant byte (bits 8 through 15) contains the secondary address. If the device has no secondary address, pass a zero in bits 8 through 15.

The `dvclr` function sends the message SDC, the meaning of which depends on the specific device. SDC usually resets all device functions. `dvclr` sends the following commands and information:

- Talk address of the GPIB interface
- Secondary address of the GPIB interface, if applicable
- Unlisten (UNL)
- Listen address of the device
- Secondary address of the device, if applicable
- Selected Device Clear (SDC)

Examples

1. Clear the device at address 3.

```
dvclr(3);
```

2. Clear the device at primary address 5 and secondary address 0x61.

```
dvclr(0x6105);
```

See Also

ibcmd(3)
Chapter 2, *The C Language Library*

DVRD(3) device-level DVRD(3)

Name

`dvrd` - read data from a GPIB device into a buffer

Synopsis

```
#include "ugpib.h"
dvrd (a,buf,cnt)
int a,cnt;
char buf[];
```

Description

`a` is the GPIB address of the device. The least significant byte (bits 0 through 7) contains the primary address and the next least significant byte (bits 8 through 15) contains the secondary address. If the device has no secondary address, pass a zero in bits 8 through 15. `buf` identifies the buffer to use. `cnt` specifies the number of bytes to read from the GPIB.

The `dvrd` function reads `cnt` bytes of data from a GPIB device. Prior to reading the data, `dvrd` sends the following commands and information:

- Unlisten (UNL)
- Listen address of the GPIB interface
- Secondary address of the GPIB interface, if applicable
- Talk address of the device
- Secondary address of the device, if applicable

When the `dvrd` function returns, `ibsta` holds the latest GPIB status; `ibcnt` is the actual number of data bytes read from the device; and `iberr` is the first error detected if the ERR bit in `ibsta` is set.

The `dvrd` operation terminates on any of the following events.

- Allocated buffer becomes full.
- Error is detected.
- Time limit is exceeded.
- END message is detected.

After termination, `ibcnt` contains the number of bytes read. A short count can occur on any of the above events but the first.

Examples

1. Read 56 bytes of data from the device at address 5 and secondary address 0x61.

```
dvrd(0x6105,rdbuf,56);
/* Check ibsta to see how the read terminated: on CMPL,   */
/* END, TIMO, or ERR.                                     */
/* Data is stored in rdbuf.                               */
```

2. Read 1024 bytes of data from the device at talk address 0x4C (ASCII L).

```
dvrd(0x4c,rdbuf,1024);
/* Check ibsta to see how the read terminated: on CMPL,   */
/* END, TIMO, or ERR.                                     */
/* Data is stored in rdbuf.                               */
```

See Also

ibcmd(3) and *ibrd(3)*
Chapter 2, *The C Language Library*

DVRSP(3) device-level DVRSP(3)

Name

`dvrsp` - return serial poll status byte from a GPIB device

Synopsis

```
#include "ugpib.h"
dvrsp (a,spr)
int a;
char spr[];
```

Description

`a` is the GPIB address of the device. The least significant byte (bits 0 through 7) contains the primary address and the next least significant byte (bits 8 through 15) contains the secondary address. If the device has no secondary address, pass a zero in bits 8 through 15. `spr` is the buffer in which the poll response is stored.

The `dvrsp` function is used to serial poll one device and obtain its status byte. If the 0x40 (RQS) bit of the response is set, the status response is positive, that is, the device is requesting service.

`dvrsp` sends the following commands and information:

- Unlisten (UNL)
- Listen address of the GPIB interface
- Secondary address of the GPIB interface, if applicable
- Serial Poll Enable (SPE)
- Talk address of the device
- Secondary address of the device, if applicable

After the response byte is read, `dvrsp` sends the following commands:

- Serial Poll Disable (SPD)
- Untalk (UNT)

The interpretation of the response in `spr`, other than the RQS bit, is device-specific. For example, the polled device might set a particular bit in the response byte to indicate that it has data to transfer, and another bit to indicate a need for reprogramming. Consult the documentation for the device for interpretation of the response byte.

Example

Obtain the serial poll response byte from the device at address 7.

```
dvrsp (7,spr);
/* The application program would then analyze the response*/
/* in spr.   */
```

See Also

ibcmd(3) and *ibrd(3)*
Chapter 2, *The C Language Library*

DVTRG(3) device-level DVTRG(3)

Name

`dvtrg` - send Group Execute Trigger (GET) to a GPIB device

Synopsis

```
#include "ugpib.h"
dvtrg (a)
int a;
```

Description

`a` is the GPIB address of the device. The least significant byte (bits 0 through 7) contains the primary address and the next least significant byte (bits 8 through 15) contains the secondary address. If the device has no secondary address, pass a zero in bits 8 through 15.

The `dvtrg` function addresses and triggers the specified device. `dvtrg` sends the following commands and information:

- Talk address of the GPIB interface
- Secondary address of the GPIB interface, if applicable
- Unlisten (UNL)
- Listen address of the device
- Secondary address of the device, if applicable
- Group Execute Trigger (GET)

The response to a trigger is device-dependent.

Examples

1. Trigger the device at address 3.

```
dvtrg(3);
```

2. Trigger the device at primary address 5 and secondary address 0x61.

```
dvtrg(0x6105);
```

See Also

ibcmd(3)
Chapter 2, *The C Language Library*

DVWRT(3) device-level DVWRT(3)

Name

`dvwrt` - write data to a GPIB device from a buffer

Synopsis

```
#include "ugpib.h"
dvwrt (a,buf,cnt)
int a,cnt;
char buf[];
```

Description

`a` is the GPIB address of the device. The least significant byte (bits 0 through 7) contains the primary address and the next least significant byte (bits 8 through 15) contains the secondary address. If the device has no secondary address, pass a zero in bits 8 through 15. `buf` contains the data to be sent over the GPIB. `cnt` specifies the number of bytes to be sent over the GPIB.

The `dvwrt` function writes `cnt` bytes of data to a GPIB device. Prior to writing the data, `dvwrt` sends the following commands and information:

- Talk address of the GPIB interface
- Secondary address of the GPIB interface, if applicable
- Unlisten (UNL)
- Listen address of the device
- Secondary address of the device, if applicable

When the `dvwrt` function returns, `ibsta` holds the latest GPIB status, `ibcnt` is the actual number of data bytes written to the device, and `iberr` is the first error detected if the ERR bit in `ibsta` is set.

The `dvwrt` operation terminates on any of the following events:

- All bytes are transferred.
- Error is detected.
- Time limit is exceeded.

After termination, `ibcnt` contains the number of bytes written. A short count can occur on any of the above events but the first.

Examples

1. Write 10 instruction bytes to the device at address 5 and secondary address 0x61.

```
dvwrt(0x6105,"F3R1X5P2G0",10);
```

2. Write five instruction bytes terminated by a carriage return and a linefeed to the device at address 3.

```
dvwrt(3,"IP2X5\r\n",7);
```

See Also

ibcmd(3) and *ibwrt(3)*
Chapter 2, *The C Language Library*

IBCAC(3) low-level IBCAC(3)

Name

`ibcac` - become Active Controller

Synopsis

```
#include "ugpib.h"
ibcac (v)
int v;
```

Description

`v` identifies the method used to take control.

If `v` is non-zero, the GPIB interface takes control synchronously with respect to data transfer operations; otherwise, the GPIB interface takes control immediately (and possibly asynchronously).

To take control synchronously, the GPIB interface waits before asserting the ATN signal so that data being transferred on the GPIB will not be corrupted. If a data handshake is in progress, the take control action is postponed until the handshake is complete; if a handshake is not in progress, the take control action is done immediately. Synchronous take control is not guaranteed if an `ibrd` or `ibwrt` operation completed with a timeout or error.

Asynchronous take control should be used in situations where it appears to be impossible to gain control synchronously (for example, after a timeout error).

It is generally not necessary to use the `ibcac` function. Functions, such as `ibcmd` and `ibrpp` (which require that the GPIB interface take control), take control automatically.

The ECIC error results if the GPIB interface is not Controller-In-Charge.

Examples

1. Take control immediately without regard to any data handshake in progress.

```
ibcac(0);
```

2. Take control synchronously and assert ATN following a read operation.

```
ibrd(rd,512);
ibcac(1);
```

See Also

Chapter 2, *The C Language Library*

IBCMD(3) low-level IBCMD(3)

Name

`ibcmd` - send command message to GPIB

Synopsis

```
#include "ugpib.h"
ibcmd (cmd,cnt)
int cnt;
char cmd[];
```

Description

`cmd` contains the commands to be sent over the GPIB. `cnt` specifies the number of bytes to be sent over the GPIB.

The `ibcmd` function is used to transmit interface messages (commands) over the GPIB. These commands, which are listed in Appendix A, *Multiline Interface Command Messages*, include device talk and device listen addresses, secondary addresses, serial and parallel poll configuration messages, and device clear and device trigger instructions. The `ibcmd` function is also used to pass GPIB control to another device. This function is *not* used to transmit programming instructions to devices; programming instructions and other device-dependent information are transmitted with the `ibwrt` or `dvwrt` functions.

The `ibcmd` operation terminates on any of the following events:

- All commands are successfully transferred.
- Error is detected.
- Time limit is exceeded.
- Take Control (TCT) command is sent.

After termination, the `ibcnt` variable contains the number of commands sent. A short count can occur on any of the above events but the first.

An ECIC error results if the GPIB interface is not Controller-In-Charge. If it is not Active Controller, it takes control and asserts ATN prior to sending the command bytes. It remains Active Controller afterward.

In the examples that follow, GPIB commands and addresses are coded as printable ASCII characters. When the hex values to be sent over the GPIB correspond to printable ASCII characters, this is the simplest means of specifying the values. Refer to Appendix A for conversions of hex values to ASCII characters.

Examples

1. Unaddress all listeners with the Unlisten command (ASCII ?) and address a talker at 0x46 (ASCII F) and a listener at 0x31 (ASCII 1).

```
ibcmd("?F1",3);          /* UNL TAD LAD                    */
```

2. Unaddress all listeners with the Unlisten command (ASCII ?) and address a talker at 0x46 (ASCII F) and a listener at 0x31 (ASCII 1) and 0x6E (ASCII n).

```
ibcmd("?F1n",4);         /* UNL TAD LAD SAD                */
```

3. Clear all GPIB devices (that is, reset internal functions) with the Device Clear (DCL) command (0x14).

```
ibcmd("\024",1);         /* DCL (octal 24 or hex 14)       */
```

4. Clear two devices with listen addresses of 0x21 (ASCII !) and 0x28 (ASCII () with the Selected Device Clear (SDC) command (0x4).

```
ibcmd("?!(\004",4);      /* UNL LAD LAD SDC                */
```

5. Trigger any devices previously addressed to listen with the Group Execute Trigger (GET) command (0x8).

```
ibcmd("\010",1);         /* GET                            */
```

6. Unaddress all listeners and serial poll a device at talk address 0x52 (ASCII R) using the Serial Poll Enable (0x18) and Serial Poll Disable (0x19) commands (the listen address of the GPIB interface is 0x20 or ASCII blank).

```
ibcmd("?R \030",4);      /* UNL TAD MLA SPE                */
ibrd(rd,1);              /* read one byte                  */
/*   After checking the status byte in rd[0], disable this*/
/*   device and unaddress it with the Untalk (UNT) command*/
/*   (0x5F or ASCII _) before polling the next one.       */
ibcmd("\031_",2);        /* SPD UNT                        */
```

See Also

dvtrg(3), *dvclr(3)*, *dvrsp(3)*, *ibcac(3)*, *ibgts(3)*, and *ibtmo(3)*.
Chapter 2, *The C Language Library*

IBEOS(3) low-level IBEOS(3)

Name

`ibeos` - change or disable end-of-string mode

Synopsis

```
#include "ugpib.h"
ibeos (v)
int v;
```

Description

`v` selects the EOS character and the data transfer termination method according to Table A-5. `ibeos` is needed only to alter the value from its default setting of zero.

The assignment made by this function remains in effect until `ibeos` is called again or the `ibonl` function is called.

Table 4-5. Data Transfer Termination Method

Method	Value of v	
	High Byte	Low Byte
A. Terminate read when EOS is detected.	0x04 (`REOS`)	EOS
B. Set EOI with EOS on write function.	0x08 (`XEOS`)	EOS
C. Compare all 8 bits of EOS byte rather than low 7 bits (all read and write functions).	0x10 (`BIN`)	EOS

Methods A and C determine how read operations terminate. If Method A alone is chosen, reads terminate when the low seven bits of the byte that is read match the low seven bits of the EOS character. If Methods A and C are chosen, a full 8-bit comparison is used.

Methods B and C together determine when write operations send the END message. If Method B alone is chosen, the END message is sent automatically with the EOS byte when the low seven bits of that byte match the low seven bits of the EOS character. If Methods B and C are chosen, a full 8-bit comparison is used.

The options coded in `v` are used for both low-level and device-level reads and writes.

Examples

1. Send END when the linefeed character is written for all subsequent write operations.

```
v = (XEOS<<8) | '\n';   /* or v = 0x080A                    */
ibeos(v);
wrt[0] = '1';           /* data bytes to be written         */
wrt[1] = '2';
wrt[2] = '3';
wrt[3] = '\n';          /* EOS character is last byte       */
dvwrt(3,wrt,4);
```

2. Program the GPIB interface to terminate a read on detection of the linefeed character (`'\n'==0x0A`) that is expected to be received within 512 bytes.

```
v = (REOS<<8) | '\n';   /* or v = 0x040A                    */
ibeos(v);
/* assume interface has been addressed; do low-level read */
ibrd(rd,512);
/* The END bit in ibsta is set if the read terminated     */
/* on the EOS character, with the actual number of bytes  */
/* received contained in ibcnt.                           */
```

3. Program the GPIB interface to terminate read operations on the 8-bit value 0x82 rather than the 7-bit character 0x0A.

```
v = ((BIN | REOS)<<8) | 0x82;  /* or v = 0x1482           */
ibeos(v);
/* assume interface has been addressed; do low-level read */
ibrd(rd,512);
/* The END bit in ibsta is set if the read terminated     */
/* on the EOS character, with the actual number of bytes  */
/* received contained in ibcnt.                           */
```

4. Disable use of the EOS character for all subsequent read and write operations.

```
ibeos(0);              /* No EOS modes enabled             */
```

5. Send END with linefeeds and terminate reads on linefeeds for all subsequent I/O operations.

```
v = ((REOS | XEOS)<<8) | 0x0A;/* or v = 0x180A            */
ibeos(v);
wrt[0] = '1';              /* data bytes to be written    */
wrt[1] = '2';
wrt[2] = '3';
wrt[3] = 0x0A;             /* EOS character is last byte  */
ibwrt(wrt,4);
```

See Also

ibeot(3) and *ibonl(3)*
Chapter 2, *The C Language Library*

IBEOT(3) low-level IBEOT(3)

Name

`ibeot` - change or disable END termination mode

Synopsis

```
#include "ugpib.h"
ibeot (v)
int v;
```

Description

If `v` is non-zero, the END message is sent automatically with the last byte of each write operation. If `v` is zero, END is not sent. `ibeot` is needed only to alter the value from its default setting of one.

The END message is sent by asserting the GPIB EOI signal during a data transfer. It is used to identify the last byte of a data string without having to use an End-Of-String character. `ibeot` is used primarily to send variable length binary data.

The option specified in `v` is used for both low-level and device-level write operations. The assignment made by this function remains in effect until `ibeot` is called again or the `ibonl` function is called.

Examples

1. Send the END message with the last byte of all subsequent write operations.

```
ibeot(1);                /* enable sending of EOI         */
/* It is assumed that wrt contains the data to be written */
/* to the GPIB device at address 7                        */
dvwrt(7,wrt,3);          /* write 3 bytes                 */
```

2. Stop sending END with the last byte for all subsequent write operations.

```
ibeot(0);                /* disable sending EOI           */
```

See Also

ibeos(3) and *ibonl(3)*
Chapter 2, *The C Language Library*

IBGTS(3) low-level IBGTS(3)

Name

`ibgts` - go from Active Controller to standby

Synopsis

```
#include "ugpib.h"
ibgts ()
```

Description

The `ibgts` function causes the GPIB interface to go to the Controller Standby state and to deassert the ATN signal if it is the Active Controller. `ibgts` permits GPIB devices to transfer data without the GPIB interface being a party to the transfer.

The ECIC error results if the GPIB interface is not Controller-In-Charge.

Example

Turn the ATN line off.

```
ibgts();
```

See Also

ibcmd(3) and *ibcac(3)*
Chapter 2, *The C Language Library*

IBLINES(3) low-level IBLINES(3)

Name

`iblines` - return the status of the GPIB control lines

Synopsis

```
#include "ugpib.h"
iblines (clines)
int *clines;
```

Description

A *valid* mask is returned along with the GPIB control line state information in `clines`. The low-order byte (bits 0 through 7) of `clines` contains a mask indicating the capability of the GPIB interface to sense the status of each GPIB control line. The next-order byte (bits 8 through 15) contains the GPIB control line state information. Bits 16 through 31 are undefined. The pattern of the defined bits is as follows:

7	6	5	4	3	2	1	0
EOI	ATN	SRQ	REN	IFC	NRFD	NDAC	DAV

To determine if a GPIB control line is asserted, first check the appropriate bit in the lower byte to determine if the line can be monitored. If the line can be monitored (indicated by a 1 in the appropriate bit position), check the corresponding bit in the upper byte. If the bit is set (1), the corresponding control line is asserted. If the bit is clear (0), the control line is deasserted.

Example

Test for Remote Enable (REN).

```
if (iblines(&clines) < 0)error();
if (!(clines & 0x10)) {
   printf("GPIB interface can't monitor REN!");
   exit();
}
if (clines & 0x1000)
   printf("REN is asserted.");
else
   printf("REN is not asserted.");
```

See Also

ibwait(3)
Chapter 2, *The C Language Library*

IBONL(3) low-level IBONL(3)

Name

`ibonl` - place the GPIB interface online or offline

Synopsis

```
#include "ugpib.h"
ibonl (v)
int v;
```

Description

`v` specifies online or offline.

`ibonl` initializes all hardware and software and is used to bring the GPIB interface online for the first time. `ibonl` must be called with `v` non-zero before any other GPIB functions can be called. If `v` is zero, the GPIB interface will be left offline, not participating in GPIB activity.

During program operation, call `ibonl` with `v` non-zero to reset the GPIB hardware and software to its power-on state.

Examples

1. Bring the GPIB interface online for the first time.

```
ibonl(1);
```

2. Disable the GPIB interface.

```
ibonl(0);
```

See Also

Chapter 2, *The C Language Library*

IBPAD(3) low-level IBPAD(3)

Name

`ibpad` - change primary address of the GPIB interface

Synopsis

```
#include "ugpib.h"
ibpad (v)
int v;
```

Description

`v` specifies the primary GPIB address.

`ibpad` is used to alter the primary address from its default setting of zero. The listen address is formed by adding 0x20 to the primary address; the talk address is formed by adding 0x40 to the primary address.

Only the low five bits of `v` are significant and they must be in the range of 0 through 0x1E.

The assignment made by this function remains in effect until `ibpad` is called again or the `ibonl` function is called.

Example

Change the primary GPIB listen and talk address of the GPIB interface from its current value to 0x27 and 0x47, respectively.

```
ibpad(7);
```

See Also

ibsad(3)
Chapter 2, *The C Language Library*

IBRD(3) low-level IBRD(3)

Name

`ibrd` - read data from the GPIB into a buffer

Synopsis

```
#include "ugpib.h"
ibrd (buf,cnt)
int cnt;
char buf[];
```

Description

`buf` identifies the buffer to use. `cnt` specifies the number of bytes to read from the GPIB.

The `ibrd` function reads `cnt` bytes of data from a GPIB device. The device is assumed to be already properly initialized and addressed.

If the GPIB interface is Controller-In-Charge (CIC), the `ibcmd` function must be called prior to `ibrd` to address a device to talk and the interface to listen. If the interface is not CIC, the device on the GPIB that is the CIC must perform the addressing.

If the GPIB interface is Active Controller, the interface is first placed in Standby Controller state, with ATN off, and remains there after the read operation is completed. An EADR error results if the interface is CIC but has not been addressed to listen with the `ibcmd` function. An EABO error results if the interface is not the CIC and is not addressed to listen within the time limit. An EABO error also results if the device that is to talk is not addressed and/or the operation does not complete for whatever reason within the time limit.

The `ibrd` operation terminates on any of the following events.

- Allocated buffer becomes full.
- Error is detected.
- Time limit is exceeded.
- END message is detected.

After termination, `ibcnt` contains the number of bytes read. A short count can occur on any of the above events but the first.

Example

Read 1024 bytes of data from a device at talk address 0x4C (ASCII L) and then unaddress it (the GPIB interface is at listen address 0x20 or ASCII blank).

```
ibcmd("?L ",3);    /* UNL TAD MLA                              */
ibrd(rdbuf,1024);
/* Check ibsta to see how the read terminated: on CMPL,   */
/* END, TIMO, or ERR.                                     */
/* Data is stored in rdbuf.                               */
/* Unaddress the talker and listener.                     */
ibcmd("_?",1);     /* UNT UNL                              */
```

See Also

ibcmd(3) and *dvrd(3)*
Chapter 2, *The C Language Library*

IBRPP(3) low-level IBRPP(3)

Name

`ibrpp` - conduct a parallel poll

Synopsis

```
#include "ugpib.h"
ibrpp (ppr)
char *ppr;
```

Description

`ppr` identifies the address where the parallel poll response byte is stored.

The `ibrpp` function causes the GPIB interface to conduct a parallel poll of previously configured devices by sending the Identify (IDY) message (ATN and EOI both asserted).

An ECIC error results if the GPIB interface is not Controller-In-Charge. If the GPIB interface is Standby Controller, it takes control and asserts ATN (becomes Active) prior to polling. It remains Active Controller afterward.

Examples

1. Remotely configure a device at listen address 0x23 to respond positively on DIO3 if its individual status bit is one, and then parallel poll all configured devices.

```
cmd[0] = 0x23;          /* device listen address          */
cmd[1] = PPC;
cmd[2] = PPE | S | 2;   /* send PPR3 if ist = 1           */
cmd[3] = UNL;
ibcmd(cmd,4);
ibrpp(&ppr);            /* PPR returned in ppr            */
```

2. Disable and unconfigure all GPIB devices from parallel polling using the PPU command.

```
ibcmd("\x15",1);        /* PPU                            */
```

See Also

ibcmd(3)
Chapter 2, *The C Language Library*

IBRSV(3) low-level IBRSV(3)

Name

`ibrsv` - request service and/or set serial poll status byte

Synopsis

```
#include "ugpib.h"
ibrsv (v)
int v;
```

Description

`v` specifies the serial poll response byte of the GPIB interface.

If the 0x40 bit is set in `v`, the GPIB interface additionally requests service from the controller by asserting the GPIB SRQ line.

The `ibrsv` function is used to request service from the controller using the SRQ signal and to provide a system-dependent status byte when the controller serial polls the GPIB interface.

It is not an error to call the `ibrsv` function when the GPIB interface is the Controller-In-Charge (CIC), although doing so makes sense only if control will be passed later to another device. In this case, the call updates the status byte, but the SRQ signal is asserted only if the 0x40 bit is set and only when control is passed.

Examples

1. Set the serial poll status byte to 0x41, which simultaneously requests service from an external CIC.

```
ibrsv(0x41);
```

2. Stop requesting service (unassert SRQ).

```
ibrsv(0);
```

3. Change the status byte without requesting service.

```
ibrsv(0x01);  /* new status byte value                    */
```

See Also

dvrsp(3)
Chapter 2, *The C Language Library*

IBSAD(3) low-level IBSAD(3)

Name

`ibsad` - change or disable secondary address of the GPIB interface

Synopsis

```
#include "ugpib.h"
ibsad (v)
int v;
```

Description

`v` is a valid secondary address.

If `v` is a number between 0x60 and 0x7E, that number becomes the secondary GPIB address of the GPIB interface. If `v` is 0 or 0x7F, secondary addressing is disabled. `ibsad` is needed only to alter the value from its default setting of zero (disabled).

The assignment made by this function remains in effect until `ibsad` is called again or the `ibonl` function is called.

Examples

1. Change the secondary GPIB address of the GPIB interface from its current value to 0x6A.

   ```
   ibsad(0x6A);
   ```

2. Disable secondary addressing for the GPIB interface.

   ```
   ibsad(0);
   ```

See Also

ibpad(3) and *ibcmd(3)*
Chapter 2, *The C Language Library*

IBSIC(3) low-level IBSIC(3)

Name

`ibsic` - send Interface Clear (IFC)

Synopsis

```
#include "ugpib.h"
ibsic ()
```

Description

The `ibsic` function causes the GPIB interface to assert the IFC signal for at least 100 µs. This action initializes the GPIB and makes the interface Controller-In-Charge (CIC). It is generally used to become CIC or to clear a bus fault condition.

The IFC signal is supposed to reset only the GPIB interface functions of bus devices and is not intended to reset internal device functions. Device functions are reset with the Device Clear (DCL) and Selected Device Clear (SDC) commands. To determine the effect of these messages, consult the device documentation.

Example

Initialize the GPIB and become CIC at the beginning of a program.

```
ibsic();
```

See Also

dvclr(3) and *ibcmd(3)*
Chapter 2, *The C Language Library*

IBSRE(3) low-level IBSRE(3)

Name

`ibsre` - set or clear the Remote Enable (REN) line

Synopsis

```
#include "ugpib.h"
ibsre (v)
int v;
```

Description

`v` specifies set or clear.

If `v` is non-zero, the Remote Enable (REN) signal is asserted. If `v` is zero, the signal is deasserted.

The `ibsre` function turns the REN signal on and off. REN is used by devices to select between local and remote modes of operation. REN enables the remote mode. A device does not actually enter remote mode until it receives its listen address.

Examples

1. Place a device at listen address 0x23 (ASCII #) in remote mode with local ability to return to local mode.

```
ibsre(1);            /* set REN to true                  */
ibcmd("#",1);        /* LAD                              */
```

2. Exclude the local ability of the device to return to local mode by sending the Local Lockout command (0x11), or include it in the command string in Example 1.

```
ibcmd("\x11");       /* LLO                              */
```

or

```
ibsre(1);            /* REN true                         */
ibcmd("#\x11");      /* LAD LLO                          */
```

3. Return all devices to local mode.

```
ibsre(0);            /* set REN to false                 */
```

See Also

ibsic(3)
Chapter 2, *The C Language Library*

IBTMO(3) low-level IBTMO(3)

Name

`ibtmo` - change or disable time limit

synopsis

```
#include "ugpib.h"
ibtmo (v)
int v;
```

Description

`v` is a code specifying the time limit. Table 4-6 lists the timeout settings.

Table 4-6. Timeout Settings

Code	Actual Value	Minimum Timeout
TNONE	0	disabled*
T10us	1	10 µs
T30us	2	30 µs
T100us	3	100 µs
T300us	4	300 µs
T1ms	5	1 ms
T3ms	6	3 ms
T10ms	7	10 ms
T30ms	8	30 ms
T100ms	9	100 ms
T300ms	10	300 ms
T1s	11	1 s
T3s	12	3 s
T10s	13	10 s
T30s	14	30 s
T100s	15	100 s
T300s	16	300 s
T1000s	17	1000 s

* If you select TNONE, no limit will be in effect and I/O operations could proceed indefinitely.

`ibtmo` is needed only to alter the value from its default setting of `T10s`.

The time limit is an escape mechanism used to exit gracefully from a "hung bus" condition. Since the GPIB is an asynchronous bus, read and write operations can be held up indefinitely.

Timeout values are approximate, though never less than indicated.

Examples

1. Change the time limit for GPIB I/O operations to approximately 300 ms.

```
ibtmo(T300ms);
```

2. Perform I/O operations with no timeout in effect (not recommended).

```
ibtmo(0);
```

See Also

Chapter 2, *The C Language Library*

IBWAIT(3) low-level IBWAIT(3)

Name

`ibwait` - wait for selected events

Synopsis

```
#include "ugpib.h"
ibwait (mask)
int mask;
```

Description

`mask` is a bit mask with the same bit assignments as the status word, `ibsta`.

A `mask` bit is set to wait for the corresponding event to occur.

The `ibwait` function is used to monitor the events selected in `mask` and to delay processing until any of them occur. These events and bit assignments are shown in Table 4-7.

Table 4-7. Wait Mask Layout

Mnemonic	Bit Position	Hex Value	Description
TIMO	14	4000	Time limit exceeded
SRQI	12	1000	SRQ on
CIC	5	20	GPIB interface is CIC
TACS	3	8	GPIB interface is talker
LACS	2	4	GPIB interface is listener

If `mask`=0, the function returns immediately. This is used to report the current GPIB interface state.

The TIMO bit is automatically included with any non-zero `mask`. If the time limit is set to 0, timeouts are disabled. Disabling timeouts should be done only when it is certain the selected event will occur.

All activity on the GPIB interface is suspended until the event occurs.

Examples

1. Wait for a service request or a timeout.

   ```
   ibwait(SRQI|TIMO);
   ```

2. Report the current status for `ibsta`.

   ```
   ibwait(0);
   ```

3. Wait until control is passed from another Controller-In-Charge (CIC).

   ```
   ibwait(CIC);
   ```

4. Wait until addressed to talk or listen by another CIC.

   ```
   ibwait(TACS|LACS);
   ```

See Also

ibtmo(3)
Chapter 2, *The C Language Library*

IBWRT(3) low-level IBWRT(3)

Name

`ibwrt` - write data to GPIB from a buffer

Synopsis

```
#include "ugpib.h"
ibwrt (buf,cnt)
int cnt;
char buf[];
```

Description

`buf` contains the data to be sent over the GPIB. `cnt` specifies the number of bytes to be sent over the GPIB.

The `ibwrt` function writes `cnt` bytes of data to a GPIB device. The device is assumed to be already properly initialized and addressed.

If the GPIB interface is Controller-In-Charge (CIC), the `ibcmd` function must be called prior to `ibwrt` to address the device to listen and the interface to talk. Otherwise, the device on the GPIB that is the CIC must perform the addressing.

If the GPIB interface is Active Controller, the interface is first placed in Standby Controller state with ATN off and remains there after the write operation has completed. Otherwise, the write operation commences immediately. An EADR error results if the interface is CIC but has not been addressed to talk with the `ibcmd` function. An EABO error results if the interface is not the CIC and is not addressed to talk within the time limit. An EABO error also results if the operation does not complete for whatever reason within the time limit.

The `ibwrt` operation terminates on any of the following events:

- All bytes are transferred.
- Error is detected.
- Time limit is exceeded.

After termination, `ibcnt` contains the number of bytes written. A short count can occur on any of the above events but the first.

Example

Write 10 instruction bytes to a device at listen address 0x35 (ASCII 5) and then unaddress it (the talk address of the GPIB interface is 0x40 or ASCII @).

```
ibcmd("?@5",3);  /* UNL MTA LAD                          */
/* send instruction bytes                                */
ibwrt("F3R1X5P2G0",10);
/* unaddress all listeners and talkers                   */
ibcmd("_?",2);  /* UNT UNL                               */
```

See Also

ibcmd(3) and *dvwrt(3)*
Chapter 2, *The C Language Library*

Appendix A
Multiline Interface Command Messages

The following tables are multiline interface messages (sent and received with ATN TRUE).

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
00	000	0	NUL		20	040	32	SP	MLA0
01	001	1	SOH	GTL	21	041	33	!	MLA1
02	002	2	STX		22	042	34	"	MLA2
03	003	3	ETX		23	043	35	#	MLA3
04	004	4	EOT	SDC	24	044	36	$	MLA4
05	005	5	ENQ	PPC	25	045	37	%	MLA5
06	006	6	ACK		26	046	38	&	MLA6
07	007	7	BEL		27	047	39	'	MLA7
08	010	8	BS	GET	28	050	40	(	MLA8
09	011	9	HT	TCT	29	051	41	)	MLA9
0A	012	10	LF		2A	052	42	*	MLA10
0B	013	11	VT		2B	053	43	+	MLA11
0C	014	12	FF		2C	054	44	,	MLA12
0D	015	13	CR		2D	055	45	-	MLA13
0E	016	14	SO		2E	056	46	.	MLA14
0F	017	15	SI		2F	057	47	/	MLA15
10	020	16	DLE		30	060	48	0	MLA16
11	021	17	DC1	LLO	31	061	49	1	MLA17
12	022	18	DC2		32	062	50	2	MLA18
13	023	19	DC3		33	063	51	3	MLA19
14	024	20	DC4	DCL	34	064	52	4	MLA20
15	025	21	NAK	PPU	35	065	53	5	MLA21
16	026	22	SYN		36	066	54	6	MLA22
17	027	23	ETB		37	067	55	7	MLA23
18	030	24	CAN	SPE	38	070	56	8	MLA24
19	031	25	EM	SPD	39	071	57	9	MLA25
1A	032	26	SUB		3A	072	58	:	MLA26
1B	033	27	ESC		3B	073	59	;	MLA27
1C	034	28	FS		3C	074	60	<	MLA28
1D	035	29	GS		3D	075	61	=	MLA29
1E	036	30	RS		3E	076	62	>	MLA30
1F	037	31	US		3F	077	63	?	UNL

Message Definitions

DCL Device Clear
GET Group Execute Trigger
GTL Go To Local
LLO Local Lockout
MLA My Listen Address
MSA My Secondary Address
MTA My Talk Address
PPC Parallel Poll Configure
PPD Parallel Poll Disable

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
40	100	64	@	MTA0	60	140	96	`	MSA0,PPE
41	101	65	A	MTA1	61	141	97	a	MSA1,PPE
42	102	66	B	MTA2	62	142	98	b	MSA2,PPE
43	103	67	C	MTA3	63	143	99	c	MSA3,PPE
44	104	68	D	MTA4	64	144	100	d	MSA4,PPE
45	105	69	E	MTA5	65	145	101	e	MSA5,PPE
46	106	70	F	MTA6	66	146	102	f	MSA6,PPE
47	107	71	G	MTA7	67	147	103	g	MSA7,PPE
48	110	72	H	MTA8	68	150	104	h	MSA8,PPE
49	111	73	I	MTA9	69	151	105	i	MSA9,PPE
4A	112	74	J	MTA10	6A	152	106	j	MSA10,PPE
4B	113	75	K	MTA11	6B	153	107	k	MSA11,PPE
4C	114	76	L	MTA12	6C	154	108	l	MSA12,PPE
4D	115	77	M	MTA13	6D	155	109	m	MSA13,PPE
4E	116	78	N	MTA14	6E	156	110	n	MSA14,PPE
4F	117	79	O	MTA15	6F	157	111	o	MSA15,PPE
50	120	80	P	MTA16	70	160	112	p	MSA16,PPD
51	121	81	Q	MTA17	71	161	113	q	MSA17,PPD
52	122	82	R	MTA18	72	162	114	r	MSA18,PPD
53	123	83	S	MTA19	73	163	115	s	MSA19,PPD
54	124	84	T	MTA20	74	164	116	t	MSA20,PPD
55	125	85	U	MTA21	75	165	117	u	MSA21,PPD
56	126	86	V	MTA22	76	166	118	v	MSA22,PPD
57	127	87	W	MTA23	77	167	119	w	MSA23,PPD
58	130	88	X	MTA24	78	170	120	x	MSA24,PPD
59	131	89	Y	MTA25	79	171	121	y	MSA25,PPD
5A	132	90	Z	MTA26	7A	172	122	z	MSA26,PPD
5B	133	91	[	MTA27	7B	173	123	{	MSA27,PPD
5C	134	92	\	MTA28	7C	174	124	|	MSA28,PPD
5D	135	93	]	MTA29	7D	175	125	}	MSA29,PPD
5E	136	94	^	MTA30	7E	176	126	~	MSA30,PPD
5F	137	95	_	UNT	7F	177	127	DEL	

PPE	Parallel Poll Enable	SPE	Serial Poll Enable
PPU	Parallel Poll Unconfigure	TCT	Take Control
SDC	Selected Device Clear	UNL	Unlisten
SPD	Serial Poll Disable	UNT	Untalk

Appendix B
Software Configuration and Installation

This appendix contains instructions for configuring and installing the GPIB software.

The ESP-488 VxWorks software is distributed as part of the VxWorks distribution for the VXIcpu-030. Please refer to *Getting Started with Your VXIcpu-030 and the NI-VXI Software for the VxWorks Operating System* for instructions on how to install the distribution media. The ESP-488 files are contained in the GPIB directory. Table B-1 describes the files included in the distribution media.

Table B-1. Software Distribution Files

File Name	Description
`Readme`	Up-to-date information not included in this manual
`esp488.o`	ESP-488 driver library module
`ibic.o`	Interface Bus Interactive Control program
`ibsta.o`	Software Installation Test, Part A
`ibstb.o`	Software Installation Test, Part B
`ugpib.h`	User include file for ESP-488 applications

A separate ESP-488 package is also available to control GPIB-1014/1014D VME boards. The kit includes the GPIB-1014/1014D board, VxWorks ESP-488 software, and a GPIB reference manual.

Please refer to *ESP-488 for VxWorks and the GPIB-1014/1014D Software Reference Manual* (part number 320429-01) for more information.

Appendix C
GPIB Programming Example

This appendix illustrates the steps involved in programming a representative IEEE-488 instrument from a terminal using the ESP-488 functions in C language. This appendix is designed to help you learn how to use the ESP-488 driver software to execute certain programming and control sequences.

The target instrument is a digital voltmeter (DVM). This instrument is otherwise unspecified. The purpose here is to explain how to use the driver software to execute certain programming and control sequences, not how to determine those sequences.

Because the instructions that are sent to program a device as well as the data that might be returned from the device are called *device-dependent messages*, the format and syntax of the messages used in this example are unique to this device. Furthermore, the *interface messages* or *bus commands* that must be sent to devices will also vary, but to a lesser degree. The exact sequence of messages to program and to control a particular device are contained in its documentation.

For example, the following sequence of actions is assumed to be necessary to program this DVM to make and return measurements of a high-frequency AC voltage signal in the autoranging mode:

1. Initialize the GPIB interface circuits of the DVM so that it can respond to messages.
2. Place the DVM in remote programming mode and turn off the front panel control.
3. Initialize the internal measurement circuits.
4. Program the DVM to perform the proper function (F3 for high-frequency AC volts), range (R7 for autoranging), and trigger source (T3 for external or remote).
5. For each measurement:
    a. Send the Group Execute Trigger (GET) command to trigger the DVM.
    b. Wait until the DVM asserts Service Request (SRQ) to indicate that the measurement is ready to be read.
    c. Serial poll the DVM to determine if the measured data is valid (status byte = 0xC0) or if a fault condition exists (the 0x40 bit and another bit of the status byte, other than 0x80, are set).
    d. If the data is valid, read 16 bytes from the DVM.
6. End the session.

    The example program given here also assumes that the GPIB interface is the designated System Active Controller of the GPIB and that the DVM is the only instrument connected to the bus.

Example Program

```
#include "ugpib.h"

char  cmd[512];     /*  command buffer          */
char  rd[512];      /*  read buffer             */
char  wrt[512];     /*  write buffer            */

unsigned int mask;  /*  events to be waited for */

main()    {
      int dvm;

      /* Bring GPIB interface online and initialize the bus.   */
      ibonl (1);
      ibsic ();

      /* Set the DVM for primary address 3, no secondary
         address.                                              */
      dvm = 3;

      /* Place the device in Remote state with Local Lockout
        (RWLS).                                               */
      if (ibsre(1) & ERR) err();
      if (ibcmd("#\021",2) & ERR) err();   /* LAD3 LLO         */

      /* Send the Selected Device Clear (SDC) message to clear
         internal device functions.                           */
      if (dvclr(dvm) & ERR) err();

      /* Write the function, range, and trigger source
         instructions to the DVM.                             */
      if (dvwrt(dvm,"F3R7T3",6) & ERR) err();

      /* Send the Group Execute Trigger (GET) message to
         trigger a measurement reading.                       */
      if (dvtrg(dvm) & ERR) err();

      /* Wait for the DVM to set SRQ or for a timeout.        */
      if (ibwait(TIMO|SRQI) & (ERR|TIMO)) err();

      /* Read serial poll response; if not equal to 0xC0,
         report dvm error.                                    */
      if (dvrsp(dvm,rd) & ERR) err();
      if ( (rd[0] & 0xFF) != 0xC0)  dvmerr();

      /* Read the measurement.                                */
      if (dvrd(dvm,rd,16) & ERR) err();

      /* Take the GPIB interface offline.                     */
      ibonl(0);

}
```

```
err(){
/* An error checking routine at this location would, among other
   things, check iberr to determine the exact cause of the error
   condition and then take action appropriate to the application.
   For errors during data transfers, ibcnt can be examined to
   determine the actual number of bytes transferred.          */
  }

dvmerr() {
/* A routine at this location would analyze the fault code
   returned in the DVM's status byte and take appropriate
   action.                                                    */
  }
```

Appendix D
Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve technical problems you might have as well as a form you can use to comment on the product documentation. Filling out a copy of the *Technical Support Form* before contacting National Instruments helps us help you better and faster.

National Instruments provides comprehensive technical assistance around the world. In the U.S. and Canada, applications engineers are available Monday through Friday from 8:00 a.m. to 6:00 p.m. (central time). In other countries, contact the nearest branch office. You may fax questions to us at any time.

Corporate Headquarters
(512) 795-8248
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices	**Phone Number**	**Fax Number**
Australia	(03) 879 9422	(03) 879 9179
Austria	(0662) 435986	(0662) 437010-19
Belgium	02/757.00.20	02/757.03.11
Denmark	45 76 26 00	45 76 71 11
Finland	(90) 527 2321	(90) 502 2930
France	(1) 48 14 24 00	(1) 48 14 24 14
Germany	089/741 31 30	089/714 60 35
Italy	02/48301892	02/48301915
Japan	(03) 3788-1921	(03) 3788-1923
Netherlands	03480-33466	03480-30673
Norway	32-848400	32-848600
Spain	(91) 640 0085	(91) 640 0533
Sweden	08-730 49 70	08-730 43 70
Switzerland	056/20 51 51	056/20 51 55
U.K.	0635 523545	0635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___)_____ Phone (___)_____

Computer brand _____ Model _____ Processor _____

Operating system _____

Speed _____MHz RAM _____MB Display adapter _____

Mouse _____yes _____no Other adapters installed _____

Hard disk capacity _____MB Brand _____

Instruments used _____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is _____

_____

_____

_____

_____

List any error messages _____

_____

_____

_____

The following steps will reproduce the problem _____

_____

_____

_____

ESP-488 Hardware and Software Configuration Form

Please fill out the *VXIcpu-030 Hardware and Software Configuration Form* found in the *Customer Communication* section of *Getting Started with Your VXIcpu-030 and the NI-VXI Software for the VxWorks Operating System*. In addition, if you are using a GPIB-1014/1014D board in your system, please also fill out the configuration form found in the *ESP-488 for VxWorks and the GPIB-1014/1014D Software Reference Manual*.

- Did you recompile the ESP-488 driver source files? _____
- If yes, did you make any changes to the driver source files? Please explain: _____

_____

_____

_____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: **ESP-488 Software Reference Manual for the VXIcpu-030®**

Edition Date: **July 1994**

Part Number: **320345-01**

Please comment on the completeness, clarity, and organization of the manual.

If you find errors in the manual, please record the page numbers and describe the errors.

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (_____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway, MS 53-02
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
MS 53-02
(512) 794-5678

Glossary

Prefix	Meaning	Value
μ-	micro-	10^{-6}
m-	milli-	10^{-3}
M-	mega-	10^{6}

AC	alternating current
ANSI	American National Standards Institute
ATN	a GPIB line that distinguishes between commands and data messages.
CIC	Controller-in-Charge
CMPL	I/O operation completed
DCL	Device Clear; a command used to reset the device or internal functions of all devices.
DVM	digital voltmeter
END	End of a data string
EOI	End Or Identify; a line used to signal either the last byte of a data message (END) or the parallel poll Identify (IDY) message.
EOS	End of String character sent as the last byte of a data message.
ERR	GPIB Error bit
GET	Group Execute Trigger; a command used to trigger a device or internal function of an address Listener.
GPIB	General Purpose Interface Bus; the industry standard IEEE 488 bus.
GTL	Go To Local; a command used to place an addressed Listener in local (front panel) control mode.
hex	hexadecimal; the numbering system with base 16, using the digits 0 to 9 and letters A to F.
Hz	hertz (cycles per second)

ibic	Interface Bus Interactive Control program; used to communicate with GPIB devices, troubleshoot problems, and develop your application.
IDY	Identify
IEEE	Institute for Electrical and Electronic Engineers
IFC	Interface Clear; a GPIB line used by the System Controller to initialize the bus.
I/O	input/output
LACS	Addressed to listen.
Listener	A GPIB device that receives data messages from a Talker.
LLO	Local Lockout; a command used to tell all devices that they may or should ignore remote (GPIB) data messages or local (front panel) controls, depending on whether the device is in local or remote program mode.
M	megabytes of memory
MLA	My Listen Address; a command used to address a device to be a Listener.
MSA	My Secondary Address; a command used to address a device to be a Listener or a Talker when extended (two byte) addressing is used.
MTA	My Talk Address; a command used to address a device to be a Talker.
octal	numbering system with base 8, using numerals 0 to 7.
PAD	primary address
PPC	Parallel Poll Configure; a command used to configure an addressed Listener to participate in polls.
PPD	Parallel Poll Disable; a command used to disable a configured device from participating in polls.
PPE	Parallel Poll Enable; a command used to enable a configured device to participate in polls and to assign a response line.
PPU	Parallel Poll Unconfigure; a command used to disable any device from participating in polls.
REN	Remote Enable; a GPIB line controlled by the System Controller but used by the CIC to place devices in remote program mode.
SAD	secondary address
SDC	Selected Device Clear; a command used to reset internal or device functions of an addressed Listener.

s	seconds
SPD	Serial Poll Disable; a command used to cancel an SPE command.
SPE	Serial Poll Enable; a command used to enable a specific device to be polled. That device must also be addressed to talk.
SRQ	Service Request; a GPIB line that a device asserts to notify the CIC that the device needs servicing.
SRQI	SRQ line is asserted.
System Controller	The single designated controller that can assert control (become CIC of the GPIB) by sending the IFC message. Other devices can become CIC only by having control passed to them.
TACS	Addressed to talk.
Talker	A GPIB device that sends data messages to Listeners.
TCT	Take Control; a command used to pass control of the bus from the current controller to an addressed Talker.
TIMO	Time limit for I/O completion has been exceeded.
Timeout	A software feature that prevents I/O functions from hanging indefinitely when there is a problem on the GPIB.
UNL	Unlisten; a command used to unaddress any active Listeners.
UNT	Untalk; a command used to unaddress an active Talker.

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP