ESP-488 Software Reference Manual for LynxOS and the AT-GPIB

National Instruments IEEE 488 Engineering Software Package for the LynxOS Operating System

August 1993 Edition

Part Number 320642-01

**© Copyright 1993, 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 AT-GPIB is warranted against defects in materials and workmanship for a period of two years from the date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

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 NI-488.2™ are trademarks of National Instruments Corporation.

Product 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

Figures

Tables

About This Manual

This manual describes the IEEE 488 Engineering Software Package (ESP-488) for the LynxOS operating system (Version 2.1 and higher) from Lynx Real-Time Systems, Inc. This package is intended for use with the National Instruments AT-GPIB interface board. To use this manual effectively, you must already have an AT-GPIB board and LynxOS for your PC AT computer.

Organization of This Manual

This manual is organized as follows:

- Chapter 1, *Introduction*, contains an overview of the ESP-488 for LynxOS software, lists the contents of the ESP-488 for LynxOS kit, and describes important considerations when using the software.

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

- Chapter 3, *ibic*, introduces you to `ibic`, the interactive control program that you can use to communicate with GPIB devices through functions you enter at your keyboard. This chapter also contains instructions for running `ibic`, contains guidelines for translating `ibic` syntax, contains a sample session, and summarizes the auxiliary functions that are compatible with the `ibic` utility.

- 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 LynxOS operating systems.

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

- Appendix B, *AT-GPIB Configuration and Installation*, describes how to configure and install the AT-GPIB interface board and the ESP-488 software package.

- 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.

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

Conventions Used in This Manual

The following conventions are used in this manual:

italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
`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.
ESP-488	ESP-488 is used throughout this manual to refer to ESP-488 for LynxOS.
IEEE 488 and IEEE 488.2	IEEE 488 and IEEE 488.2 are used throughout this manual to refer to the ANSI/IEEE Standard 488.1-1987 and the ANSI/IEEE Standard 488.2-1987, respectively, which define the GPIB.

Abbreviations, acronymns, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

Related Documentation

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

- *AT-GPIB Technical Reference Manual* (part number 320432-01)
- *Getting Started with Your AT-GPIB and the NI-488.2 Software for MS-DOS* (part number 320284-01)
- *NI-488.2 Software Reference Manual for MS-DOS* (part number 320282-01)
- ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*
- ANSI/IEEE Standard 488.2-1987, *IEEE Standard Codes, Formats, Protocols, and Common Commands*
- *LynxOS User's Manual, Volumes 1 and 2*, Lynx Real-Time Systems, Inc.

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop using 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 located in Appendix D, *Customer Communication*, at the end of this manual.

Chapter 1
Introduction

This chapter contains an overview of the ESP-488 for LynxOS software, lists the contents of the ESP-488 for LynxOS kit, and describes important considerations when using the software.

ESP-488 is a functional subset of the industry-standard NI-488.2 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 includes various functions for interface configuration and high-level device communication. Other features include timeout configuration, error reporting, and an interactive control utility (`ibic`).

What Your Kit Should Contain

Your kit should contain the following components:

Component	Part Number
ESP-488 for LynxOS software 3.5 in. distribution diskette in UNIX `tar` format	422924-55
ESP-488 Software Reference Manual for LynxOS and the AT-GPIB	320642-01

Important Considerations

Before using the ESP-488 for LynxOS software, you must install an AT-GPIB interface board and load the software from the ESP-488 for LynxOS distribution diskette. Refer to Appendix B, *AT-GPIB Configuration and Installation*, for instructions on installing the hardware and software. In addition, you must already have obtained and installed the LynxOS operating system from Lynx Real-Time Systems, Inc. before you can install the ESP-488 software package.

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

- The ESP-488 functions give you the capability to make synchronous I/O transfers through a single AT-GPIB board. 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 AT-GPIB board is normally designated to be the System Controller. Most ESP-488 functions are optimized to assume the AT-GPIB is also the Controller-In-Charge (CIC).
- You must call the `ibonl` function to initialize the AT-GPIB before any other call is made.
- Prior to calling `ibrd` or `ibwrt`, you must address the appropriate devices, including the AT-GPIB, 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.

Chapter 2
The C Language Library

This chapter contains a general description of the C language programming interface to the ESP-488 for LynxOS 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 AT-GPIB board. The status word is also available as the external variable `ibsta`.

The status word contains 32 bits, nine of which are meaningful. A bit value of 1 indicates that the corresponding condition is in effect, while a bit value of 0 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

The following paragraphs describe each status bit and its condition.

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
EDVR	0	LynxOS error (code in `ibcnt`)
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

The following paragraphs describe each error and some conditions under which it may occur.

EDVR (0) This code is returned by the language interface when an error is returned from the operating system. When this error occurs, the LynxOS error code is placed in the count variable `ibcnt` (it is also available in the external variable `errno`). You can use the LynxOS library function `perror` to print a description of the error code. Refer to the *Library Functions* section of the *LynxOS User's Manual, Volume 1*, for instructions on using this function.

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 canceled. 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 ANSI/IEEE Standard 488.1-1987 defines two methods of identifying the last byte of device-dependent (data) messages. A Talker can use either method 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.

You can use these two methods 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

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 the LynxOS development system and link it to the appropriate C language library. For example, to compile and link a program named `prog.c` that calls functions in the default ESP library, enter the following command:

```
cc prog.c cib488.o -o prog
```

To run the resulting executable file, `prog`, enter its name at the LynxOS prompt (the ESP library must already be installed). You can also generate other C language interface files to access functions in additional ESP libraries. Refer to Appendix B, *AT-GPIB Configuration and Installation*, for information on installing and using one or more ESP driver libraries.

For more information on creating and running LynxOS applications in general, refer to the description of the C compiler, `cc`, in the *Utility Programs* section of the *LynxOS User's Manual, Volume 1*, and to the *Software Development* section of the *LynxOS User's Manual, Volume 2*.

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*, for more thorough information and specific examples. Refer to Appendix B, *AT-GPIB Configuration and Installation*, for information on alternative naming conventions used in various ESP-488 driver libraries.

Device-Level Functions

The following device-level functions are performed on a GPIB device at the specified address. All Controller sequences conform to the IEEE 488.2 standard.

`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 low-level functions are 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()` Sets the interface to standby state (unasserts 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 a service request (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 `ibic`, the interactive control program that you can use to communicate with GPIB devices through functions you enter at your keyboard. This chapter also contains instructions for running `ibic`, contains guidelines for translating `ibic` syntax, contains a sample session, and summarizes the auxiliary functions that are compatible with the `ibic` utility.

Overview

With the Interface Bus Interactive Control (`ibic`) utility, you communicate with GPIB devices through functions you enter at the keyboard. For specific communication instructions, refer to the manual that came with your instrument. Then you can use `ibic` to practice communication with the instrument, troubleshoot problems, and develop your application program.

One way `ibic` helps you to learn about your instrument and to troubleshoot problems is by displaying the following information on your computer screen whenever you enter a command:

- The results of the status word (`ibsta`) in hexadecimal
- The mnemonic constant of each bit position set
- The mnemonic value of the error variable (`iberr`) if an error exists
- The count variable for each read, write, or command function
- The data received from your instrument

The following sections contain instructions for running `ibic`, guidelines for translating `ibic` syntax, a sample session, and a summary of the auxiliary functions that are compatible with the `ibic` utility. Refer to Chapter 4, *ESP-488 Functions and Utilities Reference,* for detailed descriptions of the C language functions.

Running ibic

Go to the directory containing the ESP-488 distribution files and run `ibic` by entering the following command at the LynxOS prompt:

```
ibic
```

Initially, `ibic` directs all calls to the default ESP driver module or the first module found 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)`

Example

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 are compatible with the `ibic` utility.

Table 3-1. Auxiliary Functions Compatible with ibic

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 LynxOS operating systems. The functions and utilities are arranged alphabetically within the following groups:

- GPIB
- Device-Level Functions
- Low-Level Functions

IBIC(1) GPIB IBIC(1)

Name

`ibic` - interface bus interactive control program

Synopsis

```
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 `cib488/esp488` are compatible with `ibic`.

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

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

(continues)

Table 4-1. Syntax of ESP-488 Functions in ibic (Continued)

Description	Function Syntax	Function Type
Read data	`ibrd v` (5)	low-level
Conduct a parallel poll	`ibrpp`	low-level
Request service	`ibrsv v` (3)	low-level
Change secondary address	`ibsad v` (3)	low-level
Send interface clear	`ibsic`	low-level
Set/clear remote enable line	`ibsre [v]` (2, 3)	low-level
Change/disable time limit	`ibtmo v` (3)	low-level
Wait for selected event	`ibwait [mask]` (2, 6)	low-level
Write data	`ibwrt string` (4)	low-level

Notes for Table 4-1

(1) `a` is the hex, octal, or decimal integer (see note 3) that designates 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 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.

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.

Table 4-3. GPIB Error Codes

Suggested Mnemonic	Decimal Value	Explanation
EDVR	0	LynxOS error (code in `ibcnt`)
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 are compatible with the `ibic` utility.

Table 4-4. Auxiliary Functions Compatible with ibic

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

```
ibtsta [x]
ibtstb [x]
```

Description

`ibtsta` and `ibtstb` verify the correct installation and operation of an ESP-488 library. You can use the optional argument `x` to run the test on a specific 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 user interaction. `ibtstb` performs a more thorough check of I/O and interrupt operation and requires the use of a GPIB analyzer. Both tests give onscreen instructions at program startup for you to set up and run the test.

You should run `ibtsta` first. If `ibtsta` completes with no errors and a GPIB analyzer is available, you should then run `ibtstb`. You can omit `ibtstb` 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 addresses the GPIB interface and device appropriately, and then sends the message SDC, the meaning of which depends on the specific device. SDC usually resets all device functions.

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` is the number of bytes to read from the GPIB.

The `dvrd` function addresses the GPIB interface to listen, the device at address `a` to talk, and then reads `cnt` bytes of data from the device.

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.

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. 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 addresses the GPIB interface to talk, the device at address `a` to listen, and then writes `cnt` bytes of data to the device.

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 ten 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 is not 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.

Use asynchronous take control 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 that require that the GPIB interface take control, such as `ibcmd` and `ibrpp`, 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 sent over the GPIB. `cnt` is the number of bytes to be sent over the GPIB.

You use the `ibcmd` function to transmit GPIB interface messages (commands). 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. You also use the `ibcmd` function to pass GPIB control to another device. You do *not* use this function 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. Refer to Table 4-5. `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 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 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         */
/* 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 unassert the ATN signal if it is the Active Controller. `ibgts` permits GPIB devices to transfer data without the GPIB interface participating in 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 board can monitor the line (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 unasserted.

Example

Test for Remote Enable (REN).

```
if (iblines(&clines) < 0) error();
if (!(clines & 0x10)) {
   printf("GPIB interface cannot 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` indicates 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 is 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` indicates 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` is 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 (CIC). If the GPIB interface is Standby Controller, it takes control and asserts ATN (becomes Active) prior to polling and 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` indicates 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.

An error does not result if the `ibrsv` function is called 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 you call `ibsad` again or you call the `ibonl` function.

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 should 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` indicates set or clear.

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

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 `ibcmd` 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 that indicates the time limit. Table 4-6 lists the timeout settings.

Table 4-6. Timeout Settings

Actual Code	Minimum Value	Timeout
`TNONE`	0	disabled (1)
`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

Note for Table 4-6

(1) If you select `TNONE`, no limit is 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 from a *hung bus* condition. Because 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 is asserted
CIC	5	20	Controller-In-Charge
TACS	3	8	Talker
LACS	2	4	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` is 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 ten 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
AT-GPIB Configuration and Installation

This appendix describes how to configure and install the AT-GPIB interface board and the ESP-488 software package. If your board is already installed, make a note of the current hardware configuration settings and proceed to the *Software Installation and Configuration* section later in this appendix. If you encounter any problems with the current hardware settings, you should review the following section, *AT-GPIB Hardware Configuration*.

AT-GPIB Hardware Configuration

Table B-1 lists the factory default hardware configuration settings, which are the recommended settings for correct operation of the AT-GPIB board in a LynxOS-based computer. The following sections explain each of these options in more detail and describe configuration.

Table B-1. AT-GPIB Hardware Configuration Settings

Configuration Option	Recommended Setting	Hardware Jumper Setting
Base I/O Address	2C0 hex (1)	U25 (4) = 10110
Interrupt Request Line	11 (2)	IRQ11
DMA Channel Number	5 (3)	DRQ5, DACK5
Shield Ground	Logic Ground Connected to Shield Ground	W1 Connected

Notes for Table B-1

(1) Optional base I/O address settings range from 100 hex to 3E0 hex, in multiples of 20 hex.

(2) Optional interrupt request lines range from 3 to 7, 9 to 12, 14, 15, or not used. Notice that on most ISA computers, IRQ6 is normally used by the floppy disk drive controller and IRQ14 is normally used by the hard disk drive controller. Other interrupt request lines may also be used by various devices, depending on the system configuration. In particular, the Adaptec SCSI controller (if present) normally uses IRQ11. Select an IRQ line that does not conflict with any other installed devices.

(3) Optional DMA channels are 5, 6, 7, or not used. Notice that the Adaptec SCSI controller (if present) normally uses DMA Channel 5. Select a DMA channel that does not conflict with any other installed devices.

(4) This position number may vary depending on the revision of your board.

If you select values other than the recommended settings shown for any of the options in Table B-1, be sure to make the corresponding changes in the software as described in the *Software Installation and Configuration* section, later in this appendix. For additional information concerning base I/O address, interrupt request line, and DMA channel selection on your particular computer, refer to your computer documentation.

Figure B-1 shows the locations of the switches and jumpers on the AT-GPIB.

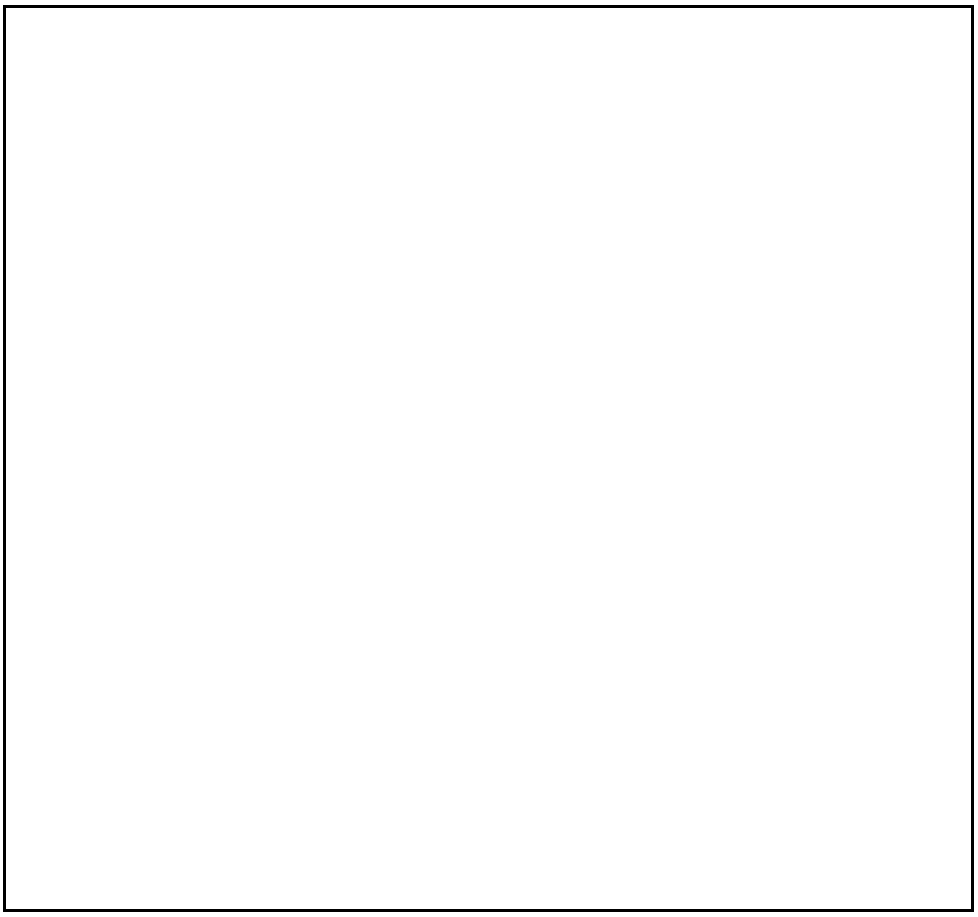


Figure B-1. AT-GPIB Parts Locator Diagram

Base I/O Address Selection

PC AT computers have a segment of memory reserved for input and output. This segment is referred to as the I/O address space. The base I/O address of a PC AT plug-in board such as the AT-GPIB is the position that it occupies in the I/O address space. The base I/O address for the AT-GPIB is determined by the switches at position U25. The switches are set at the factory for the base I/O address 2C0 hex. The AT-GPIB uses the base I/O address space 2C0 through 2DF hex with this setting.

Note: Check to determine that this space is not already used by other equipment installed in your computer. If any device in your computer uses this base I/O address space, change the base I/O address of the AT-GPIB or the other device. Remember that you must make a corresponding change to the software setting as described later in this appendix.

All possible base I/O addresses are multiples of 20 hex between 100 hex and 3E0 hex. Table B-2 lists the possible switch settings you can use for the AT-GPIB, the corresponding base I/O address, and the I/O address space used for that setting.

Table B-2. Possible Base I/O Address Switch Settings for the AT-GPIB

Switch Setting					Base I/O Address (hex)	I/O Address Space Used (hex)
A9	**A8**	**A7**	**A6**	**A5**		
0	1	0	0	0	100	100 to 11F
0	1	0	0	1	120	120 to 13F
0	1	0	1	0	140	140 to 15F
0	1	0	1	1	160	160 to 17F
0	1	1	0	0	180	180 to 19F
0	1	1	0	1	1A0	1A0 to 1BF
0	1	1	1	0	1C0	1C0 to 1DF
0	1	1	1	1	1E0	1E0 to 1FF
1	0	0	0	0	200	200 to 21F
1	0	0	0	1	220	220 to 23F
1	0	0	1	0	240	240 to 25F
1	0	0	1	1	260	260 to 27F
1	0	1	0	0	280	280 to 29F
1	0	1	0	1	2A0	2A0 to 2BF
1	0	1	1	0	2C0	2C0 to 2DF
1	0	1	1	1	2E0	2E0 to 2FF
1	1	0	0	0	300	300 to 31F
1	1	0	0	1	320	320 to 33F
1	1	0	1	0	340	340 to 35F
1	1	0	1	1	360	360 to 37F
1	1	1	0	0	380	380 to 39F
1	1	1	0	1	3A0	3A0 to 3BF
1	1	1	1	0	3C0	3C0 to 3DF
1	1	1	1	1	3E0	3E0 to 3FF

To change the base I/O port address, press each switch to the desired position, checking each switch to make sure it is pressed down all the way. Each switch in U25 corresponds to one of the address lines A9 through A5. The top switch (1) corresponds to address line A9, the next switch from the top (2) corresponds to address line A8, and so on. The five least significant bits of the address (A4 through A0) are decoded by the AT-GPIB to select the appropriate GPIB interface register and cannot be changed.

Press the side marked OFF to select a binary value of 1 for the corresponding address bit. Press down on the ON side of the switch to select a binary value of 0 for the corresponding address bit. Figure B-2 shows two possible switch settings. Each of the address selections shows how the base I/O address was calculated from the switch positions.

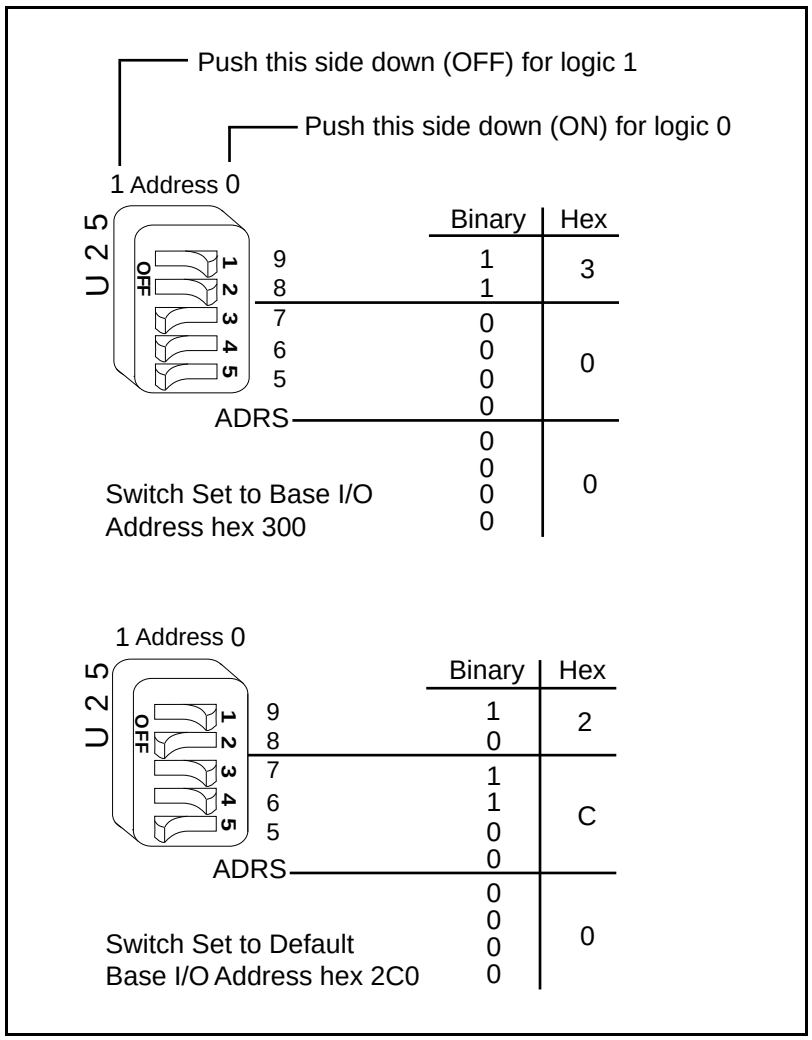


Figure B-2. AT-GPIB Base I/O Address Options

If you change this hardware setting from the default, make a note of the new base address for the AT-GPIB for reference so that you can make the corresponding change when configuring the software.

Interrupt Selection

PC AT computers have a series of interrupt lines available to devices. Devices use interrupts to get immediate service from the CPU for asynchronous events. The AT-GPIB and the ESP-488 software use interrupts to get service from the CPU when necessary. As a result, the CPU can service other processes in the LynxOS environment while the ESP-488 driver is waiting for an asynchronous event to occur.

You select the interrupt line by the position of a jumper on one of two jumper locations located above the I/O slot edge connector on the AT-GPIB board. Figure B-1 shows the location of the two jumper sets you can use to make your interrupt request line selection.

The AT-GPIB board uses interrupt line 11 by default. The AT-GPIB board can use one of 11 interrupt lines on the PC AT I/O channel. The AT-GPIB board can use interrupt lines IRQ3, 4, 5, 6, 7, 9, 10, 11, 12, 14, and 15.

Note: Do not use interrupt line 6 or interrupt line 14. Interrupt line 6 is used by the diskette drive controller and interrupt line 14 is used by the hard disk drive controller on most PC ATs.

If you need to change the interrupt line, place the jumper on the appropriate pins to enable a different interrupt line. For lines 10, 11, 12, 14, or 15, use the jumper set shown in Figure B-3. As mentioned previously, you should *not* select IRQ14. Figure B-3 shows the default setting, IRQ11.

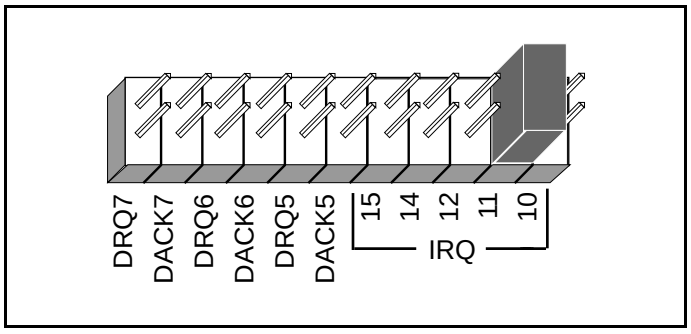


Figure B-3. Interrupt Jumper Setting for IRQ11 (Default Setting)

For interrupt lines 3 through 7, or 9, use the jumper set shown in Figure B-4. As mentioned previously, you should *not* select IRQ6. Figure B-4 shows the selection of IRQ5.

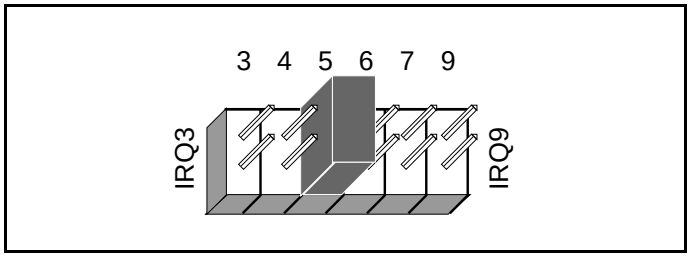


Figure B-4. Interrupt Jumper Setting for IRQ5

If you do not want to use interrupts, you need to logically disable interrupt levels on the AT-GPIB board. In this case, follow the instructions in the *Configuring the Driver* section, later in this appendix. You do not need to move any jumpers. However, you may want to move or remove the jumper just to remind yourself that you are disabling interrupts. You can store the jumper on the board in the position shown in Figure B-5.

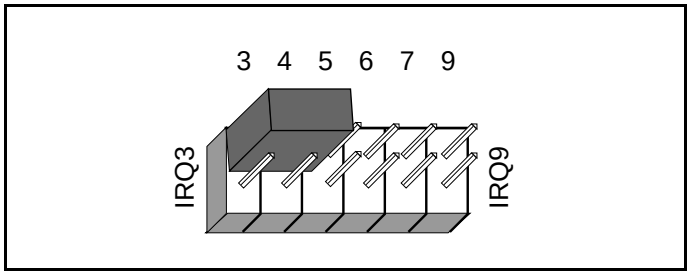


Figure B-5. Interrupt Jumper Setting for Physically Disabling Interrupts

If you change this hardware setting from the default, make a note of the new interrupt request line for the AT-GPIB for reference so that you can make the corresponding change when configuring the software.

DMA Channel Selection

DMA (direct memory access) refers to data transfers directly to or directly from devices such as the AT-GPIB board and the computer memory. Your AT-GPIB hardware and the ESP-488 software are designed to perform DMA. Data transfers using DMA are significantly faster than programmed I/O data transfers that require the use of the CPU.

Figure B-1 shows the location of the DRQ/DACK jumpers used to select a DMA channel for the AT-GPIB. The AT-GPIB is set at the factory to use DMA channel 5. Verify that this DMA channel is not used by equipment already installed in your computer. If any device uses DMA channel 5, change the DMA channel of either the AT-GPIB or the other device.

Note: DMA channels 5, 6, and 7 are the three 16-bit channels on the AT bus. The AT-GPIB does *not* use and *cannot* be configured to use the 8-bit DMA channels on the AT bus.

Each DMA channel consists of two signal lines as shown in Table B-3.

Table B-3. DMA Channels for the AT-GPIB

DMA Channel	DMA Acknowledge	DMA Request
5	DACK5	DRQ5
6	DACK6	DRQ6
7	DACK7	DRQ7

You must position two jumpers to select a DMA channel. The DMA Acknowledge and DMA Request lines selected must have the same numeric suffix for proper operation. Figure B-6 displays the jumper position for selecting DMA channel 7.

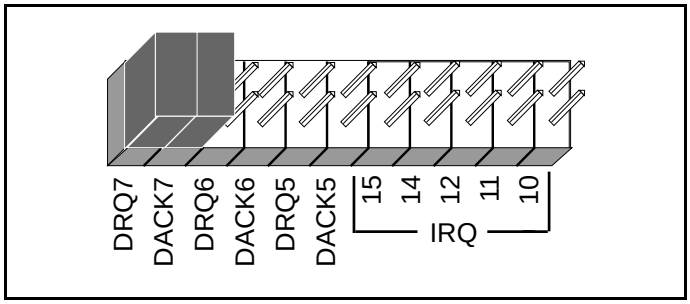


Figure B-6. DMA Channel Jumper Setting for DMA Channel 7

Using Programmed I/O for GPIB Transfers

As an alternative, you can use programmed I/O GPIB transfers if you do not want to use DMA for GPIB transfers. In this case, you need to logically disable DMA on the AT-GPIB board. Do this by following the instructions in the *Configuring the Driver* section, later in this appendix. You do not need to move any jumpers. However, you may want to move or remove the jumpers just to remind yourself that you are disabling DMA. You can store the jumpers on the board in the positions shown in Figure B-7.

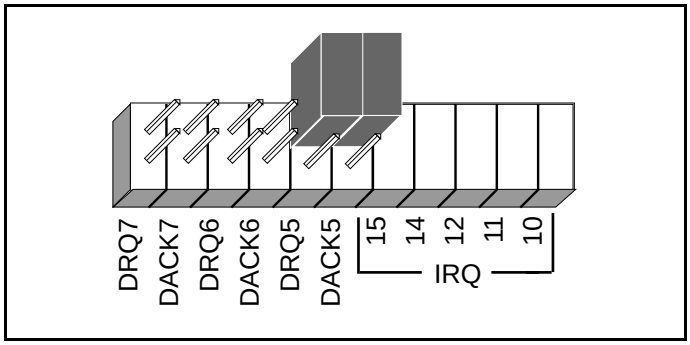


Figure B-7. DMA Jumper Setting for No DMA Channel

If you change this hardware setting from the default, make a note of the new DMA channel for the AT-GPIB for reference so that you can make the corresponding change when configuring the software.

Shield Ground Selection

Connecting the AT-GPIB logic ground to its shield ground minimizes EMI emissions. The AT-GPIB board is set at the factory with the jumper in place to connect the logic ground of the AT-GPIB board to its shield ground.

If your application requires that logic ground be disconnected from shield ground, remove the jumper (W1) and place it across only one of the jumper pins. Jumper settings for logic ground connected and disconnected from shield ground are shown in Figure B-8.

Caution: The AT-GPIB board was tested for compliance with FCC standards with the shield ground connected to logic ground. Removing the jumper may cause EMI emissions to exceed any or all of the applicable standards.

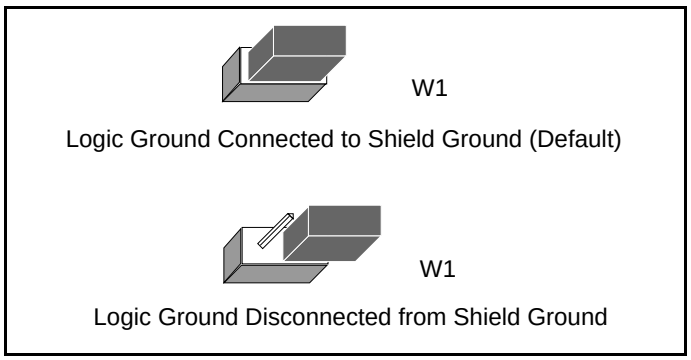


Figure B-8. Ground Configuration Jumper Settings

AT-GPIB Hardware Installation

Before installing the AT-GPIB board, verify that none of the selected configuration settings conflict with those of any other device already installed in your computer.

You can install the AT-GPIB board in any 16-bit ISA expansion slot in your computer. The AT-GPIB board does *not* work if installed in an 8-bit expansion slot (PC-style). Consult the user manual or technical reference manual of your personal computer for specific instructions and warnings.

Perform the following steps to install the AT-GPIB board:

1. Shut down your computer and turn off the power switch. Keep the computer plugged in so that it remains grounded while you install the AT-GPIB board.

2. Remove the top cover or access port of the I/O channel.

3. Remove the expansion slot cover on the back panel of the computer.

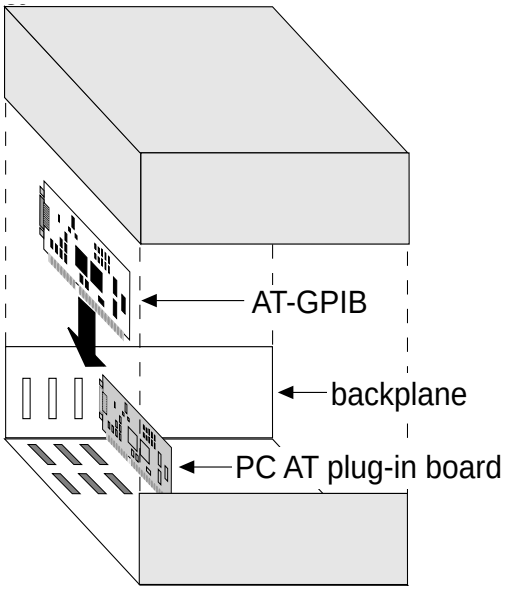
4. Insert the AT-GPIB board in an unused 16-bit slot with the IEEE 488 receptacle sticking out of the opening on the back panel. It may be a tight fit, but do not force the board into place.

5. Screw the mounting bracket of the AT-GPIB board to the back panel rail of the computer.

6. Check the installation.

7. Replace the top cover or access port to the I/O channel.

8. Turn on your computer.



Software Installation and Configuration

The ESP-488 software is distributed on a distribution diskette in `tar` format. To read the diskette on a LynxOS system, make a suitable working directory, change to that directory, and enter the following command at the LynxOS prompt:

```
tar xvf /dev/rfd0
```

The driver library is distributed in both binary and source form. The utility programs are given in binary form only. Table B-4 describes the files included in the software distribution.

Table B-4. Software Distribution Files

File Name	Description
`Readme`	Up-to-date information not included in this manual
`cib488.o`	C language driver interface module
`esp488.o`	ESP-488 driver library module
`ib.cfg`	`CONFIG.TBL` include file for statically linking the ESP-488 driver into the LynxOS kernel
`ibic`	Interface Bus Interactive Control program
`ibinfo.o`	Device configuration library module for statically linking the ESP-488 driver into the LynxOS kernel
`ibtsta`	Software Installation Test, Part A
`ibtstb`	Software Installation Test, Part B

`Source/`
`  Makefile`
`  brd488.h`
`  brd488_master.c`
`  cib488.h`
`  cib488_master.c`
`  esp488.h`
`  esp488_master.c`
`  ib_master.cfg`
`  ibinfo_master.c`
`  sys488.h`
`  sys488_master.c`
	Driver Source files

`ugpib.h`	User include file for ESP-488 applications

Configuring the Driver

The `esp488.o` driver library module for the AT-GPIB is configured for the following default settings.

Base I/O Address	0x2C0
Interrupt Request Line	11
DMA Channel	5

If you are installing or using more than one GPIB interface board in your system, or if the default settings are inappropriate for your hardware configuration, you must change these settings. To do this, change to the driver source directory, edit the configuration file `ibinfo_master.c`, and make the appropriate changes to the fields in the `ibinfo` structure. Save the file and enter the following line to rebuild the object module:

```
make ibinfo.o
```

For more extensive software configuration changes (for example, to disable DMA and use programmed I/O for all data transfers), you must recompile the driver. First, edit the file `sys488.h` and make any necessary changes to the `Editable Parameters` section found at the top of the file. Then rebuild the desired modules using the `Makefile` included with your software distribution files. For example, enter the following line to rebuild the default ESP module:

```
make esp488.o
```

For information on generating additional object modules for use with other GPIB interface boards, refer to the section *Installing Multiple Driver Modules*, later in this appendix.

Installing the Driver

The `esp488.o` driver module can be installed either statically or dynamically into the LynxOS kernel. Unless you are installing a number of optional devices on your system and you plan to use the AT-GPIB board only infrequently, the static linking method is recommended. This section contains instructions for statically linking the driver into the kernel. For information on dynamically loading a LynxOS device driver, refer to the *Device Driver Writer's Guide* in the *LynxOS User's Manual, Volume 2*, and to the documentation for `drinstall` and `devinstall` in the *Utility Programs* section of the *LynxOS User's Manual, Volume 1*.

To link the driver statically into the LynxOS kernel, complete the following steps:

1. Log on to the system as root and go to the directory where you unloaded the distribution files.

2. Add the ESP device configuration module to the system device library by entering the following line:

```
libr r /sys/devlib.a ibinfo.o
```

3. Add the ESP driver module to the system driver library by entering the following line:

```
libr r /sys/drivlib.a esp488.o
```

4. Copy the `CONFIG.TBL` include file `ib.cfg` to the kernel configuration directory and then change to that directory by entering the following lines:

```
cp ib.cfg /sys/lynx.os
cd /sys/lynx.os
```

5. Edit the file `CONFIG.TBL` and add the following line to the end of the file:

```
I:ib.cfg
```

6. Build a new LynxOS kernel and reboot the system by entering the following lines:

```
make install
sync;sync;reboot -aN
```

After installing the driver, refer to *IBTEST(1)* in Chapter 4, *ESP-488 Functions and Utilities Reference*, for information on testing the driver.

Installing Multiple Driver Modules

The ESP-488 driver has been designed and optimized to work with a single AT-GPIB board only. To use two or more AT-GPIB boards, you must make and install multiple copies of the ESP-488 driver. You can use the `Makefile` included with your software distribution files to generate up to four additional versions of the driver, each with a name of the form `esp488_`*X*`.o`, where *X* = 1 to 4.

In addition, you can generate corresponding files of the form `ib`*X*`.cfg`, `ib`*X*`info.o`, and `cib488_`*X*`.o` for each module. The `cib488_`*X*`.o`/`esp488_`*X*`.o` modules are functionally equivalent to the `cib488.o`/`esp488.o` modules, but each contains a unique set of function and variable names in the form `ib`*X* and `dv`*X*.

For example, to install and use two AT-GPIB boards in the system at the same time, you might choose to load the modules `esp488.o` and `esp488_2.o`. First, go to the driver source directory and edit `ibinfo_master.c` to specify the hardware settings for the second interface board (the settings must be unique from those of the first board). Then use the `make` command to generate the necessary files for the second board as follows:

```
make cib488_2.o
make esp488_2.o
make ib2.cfg
make ib2info.o
```

To install the second driver, repeat steps 2 through 6 as listed in the previous section, *Installing the Driver*, substituting the last three files above for the default installation files. For example:

```
libr r /sys/devlib.a ib2info.o
libr r /sys/drivlib.a esp488_2.o
cp ib2.cfg /sys/lynx.os
cd /sys/lynx.os
vi CONFIG.TBL          /* add line "I:ib2.cfg" */
make install
sync;sync;reboot -aN
```

When compiling an application program, you must include both C language interface modules on the link line if you want the same program to control both GPIB boards. For example:

```
cc prog.c cib488.o cib488_2.o -o prog
```

In the application program, you should code all function calls directed to the first GPIB board using the format described in Chapter 2, *The C Language Library*. You should code all function calls directed to the second board to use the entry points in the `esp488_2.o` module, as shown in Table B-5.

Table B-5. Naming Syntax for esp488_2.o

Default Name	New Name for Second GPIB Port
dvclr	dv2clr
dvrd	dv2rd
dvrsp	dv2rsp
dvtrg	dv2trg
dvwrt	dv2wrt
ibcac	ib2cac
ibcmd	ib2cmd
ibeos	ib2eos
ibeot	ib2eot
ibgts	ib2gts
iblines	ib2lines
ibonl	ib2onl
ibpad	ib2pad
ibpoke	ib2poke

(continues)

Table B-5. Naming Syntax for esp488_2.o (Continued)

Default Name	New Name for Second GPIB Port
ibrd	ib2rd
ibrpp	ib2rpp
ibrsv	ib2rsv
ibsad	ib2sad
ibsic	ib2sic
ibsre	ib2sre
ibtmo	ib2tmo
ibwait	ib2wait
ibwrt	ib2wrt
ibsta	ib2sta
ibcnt	ib2cnt
iberr	ib2err

Similarly, to install four AT-GPIB boards in the system, you could make and load the modules `esp488_1.o`, `esp488_2.o`, `esp488_3.o`, and `esp488_4.o`, and use function call and variable references in the form `ib1wrt`, `ib2wrt`, `ib3wrt`, and so on.

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). 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 Active System 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 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 status byte of the DVM 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 _____ Rev. _____

Configuration _____

National Instruments software product _____ Rev. _____

Configuration _____

The problem is _____

_____

_____

_____

_____

List any error messages _____

_____

_____

_____

_____

The following steps will reproduce the problem _____

_____

_____

ESP-488 Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use 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.

National Instruments Products

- ESP-488 for LynxOS Software Version Number on Disk _____
- Types of National Instruments GPIB boards installed and their respective configuration settings:

Board Type	Base I/O Address	Interrupt Request Line	DMA Channel

- Did you recompile the ESP-488 driver source files? _____

  If yes, did you make any changes to the driver source files? _____
  Please include a listing with the changes you made.

Other Products

- Computer Model _____
- Microprocessor _____
- LynxOS Version _____
- Types of other boards installed in your system and their respective configuration settings:

Board Type	Base I/O Address	Interrupt Request Line	DMA Channel

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 LynxOS and the AT-GPIB**

Edition Date: **August 1993**

Part Number: **320642-01**

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	Attention
CIC	Controller-In-Charge
CMPL	Complete
DACK	DMA Acknowledge
DCL	Device Clear
DMA	direct memory access
DRQ	DMA Request
DVM	digital voltmeter
EMI	electromagnetic interference
EOI	End Or Identify
EOS	End-Of-String
ERR	Error
ESP	Engineering Software Package
GET	Group Execute Trigger
GPIB	General Purpose Interface Bus
GTL	Go To Local
hex	hexadecimal
Hz	Hertz
I/O	input/output
ibic	Interface Bus Interactive Control utility
IDY	Identify
IEEE	Institute of Electric and Electronic Engineers
IFC	Interface Clear
in.	inches
IRQ	Interrupt Request
ISA	Industry Standard Architecture
LACS	Listener
LADS	Listener Addressed State
LLO	Local Lockout
M	megabytes of memory
MLA	My Listen Address
MSA	My Secondary Address
MTA	My Talk Address
PAD	Primary Address

PPC	Parallel Poll Configure
PPD	Parallel Poll Disable
PPE	Parallel Poll Enable
PPU	Parallel Poll Unconfigure
REN	Remote Enable
RQS	Request Service
s	seconds
SAD	Secondary Address
SDC	Selected Device Clear
SPE	Serial Poll Enable
SRQ	Service Request
SRQI	Service Request Input
TACS	Talker
TADS	Talker Addressed State
TCT	Take Control
TIMO	Timeout
UNL	Unlisten
UNT	Untalk