VXI bus

HP 75000 SERIES B

Quad 8-Bit Digital I/O Module HP E1330A/B

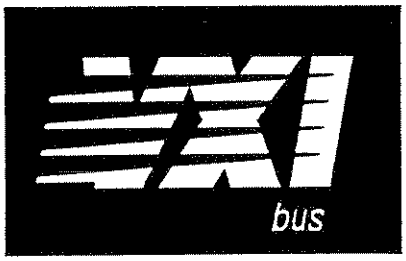
User's Manual

HEWLETT PACKARD

E1330-90003
E0492

Manual Part Number: E1330-90003
Microfiche Part Number: E1330-99003

Printed: APRIL 1992 Edition 3
Printed in U.S.A.



HP 75000 SERIES B

Quad 8-Bit Digital I/O Module HP E1330A/B

User's Manual

Manual Part Number: E1330-90003
Microfiche Part Number: E1330-99003

Printed: APRIL 1992 Edition 3
Printed in U.S.A.

CERTIFICATION

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology (formerly National Bureau of Standards), to the extent allowed by that organization's calibration facility, and to the calibration facilities of other International Standards Organization members.

WARRANTY

This Hewlett-Packard product is warranted against defects in materials and workmanship for a period of three years from date of shipment. Duration and conditions of warranty for this product may be superceded when the product is integrated into (becomes a part of) other HP products. During the warranty period, Hewlett-Packard Company will, at its option, either repair or replace products which prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Hewlett-Packard (HP). Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country.

HP warrants that its software and firmware designated by HP for use with a product will execute its programming instructions when properly installed on that product. HP does not warrant that the operation of the product, or software, or firmware will be uninterrupted or error free.

LIMITATION OF WARRANTY

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied products or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

The design and implementation of any circuit on this product is the sole responsibility of the Buyer. HP does not warrant the Buyer's circuitry or malfunctions of HP products that result from the Buyer's circuitry. In addition, HP does not warrant any damage that occurs as a result of the Buyer's circuit or any defects that result from Buyer-supplied products.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

EXCLUSIVE REMEDIES

THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES. HP SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

NOTICE

The information contained in this document is subject to change without notice. HEWLETT-PACKARD (HP) MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HP shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company. HP assumes no responsibility for the use or reliability of its software on equipment that is not furnished by HP.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013. Hewlett-Packard Company; 3000 Hanover Street; Palo Alto, California 94304

Declaration of Conformity

According to ISO/IEC Guide 22 and EN 45014

The Hewlett-Packard Company declares that the HP E1330 conforms to the following Product Specifications.

Safety: IEC 1010
CSA 231
UL 1244

EMC: CISPR 11,EN 55011, Class A
IEC 801-2, EN 50082-1, 4kVCD, 8kVAD
IEC 801-3, EN 50082-1, 3 V/M
IEC 801-4, EN 50082-1, 1kV

Jon White

Q.A. Manager
April 1992

Hewlett-Packard Company
P.O. Box 301
815 14th Street S.W.
Loveland, Colorado 80539 U.S.A

Printing History

The Printing History shown below lists all Editions and Updates of this manual and the printing date(s). The first printing of the manual is Edition 1. The Edition number increments by 1 whenever the manual is revised. Updates, which are issued between Editions, contain replacement pages to correct the current Edition of the manual. Updates are numbered sequentially starting with Update 1. When a new Edition is created, it contains all the Update information for the previous Edition. Each new Edition or Update also includes a revised copy of this printing history page. Many product updates or revisions do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

Edition 1 (Part NumberE1330-90001) .....

Edition 2 (Part Number E1330-90002) ..... Sept 1990

Edition 3 (Part Number E1330-90003) ..... Apr 1992

Safety Symbols

Instruction manual symbol affixed to product. Indicates that the user must refer to the manual for specific Warning or Caution information to avoid personal injury or damage to the product.

Indicates the field wiring terminal that must be connected to earth ground before operating the equipment—protects against electrical shock in case of fault.

Frame or chassis ground terminal—typically connects to the equipment's metal frame.

Alternating current (AC).

Direct current (DC).

Indicates hazardous voltages.

WARNING Calls attention to a procedure, practice, or condition that could cause bodily injury or death.

CAUTION Calls attention to a procedure, practice, or condition that could possibly cause damage to equipment or permanent loss of data.

WARNINGS

The following general safety precautions must be observed during all phases of operation, service, and repair of this product. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the product. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground the equipment: For Safety Class 1 equipment (equipment having a protective earth terminal), an uninterruptible safety earth ground must be provided from the mains power source to the product input wiring terminals or supplied power cable.

DO NOT operate the product in an explosive atmosphere or in the presence of flammable gases or fumes.

For continued protection against fire, replace the line fuse(s) only with fuse(s) of the same voltage and current rating and type. DO NOT use repaired fuses or short-circuited fuseholders.

Keep away from live circuits: Operating personnel must not remove equipment covers or shields. Procedures involving the removal of covers or shields are for use by service-trained personnel only. Under certain conditions, dangerous voltages may exist even with the equipment switched off. To avoid dangerous electrical shock, DO NOT perform procedures involving cover or shield removal unless you are qualified to do so.

DO NOT operate damaged equipment: Whenever it is possible that the safety protection features built into this product have been impaired, either through physical damage, excessive moisture, or any other reason, REMOVE POWER and do not use the product until safe operation can be verified by service-trained personnel. If necessary, return the product to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

DO NOT service or adjust alone: Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

DO NOT substitute parts or modify equipment: Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the product. Return the product to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

How to Use This Manual

Manual Overview

This manual shows how to operate, configure, and program the HP E1330A/B Quad 8-bit Digital I/O Module (referred to as the Digital I/O Module). The Digital I/O Module is a VMEbus Extensions for Instrumentation (VXIbus) and VMEbus B-sized register-based slave. The Digital I/O Module can operate in a B-size VXIbus or VMEbus mainframe or in a C-size VXIbus mainframe.

Throughout this manual special attention is given to the use of the module with the Hewlett-Packard 75000 Series B System mainframes, the HP E1300/E1301, and the Standard Commands for Programmable Instruments (SCPI). In many instances, module functions may exist that are not specifically supported by either mainframe or by SCPI; where appropriate, these are addressed by register-based programming in Appendix B.

Chapters in the Manual

This manual is divided into the following chapters:

Chapter 1 - Getting Started with the Quad 8-bit Digital I/O Module

- Description
- Standard Commands for Programmable Instruments (SCPI) Programming

Chapter 2 - Configuring the Quad 8-bit Digital I/O Module

- Setting the Address Switch
- Setting the Interrupt Switch
- Setting the Jumpers
- Peripheral Interface Pin-out
- Configuring the Digital I/O Module for Isolated I/O

Chapter 3 - Using the Quad 8-bit Digital I/O Module

- Inputting and Outputting Bytes
- Inputting and Outputting Bits
- Setting Polarity
- Setting Up Handshaking
- Outputting 16-bit words to a GPIO Interface

Chapter 4 - Understanding the Quad 8-bit Digital I/O Module

- The Handshaking Lines
- Typical Driver/Receiver Circuits
- Word Output Transfers

Chapter 5 - Command Reference

- Syntax Usage Summary
- Command Reference
- IEEE-488.2 Command Reference
- Command Quick Refernce

Appendix A - Quad 8-bit Digital I/O Specifications

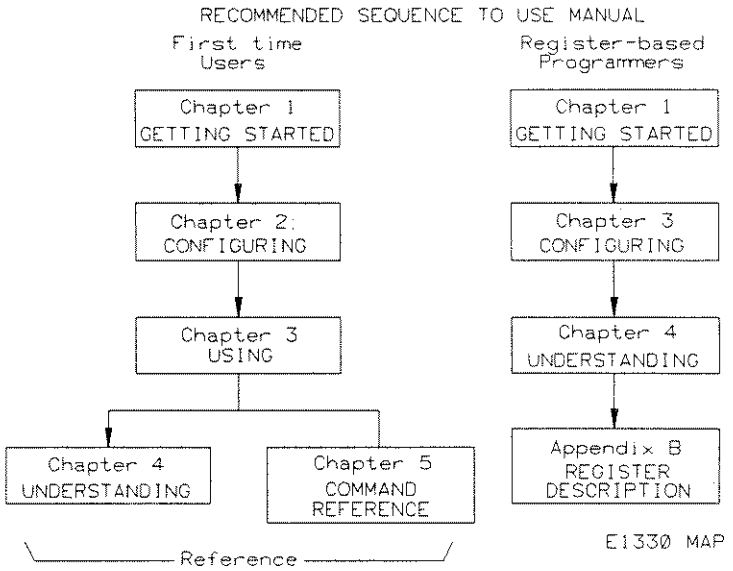
Appendix B - Quad 8-bit Digital I/O Module Register Description

Appendix C- Error Messages

Suggested Sequence to Use This Manual

This manual has five chapters and three appendixes. Chapters 1 and 2 provide description and configuration for the Digital I/O module, while Chapter 3 shows specific tasks as examples of how to use and program the module with SCPI. Chapters 4 and 5 are primarily reference material.

For basic operation using SCPI programming, see Chapters 1, 2, and 3. For basic operation using register-based programming, see Appendix B. If you need additional operating information, see Chapter 4. Chapter 5 is the SCPI command reference for the Digital I/O Module



Contents

5. Quad 8-bit Digital I/O Module Command Reference

B. Quad 8-bit Digital I/O Module Register Description

C. Quad 8-bit Digital I/O Module Error Messages

1

Getting Started

Using this Chapter

This chapter describes the Quad 8-bit Digital I/O Module and how to program the Module using SCPI (Standard Commands for Programming Instruments) commands. This chapter contains the following sections:

Instrument Definition

HP plug-in modules installed in an HP mainframe are treated as independent instruments each having a unique secondary HP-IB address. Each instrument is also assigned a dedicated error queue, input and output buffers, status registers and, if applicable, dedicated mainframe memory space for readings or data. An instrument may be composed of a single plug-in module (such as a counter) or multiple plug-in modules (for a Switchbox or Scanning Voltmeter Instrument).

Technical Description

The Quad 8-bit Digital I/O Module (referred to as the Digital I/O) is a four port digital Input/Output module intended for data communication and digital control in electronic environments compatible with TTL levels (0-5V). Each port is identical and consists of 6 control lines and 8 data lines. Resident on each port are 6 registers for control and status. In addition, the module also has a module interrupt and identification register. The Digital I/O module complies with VMEbus Extensions for Instrumentation (VXIbus) definitions for the P1 bus connector on B-sized modules. Jumpers on the module connect the flag lines for multiport data transmission, select either open collector or TTL compatible levels on the data lines, and set the VXI interrupt priority level.

Firmware provided in the mainframe for the Digital I/O allows operations in 8-bit "BYTE" format, 16-bit "WORD" format (using 2 ports) or 32-bit "LWORd" format (using all 4 ports). When using "WORD" commands, 8 bit ports 0 and 1 are treated as a single port, with port 0 being the high order byte . When using "LWORd" commands all 4 8-bit ports are used, with 8-bit port 0 being the highest order byte. Table 1-1 shows the mapping of bit numbers from the 8-bit ports to the 16-bit and 32-bit ports.

Two 3-meter, 60 wire ribbon cables with an insulation displacement header connector (ribbon cable connector) on one end are included with the Digital I/O module.

Each Digital I/O port can be configured for input or output, positive or negative true logic, six different handshaking modes, and byte, word, long word, or bit data transmission. Register-based programming can access control and status registers for custom handshaking and interrupt functions.

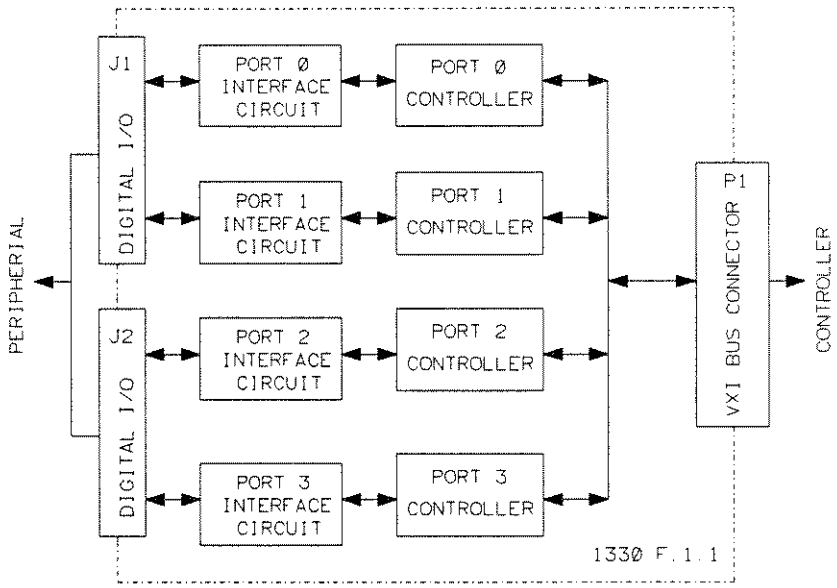


Figure 1-1. Digital I/O Module Block Diagram

Data and Handshake Lines

Each port has eight data lines and six handshake (or peripheral control) lines. The data and handshake lines for ports 0 and 1 are brought out through connector J1. The corresponding lines for ports 2 and 3 are through connector J2. With either SCPI commands or through register-based programming, you can control the state of these lines. Control of 8-bit byte transfers of data (data, flag, and control line polarity) and handshaking are available through SCPI. Some control, like peripheral interrupt control or peripheral reset, is not available through SCPI but may be allowed with register-based programming. Let's look at what these lines do.

The Data lines (Input and Output)

Each 8-bit port has eight data lines for parallel data transmission. Table 1-1 shows the data lines for the 8-bit ports and their mapping into 16-bit and 32-bit ports.

Table 1-1. Data Lines

8-bit (BYTE) operations				
Port #	0	1	2	3
Bit designations	7 - - - - - - 0	7 - - - - - - 0	7 - - - - - - 0	7 - - - - - - 0
16-bit (WORD) operations				
Port #	0		2	
Bit designations	15 - - - - - - 8	7 - - - - - - 0	15 - - - - - - 8	7 - - - - - - 0
32-bit (LWORd) operations				
Port #	0			
Bit designations	31 - - - - - - 24	23 - - - - - - 16	15 - - - - - - 8	7 - - - - - - 0

The most significant bit for BYTE is bit 7, for WORD is bit 15, and for LWORd is bit 31. The data lines of each port are bi-directional. You can enable a port for either output or input by setting its I/$\overline{O}$ line FALSE or TRUE.

The FLG Line (Input)

Each port has a flag (FLG) line. These lines are FLG(0-3) for ports 0-3. A flag line is an input line from a peripheral and has two states: READY and BUSY. A flag line is normally used in conjunction with the corresponding control (CTL) line to establish a handshake between a peripheral and the Digital I/O Module. The exact use of the flag line depends on the type of handshake in use. Refer to Chapter 3, Using the Quad 8-bit Digital I/O Module, for the handshaking modes used with SCPI.

The CTL Lines (Output)

Each port has a control line (CTL), CTL(0-3). A control line is an output line from the Digital I/O Module to the peripheral and has two states: FALSE and TRUE. A control line is normally used in conjunction with the corresponding flag line on the same port to establish a handshake between a peripheral and the Digital I/O Module. The exact use of the control line depends on the type of handshake in use. Refer to Chapter 3, Using the Quad 8-bit Digital I/O Module, for handshake description.

The I/$\overline{O}$ Lines (Output)

Each port has an I/$\overline{O}$ line, I/O(0-3). An I/O line is an output to the peripheral and has two states: FALSE and TRUE.

- When the I/$\overline{O}$ line is FALSE, the data transceiver of that port is enabled for output. The peripheral should respond to the signal by enabling itself to receive data.
- When the I/$\overline{O}$ line is TRUE, the data transceiver of that port is enabled for input. The peripheral should respond to the signal by enabling itself to send data.

Other control lines

Each port has several other control lines which are not supported by SCPI commands and are accessible only at the register level. Please see Appendix B for more details on register level programming of the Digital I/O Module.

Programming the Digital I/O Module

To program the Digital I/O Module using SCPI, you must select the controller language, interface address, and SCPI commands. Guidelines for SCPI command selection for the Digital I/O Module are covered in this manual. See the HP 75000 Series B Installation and Getting Started Guide for detailed interface addressing and controller language information.

The SCPI commands are sent to the instrument over the Hewlett-Packard Interface Bus (HP-IB). HP-IB is Hewlett-Packard's implementation of IEEE Std 488.2-1987,"Standard Digital Interface for Programmable Instrumentation."

Note

This discussion applies only to SCPI (Standard Commands for Programmable Instruments) programming. See Appendix B, Quad 8-bit Digital I/O Register Description, for details on register addressing.

Sending SCPI Commands

To send SCPI commands, specify:

- the instrument's HP-IB interface select code
- the instrument's primary HP-IB address
- the instrument's secondary HP-IB address
- the SCPI command string

The interface select code is 7. The primary address is the same as the mainframe address and can be changed in the mainframe. For the HP E1300/E1301 Mainframe, the factory setting is 09. A secondary address is assigned to the Digital I/O when used with an E1300/E1301 Mainframe or an E1405A/B or E1406A Command Module. The secondary address is normally the value obtained by dividing the logical address of the Digital I/O by 8. The logical address is set on the Digital I/O module.

The Digital I/O Module is set to a logical address of 144 at the factory. This results in a secondary address of 18 (144/8).

Figure 2-2 shows how to set the logical address.

Note

Logical addresses that are not multiples of 8 are normally not recognized as instruments by the HP E1300/1301, HP E1405 and E1406 cannot be addressed directly by SCPI commands.

The actual address which must be sent combines the three elements above. The address for this module is set to 70918 (interface select code 7, primary address 9, and secondary address 18) at the factory.

Specifying SCPI Commands

In this section you will see the SCPI commands used on a simple example which sends and receives a byte of data with the Digital I/O Module. The commands apply to four identical I/O ports, numbered from 0 through 3. To identify the port, SCPI asks you to append the port number to all commands. For example, to send data to a peripheral connected to port *n*, you send the SCPI command string:

[SOURce:]DIGital:DATA*n* [:BYTE][:VALue] < parameter>

This command string writes the data represented by the parameter to the data lines on port *n*. The parameter can be a binary, octal, decimal, or hexadecimal number.

Note

All lower case letters in these examples are optional and commands enclosed in square brackets "[ ]" are implied. Do not type brackets as shown when you use the command string in your program. They are not valid syntax in the command. You may choose to use only the upper case letters or all of the letters in a command string. A command string written with incomplete commands not represented by just upper case letters results in a syntax error.

The example above could be written as:

DIG:DATA*n* < parameter>

Similarly, reading data requires that you send the following string:

MEASure:DIGital:DATA*n*?

or, in shortened form:

MEAS:DIG:DATA*n*?

The data received is read in decimal format.

Example To write binary equivalent of decimal 135 to port 0, you use

DIG:DATA0 135

Example To read the byte on port 0 in decimal format, you use

MEAS:DIG:DATA0?

For example, a program to output the 8-bit binary equivalent of decimal 25 to port 3 using HP BASIC with an HP-IB interface at select code 7, primary address of 09 and secondary address of 18 is:

```
10 ASSIGN @Dio TO 70918
20 OUTPUT @Dio; "DIG:DATA3 25"
30 END
```

Line 10 assigns an I/O path @Dio to the device at 70918. Line 20 writes the SCPI command string to @Dio and places the binary equivalent of decimal 25 on the port 3 data lines. Line 30 ends the BASIC program.

To input a byte of data from the peripheral at the same address:

```
10 ASSIGN @Dio TO 70918
20 OUTPUT @Dio; "MEAS:DIG:DATA3?"
30 ENTER @Dio; Data
40 END
```

Line 10 assigns an I/O path @Dio to the device at 70918. Line 20 writes the SCPI command string to @Dio and read the decimal equivalent of the 8-bit byte on the port 3 data lines. Line 30 enters the data into the computer from device @Dio. Line 40 ends the BASIC program.

Note

These examples may not work with your peripheral. The Digital I/O Module's handshaking protocol, timing, and polarity have to match the peripheral's before communication can be successful. Additional SCPI commands are covered in Chapter 3, Using the Quad 8-bit Digital I/O Module, and Chapter 5, Quad 8-bit Digital I/O Command Reference.

Configuring the Quad 8-bit Digital I/O Module

Using this Chapter

In this chapter you will learn how to

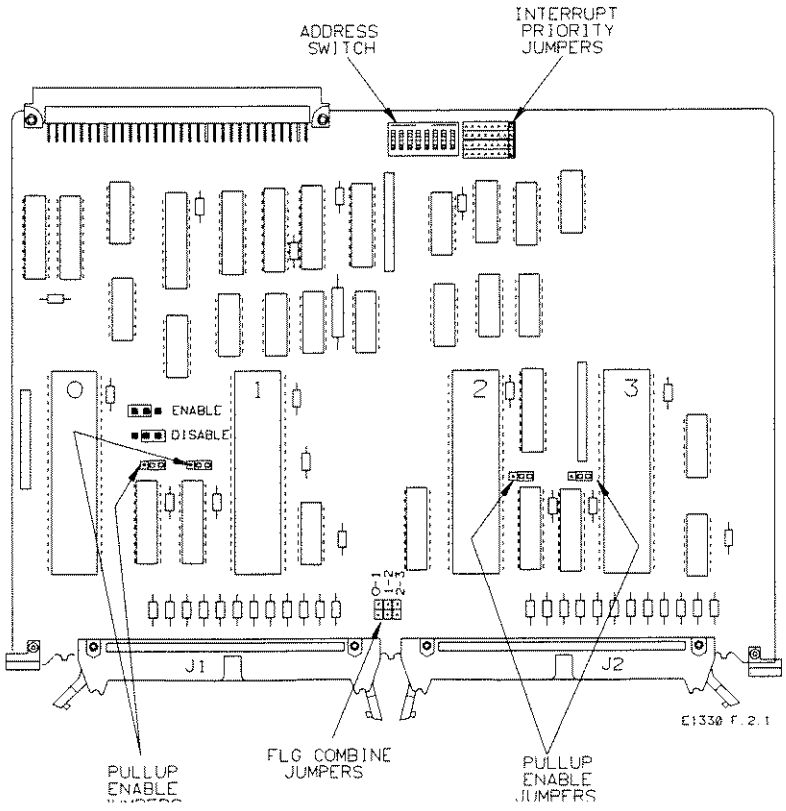


Figure 2-1. Jumper, Switch, and Connector Locations

Setting the Address Switch

Refer to Figure 2-1. In the center rear of the module, next to the P1 connector you will find the address select switch. Its factory setting is 144; rockers 4 and 7 are closed, all others are open. You can select the address of the Digital I/O Module to any number 0-255 (decimal). For example, to set the address 16, open the switch rockers 0, 1, 2, 3, 5, 6, 7 and close 4 as shown in Figure 2-2.

Note

To be recognized as an instrument when you are using the Digital I/O Module in an HP E1300/1301 Mainframe or with an HP E1405 or E1406 Command Module, the logical address *must* be set to a multiple of 8.

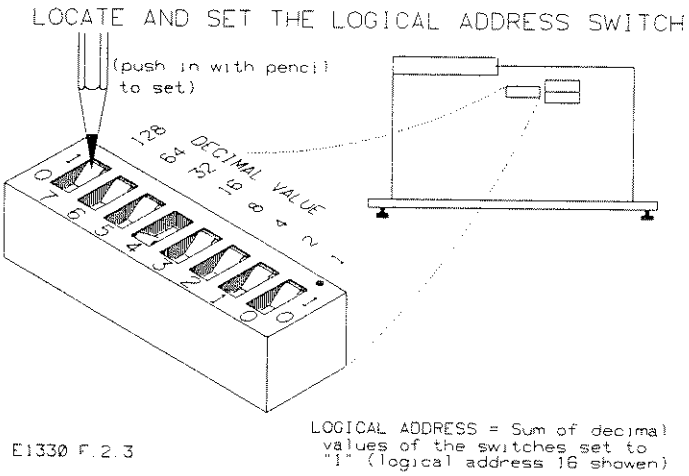


Figure 2-2. Address Switch Set at 16

Enabling Pull-ups

Referring to Figure 2-1, note the pull-up enable jumpers near the middle of each of the large ICs. Each port can be configured separately for open collector configuration or a pull up to TTL high levels. The jumpers can be configured for one of two modes: enabled or disabled; the factory-shipped condition is pull-up disabled. The enabled mode is convenient for detecting dry switch closure.

Selecting the Interrupt Line

The VXI peripheral interrupt bus consists of seven lines which can carry the interrupt signal to the controller. Module interrupt priority can be established with these lines; in general, the higher the line number, the higher the priority. Referring to Figures 2-1 and 2-3, near the P1 connector you will find two sets of jumper pins labeled x and 1-7 (JM15 and JM16). The Digital I/O Module is factory-shipped with the interrupt set to 1. If you need to change the interrupt level you must move the jumpers on both jumper blocks. Spare jumpers used for connecting the flag (FLG) lines are stored on the unused ground pins of this connector when it ships from the factory.

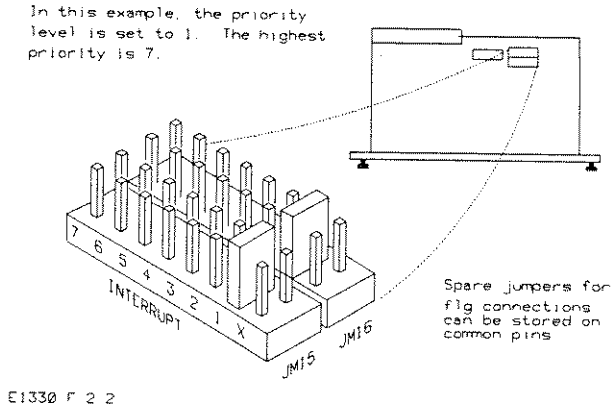


Figure 2-3. Priority Interrupt Connector (Factory Settings)

Note

The interrupt circuitry for the E1330B is implemented as release on interrupt acknowledge (ROAK). The Digital I/O module will de-assert (or release) the interrupt request line during an interrupt acknowledge cycle.

The interrupt circuitry on the E1330A was implemented as release on register access (RORA). The E1330A digital I/O module would continue to assert the interrupt request line until the Port Control/status register on the digital I/O module is accessed.

Both the E1330A and E1330B may be used with the E1300A/E1301A and with the E1405A/B and E1406A. Only the E1330B can be used with the E1570A, however.

Combining the Flag Lines

Each port has its own flag line. For the transmission of 8-bit bytes, the flag line from each of the individual ports can be used independently to handshake with a peripheral. On the other hand, the flag lines can also be combined for up to 32 bits in a word. The jumpers used to implement this configuration are shown in Figure 2-1. Extra jumpers for combining flag lines are stored on the unused pins of the interrupt priority connector.

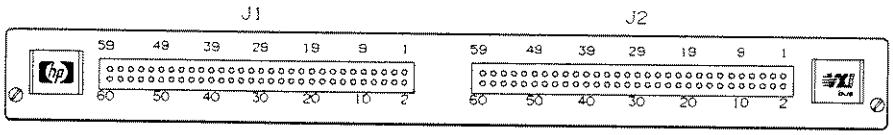
For 16-bit data width you can combine the flag lines on ports 0-1, or 1-2, or 2-3 with one jumper. The jumpered flag lines are physically tied together. Accessing either of the flag lines accesses them both.

For 32-bit data width you can combine ports 0-1, 1-2, and 2-3 with 3 jumpers. Accessing any one of the four flag lines accesses all four in this configuration.

Digital I/O Module Peripheral Pin-out

Figure 2-4 shows pinouts for the Digital I/O Module,which is compatible with easy crimp connections to ribbon cables for standard digital I/O interfacing. Figure 2-5 shows the data line location on the supplied ribbon cables. Figure 2-7 shows how to connect the cables. Details about the functioning of these pins is covered in Chapter 4 - Understanding the Quad 8-bit Digital I/O Module, but line names are as follows for ports 0 through 3:

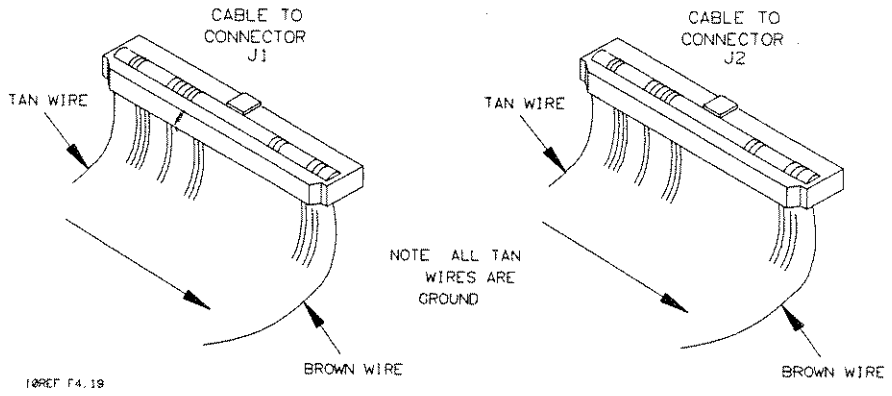
$\overline{\text{RES}}$	the reset line - used to reset a peripheral.
STS	the status line - used as an ancillary handshake line.
PIR	the peripheral interrupt line - used to signal a peripheral interrupt.
FLG	the flag line - used to handshake data between a peripheral and the Digital I/O Module. Controlled by the peripheral.
CTL	the control line - used to handshake data between a peripheral and the Digital I/O Module. Controlled by the Digital I/O Module.
I/$\overline{\text{O}}$	the input/output line - used to establish input or output on a port.
D(0-7)	the data lines - transmit information between the peripheral and Digital I/O Module.



10REF F4.17

Pin	Assignment	Pin	Assignment	Pin	Assignment	Pin	Assignment
1	——	31	D1-2	1	——	31	D3-2
3	RES1	33	D1-3	3	RES3	33	D3-3
5	RES0	35	D1-4	5	RES2	35	D3-4
7	STS1	37	D1-5	7	STS3	37	D3-5
9	STS0	39	D1-6	9	STS2	39	D3-6
11	PIR1	41	D1-7	11	PIR3	41	D3-7
13	PIR0	43	D0-0	13	PIR2	43	D2-0
15	FLG1	45	D0-1	15	FLG3	45	D2-1
17	FLG0	47	D0-2	17	FLG2	47	D2-2
19	CTL1	49	D0-3	19	CTL3	49	D2-3
21	CTL0	51	D0-4	21	CTL2	51	D2-4
23	I/O1	53	D0-5	23	I/O3	53	D2-5
25	I/O0	55	D0-6	25	I/O2	55	D2-6
27	D1-0	57	D0-7	27	D3-0	57	D2-7
29	D1-1	59	——	29	D3-1	59	——

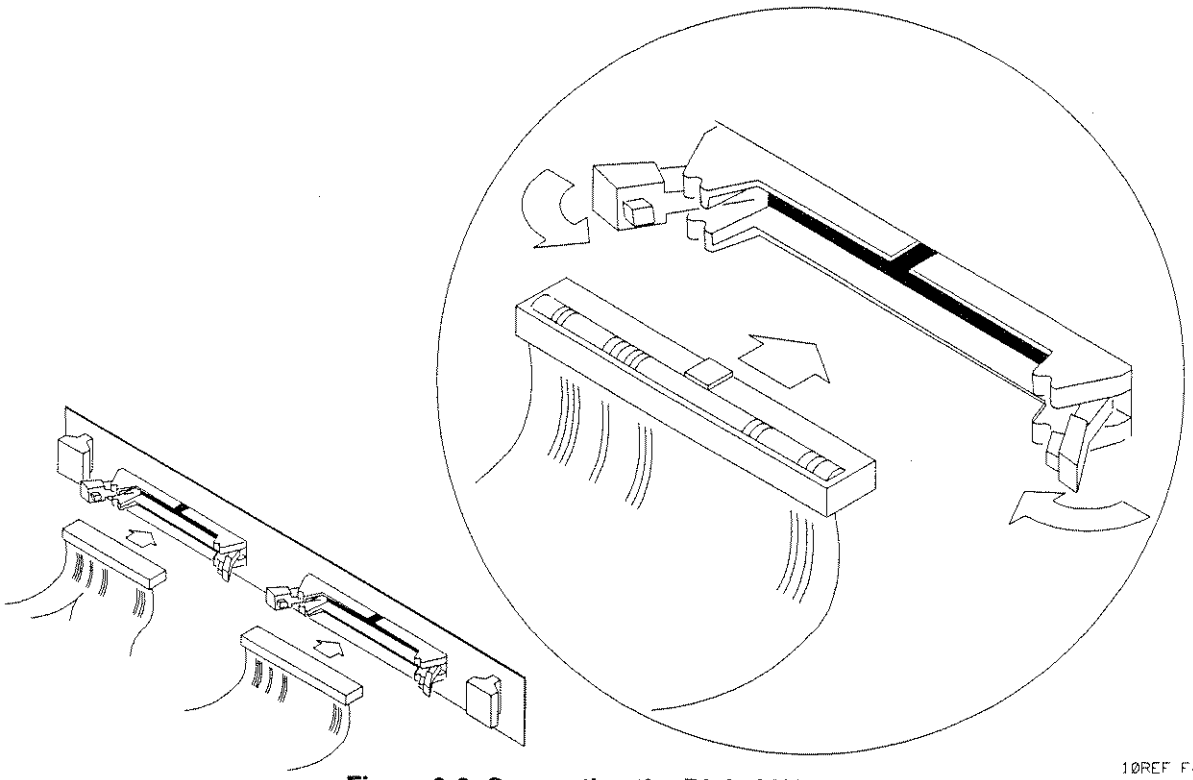
Figure 2-4. J1 and J2 Connector Pinouts



BYTE Line	WORD Line	LWORd Line	Wire Color	BYTE Line	WORD Line	LWORd Line	Wire Color
D0-7	W0-15	L0-31	White	D2-7	W2-15	L0-15	White
D0-6	W0-14	L0-30	Grey	D2-6	W2-14	L0-14	Grey
D0-5	W0-13	L0-29	Purple	D2-5	W2-13	L0-13	Purple
D0-4	W0-12	L0-28	Blue	D2-4	W2-12	L0-12	Blue
D0-3	W0-11	L0-27	Green	D2-3	W2-11	L0-11	Green
D0-2	W0-10	L0-26	Yellow	D2-2	W2-10	L0-10	Yellow
D0-1	W0-9	L0-25	Orange	D2-1	W2-9	L0-9	Orange
D0-0	W0-8	L0-24	Red	D2-0	W2-8	L0-8	Red
D1-7	W0-7	L0-23	Brown	D3-7	W2-7	L0-7	Brown
D1-6	W0-6	L0-22	Black	D3-6	W2-6	L0-6	Black
D1-5	W0-5	L021	White	D3-5	W2-5	L0-5	White
D1-4	W0-4	L020	Grey	D3-4	W2-4	L0-4	Grey
D1-3	W0-3	L0-19	Purple	D3-3	W2-3	L0-3	Purple
D1-2	W0-2	L0-18	Blue	D3-2	W2-2	L0-2	Blue
D1-1	W0-1	L0-17	Grey	D3-1	W2-1	L0-1	Grey
D1-0	W0-0	L0-16	Yellow	D3-0	W2-0	L0-0	Yellow

Notes:
Read wire colors from left to right
All even wires (tan) are ground

Figure 2-5. Data Line Location on Ribbon Cables



10REF F4.18

Figure 2-6. Connecting the Digital I/O Cable

Configuring the Digital I/O Module for Isolated Digital I/O

The two Digital I/O Module peripheral connectors, J1 and J2, each have sixty pins. An industry standard isolated digital I/O peripheral, like the **Opto 22®** 16 Position Single Channel Mounting Rack, is a 50 pin connection. The connector is either a card edge or a header connector (similar to J1 on the digital I/O module). For example, the Opto 22 rack, PB16C, uses a card edge connector; PB16H uses a header connector. They both have the same pinout for the ribbon cable. Both can accommodate up to 16 single channel I/O lines.

Since the isolation peripheral only uses 50 pins, 10 of the wires on the ribbon cable supplied are left unconnected on one end of the cable. The method of connection to the ribbon cable can be facilitated by the use specialty fixtures for these connectors, but there is no standard for connector keys or spacing. Check the connector you need before proceeding to interface to the rack.

For the Opto 22® rack, lines 1-10 are not used on the peripheral connector. Pins 27 - 57 on the ribbon cable, odd numbered pins only, correspond to the pins 17 - 47 on the Opto 22® rack. All even numbered pins are ground.

The Module Input/Output at the top of Figure 2-7 is the interface to the isolated input/output lines.

Opto 22® is a registered trademark of Opto 22, Huntington Beach, CA 92649

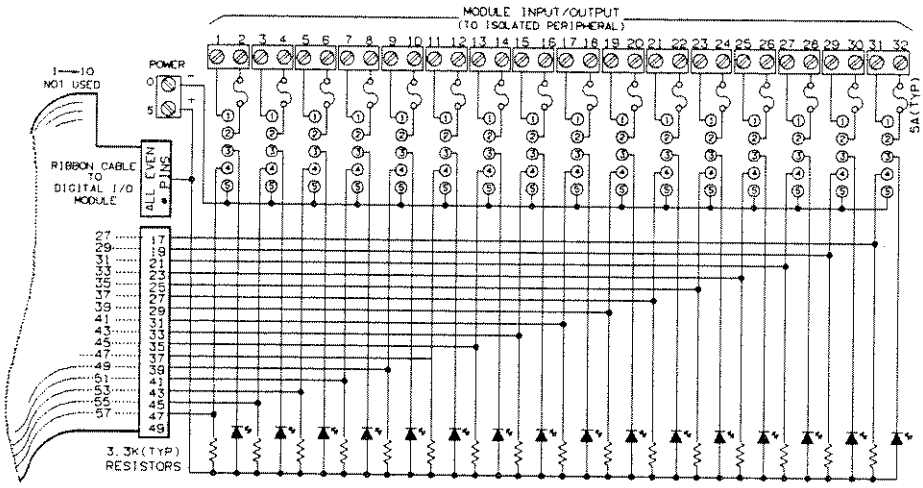


Figure 2-7. Typical Isolated Peripheral Hookup

Procedure

1. Carefully cut lines 1-10 on the ribbon cable. The blue wire should be the first wire on the ribbon cable after you make the cut.
2. Select the 50 pin connector you need, either edge connector or header connector and attach the ribbon cable.
3. Connect the ribbon cable to the Opto 22® rack for optically isolated digital operation.

Connecting the Digital I/O Module to a GPIO Peripheral

The GPIO interface is a widely used standard parallel interface for connecting computers to peripherals. The GPIO interface may employ up to 32-bits of bidirectional data transfer. The Digital I/O Module and the GPIO interface have identical line definitions but different pin assignments. Ports A-D on the GPIO are defined as ports 0-3 on the Digital I/O Module.

Procedure:

1. Connect the ribbon cable to connector J1 and/or J2 on the Digital I/O Module.
2. Connect the wires on the ribbon cable to the peripheral as described in Table 2-1 for the GPIO interface.

Table 2-1. Digital I/O Pinout to GPIO Pinout

	Port 0 Digital I/O	GPIO			Port1 Digital I/O	GPIO
Connector	J1	J2			J1	J2
Name	Pin #	Pin#		Name	Pin#	Pin#
D00	43	33		D10	27	4
D01	45	15		D11	29	22
D02	47	34		D12	31	3
D03	49	16		D13	33	21
D04	51	35		D14	35	2
D05	53	17		D15	37	20
D06	55	36		D16	39	1
D07	57	18		D17	41	19
RES0	5	12		RES1	3	29
STS0	9	26		STS1	7	8
PIR0	13	9		PIR1	11	25
FLG0	17	27		FLG1	15	7
CTL0	21	13		CTL1	19	30
I/O0	25	31		I/O1	23	11
	Port 2 Digital I/O	GPIO			Port3 Digital I/O	GPIO
Connector	J2	J1			J2	J1
Name	Pin #	Pin#		Name	Pin#	Pin#
D20	43	33		D30	27	4
D21	45	15		D31	29	22
D22	47	34		D32	31	3
D23	49	16		D33	33	21
D24	51	35		D34	35	2
D25	53	17		D35	37	20
D26	55	36		D36	39	1
D27	57	18		D37	41	19
RES2	5	12		RES3	3	29
STS2	9	26		STS3	7	8
PIR2	13	9		PIR3	11	25
FLG2	17	27		FLG3	15	7
CTL2	21	13		CTL3	19	30
I/O2	25	31		I/O3	23	11

For the Digital I/O connectors,
all even numbered pins are ground.
For the GPIO connector,
pins 5,6,10,14,23,24,28 and 32 are ground.

3

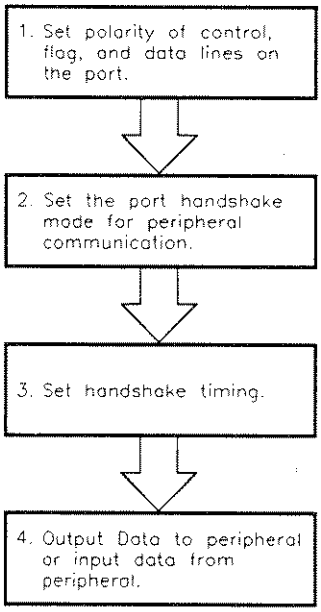
Using the Quad 8-bit Digital I/O Module

Using this Chapter

This chapter is divided into six sections about transferring data to and from the Digital I/O Module and a peripheral:

A Digital Operation Algorithm

The basic algorithm used to read data from or to write data to a peripheral is to use the following steps:



Inputting and Outputting Bytes, Words, and Long Words

For data transfers directly to or from a peripheral device, the Digital I/O transfers data using BIT, BYTE (8-bit), WORD (16-bit), or LWORd (32-bit) SCPI commands. For all transfers either binary, octal, decimal, or hexadecimal formats may be used to output data to a peripheral. All input data from the peripheral is read in decimal.

OUTPUT

The syntax used to output data follows:

[SOURce:]DIGital:DATA*n***[:type][:VALue] [base]<** *numeric data*> for output on port *n*.

[:type] is [:BYTE], :WORD, or :LWORd. The command works on 8-bit bytes if [:type] is not included. When :WORD is specified only ports 0 and 2 are available, with port 0 being the combined 8-bit ports 0 and 1 (8-bit port 0 is the high order byte of 16-bit port 0), and port 2 being the combined 8-bit ports 2 and 3 (8-bit port 2 is the high order byte of 16-bit port 2). When LWORd is specified only port 0 is available, with 8-bit port 0 being the highest order byte of the 32-bit LWORd port.

[base] sets the *numeric* string to be binary, octal, decimal, or hexadecimal by using the following coding.

Binary	[base] = # B
Octal	[base] = # Q
Decimal	[base] = no entry
Hexadecimal	[base] = # H

Example

DIG:DATA3 # HFF

Outputs hexadecimal FF on port 3.

INPUT

The syntax used to input data follows:

MEASure:DIGital:DATA*n***[:type]?** for input from port *n*.

This is always read as a decimal number.

[:type] is [:BYTE], :WORD, or :LWORd. The command works on 8-bit bytes if [:type] is not included. When :WORD is specified only ports 0 and 2 are available, with port 0 being the combined 8-bit ports 0 and 1 (8-bit port 0 is the high order byte of 16-bit port 0), and port 2 being the combined 8-bit ports 2 and 3 (8-bit port 2 is the high order byte of 16-bit port 2). When LWORd is specified only port 0 is available, with 8-bit port 0 being the highest order byte of the 32-bit LWORd port.

Example

MEAS:DIG:DATA2?

Inputs 8-bit data from the peripheral connected to port 2 in decimal format.

Inputting and Outputting Bits

The Digital I/O module can also input and output single bits to and from ports as required. This is done using the :BIT*m* keyword.

OUTPUT

The syntax for bit transfers is:

[SOURce:]DIGital:DATA*n***[:type]:BIT***m* < *1 or 0*> for output of a "1" or a "0" at bit *m* of port *n*.

[:type] is [:BYTE], :WORD, or :LWORd. The command works on 8-bit bytes if [:type] is not included. When :WORD is specified only ports 0 and 2 are available, with port 0 being the combined 8-bit ports 0 and 1 (8-bit port 0 is the high order byte of 16-bit port 0), and port 2 being the combined 8-bit ports 2 and 3 (8-bit port 2 is the high order byte of 16-bit port 2). When LWORd is specified only port 0 is available, with 8-bit port 0 being the high order byte of the 32-bit LWORd port.

Examples

DIG:DATA2:BIT3 0

Sets bit 3 of port 2 to logical 0.

INPUT

The syntax for inputting individual bits is:

MEASure:DIGital:DATA*n***[:type]:BIT***m***?** for input from bit *m* of port *n*.

[:type] is [:BYTE], :WORD, or :LWORd. The command works on 8-bit bytes if [:type] is not included. When :WORD is specified only ports 0 and 2 are available, with port 0 being the combined 8-bit ports 0 and 1 (8-bit port 0 is the high order byte of 16-bit port 0), and port 2 being the combined 8-bit ports 2 and 3 (8-bit port 2 is the high order byte of 16-bit port 2). When LWORd is specified only port 0 is available, with 8-bit port 0 being the high order byte of the 32-bit LWORd port.

Example

MEAS:DIG:DATA1:BIT7?

Reads bit 7 of port 1.

Setting the Polarity

The true level, either TTL high (> 2.5V) or TTL Low (< 1.4V), can be set on the control (CTL) line, the flag (FLG) line, and the data lines of each port with the POLarity keyword. The SCPI commands are:

[SOURce:]DIGital:CONTrol*n*:POLarity < POSitive or NEGative> to set the control line's polarity on port *n*.

[SOURce:]DIGital:FLAG*n*:POLarity < POSitive or NEGative> to set the flag line's polarity on port *n*..

[SOURce:]DIGital:DATA*n*:POLarity < POSitive or NEGative> to set the data line polarity on port *n*.

Example **DIG:DATA1:POL POS**

Sets the polarity to positive on port 1 data lines.

The ***RST** (reset) condition is positive polarity for control (CTL), flag (FLG), and data lines.

Using the Handshaking Modes

Handshaking modes used by the Digital I/O Module are designed to allow data transfer in and out of the module so that both the peripheral and the Module are ready to send or accept data. SCPI supports the following modes of handshaking which are illustrated in the timing diagrams of Figure 3-1.

- Leading Edge
- Trailing Edge
- Pulse
- Partial
- Strobe

The handshaking commands control the behavior of the control (CTL) line and the flag (FLG) lines (the interactive timing and sequence of control and flag signals). CTL is controlled by the Digital I/O Module; FLG is controlled by the peripheral. The data flow is controlled by the peripheral during an input cycle or by the Digital I/O Module during an output cycle. To interpret the timing diagrams in Figure 3-1, let's look at the Leading Edge Handshake (upper left hand corner of Figure 3-1).

The Digital I/O Module monitors the flag (FLG) line for a READY condition. Once this line is ready, the Digital I/O Module places data on the output port. After a delay time, Td, the Digital I/O Module sets the control (CTL) line TRUE. The peripheral senses the CTL condition and sets the FLG to BUSY and reads the data. The Digital I/O Module senses the FLG BUSY state and returns CTL to FALSE. After a minimum of 250 ns, the peripheral sets the FLG to READY, signaling the end of the data transfer. When FLG returns to the READY, the Digital I/O Module is ready for another cycle.

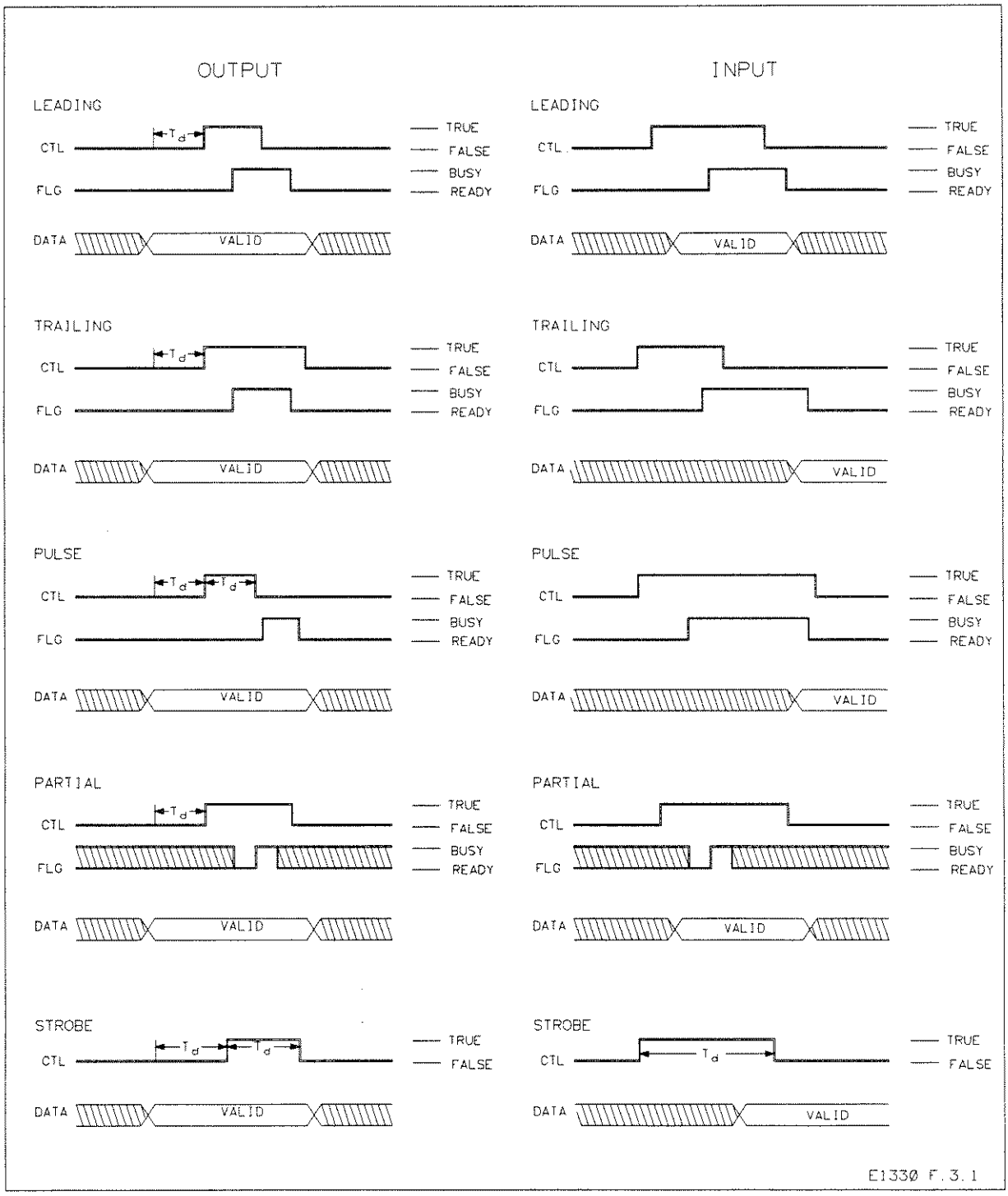


Figure 3-1. Handshake Mode Timing Diagrams

Handshaking Commands

The SCPI HANDshake keyword has six modes of operation: five corresponding to the timing diagrams in Figure 3-1 plus the no handshake mode, NONE. You can set the handshake for 2 ports by using WORD commands, and for all 4 ports by using LWORd commands.

The **DIG:HAND***n* version of this command operates only on 8-bit ports. The **DIG:DATA***n***[:type]:HAND** version of this command operates on 8-bit ([:type]= :BYTE), 16-bit ([:type]= :WORD), and 32-bit ([:type]= :LWORd) ports. The following syntax examples use all of the handshaking modes and various data output and input formats. All of the following examples use HP BASIC to communicate with the Digital I/O.

Handshaking Mode NONE

DIG:HAND*n* NONE sets the port *n* handshake mode to none. NONE is the reset state for all ports.

```
10 OUTPUT @Dio;"DIG:HAND1  NONE"
20 OUTPUT @Dio;"DIG:DATA1 # B00001010"
```

Line 10 sets the handshake mode to none on port 1 for device @Dio. Line 20 outputs the binary byte 00001010 on port 1 for device @Dio.

```
10 OUTPUT @Dio;"DIG:DATA2:WORD:HAND  NONE"
20 OUTPUT @Dio;"DIG:DATA2:WORD # B1100001100001010"
```

Line 10 sets the handshake mode to none on 16-bit port 2 for device @Dio. Line 20 outputs the binary data 1100001100001010 on 16 bit port 2 for device @Dio.

Handshaking Mode LEADing

DIG:HAND*n* LEAD sets the port *n* handshake mode to leading edge.

```
10 OUTPUT @Dio;"DIG:DATA3:HAND  LEAD"
20 OUTPUT @Dio;"MEAS:DIG:DATA3?"
30 ENTER @Dio; Data
```

Line 10 sets the handshake mode to LEADing on port 3 for device @Dio. Line 20 commands the module at address @Dio to measure the byte of data on port 3. The data from the peripheral is returned to the computer with the ENTER statement as a decimal number.

```
10 OUTPUT @Dio;"DIG:DATA0:LWOR:HAND  LEAD"
20 OUTPUT @Dio;"MEAS:DIG:DATA0:LWOR?"
30 ENTER @Dio; Data
```

Line 10 sets the handshake mode to LEADing on the 32-bit port for device @Dio. Line 20 commands the module at address @Dio to measure the 4 bytes of data on the 32 bit port. The data from the peripheral is returned to the computer with the ENTER statement as a decimal number.

Handshaking Mode TRAiling

DIG:HAND*n* TRA sets the port *n* handshake mode to trailing edge.

```
10 OUTPUT @Dio;"DIG:DATA2:WORD:HAND  TRA"
20 OUTPUT @Dio;"DIG:DATA2:WORD  # HABAB"
```

Line 10 sets the handshake mode to TRAiling on 16 bit port 2 for device @Dio. Line 20 outputs the hexadecimal data ABAB on 16-bit port 2 for device @Dio.

```
10 OUTPUT @Dio;"DIG:DATA0:LWOR:HAND  TRA"
20 OUTPUT @Dio;"DIG:DATA0:LWOR  # HABABABAB"
```

Line 10 sets the handshake mode to TRAiling on the 32-bit port for device @Dio. Line 20 outputs the hexadecimal data ABABABAB on the 32-bit port for device @Dio.

Handshaking Mode PULSe

DIG:HAND*n* PULS sets the port *n* handshake mode to pulse.

```
10 OUTPUT @Dio;"DIG:HAND3  PULS"
20 OUTPUT @Dio;"MEAS:DIG:DATA3?"
30 ENTER @Dio; Data
```

Line 10 sets the handshake mode to PULSe on 8-bit port 3 for device @Dio. Line 20 commands the module at address @Dio to measure the byte of data on 8-bit port 3. The data from the peripheral is returned to the computer with the ENTER statement as a decimal number.

Handshaking Mode PARTial

DIG:HAND*n* PART sets the port *n* handshake mode to a partial handshake.

```
10 OUTPUT @Dio;"DIG:DATA0:WORD:HAND  PART"
20 OUTPUT @Dio;"DIG:DATA0:WORD  # Q53"
```

Line 10 sets the handshake mode to PARTial on 16-bit port 0 for device @Dio. Line 20 outputs the octal byte 53 on 16-bit port 0 for device @Dio.

Handshaking Mode STRobe

DIG:HAND*n* STR sets the port *n* handshake mode to a strobe handshake.

```
10 OUTPUT @Dio;"DIG:DATA0:HAND  STR"
20 OUTPUT @Dio;"DIG:DATA0  47"
```

Line 10 sets the handshake mode to STRobe on the 32-bit port for device @Dio. Line 20 outputs the decimal byte 47 on the 32-bit for device @Dio.

Note

Output data can be either in binary, octal, decimal, or hexadecimal formats. Input data is read in decimal only. SCPI queries must be read or cleared or an error will result.

Programming Examples for Digital I/O Module

Although there are too many possible command combinations to list all of them specifically in this book, the following examples are an attempt to set forth some tasks you may require and the solutions to these tasks.

All of the following task examples assume that the task starts with the card in the *RST state. All of the examples are shown using HP Basic to communicate with the Digital I/O.

TASK Write 16-bit data 1234 hex to port 2,with handshake mode none.

```
OUTPUT @Dio;"DIGital:DATA2:WORD #h1234"
```

TASK Write 32-bit data 12345678 hex to port 0,with handshake mode none.

```
OUTPUT @Dio;"DIGital:DATA0:LWORd #h12345678"
```

TASK Write 16-bit data 1234 hex to port 0,with handshake mode LEADING.

```
10 OUTPUT @Dio;"DIGital:DATA0:WORD:HAND lead"
20 OUTPUT @Dio;"DIGital:DATA0:WORD #h1234"
```

Connect the acknowledge signal to the flag of port 0 and port 1. Use the control signal from either port 0 or port 1.

TASK Write 16-bit data 1234 hex to port 2,with handshake mode LEADING, data polarity inverted, and also flag polarity inverted.

```
10 OUTPUT @Dio;"DIGital:FLAG2:POL NEG"
20 OUTPUT @Dio;"DIGital:FLAG3:POL NEG"
30 OUTPUT @Dio;"DIGital:DATA2:WORD:POL NEG"
40 OUTPUT @Dio;"DIGital:DATA2:WORD:HAND lead"
50 OUTPUT @Dio;"DIGital:DATA2:WORD #h1234"
```

Connect the acknowledge signal to the flag of port 2 and port 3. Use the control signal from either port 2 or port 3.

TASK Write 32-bit data 12345678 hex to port 0,with handshake mode TRAiling, data polarity inverted,and also control polarity inverted. Then write 9abcdef0 hex after first the data.

```
10 OUTPUT @Dio;"DIGital:CONT0:POL NEG"
20 OUTPUT @Dio;"DIGital:CONT1:POL NEG"
30 OUTPUT @Dio;"DIGital:CONT2:POL NEG"
40 OUTPUT @Dio;"DIGital:CONT3:POL NEG"
50 OUTPUT @Dio;"DIGital:DATA0:LWOR:POL NEG"
60 OUTPUT @Dio;"DIGital:DATA0:LWORd:HAND TRA"
70 OUTPUT @Dio;"DIGital:DATA0:LWORD #h12345678"
80 ENTER @Dio;A(*) ! make sure previous command is done
90 OUTPUT @Dio;"DIGital:DATA0:LWORD #h9abcdef0"
```

Connect the acknowledge signal to the flag of all ports. Use the control signal from any port.

TASK Read 16-bit data from port 2,with handshake mode none.

```
10 OUTPUT @Dio;"MEAS:DIGital:DATA2:WORD?"
20 ENTER @Dio;A(*) ! make sure previous command is done
```

TASK Read 31st bit (i.e. bit 7) of port 0 (physical port 0) or port 0 of width 32 (logical port 0).

```
10 OUTPUT @Dio;"MEAS:DIGital:DATA0:LWORD:BIT31?"
20 ENTER @Dio;A(*) ! make sure previous command is done
```

or

```
10 OUTPUT @Dio;"MEAS:DIGital:DATA3:BIT7?"
20 ENTER @Dio;A(*) ! make sure previous command is done
```

TASK Read two 32-bit data from port 0,with handshake mode TRAiling, data polarity inverted,and also control polarity inverted.

```
10 OUTPUT @Dio;"DIGital:CONT0:POL  NEG
20 OUTPUT @Dio;"DIGital:CONT1:POL  NEG"
30 OUTPUT @Dio;"DIGital:CONT2:POL  NEG"
40 OUTPUT @Dio;"DIGital:CONT3:POL  NEG"
50 OUTPUT @Dio;"DIGital:DATA0:LWOR:POL  NEG"
60 OUTPUT @Dio;"DIGital:DATA0:LWORd:HAND  TRA"
70 OUTPUT @Dio;"MEAS:DIGital:DATA0:LWORD?"
80 ENTER @Dio;A(*) ! make sure previous command is done
90 OUTPUT @Dio;"MEAS:DIGital:DATA0:LWORD?"
```

Connect the acknowledge signal to the flag of all ports. Use the control signal from any port.

TASK Write an array of 100 bytes to port 1 and port 0. There is no memory card in the system.

The trace format (IEEE 488.2) needs binary input. We will use HP BASIC to show how it can be done.

```
10 DATA  123,1,2,3,4,5,6,7,8,9
20 INTEGER  A(1:10)  !10 words =  20 bytes
30 READ  A(*)  ! fill array in controller memory.
40 ASSIGN  @Dio TO 70907  ! dig io at sec address 7
50 OUTPUT  @Dio;"DIG:TRAC:DEF  first_track,100"  ! define a trace
named first_trace of 100 bytes length
60 OUTPUT  @Dio;"DIG:TRAC  first_trace,"# 220";A(*)  ! fill 20 bytes
of first_trace with array A
70 OUTPUT  @Dio USING "K,10(W)";"DIG:TRAC
first_trace,# 220";A(*)  ! fill 20 bytes of first_trace with  array A
```

More program lines

```
90 OUTPUT  @Dio;"DIG:DATA  first_trace"  ! output first_trace data
at port 0.
```

TASK Same task as above but using external memory card.

```
10 DATA 123,1,2,3,4,5,6,7,8,9
20 INTEGER A(1:10) !10 words = 20 bytes
30 READ A(*) ! fill array in controller memory.
40 ASSIGN @Dio TO 70907 ! digio at sec address 7
41 OUTPUT @Dio;"MEM:VME:ADDR # H200000" ! tell where
memory card is.
42 OUTPUT @Dio;"MEM:VME:SIZE 100" ! how much memory
can be used by this digital I/O
43 OUTPUT @Dio;"MEM:VME:STATE ON" ! turn state on so
trace will be placed in ext memory.
50 OUTPUT @Dio;"DIG:TRAC:DEF first_trace,100" ! define trace
named first_trace of 100 bytes length
60 OUTPUT @Dio USING "K,10(W)";"DIG:TRAC
first_trace,# 220";A(*) ! fill 20 bytes of first_trace with array A
70 OUTPUT @Dio;"DIG:DATA1 first_trace" ! output first_trace data
at port 1.
```

More program lines

```
560 OUTPUT @Dio;"DIG:DATA first_trace" ! output first_trace data at
port 0.
```

TASK Read 40 words (16 bits) from port 2 at fastest speed.

```
10 ASSIGN @Dio TO 70907 ! dig io at sec address 7
20 INTEGER A(1:40)
30 OUTPUT @Dio;"DIG:TRAC:DEF read_trace,80" ! define trace
named read_trace of 80 bytes (40 words) length
40 OUTPUT @Dio;"MEAS:DIG:DATA2:WORD:TRAC read_trace" !
fast read data at port 2 into read_trace.
50 OUTPUT @Dio;"DIG:TRAC? read_trace" ! put read_trace data
into DIO output buffer
60 ENTER @Dio USING "4A,40(W)";D$;A(*) ! read data from DIO
output buffer into controller memory (D$ gets # 280, A(*) gets data)
```

4

Understanding the Quad 8-bit Digital I/O Module

Using this Chapter

This chapter describes the signal lines used in the Digital I/O Module. An algorithm for word output transfers of 16-, 24-, and 32-bit words is also discussed. This topic is introduced in Chapter 3, Using the Quad 8-bit Digital I/O Module, for a 16-bit output to a GPIO peripheral and it is extended here.

The information in this chapter provides the user with additional background for a thorough understanding of the Digital I/O module. In particular, this information forms the basis of register programming, discussed further in Appendix B.

Chapter Contents:

A System Overview

The Digital I/O Module serves as a unique interface to peripherals that may be connected to it. In most system applications, the real human interface is the computer that controls the VXIbus mainframe. The interface that communicates between the computer and the mainframe may be RS-232 or HP-IB or some other interface. The VXIbus is the interface that the mainframe uses to communicate with the Digital I/O Module to establish setups and transfer data. And, finally, the interface with the peripheral is through the data and handshaking (peripheral control) lines. (Refer to Figure 4-1.)

Corresponding to the hardware interfaces are software (firmware) instructions to control the operation of the hardware. SCPI is the command language the HP E1300/1301 Mainframe and HP E1405 and E1406 Command Module understands. You use these commands on a high level to communicate with the peripheral connected to the Digital I/O Module. Communication protocols and device drivers are built in to this language to simplify communication. But, some modes of operation are not supported at this higher level. For example, direct control of the data transfer modes and peripheral resets can only be accomplished with direct Digital I/O Module register communication. If you have a special need, refer to Appendix B, Register Description, for more information.

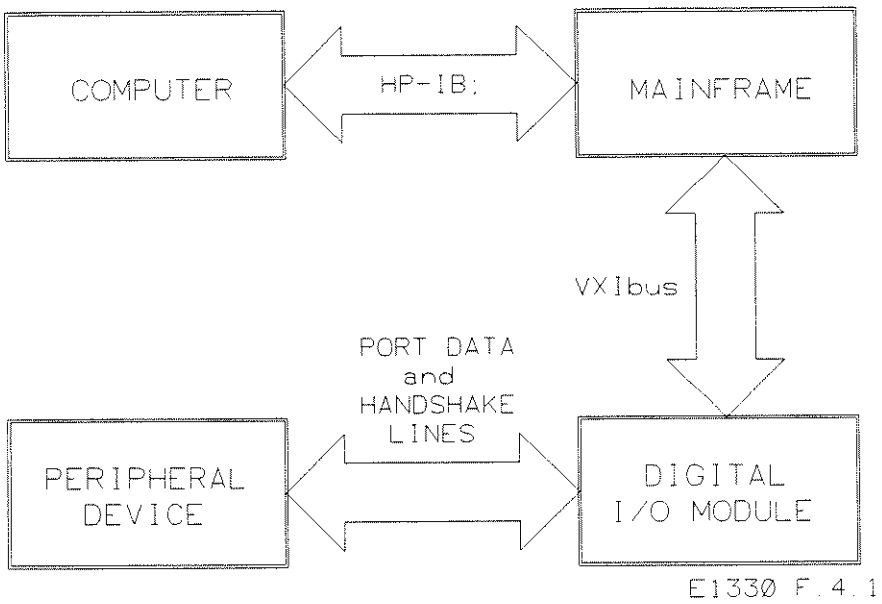


Figure 4-1. System Overview

Direction of Data Flow

To avoid confusion about the direction of the flow of data:

- Output is data from the Digital I/O Module to the peripheral.
- Input is data from the peripheral to the Digital I/O Module.

Figure 4-2 shows a data flow diagram of the Digital I/O Module. The module consists of four 8-bit I/O ports, each with its own controller, interface circuit, data lines, and handshake lines.

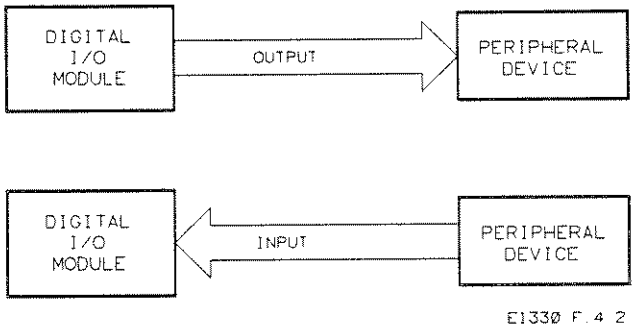


Figure 4-2. Digital I/O Module Data Flow

VXIbus Connector and Interface Circuit

The VXIbus Connector P1 plugs into your VXIbus mainframe. The VXIbus interface circuit decodes addresses, generates and responds to handshaking signals, and provides data buffering.

Port Controllers

Each of the four 8-bit ports (0,1,2, and 3) has its own controller integrated circuit (IC); hence, each port can operate independently. Each port has six hardware registers for port control. These registers let you transmit and receive data and enable handshaking and interrupts for each port.

Port Interface Circuits

Each port has an interface circuit that buffers data and handshaking between the controller IC and the port signal lines. For each port,this circuit is an 8-bit data transceiver for the eight data lines, a receiver for each of the three input handshake lines; and a driver for each of the three output handshake lines.

Peripheral Interface Connectors

The peripheral interface connectors are two 60 pin connectors. Ports 0 and 1 share connector J1, while ports 2 and 3 share connector J2. This makes it easy to pair ports 0 and 1 or 2 and 3 for 16-bit I/O operations. The pin out for these connectors is given in Chapter 2, Configuring the Quad 8-bit Digital I/O Module.

Data and Handshake Lines

Each port has eight data lines and six handshake (or peripheral control) lines. The data and handshake lines for ports 0 and 1 are brought out through connector J1. The corresponding lines for ports 2 and 3 are through connector J2. With either SCPI commands or through register-based programming, you can control the state of these lines. Control of 8-bit byte transfers of data; (data, flag, and control line polarity); and handshaking are available through SCPI. Some control, like peripheral interrupt control or peripheral reset, is not available through SCPI but may be allowed with register-based programming. Let's look at what these lines do.

The Data lines (Input and Output)

Each 8-bit port has eight data lines for parallel data transmission. Table 4-1 shows the data lines for the 8-bit ports and their mapping into 16-bit and 32-bit ports.

Table 4-1. Data Lines

8-bit (BYTE) operations				
Port #	0	1	2	3
Bit designations	7 - - - - - - 0	7 - - - - - - 0	7 - - - - - - 0	7 - - - - - - 0
16-bit (WORD) operations				
Port #	0		2	
Bit designations	15 - - - - - - 8	7 - - - - - - 0	15 - - - - - - 8	7 - - - - - - 0
32-bit (LWORd) operations				
Port #	0			
Bit designations	31 - - - - - - 24	23 - - - - - - 16	15 - - - - - - 8	7 - - - - - - 0

The most significant bit is 7 for bytes, 15 for words, and 31 for long words. The data lines of each port are bi-directional. You can enable a port for either output or input by setting its I/$\overline{O}$ line FALSE or TRUE.

The FLG Line (Input)

Each port has a flag (FLG) line. These lines are FLG(0-3) for ports 0-3. A flag line is an input line from a peripheral and has two states: READY and BUSY. A flag line is normally used in conjunction with the corresponding control (CTL) line to establish a handshake between a peripheral and the Digital I/O Module. The exact use of the flag line depends on the type of handshake in use. Refer to Chapter 3, Using the Quad 8-bit Digital I/O Module, for the handshaking modes used with SCPI.

The CTL Lines (Output)

Each port has a control line (CTL), CTL(0-3). A control line is an output line from the Digital I/O Module to the peripheral and has two states: FALSE and TRUE. A control line is normally used in conjunction with the corresponding flag line on the same port to establish a handshake between a peripheral and the Digital I/O Module. The exact use of the control line depends on the type of handshake in use. Refer to Chapter 3, Using the Quad 8-bit Digital I/O Module, for handshake description.

The I/$\overline{O}$ Lines (Output)

Each port has an I/$\overline{O}$ line, I/O(0-3). An I/O line is an output to the peripheral and has two states: FALSE and TRUE.

- When the I/$\overline{O}$ line is FALSE, the data transceiver of that port is enabled for output. The peripheral should respond to the signal by enabling itself to receive data.
- When the I/$\overline{O}$ line is TRUE, the data transceiver of that port is enabled for input. The peripheral should respond to the signal by enabling itself to send data.

Other control lines

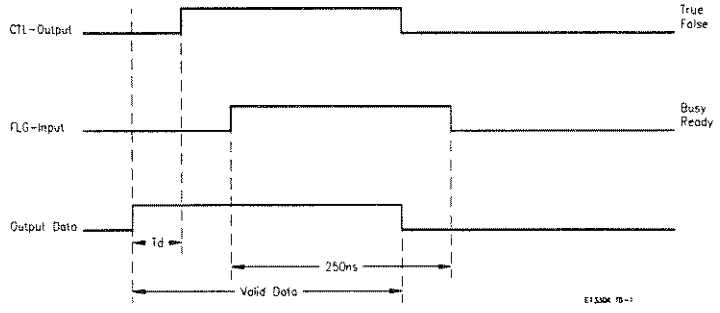
Each port has several other control lines which are not supported by SCPI commands and are accessible only at the register level. Please see Appendix B for more details on register level programming of the Digital I/O Module.

Output Handshake Timing

All output handshake timing starts with the HP E1330's I/$\overline{O}$ line in the LOW state (output function). The module then senses the FLG line and changes data. After a specified time delay (Td), the module change CTL in accordance with the following handshake modes. The peripheral device must provide at least a 250nS FLG pulse indicating it has latched the data. Td is a programmable time delay using the SOURCE:DIG:DATAn:HAND:DELAY(number) command or using register based programming (see Appendix B). The following timing diagrams show the possible handshaking modes.

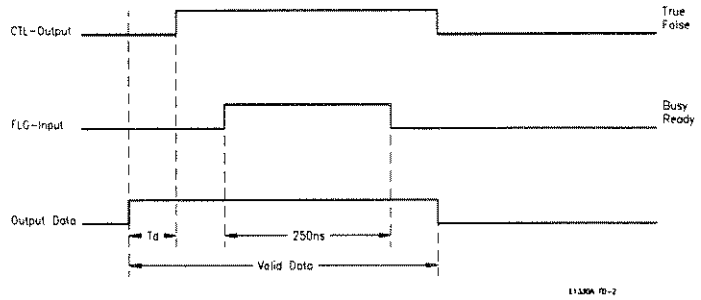
Output Leading Handshake

The ready to busy transition of FLG should latch the output data.



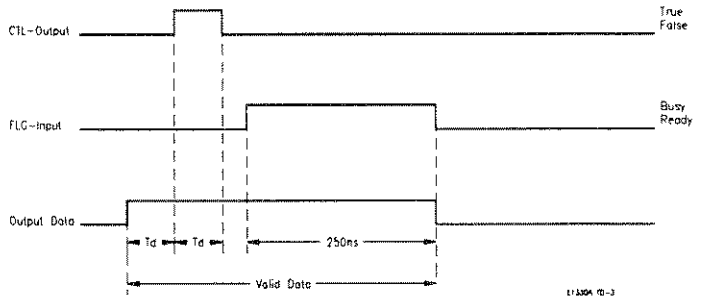
Output Trailing Handshake

The busy to ready transition of the FLG line should latch the output data.



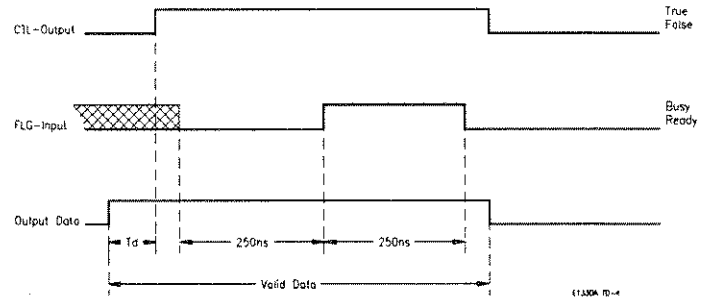
Pulse Handshake

The output data may be latched on either transition of FLG or CTL.



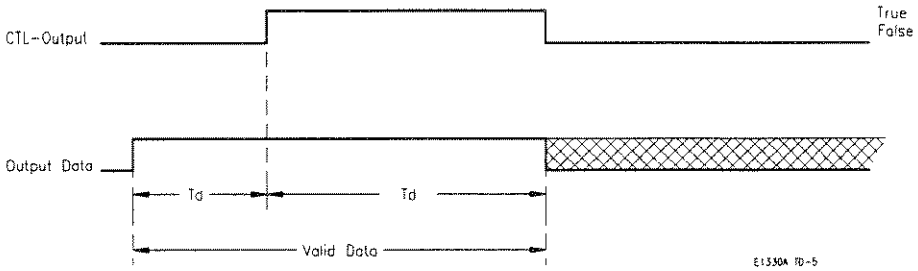
Partial Handshake

The HP E1330 doesn't check for Ready FLG. Output data should be latched on the ready to busy transition of FLG.



Strobe Handshake

No FLG line is used. The Output data may be latched on either transition of CTL.

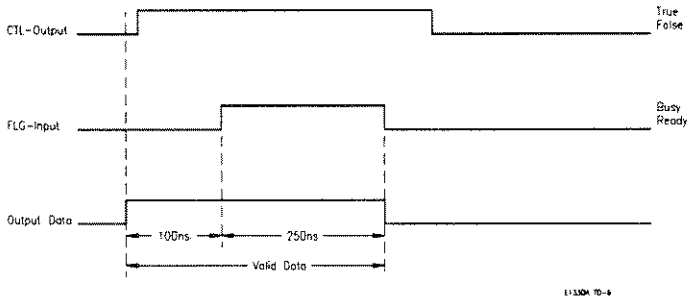


Input Handshake Timing

All input handshakes start with the HP E1330 setting its I/O line high (input mode). The module then senses the FLG line and changes its CTL line according to the handshake mode selected. The peripheral device must provide at least a 250nS FLG pulse with data true at least 100nS before it is latched. The following diagrams show the possible Input Handshaking Modes.

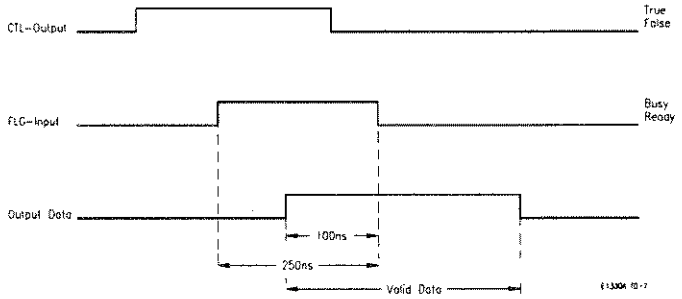
Input Leading Handshake

The ready to busy transition of the FLG line latches the input data.



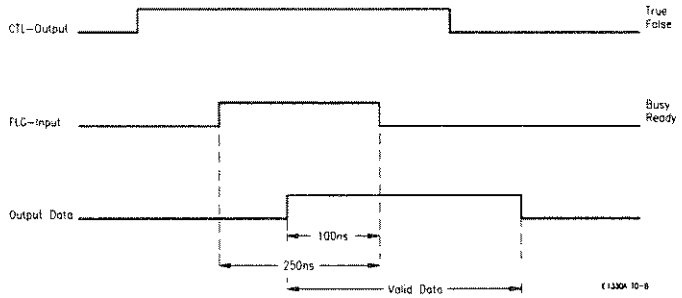
Input Trailing Handshake

The busy to ready transition of the FLG line latches the input data.



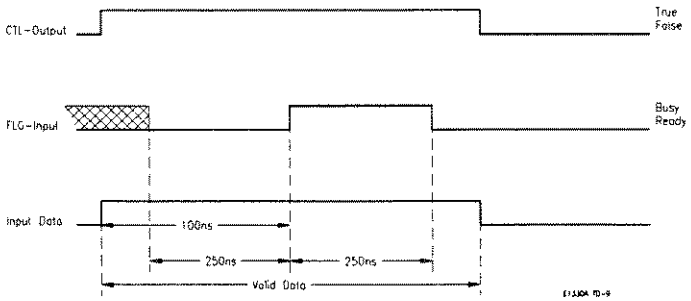
Pulse Handshake

The busy to ready transition of FLG latches the data.

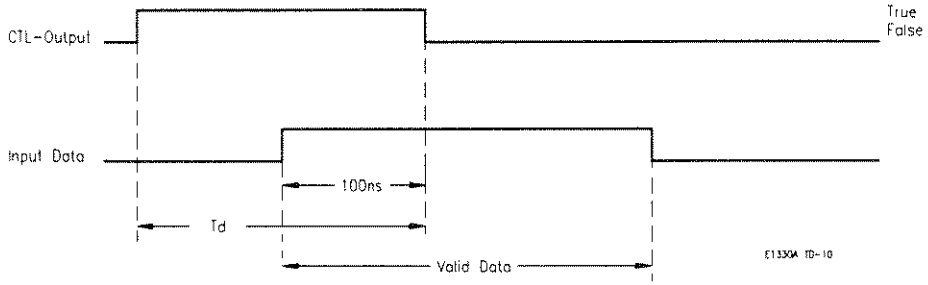


Partial Handshake

The HP E1330 doesn't check for ready FLG. Input data is latched on the true to false transition of CTL.



Strobe Handshake No FLG line is used. Input data is latched on the true to false transition of CTL.



Typical Driver/Receiver Circuits

Figure 4-3 shows the typical driver/receiver circuits that are used for digital I/O with the Digital I/O Module. Dry contact closures can be detected by using the pull-up enable jumpers shown in figure 2-1.

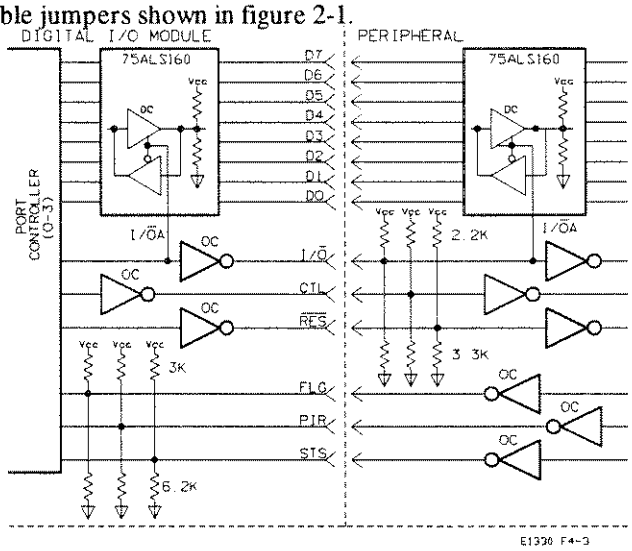


Figure 4-3. Typical Driver/Receiver Circuit

Quad 8-bit Digital I/O Module Command Reference

Chapter Contents and Description

This chapter describes Standard Commands for Programmable Instruments (SCPI) commands and summarizes IEEE 488.2 Common (*) commands applicable to the Quad 8-bit Digital I/O Module. This chapter contains the following sections:

Command Types

Commands are separated into two types: IEEE 488.2 Common Commands and SCPI Commands.

Common Command Format

The IEEE 488.2 standard defines the Common commands that perform functions like reset, self-test, status byte query, etc. Common commands are four or five characters in length, always begin with the asterisk character (*), and may include one or more parameters. The command keyword is separated from the first parameter by a space character. Some examples of Common commands are shown below:

```
*RST    *ESR 32    *STB?
```

SCPI Command Format

The SCPI commands perform functions like closing switches, making measurements, and querying instrument states or retrieving data. A subsystem command structure is a hierarchical structure that usually consists of a top level (or root) command, one or more lower level commands, and their parameters. The following example shows part of a typical subsystem:

```
[SOURce:]
    DIGital
        :DATAn
            [:VALue]?
            :BITm?
```

SOURce: is the root command, DIGital is a second level command, :DATA*n* is a third level command where *n* is the port number 0 - 3, and :VALue and :BIT*m* are fourth level commands where *m* is the queried bit location.

Command Separator

A colon (:) always separates one command from the next lower level command. This is illustrated as follows:

MEASure:DIGital:DATA*n*:VALue?

Colons separate the root command from the second level (MEASure:DIGital) and the second from third level (DIGital:DATA*n*), and so forth.

Abbreviated Commands

The command syntax shows most commands as a mix of upper and lower case letters. The upper case letters indicate the abbreviated spelling for the command. For shorter program lines, send only the abbreviated form. For better program readability you may send the entire command. The instrument will accept either the abbreviated form or the entire command.

For example, if the command reference syntax shows the command MEASure, then MEAS and MEASURE are both acceptable forms. Other forms of MEASure, such as MEASU or MEASUR will generate an error.

The instrument does not distinguish between upper case and lower case characters. Therefore MEASURE, measure, and MeAsUrE are all acceptable.

Implied Commands

Implied commands appear in square brackets ([ ]) in the command syntax. (Note that the brackets are not part of the command and are not sent to the instrument.) Suppose you send a second level command but do not send the preceding implied command. In this case, the instrument assumes you intend to use the implied command and it responds as if you had sent it. Examine this excerpt from the SOURce subsystem shown below:

```
[SOURce:]
    DIGital
    :DATAn
        [:VALue] < parameter>
        :BITm< parameter>
```

The root command SOURce and the keyword VALue are implied. To set the instrument to output a logical 1 to bit 0 of port 3, you may send either **SOURce:DIGital:DATA3:BIT0 1**, or **DIGital:DATA3:BIT0 1**.

For examples in this manual the root command and implied keywords are not used.

Short Commands

Command keywords can be entered in their full form, as shown above, or can be entered in their short form. In this manual, the entry required in short form commands is always capitalized. The example above may be entered as **SOURce:DIGital:DATA3:BIT0 1**, or **SOUR:DIG:DATA3:BIT0 1**

For examples in this manual the short form is generally used.

Parameters

Parameter Types. The following table contains explanations and examples of parameter types you might see later in this chapter.

Parameter Type	Explanations and Examples
Numeric	Accepts all commonly used decimal representations of numbers including optional signs, decimal points, and scientific notation. 123, 123E2, -123, -1.23E2, .123, 1.23E-2, 1.23000E-01. Special cases include MIN, MAX, and INF.
Boolean	Represents a single binary condition that is either true or false. ON, OFF, 1, 0.
Discrete	Selects from a finite number of values. These parameters use mnemonics to represent each valid setting. An example is the DIGital:CONTrol*n*::POLarity < *polarity*> command where *polarity* can be POS or NEG.

Optional Parameters. Parameters shown within square brackets ([ ]) are optional parameters. (Note that the brackets are not part of the command and are not sent to the instrument.) If you do not specify a value for an optional parameter, the instrument chooses a default value. For example, consider the DISPlay:MONitor:PORT? [< MIN| MAX| DEF>] command. If you send the command without specifying a parameter or send the DEF parameter, the mainframe displays the state of the port last addressed. If you send the MIN parameter, the mainframe displays the port 0 state. If you send the MAX parameter, the mainframe displays the port 3 state. Be sure to place a space between the command and the parameter.

Linking Commands

Linking IEEE 488.2 Common Commands with SCPI Commands. Use a semicolon between the commands. For example:

***RST;DIG:CONT2 1** *or* **DIG:CONT2:POL POS;*IDN?**

Linking Multiple SCPI Commands. Use both a semicolon and a colon between the commands. For example:

DIG:DATA2:POL NEG;:DIG:DATA2:BIT1 1

DISPlay:MONitor [:STATe]

SCPI Command Reference

This section describes the Standard Commands for Programmable Instruments (SCPI) commands for the Quad 8-bit Digital I/O Module. Commands are listed alphabetically by subsystem and also within each subsystem. Command guides are printed in the top margin of each page. The left guide indicates the first command listed on that page. The right guide indicates the last command listed on that page. Where only a single command appears on a page, the left and right guides will be the same.

DISPlay Subsystem

The DISPlay subsystem turns on the Monitor mode of the display and shows the module identification in the display (HP E1301 Mainframes only). The parameters displayed are:

- port number
- polarity
- handshake mode
- state of the control line
- state of the flag line
- values on the data lines in both decimal and hexadecimal

Syntax

```
DISPlay
    :MONitor
        [:STATe] < Boolean>
        :PORT < numeric | AUTO>
        :PORT? [< MAX| MIN| DEF> ]
```

DISPlay:MONitor [:STATe]

DISPlay:MONitor[:STATe] < *Boolean*> turns the monitor mode on and off.

Parameter

Parameter Name	Parameter Type	Range of Values	Default Units
STATe	Boolean	0 of 1, OFF or ON	None

Comments

- Shows state of the last port programmed by a command.
- **Related Commands:**
  DISPlay:MONItor:PORT
  DISPlay:MONItor:PORT?
- ***RST Condition: OFF**

Example

DISP:MON ON displays the state of the last port programmed.

DISPlay:MONitor: PORT*n*

DISPlay:MONitor:PORT*n* turns the monitor mode ON for port *n*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
PORT	Numeric or Discreet	none, 0, 1, 2, or 3 AUTO	0

Comments

- In the **AUTO** mode of operation, the display shows the state of the port last programmed.
- **Related Commands:**
  DISPlay:MONItor[:STATe]
  DISPlay:MONItor:PORT?
- ***RST Condition: AUTO**

Example **DISP:MON:PORT3** displays the state of port 3.

DISPlay:MONitor: PORT?

DISPlay:MONitor:PORT? [< MAX| MIN| DEF>], with no parameter, returns the identification of the port monitored. If AUTO was selected as the port parameter in the **DISP:MON:PORT AUTO** command, the query returns a –1. If DEF is specified, the query always returns 0. If MAX is specified, the query returns the maximum port (always 3). If MIN is specified, the query returns the minimum port (always 0).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
PORT?	Optional or Discrete	None MAX, MIN, or DEF	None

Comments

- This command is included to comply with IEEE-488.2 syntax requirements.
- **Related Commands:**
  DISPlay:MONitor:PORT
  DISPlay:MONitor[:STATe]
- ***RST Condition:** Not applicable.

Example **DISP:MON:PORT?** identifies the port being monitored.

MEASure Subsystem

The measure subsystem defines the command set for the Digital I/O module input statements

Syntax

```
MEASure
    :DIGital
        :DATAn
            [:BYTE][:VALue]?
            [:BYTE]:BITm?
            [:BYTE]:TRACe < name>
            :WORD[:VALue]?
            :WORD:BITm?
            :WORD:TRACe < name>
            :LWORd[:VALue]?
            :LWORd:BITm?
            :LWORd:TRACe < name>
        :FLAGn?
```

MEASure:DIGital :DATA*n*[:type] [VALue]?

MEASure:DIGital:DATA*n*[:BYTE][:VALue]? reads one byte from 8-bit port *n* after the completion of the handshake.

MEASure:DIGital:DATA*n*:WORD[:VALue]? reads one word (2 bytes) from 16-bit port *n* after the completion of the handshake.

MEASure:DIGital:DATA*n*:LWORd[:VALue]? reads one longword (4 bytes) from the 32-bit port after the completion of the handshake.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
DATA	Numeric	BYTE none, 0, 1, 2, 3 WORD none, 0, or 2 LWORd none or 0	0

Comments

- **:DATA*n*** is the keyword used for commands relating to the data at port *n*. The port number *n* must be next to the last character of the keyword without spaces.
- Input data from the Digital I/O is always assumed to be in decimal format. Other formats are not supported for input. However, data output to the Digital I/O may be in binary, octal, decimal, or hexadecimal.
- **Related Commands:**
  [SOURce:]DIGital:DATA*n*[:type][:VALue]
  MEASure:DIGital:DATA*n*[:type]:BIT*m*?
- ***RST Condition:** Set to input positive true on all ports.

Example **MEAS:DIG:DATA1?** reads 8-bit port 1 data.

MEASure:DIGital :DATA*n*[:type]:BIT*m*?

MEASure:DIGital :DATA*n*[:type]:BIT*m*?

MEASure:DIGital:DATA*n*:BYTE:BIT*m*? reads bit *m* of 8-bit port *n* after the completion of the handshake.

MEASure:DIGital:DATA*n*:WORD:BIT*m*? reads bit *m* of 16-bit port *n* after the completion of the handshake.

MEASure:DIGital:DATA*n*:LWORd:BIT*m*? reads bit *m* of the 32-bit port after the completion of the handshake.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
DATA	Numeric	BYTE none, 0, 1, 2, 3 WORD none, 0, or 2 LWORd none or 0	0
BIT	Numeric	BYTE 0-7 WORD 0-15 LWORd 0-31	None

Comments

- **:DATA*n*** is the keyword used for commands relating to the data at port *n*. The port number *n* must be next to the last character of the keyword without spaces.
- **:BIT*m*** is the keyword that specifies the bit within the eight-bit byte that is read by this command. Like the **DATA*n*** keyword, no space can be between the keyword **BIT** and the bit number *m* parameter.
- Input data is always assumed to be in binary format, since only a single bit of data is being read.
- ***RST Condition:** Set to input on all ports.

Example

MEAS:DIG:DATA1:BIT4? reads port 1, bit 4.

MEASure:DIGital :DATA*n*[:type]:TRACe < *name*>

MEASure:DIGital :DATA*n*[:type]:TRACe < *name*>

MEASure:DIGital:DATA*n*[:BYTE]:TRACe< *name*> reads 8-bit port *n* after the completion of the handshake.

MEASure:DIGital:DATA*n*:WORD:TRACe< *name*> reads 16-bit port *n* after the completion of the handshake.

MEASure:DIGital:DATA*n*:LWORd:TRACe< *name*> reads the 32-bit port after the completion of the handshake.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
DATA	Numeric	BYTE none, 0, 1, 2, or 3 WORD none, 0, or 2 LWORd none or 0	0
< name>	String	previously defined block name (max 13 characters)	None

Comments

- **:DATA*n*** is the keyword used for commands relating to the data at port *n*. The port number *n* must be next to the last character of the keyword without spaces.
- **:TRACe**< *name*> is the keyword (maximum 13 characters) that specifies the block where the data should be stored. This block must previously have been previously defined by the **[SOURce]:DIGital:TRACe:DEFINE** command.
- Input data is always assumed to be in decimal format. Other formats are not supported for input; however, data output may be in binary, octal, decimal or hexadecimal.
- **Related Commands:**
  MEASure:DIGital:DATA*n*[:VALue]?
  SOURce:DIGital: TRACe:DEFINE
- ***RST Condition:** Set to input on all ports.

Example

MEAS:DIG:DATA0:WORD:TRACe *first_block* reads data from port 0 and stores it in the user memory location *first_block*.

MEASure:DIGital :FLAG*n*?

MEASure:DIGital :FLAG*n*?

MEASure:DIGital:FLAG*n*? reads the status of the flag line on port *n* and returns a 0 or 1 to show whether a peripheral has set the flag line to READY or BUSY.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
FLAG	Numeric	none, 0, 1, 2, or 3	0

Comments

- **MEASure:DIGital:FLAG*n*?** is used to implement custom handshakes.
- **:FLAG*n*** is the keyword used for commands relating to the flag line at port *n*. The port number *n* must be next to the last character of the keyword without spaces.
- **Related Commands:**
  [SOURce:]DIGital:CONTrol*n*:POLarity?
  [SOURce:]DIGital:CONTrol*n*[:VALue]
  [SOURce:]DIGital:FLAG*n*:POLarity
  [SOURce:]DIGital:FLAG*n*:POLarity?

Example

MEAS:DIG:FLAG1? reads the port1 flag line.

MEMory Subsystem

The memory subsystem defines the command set for enabling the use of external VME memory commands and for managing macros. The incoming data (from user) is stored in IEEE block format and output data is stored in NR1 format. The addressable range is # h200000 through # hDFFFF8 in A24 space.

Syntax

MEMory
DELete:MACRo< *string*>
VME:
ADDRess < *numeric*>
ADDRess?
SIZE < *numeric*>
STATe< *boolean*>
STATe?

MEMory:DELete :MACRo< *name*>

MEMory:DELete:MACRo< *name*> deletes a macro previously recorded using Common commands.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
< name>	String	Previously defined block name (maximum 13 characters)	None

Comments

- < name> must have been previously defined by a *DMC (Define Macro) Common command.
- The maximum length for < *name*> is 13 characters.
- The difference between this statement and the Common command statement *PMC (Purge Macros) is that the Common command has no provisions for deleting a single macro.

Example

MEM:DEL:MACR test_macro deletes the macro named *test_macro* which has been previously defined using IEEE 488.2 Common commands.

MEMory:VME :ADDRess *< base> < address>*

MEMory:VME :ADDRess *< base> < address>*

MEMory:VME:ADDRess *< base> < address>* establishes the address of add-on VME memory in the system.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ADDRess	Numeric or Discrete	# H200000 - # HDFFFF8 MIN or MAX	None

Comments

- Addresses are accepted in either decimal or hexadecimal.
- *< base>* specifies the numeric format as decimal, hexadecimal, octal, or binary. IEEE-488.2 specifies the following values for this parameter:

  Decimal = no parameter
  Hexadecimal = # H
  Octal = # Q
  Binary = # B

- Valid values for *< address>* are # H200000 (2,097,152 decimal) through # HDFFFF8 (14,680,056 decimal).
- For this memory to actually be used it must also have a defined length and have been turned ON using the MEMory:VME:STATe command.
- **Related Commands:**
  MEMory:VME:SIZE*< size>*
  MEMory:VME:STATe*< ON or OFF>*
- ***RST Condition:** # h200000.

Example

MEM:VME:ADDR # H200000 sets the starting VME address to 200000_{16}.

MEMory:VME :ADDRess? [< MIN | MAX>]

MEMory:VME:ADDRess? [< *MIN* | *MAX*>] queries for the current VME memory address. The optional parameter lets you query for the fixed minimum or maximum address.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ADDRess	Discrete	none, MIN, or MAX	None

Comments

- This command always returns the address in decimal format
- The address returned using *MIN* is always 2,097,152.
- The address returned using *MAX* is always 14,680,056.
- **Related Commands:**
  MEMory:VME:ADDRess?
  MEMory:VME:STATe?
  MEMory:VME:SIZE?[< *MIN or MAX*>]

MEMory:VME:SIZE < size>

MEMory:VME:SIZE [< *base*>]< *size*> sets the size in bytes of the external memory board.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
< base>	Discrete	None or # H	None
< size>	Numeric or Discrete	0 - # hE00000 or MIN or MAX	None None

Comments

- Address plus size must not exceed # hE00000
- Sizes are accepted in either decimal or hexadecimal.
- < *base*> specifies the numeric format as decimal, hexadecimal, octal, or binary. IEEE-488.2 specifies the following values for this parameter:

  Decimal = no parameter
  Hexadecimal = # H
  Octal = # Q
  Binary = # B

- **Related Commands:**
  MEMory:VME:ADDRess?
  MEMory:VME:STATe?
  MEMory:VME:SIZE?

- ***RST Condition:** # h000000.

MEMory:VME:SIZE? [< *MIN* | *MAX*>]

MEMory:VME:SIZE? [< *MIN* | *MAX*>] queries for the current VME memory size. The optional parameter lets you query for the fixed maximum or minimum VME memory size.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
SIZE	Discrete	MIN or MAX	None

Comments

- This command always returns the memory size in decimal format.
- The size returned using *MIN* is always 0.
- The size returned using *MAX* is always 12582912
- **Related Commands:**
  MEMory:VME:ADDRess?[< *MIN or MAX*>]
  MEMory:VME:STATe?
  MEMory:VME:SIZE < *size*>

MEMory:VME:STATe < *state*>

MEMory:VME:STATe < *state*> enables/disables the use of VME memory for storage.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
STATe	Boolean	0 or 1, OFF or ON	None

Comments

- **Related Commands:**
  MEMory:VME:ADDress < *address*>
  MEMory:VME:SIZE < *size*>
- ***RST Condition:** Set to OFF.

Example **MEM:VME:STAT ON** enables access to the VMEmemory.

MEMory:VME:STATe?

MEMory:VME:STATe? queries the state of the external memory flag.

Parameters None

Comments

- **Related Commands:**
  MEMory:VME:ADDRess?[< *MIN or MAX*>]
  MEMory:VME:SIZE?[< *MIN or MAX*>]

[SOURce:] Subsystem

The SOURce subsystem defines the command set for the Digital I/O module output statements. It also defines the state and polarity of the control line (CTL), the polarity of the flag line (FLG), the handshaking mode, and delay for both data input and output. The root command, SOURce, is optional.

Syntax

```
[SOURCE:]
    DIGital
        :TRACe
            :CATalog?
            [:DATA]< string> < block_data>
            [:DATA]?< string>
            :DEFine< string> < numeric> [numeric]
            :DELete[:NAME]< string>
            :DELete:ALL
        :CONTroln
            :POLarity< POS| NEG>
            :POLarity?
            [:VALue]< 0 or 1>
        :DATAn
            [:BYTE]
                :BITm < 0 or 1>
                :POLarity< POS or NEG>
                :POLarity?
                [:VALue]< numeric>
                :HANDshake
                    :DELay< numeric>
                    :DELay?
                    [:MODE]    < NONE| LEADing| TRAiling
                               | PULSe| PARTial| STRobe>
                    [:MODE]?
                    :TRACe < name>
            :WORD
                :BITm < 0 or 1>
                :POLarity< POS or NEG>
                :POLarity?
                [:VALue]< numeric>
                :HANDshake
                    :DELay< numeric>
                    :DELay?
                    [:MODE]    < NONE| LEADing| TRAiling
                               | PULSe| PARTial| STRobe>
                    [:MODE]?
                    :TRACe < name>
```

```
        :LWORD
            :BITm < 0 or 1>
            :POLarity < POS or NEG>
            :POLarity?
            [:VALue] < numeric>
            :HANDshake
                :DELay < numeric>
                :DELay?
                [:MODE]   < NONE| LEADing| TRAiling
                          | PULSe| PARTial| STRobe>
                [:MODE]?
                :TRACe < name>
    :HANDshaken
        :DELay < numeric>
        :DELay?
        [:MODE]   < NONE| LEADing| TRAiling
                  | PULSe| PARTial| STRobe>
        [:MODE]?
    :FLAGn
        :POLarity < POS or NEG>
        :POLarity?
```

DIGital:TRACe :CATalog?

[SOURce:]DIGital:TRACe:CATalog lists the currently available data blocks.

Parameters

None

Comments

- This command catalogs all blocks in VME memory and all blocks in the mainframe system memory.

DIGital:TRACe [:DATA] < name> < block_data>

[SOURce:]DIGital:TRACe[:DATA] < *name*> < *block_data*> writes a block of data to a previously defined user memory block.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
< name>	String	Name of user memory block (maximum 13 characters)	None
< block_data>	Numeric/String	Numeric header and ASCII block data	None

Comments

- < name> must have been previously defined by a **DIGital:TRACe:DEFine** command.
- The maximum length for < *name*> is 13 characters.

- < block_data> is of the form # < *digits*> < *length*> < *block*> where:
  < *digits*> tells how many digits are used to define < *length*> ;
  < *length*> tells how many bytes are to be transferred in < *data*> ;
  < *data*> contains the actual data to transfer.

Example **DIG:TRAC:DATA first_block, # 210ABCDEFGHIJ** sends the data "ABCDEFGHIJ" to the user memory block *first_block*.

DIGital:TRACe [:DATA]? < *name*>

[SOURce:]DIGital:TRACe[:DATA]?< *name*> reads a block of data from a previously defined user memory block.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
< name>	String	Name of user memory block (maximum 13 characters)	None

Comments

- < name> must have been previously defined by a **DIGital:TRACe:DEFine** command.
- The maximum length for < *name*> is 13 characters.

Example **DIG:TRACe? first_block** reads data from a block named *first_block*.

DIGital:TRACe :DEFine < *name*> , < *size*> , [< *fill*>]

[SOURce:]DIGital:TRACe:DEFine < *name*> , < *size*> , [< *fill*>] defines a block of data as a user memory block, names the block for future reference, and fills the block with the last parameter. If the last parameter is absent, the block is filled with zeros.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
< name>	String	Name of user memory block (maximum 13 characters)	None
< size>	Numeric	Up to 4Gbytes (depending on memory installed)	None
< fill>	Numeric	0 - 255	None

Comments

- The firmware can handle blocks with a total memory space of up to 12Mbytes of memory space. The actual amount available depends on the memory installed.
- If the **MEMory:VME:STATe ON** command has been used, this command will create blocks in the external add-on memory. If the **MEMory:VME:STATe OFF** command has been used, this command will create blocks in the system memory.

Example **DIG:TRAC:DEF first_block, 256** defines a 256 byte user memory block named *first_block*.

DIGital:TRACe :DEFine? < name>

[SOURce:]DIGital:TRACe:DEFine? < *name*> returns the size of a previously defined user memory block in bytes.

Parameters

< name>

Parameter Name	Parameter Type	Range of Values	Default Units
< name>	String	Name of user memory block (maximum 13 characters)	None

Comments

- < name> must have been previously defined by a **DIGital:TRACe:DEFine** command. The maximum length for < *name*> is 13 characters.

DIGital:TRACe :DELete < name>

[SOURce:]DIGital:TRACe:DELete < *name*> deletes a previously defined user memory data block.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
< name>	String	Name of user memory block (maximum 13 characters)	None

Comments

- < name> must have been previously defined by a **DIGital:TRACe:DEFine** command. The maximum length for < *name*> is 13 characters.

Example **DIG:TRACe:DEL first_block** deletes a user memory block named *first_block*.

DIGital:TRACe :DELete:ALL

[SOURce:]DIGital:TRACe:DELete:ALL deletes all previously defined user memory data blocks.

Parameters

None

DIGital:CONTroln :POLarity< polarity>

[SOURce:]DIGital:CONTroln:POLarity < *polarity*> sets the voltage level for logical true in port *n* to either TTL high for **POSitive** polarity or TTL low for **NEGative** polarity.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
CONTrol	Numeric	none, 0, 1, 2, or 3	0
POLarity	Discrete	POSitive or NEGative	None

Comments

- Control lines are always accessed by their 8-bit port number
- **:CONTrol***n* is the keyword used for commands relating to the control (CTL) line at port *n*. The port number *n* must be next to the last character of the keyword without spaces.
- The control line is used with the flag line to handshake data to and from peripherals.
- **Related Commands:**
  [SOURce:]DIGital:CONTrol*n*:POLarity?
  [SOURce:]DIGital:CONTrol*n*[:VALue]
  [SOURce:]DIGital:FLAG*n*:POLarity
  [SOURce:]DIGital:FLAG*n*:POLarity?
- ***RST Condition: POLarity = POSitive**

Example **DIG:CONT0:POL POS** sets logical true to TTL high on port 0 control line.

DIGital:CONTrol*n* :POLarity?

[SOURce:]DIGital:CONTrol*n*:POLarity? returns either POSitive or NEGative for the logical true condition of the control (CTL) line of port *n*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
CONTrol	Numeric	none, 0, 1, 2, or 3	0

Example **DIG:CONT0:POL?** queries the state of the logical true condition on port 0.

DIGital:CONTrol*n* [:VALue]< *value*>

[SOURce:]DIGital:CONTrol*n*[:VALue]< *value*> sets or clears the control line on the selected port *n*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
CONTrol	Numeric	none, 0, 1, 2, or 3	0
< value>	Boolean	0 or 1, OFF or ON	None

Comments

- This command is used to create custom handshakes when the **HANDshake** is set to **NONE**.

DIGital:DATA*n* [:type]:BIT*m* < *value*>

- **:CONTrol***n* is the keyword used for commands relating to the control (CTL) line at port *n*. The port number *n* must be next to the last character of the keyword without spaces.
- The control line is used with the flag line to handshake data to and from peripherals.
- **Related commands:**
  [SOURce:]DIGital:CONTrol*n*:POLarity
  [SOURce:]DIGital:CONTrol*n*:POLarity?
  [SOURce:]DIGital:FLAG*n*:POLarity
  [SOURce:]DIGital:FLAG*n*:POLarity?
- ***RST Condition:** Clears the control line; i.e., sets the control line to logical 0.

Example **DIG:CONT2 1** sets the 8-bit port 2 control line true.

DIGital:DATA*n* [:type]:BIT*m* < *value*>

[SOURce:]DIGital:DATA*n*[:BYTE]:BIT*m* < value> writes data to port *n* and bit *m*.
[SOURce:]DIGital:DATA*n*:WORD:BIT*m* < value> writes data to word *n* and bit *m*.
[SOURce:]DIGital:DATA*n*:LWORd:BIT*m* < value> writes data to longword *n* and bit *m*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
DATAn	Numeric	BYTE none, 0, 1, 2, or 3 WORD none, 0, or 2 LWORd none or 0	0
BITm	Numeric	BYTE 0 - 7 WORD 0 - 15 LWORd 0 - 31	None
< value>	Numeric	0 or 1	None

Comments

- **:DATA*n*** and **:BIT*m*** are the keywords used to write data to port *n* and bit *m*. The port number *n* and bit number *m* must be next to the last character of the keyword without spaces.
- For 16-bit operations using **:WORD**, *n* must be 0 or 2.
- For 32-bit operations using **:LWORd**, *n* must be 0.
- **Related commands:**
  [SOURce:]DIGital:DATA*n*[:VALue]
  [SOURce:]DIGital:DATA*n*:POLarity
- ***RST Condition:** All ports are set for data input.

Example **DIG:DATA3:BIT4 1** sets bit 4 (the 5th bit) of port 3 to logical 1.

DIGital:DATA*n* [:type]:TRACe< *name*>

[SOURce:]DIGital:DATA*n*[:BYTE]:TRACe < *name*> writes the named block of data to 8-bit port *n* whenever the port is in ready state to start a new handshake.

[SOURce:]DIGital:DATA:WORD:TRACe < *name*> writes the named block of data to 16-bit port *n* whenever the port is in ready state to start a new handshake.

[SOURce:]DIGital:DATA*n*:LWORd:TRACe < *name*> writes the named block of data to the 32-bit port whenever the port is in ready state to start a new handshake.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
DATAn	Numeric	BYTE none, 0, 1, 2, or 3 WORD none, 0, or 2 LWORd none or 0	0
< name>	String	Name of user memory block (maximum 13 characters)	None

Comments

- **:DATA*n*** and **:TRACe** are the keywords used to write data to port *n* from block < *name*> . The port number *n* must be next to the last character of the keyword without spaces.
- For 16-bit operations using **:WORD**, *n* must be 0 or 2.
- For 32-bit operations using **:LWORd**, *n* must be 0.
- **Related commands:**
  [SOURce:]DIGital:DATA*n*[:VALue]
  [SOURce:]DIGital:DATA*n*:POLarity
- ***RST Condition:** All ports are set for data input.

Example

DIG:DATA2:TRAC:WORD first_block writes data from the user memory block *first_block* to 16-bit port 2.

DIGital:DATA*n* [:type]:POLarity < *polarity*>

[SOURce:]DIGital:DATA*n*:POLarity < *polarity*> sets the voltage level for logical true in port *n* to either TTL high for **POSitive** polarity or TTL low for **NEGative** polarity.

[SOURce:]DIGital:DATA*n*:POLarity:WORD < *polarity*> sets the voltage level for logical true in both 8-bit ports involved to either TTL high for **POSitive** polarity or TTL low for **NEGative** polarity.

[SOURce:]DIGital:DATA*n*:POLarity:LWORd < *polarity*> sets the voltage level for logical true in all ports to either TTL high for **POSitive** polarity or TTL low for **NEGative** polarity.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
DATA	Numeric	BYTE none, 0, 1, 2, or 3 WORD none, 0, or 2 LWORd none or 0	0
POLarity	Discrete	POSitive or NEGative	None

Comments

- **:DATA*n*** is the keyword used for commands relating to the control line at port *n*. The port number *n* must be next to the last character of the keyword without spaces.
- **Related Commands:**
  [SOURce:]DIGital:DATA*n*:POLarity?
  [SOURce:]DIGital:DATA*n*[:VALue]
  [SOURce:]DIGital:DATA*n*:BIT*m*
- ***RST Condition: POLarity = POSitive**

Example

DIG:DATA0:POL POS sets logical true to TTL high on port 0 data lines.

DIGital:DATA*n* [:type]:POLarity?

[SOURce:]DIGital:DATA*n*[:BYTE]:POLarity? returns either POSitive or NEGative as the logical true condition of the data lines of 8-bit port*n*.

[SOURce:]DIGital:DATA*n*:WORD:POLarity? returns either POSitive or NEGative as the logical true condition of the data lines of 16-bit port*n*.

[SOURce:]DIGital:DATA*n*:LWORd:POLarity? returns either POSitive or NEGative as the logical true condition of the data lines of the 32-bit port.

DIGital:DATA*n*[:type] [:VALue]< *value*>

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
DATA	Numeric	BYTE none, 0, 1, 2, or 3 WORD none, 0, or 2 LWORd none or 0	None
POLarity	Discrete	POSitive or NEGative	None

Example

DIG:DATA0:POL? returns the state of the logical true condition on port 0 as either POSitive or NEGative.

DIGital:DATA*n*[:type] [:VALue]< *value*>

[SOURce:]DIGital:DATA*n*[:BYTE][:VALue] [< *base*>]< *value*> writes data to 8-bit port *n* . Values can be binary, octal, decimal, or hexadecimal.

[SOURce:]DIGital:DATA*n*:WORD[:VALue] [< *base*>]< *value*> writes data to 16-bit port *n* . Values can be binary, octal, decimal, or hexadecimal.

[SOURce:]DIGital:DATA*n*:LWORd[:VALue] [< *base*>]< *value*> writes data to the 32-bit port. Values can be binary, octal, decimal, or hexadecimal.

Parameter

Parameter Name	Parameter Type	Range of Values	Default Units
DATA	Numeric	BYTE none, 0, 1, 2, or 3 WORD none, 0, or 2 LWORd none or 0	0
< base>	Discrete	None, # H, # Q, or # B	None
< value>	Numeric	BYTE -2^{7} to $(2^{8}-1)$ WORD -2^{15} to + $(2^{16}-1)$ LWORd -2^{31} to + $(2^{31}-1)$	Decimal

Comments

- < *base*> specifies the numeric format as decimal, hexadecimal, octal, or binary. IEEE-488.2 specifies the following values for this parameter:

  Decimal = no parameter
  Hexadecimal = # H
  Octal = # Q
  Binary = # B

- **:DATA*n*** is the keyword used for commands relating to data output at port *n*. The port number *n* must be next to the last character of the keyword without spaces.

- **Related commands:**
  [SOURce:]DIGital:DATA*n*:BIT*m*
  [SOURce:]DIGital:DATA*n*:POLarity

- ***RST Condition:** All ports are set for data input.

Example

DIG:DATA3 27 writes the binary equivalent of the decimal 27 (00011011) to 8-bit port 3.

DIG:DATA3 # B00011011 writes the same byte of data as above to port 3 in binary format.

DIGital:FLAG*n* :POLarity < *polarity*>

[SOURce:]DIGital:FLAG*n*:POLarity < *POS or NEG*> sets the voltage level for logical true to either TTL high, **POSitive**, or TTL low, **NEGative** on the FLAG line.

Parameter Name	Parameter Type	Range of Values	Default Units
FLAG	Numeric	none, 0, 1, 2, or 3	
POLarity	Discrete	POSitive or NEGative	None

Comments

- **:FLAG*n*** is the keyword used for commands relating to the flag line at port *n*. The port number *n* must be next to the last character of the keyword without spaces.
- **Related Commands:**
  [SOURce:]DIGital:FLAG*n*:POLarity?
  [SOURce:]DIGital:CONTrol*n*:POLarity
  [SOURce:]DIGital:CONTrol*n*:POLarity?
- ***RST Condition: POLarity = POSitive**

Example

DIG:FLAG0:POL POS sets logical true to TTL high on the port 0 flag line.

DIGital:FLAG*n* :POLarity?

[SOURce:]DIGital:FLAG*n*:POLarity? returns either POSitive or NEGative as the logical true condition of the flag (FLG) line.

Parameter Name	Parameter Type	Range of Values	Default Units
FLAG	Numeric	none, 0, 1, 2, or 3	None

Example

SOURCE:DIGITAL:FLAG0:POLARITY ? uses long commands to query the state of the logical true condition on port 0.

DIG:FLAG0:POL? performs the same function as the example above with short commands.

DIGital:DATA*n*[:type] :HANDshake:DELay < *time*>

[SOURce:]DIGital:DATA*n*[:BYTE]:HANDshake:DELay < *time*> sets the delay between data output and the control line for data output at 8-bit port *n*. It also sets the strobe pulse width for both output and input STROBE handshakes.

[SOURce:]DIGital:DATA*n*:WORD:HANDshake:DELay < *time*> sets the delay between data output and the control line for data output at 16-bit port *n*

[SOURce:]DIGital:DATA*n*:LWORd:HANDshake:DELay < *time*> sets the delay between data output and the control line for data output at the 32-bit port.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
DATA	Numeric	BYTE none, 0, 1, 2, or 3 WORD none, 0, or 2 LWORd none or 0	0
< time>	Numeric	2 us to 15 us 20 us to 150 us 200 us to 1.5 ms 2ms to 15ms	Seconds

Comments

- **:DATA*n*[:type]:HANDshake** is the sequence used for commands relating to data handshaking at ports defined by *n*. The port number *n* must be next to the last character of **:DATA** without spaces.
- **DIGital:HANDshake*n* NONE** command sets the delay to 0. For all other modes of handshaking 2 us is the minimum.
- Bands of delay settings are not allowed. these are:

  1.5 us to 2.0 us
  150 us to 200 us
  1.5 ms to 2.0 ms

  The controller places closest rounded-up value in the parameter field if these values are specified.

- **Related commands:**
  [SOURce:]DIGital:HANDshake*n*[:MODE]
  [SOURce:]DIGital:CONTrol*n*:POLarity
  [SOURce:]DIGital:CONTrol*n*[:VALue]
  [SOURce:]DIGital:FLAG*n*:POLarity

- ***RST Condition:** Delay is set to 2 us.

Example

DIG:HAND3:DEL .005 sets the delay between the data output and the assertion of the control line to true on 8-bit port 3 to 5 ms.

DIGital:DATA*n*[:type] :HANDshake:DELay?

[SOURce:]DIGital:DATA*n*[:BYTE]:HANDshake:DELay? queries for the delay time between data output and the control line for data output at 8-bit port *n*.

[SOURce:]DIGital:DATA*n*:WORD:HANDshake:DELay? queries for the delay time between data output and the control line for data output at 16-bit port *n*

[SOURce:]DIGital:DATA*n*:LWORd:HANDshake:DELay? queries for the delay between data output and the control line for data output at the 32-bit port.

Parameters

None

Comments

- **:DATA*n*[:type]:HANDshake is** the sequence used for commands relating to data handshaking at ports defined by *n*. The port number *n* must be next to the last character of the **:DATA** without spaces.

DIGital:HANDshake*n* :DELay < *time*>

[SOURce:]DIGital:HANDshake*n*:DELay < *time*> sets the time between data valid and the assertion of the control line to TRUE for port *n*. This form of the command operates on 8-bit ports only.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
HANDshake	Numeric	0, 1, 2, or 3	None
< time>	Numeric	2 us to 15 us 20 us to 150 us 200 us to 1.5 ms 2ms to 15ms	Seconds

- **:HANDshake*n*** is the keyword used for commands relating to data handshaking at port *n*. The port number *n* must be next to the last character of the keyword without spaces.
- **DIGital:HANDshake*n* NONE** command sets the delay to 0. For all other modes of handshaking 2 us is the minimum.
- Bands of delay settings are not allowed. These are:
  1.5 us to 2.0 us
  150 us to 200 us
  1.5 ms to 2.0 ms

  The controller places closest rounded-up value in the parameter field if these values are specified.
- **Related commands:**
  [SOURce:]DIGital:HANDshake*n*[:MODE]
  [SOURce:]DIGital:CONTrol*n*:POLarity
  [SOURce:]DIGital:CONTrol*n*[:VALue]
  [SOURce:]DIGital:FLAG*n*:POLarity
- ***RST Condition:** Delay is set to 2 us.

Example **DIG:HAND3:DEL .005** sets the delay between the data output and the assertion of the control line to true on 8-bit port 3 to 5 ms.

DIGital:HANDshake*n* :DELay?

[SOURce:]DIGital:HANDshake*n*:DELay? queries for the time between data valid and the assertion of the control line to TRUE. This form of the command operates only on 8-bit ports.

Parameters **None**

Comments

- **:HANDshake*n*** is the keyword used for commands relating to data handshaking at 8-bit port *n*. The port number *n* must be next to the last character of the keyword without spaces.

Example **DIG:HAND0:DEL?** queries the delay time between data valid and the assertion of the control line to TRUE on 8-bit port 0.

DIGital:DATA*n*[:type] :HANDshake[:MODE] < *mode*>

[SOURce:]DIGital:DATA*n*[:BYTE]:HANDshake[:MODE] < *mode*> selects the type of handshake mode and defines the timing relationship between the control (CTL) line, the flag (FLG) line, and when data is transferred in either direction between the Digital I/O Module and a peripheral on 8-bit port *n*. All handshakes are initiated by execution of a DIG:DATA*n* or MEAS:DATA*n*? command.

[SOURce:]DIGital:DATA*n*:WORD:HANDshake[:MODE] < *mode*> selects the handshake mode used on the 16-bit port *n*.

[SOURce:]DIGital:DATA*n*:LWORd:HANDshake[:MODE] < *mode*> selects the handshake mode used on the 32-bit port.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
DATA	Numeric	BYTE none, 0, 1, 2, or 3 WORD none, 0, or 2 LWORd none or 0	0
< mode>	Discrete	NONE, LEADing, TRAiling, PULSe PARTial, or STRobe	Seconds

Comments

- **:DATA[:type]HANDshake*n*** is the sequence used for commands relating to data handshaking at port *n*. The port number *n* must be next to the last character of **:DATA** without spaces.
- **NONE** deletes all automatic data handshaking between the Digital I/O Module and the peripheral. For custom handshaking, the control and the flag lines are controlled by the **DIGital:CONTrol*n*** and **DIGital:FLAG*n*** commands.

- **Related commands:**
  [SOURce:]DIGital:HANDshake*n*:DELay]
  [SOURce:]DIGital:CONTrol*n*:POLarity
  [SOURce:]DIGital:CONTrol*n*[:VALue]
  [SOURce:]DIGital:FLAG*n*:POLarity

- ***RST Condition:** Mode is **NONE** on all ports.

Example **DIG:HAND3 LEAD** sets the handshake mode to LEADing on port 3.

DIGital:HANDshake*n* [:MODE] < *mode*>

[SOURce:]DIGital:HANDshake*n* [:MODE] < mode> selects the type of handshake mode and defines the timing relationship between the control (CTL) line, the flag (FLG) line, and when data is transferred in either direction between the Digital I/O Module and a peripheral on 8-bit port *n*. All handshakes are initiated by execution of a DIG:DATA or MEAS:DATA*n*? command. This form of the HANDshake command operates only on 8-bit ports.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
HANDshake	Numeric	None, 0, 1, 2, or 3	0
< mode>	Discrete	NONE, LEADing, TRAiling, PULSe PARTial, or STRobe	Seconds

Comments

- **:HANDshake*n*** is the keyword used for commands relating to data handshaking at port *n*. The 8-bit port number *n* must be next to the last character of the keyword without spaces.

- **NONE** deletes all automatic data handshaking between the Digital I/O Module and the peripheral. For custom handshaking, the control and the flag lines are controlled by the **DIGital:CONTrol*n*** and **DIGital:FLAG*n*** commands.

- **Related commands:**
  [SOURce:]DIGital:HANDshake*n*:DELay]
  [SOURce:]DIGital:CONTrol*n*:POLarity
  [SOURce:]DIGital:CONTrol*n*[:VALue]
  [SOURce:]DIGital:FLAG*n*:POLarity

- ***RST Condition:** Mode is **NONE** on all ports.

Example **DIG:HAND3 LEAD** sets the handshake mode to LEADing on port 3.

DIGital:HANDshake*n*

[SOURce:]DIGital:HANDshake*n* queries for the current handshake mode of 8-bit port *n*. This form of the HANDshake command operates only on 8-bit ports.

Parameters None

Comments

- **:HANDshake***n* is the keyword used for commands relating to data handshaking at port *n*. The port number *n* must be next to the last character of the keyword without spaces.

SYSTem Subsystem

The SYSTem subsystem reports the status of the error registers.

Syntax

```
SYSTem
    :ERRor?        :VERsion?
```

SYSTem:ERRor?

SYSTem:ERRor? queries the error register for the error value and returns an error message to identify the error type.

- **Related Commands:**
  ***ERR**
- ***RST Condition: NONE**

Example **SYST:ERR?** queries the mainframe for errors.

SYSTem:VERSion?

SYSTem:VERsion? Returns the SCPI version to which this instrument complies.

Comments

- The returned information is in the format: YYYY.R; where YYYY is the year, and R is the revision number within that year.

IEEE 488.2 Common Commands

The following table lists the IEEE 488.2 Common (*) Commands that can be executed by the Quad 8-bit Digital I/O Module. For more information on Common Commands, refer to the HP 75000 Series B Mainframe (HP Model Number E1300/E1301) User's Manual or the ANSI/IEEE Standard 488.2-1987.

Note

These commands apply to many instruments and are not documented in detail here. See the HP 75000 Series B E1300/E1301 Mainframe User's Manual or the ANSI/IEEE Standard 488.2-1987 for more information.

Command	Title	Description
*IDN?	Identification	Returns identification string of the Digital I/O Module
*RST	Reset	Sets all ports to input mode. Sets handshake to NONE. Sets Polarity to POS.
*TST?	Self-Test	Always returns 0.
*OPC	Operation Complete	See note below
*OPC?	Operation Complete Query	See note below
*WAI	Wait to Complete	See note below
*CLS	Clear status	Clears all status registers
*ESE	Event status enable	See note below
*ESE?	Event status enable query	See note below
*ESR?	Event status register query	See note below
*SRE	Service request enable	Enables status register bits
*SRE?	Service request enable query	See note below
*STB?	Read status byte query	See note below
*TRG	Trigger	See note below
*RCL	Recall instrument state	See note below
*SAV	Store instrument state	See note below
*EMC		
*EMC?		
*RMC		
*LMC?		
*DMC	Define macro	Defines a macro
*GMC?		
*PMC	Purge macros	Purges all system macros

Command Quick Reference

Command	Parameter	Description
DISPlay:MONitor[:STATe]	< 0| 1orOFF| ON>	Turns the monitor mode of the display on.
DISPlay:MONitor:PORT*n*	< number| AUTO>	Turns the monitor mode on for the specified port.
DISPlay:MONitor:PORT?*n*	[MIN| MAX| DEF]	Returns the monitored port number.
DISPlay:MONitor:STRing?	< string>	Returns the string which appears on the mainframe front panel.
MEASure:DIGital:DATA*n***[:type]?**	< number>	Reads selected 8-, 16-, or 32-bit port after completion of handshake. Assumes decimal format of input data.
MEASure:DIGital:DATA*n***[:type] :BIT***m***?**	< number> and < number>	Reads selected bit on selected 8-, 16-, or 32-bit port after completion of handshake.
MEASure:DIGital:DATA*n***[:type] :TRACe<** *name***>**	< number> and < string>	Reads selected 8-, 16-, or 32-bit port after completion of handshake and and stores block.
MEASure:DIGital:FLAG*n***?**	< POS or NEG>	Reads FLAG line on selected port. Returns 0 or 1. Used to implement custom handshakes.
MEMory:DELete:MACRo< *name***>**	< string>	Deletes a macro
MEMory:VME:ADDRess< *address***>**	< number or MIN or MAX>	Sets the address for additional VME system memory.
MEMory:VME:ADDRess?	[MIN or MAX]	Returns the current add-on VME memory address.
MEMory:VME:SIZE< *size***>**	< number>	Sets the size of the add-on VME memory to be used for the DIO card.
MEMory:VME:SIZE?		Returns the current size of the add-on VME memory assigned to the DIO module.
MEMory:VME:STATe< *state***>**	< 0 or 1, ON or OFF>	Sets the state (ON or OFF) of the assigned VME memory. When this is OFF, all memory commands refer to the base system memory.
MEMory:VME:STATe?		Returns the current state (ON or OFF) of the add-on VME memory.

Command	Parameter	Description
[SOURce:]DIGital:TRACe:CATalog		Lists the currently defined memory blocks.
[SOURce:]DIGital:TRACe[DATA] *< name> < block_data>*	< string> and < string>	Writes a block of data to a previously defined memory block.
[SOURce:]DIGital:TRACe[:DATA]? *< name>*	< string>	Reads a block of data from a previously defined memory block.
[SOURce:]DIGital:TRACe:DEFine *< name> < size> < fill>*	< string> and < number> and < number>	Defines a memor block and fills it with the specified fill.
[SOURce:]DIGital:TRACe:DEFine? *< name>*	< string>	Returns the size in bytes of a previously defined block of data.
[SOURce]:DIGital:TRACe:DELete< name>	< string>	Deletes the specified memory block.
[SOURce]:DIGital:TRACe:DELete ALL		Deletes all memory blocks.
[SOURce]:DIGital:CONTrol*n*:POLarity *< polarity>*	< number> and < POS or NEG>	POS = 1 for TTL high voltage; NEG = 1 for TTL low voltage.
[SOURce]:DIGital:CONTrol*n*:POLarity?		Returns POS or NEG.
[SOURce:]DIGital:CONTrol*n*[:VALue]	< numeric>	Sets or clears control line on selected port. Command used to create custom handshakes when HANDshake is set to NONE
[SOURce:]DIGital:DATA*n*:[:type] [:VALue] *< value>*	< number> and < number>	Writes data to selected port when port is ready to start a new handshake. Data can be binary, octal, decimal, or hexadecimal.
[SOURce:]DIGital:DATA*n*[:type] :BIT*m*< value>	< number> and < number>	Sets or clears selected bit on selected port when port is ready to start a new handshake.
[SOURce:]DIGital:DATA*n*[:type] :TRACe *< name>*	< number> and < string>	Writes the named block of data to the specified port whenever the port is in ready state to start a new handshake.
[SOURce:]DIGital:DATA*n*[:type] HANDshake:DELay *< time>*	< number>	Sets delay between data output and assertion of control line for data output. Also sets strobe pulse for both output and input STROBE handshake.
[SOURce:]DIGital:DATA*n*[:type] HANDshake:DELay?		Returns handshake delay time for selecterd port.
[SOURce:]DIGital:DATA*n*[:type] HANDshake[:MODE]	NONe, LEADing, TRailing, PULSe, PARTial,STRobe	Selects type of handshake to transfer data between the selected port and peripheral. All handshakes are initiated by execution of a DIG:DATAn or MEAS:DATAn? command.
[SOURce:]DIGital:DATA*n*[:type] HANDshake[:MODE]?		Returns NONE, LEAD, TRA, PULS, PART, or STR to show handshake type for the selected port.

Command	Parameter	Description
[SOURce:]DIGital:DATA*n*[:type]:POLarity	< POS or NEG>	POS= 1 for TTL high voltage. NEG= 1 for TTL low voltage.
[SOURce:]DIGital:DATA*n*[:type]:POLarity?		Returns POS or NEG.
[SOURce]:DIGital:FLAG*n* :POLarity< polarity>	< number> < 0 or 1, OFF or ON>	Controls interpretation of flag line. POS = 1 for TTL high voltage; NEG = 1 for TTL low voltage.
[SOURce]:DIGital:FLAG*n*:POLarity?	< number>	Returns POS or NEG.
[SOURce]:DIGital:HANDshake*n*:DELay	< number>	Sets delay between data output and assertion of control line for data output. Also sets strobe pulse for both output and input STROBE handshake.
[SOURce]:DIGital:HANDshake*n*[:MODE]	< NONE, LEADing, TRAiling, PULSe, PARTial, STRobe>	Selects type of handshake to transfer data between selected port and peripheral. All handshakes are initiated by execution of DIG:DATAn or MEAS:DATAn? command
[SOURce]:DIGital:HANDshake*n*[:MODE]?		Returns NONE, LEAD, TRA, PULS, PART, or STR.
SYStem:ERRor?		Returns the contents of the system error register

A

Specification

Logic Levels:
TTL Compatible, 5V max

Data Lines:
Iout (High): -5.2 mA
@ Vout (High): 2.5 V
(Pullup Enabled)
Iout (Low): 48 mA
@ Vout (Low): 0.5 V
Vin (High): > 2.0 V; < 5.0 V
Vin (Low): < 0.8 V
Iin (High): < 2.5 mA @ 2.5 V
Iin (Low): < -3.2 mA @ 0.4 V

Handshake Lines:
Iout (High): 250 µA
@ Vout (High): 5 V
Iout (Low): 40 mA
@ Vout (Low): 0.7 V
Iout (Low): 16 mA
@ Vout (Low): 0.4 V
Vin (High): > 2.0 V
Vin (Low): < 0.8 V
Iin (Low): < 1.75 mA

Module Size/Device Type:
B, register-based

Connectors Used: P1

Number of Slots: 1

VXIbus Interface Capability:
Slave, interrupter, A16, D16, D08EO

Interrupt Level: 1-7, selectable

Power Requirements:
Voltage: + 5 V
Peak module current, IPM (A): 0.50
Dynamic module current, IDM (A): 0.01

Watts/Slot: 2.5

Cooling/Slot:
0.04 mm H_2O @ 0.21 liter/sec

Humidity: 65%, 0 to 40°C

Operating Temperature:
0 to 55°C

Storage Temperature:
-40 to 75°C

EMC, RFI, Safety:
meets FTZ 1046/1984, CSA 556B, IEC 348, UL 1244

Net Weight (kg): 1.0

B

Quad 8-bit Digital I/O Module Register Information

Using this Appendix

The contents of this appendix are:

Addressing the Registers

To access a specific register for either read or write operations, the address of the register must be used. Register addresses for the plug-in modules are found in an address space known as VXI A16. The exact location of A16 within a VXIbus master's memory map depends on the design of the the VXIbus master you are using; for the HP E1300/1301 Mainframe and HP E1405/E1406 Command Module, the A16 space location starts at 1F0000h.

The A16 space is further divided so that the modules are addressed only at locations above 1FC000h within A16. Further, every module is allocated 64 register addresses (40h). The address of a module is determined by its logical address (set by the address switches on the module) times 64 (40h). In the case of the Digital I/O module, the factory setting is 144 or 90h, so the addresses start at 1FE400h

Register addresses for register-based devices are located in the upper 25% of VXI A16 address space. Every VXI device (up to 256) is allocated a 64 byte block of addresses.

Figure B-1 shows the register address location within A16. Figure B-2 shows the location of A16 address space in the HP E1405 Command Module.

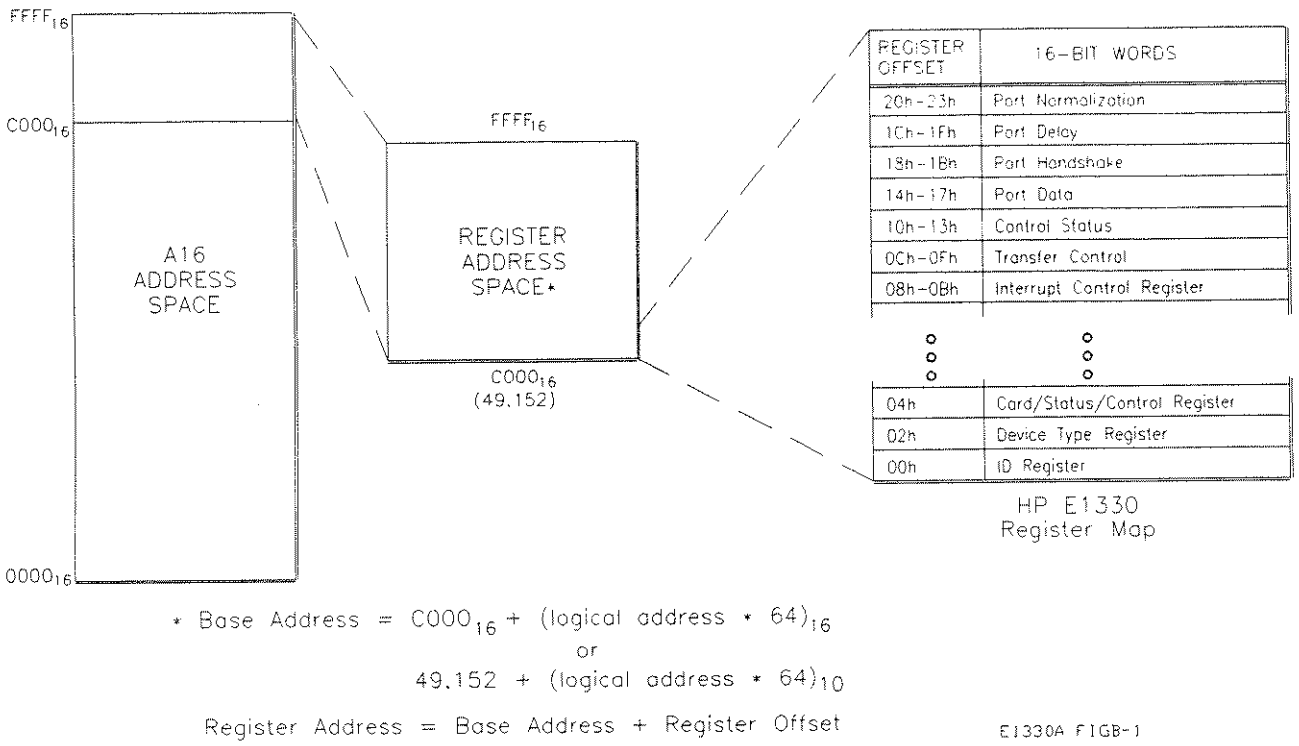


Figure B-1. Register Address Location Within A16

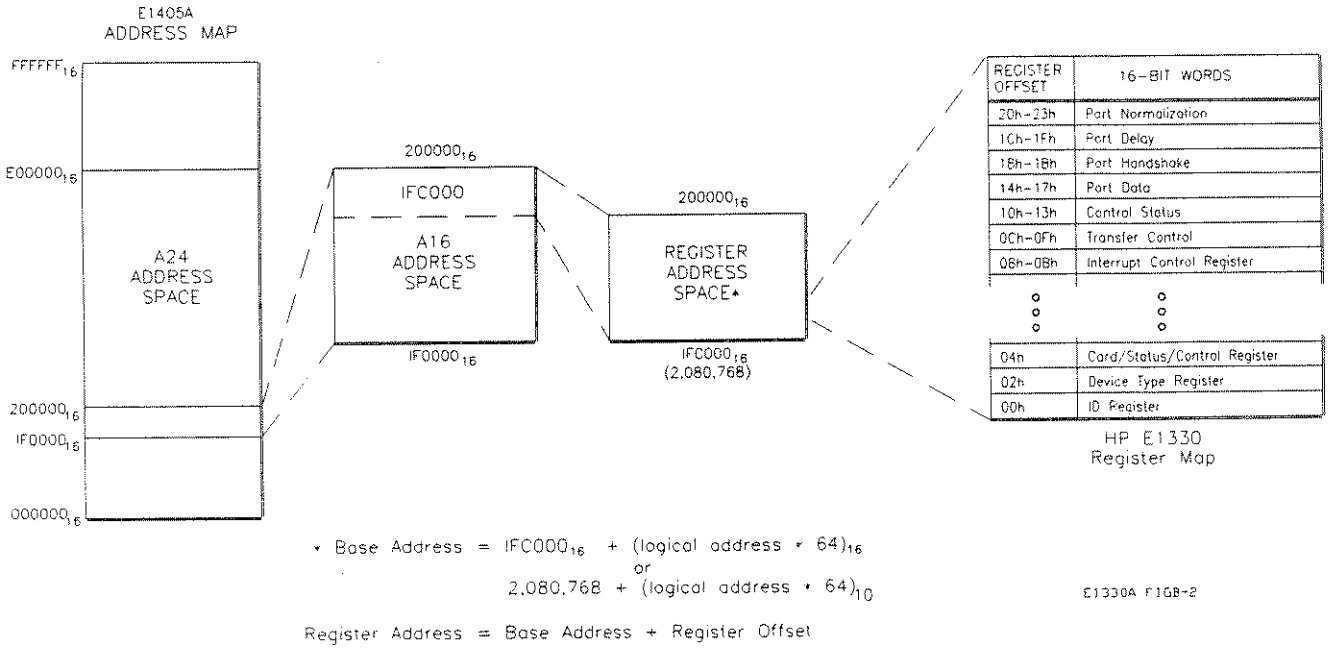


Figure B-2. A16 Address Space in the HP E1405A

The Base Address

When you are reading or writing to a module register, a hexadecimal or decimal register address is specified. This address consists of a base address plus a register offset. The base address used in register-based programming depends on whether the A16 address space is outside or inside the HP E1405 Command Module.

A16 Address Space Outside the Command Module

When the HP E1405 Command Module is not part of your VXIbus system (Figure B-1), the E1330's base address is computed as:

$A16_{base} + C000_h + (LADDR * 64)_h$

or (decimal)

$A16_{base} + 49{,}152 + (LADDR * 64)$

where C000h (49,152) is the starting location of the register addresses, LADDR is the module's logical address, and 64 is the number of address bytes per VXI device. For example, the E1330's factory set logical address is 144 (90h), therefore it will have a base address of:

$A16_{base} + C000h + (144 * 64)h = C000h + 2400h =$ **E400h**

or (decimal)

$A16_{base} + 49{,}152 + (144 * 64) = 49{,}152 + 9216 =$ **58368**

A16 Address Space Inside the Command Module or Mainframe

When the A16 address space is inside the HP E1405 Command Module (Figure B-2), the module's base address is computed as:

$1FC000h + (LADDR * 64)h$

or

$2{,}080{,}768 + (LADDR * 64)$

where 1FC000h (2,080,768) is the starting location of the VXI A16 addresses, LADDR is the module's logical address, and 64 is the number of address bytes per register-based device. Again, the E1330's factory set logical address is 144. If this address is not changed, the module will have a base address of:

$1FC000h + (144 * 64)h = 1FC000h + 2400h =$ **1FE400h**

or

$2{,}080{,}768 + (144 * 64) = 2{,}080{,}768 + 9216 =$ **2,089,984**

Register Offset

The register offset is the register's location in the block of 64 address bytes that belong to the module. For example, the module's Status/Control Register has an offset of 04h. When you write a command to this register, the offset is added to the base address to form the register address:

$E400h + 04h =$ **E404h** $\qquad$ $1FE400h + 04h =$ **1FE404h**

or

$58{,}368 + 4 =$ **58,372** $\qquad$ $2{,}089{,}984 + 4 =$ **2,089,988**

Table B-1 shows the general programming method for accessing the HP E1330 registers using different computers.

Computer	Programming Method	Base Address
E1300/E1301 IBASIC (Absolute Addressing) Select Code 8)	 READIO (-9826, Base_addr + offset) WRITEIO -9826, Base_addr + offset; data (positive select code = byte read or write negative select code = word read or write) READIO (8, Base_addr + reg number) WRITEIO 8, Base_addr + reg number; data	 Base_addr = $1fc000_{16}$ + (LADDR * 64)$_{16}$ or = 2,080,768 + (LADDR *64) offset = register number Base_addr = LADDR * 256 reg number = offset
External Computer (over HP-IB to E1300/E1301 Mainframe or E1405 Command Module)	VXI:READ? logical_addrress, offset VXI:WRITE logical_address, offset, data DIAG:PEEK? Base_addr + offset, width DIAG:POKE Base_addr +offset, width, data	Module Logical Address setting (LADDR) offset = register number Base_addr = $1FC000_{16}$ + (LADDR * 64)$_{16}$ or = 2,080,768 + (LADDR * 64) offset = register number
V/360 Embedded Computer (C-Size system)	READIO (-16, Base_addr + offset) WRITEIO -16, Base_addr + offset; data (positive select code = byte read or write negative select code = word read or write)	Base_addr = $C000_{16}$ + (LADDR * 64)$_{16}$ or = 49,152 + (LADDR * 64) offset = register number
LADDR = E1330 Logical Address = 144 (LADDR * 64)$_{16}$ = multiply quantity then convert to hexadecimal number (e.g. (80 * 64)$_{16}$ = (960)$_{16}$ = 1400_{16} When using DAIG:PEEK? and DIAG:POKE, the width must be either 8 or 16.		

Reset and Registers

When the Digital I/O Module undergoes a hardware reset (*RST in SCPI), the bits of the registers are put into the following states:

- The identification bytes at address 00 through 03, the Manufacturer ID and Device ID, remain unaffected.
- The I/$\overline{O}$ bits (bit 6 of the Port Control/Status Registers(0-3)) are set to "1", enabling all four ports for input.

All other bits of all registers are set to "0".

Register Definitions

You can program the HP E1330A Quad 8-bit Digital I/O Module using its hardware registers. *The procedures for reading or writing to a register depend on your operating system and programming language.* Whatever the access method, you will need to identify each register with its address. These addresses are given in Table B - 2.

Table B-2. Register Map

Register Name	Address			
Manufacturer ID (MSB)	00h			
Manufacturer ID (LSB)	01h			
Device ID (MSB)	02h			
Device ID (LSB)	03h			
Card /Status/Control (MSB)	04h			
Card/Status/Control (LSB)	05h			
	Address			
Register Name	**Port 0**	**Port 1**	**Port 2**	**Port 3**
Port Interrupt Control	08h	09h	0Ah	0Bh
Port Transfer Control	0Ch	0Dh	0Eh	0Fh
Port Control/Status	10h	11h	12h	13h
Port Data	14h	15h	16h	17h
Port Handshake	18h	19h	1Ah	1Bh
Port Delay	1Ch	1Dh	1Eh	1Fh
Port Normalization	20h	21h	22h	23h

The module is a register-based slave/interrupter device, supporting VME D16, D8(O), and D8(OE) transfers. The interrupt protocol supported is "release on Register access" – an interrupt is cleared only by servicing the cause of the interrupt (generally by reading or writing a byte of data). Interrupts are not cleared by a VXIbus interrupt acknowledge cycle.

Register Descriptions

The following pages detail register descriptions of the Digital I/O Module.

Manufacturer Identification Register

The Manufacturer Identification Register is a read-only register at address 00h (Most Significant Byte (MSB)) and 01h (Least Significant Byte (LSB)). Reading this register returns the Hewlett-Packard identification, FFFFh.

Device Identification Register

The Device Identification Register is a read-only register accessed at address 02h. Reading this register returns the Digital I/O Module identification of 50h for the E1330A or 51h for the E1330B. Reading address 03h always returns FFh.

Card Status/Control Register

The Card Status/Control Register is a read/write register accessed at address 04h and 05h. The following table shows the register bit patterns.

Address b+ 05h								Address b+04h							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	I3	I2	I1	I0	1	IEN	1	1	1	1	1	SR

SR(soft reset) Writing a "1" and then a "0" to this bit resets all Digital I/O Module components. SR disables all output ports (all ports become input ports) and sets all other registers to default values. Reads and writes to the other module registers will not transfer valid data when SR is asserted. This bit is cleared by a hard reset.

IEN: Main interrupt enable. Writing a 1 to this bit allows interrupts from port controller ICs to assert interrupt on the VXIbus. Writing a 0 masks these interrupts. This bit is cleared by a hard reset, but not by a soft reset.

Caution

A potential race condition exists when clearing this bit or masking interrupts by means of register 08h through 0Bh. If an interrupt occurs just before interrupts are masked, it could be asserted on the VXIbus but not acknowledged by the Digital I/O Module. Therefore, use care in disabling interrupts once they have been enabled.

I(0-3): Interrupt FLags for ports (0-3) (0 = interrupt). The MSB of this register is the module's interrupt response vector. It is asserted on the VXIbus during an interrupt acknowledge cycle.

Port Interrupt Control Register

The Port Interrupt Control Register is a read/write register and functions as the interrupt register for the port. This register shows the interrupt enable status, the level of interrupt that can signal the controller (always set to 0), and whether an interrupt is pending.

Port Address (0–3) b+08h,b+09h,b+0Ah,b+0Bh							
7	6	5	4	3	2	1	0
PIEN	IP	IL1	IL0	—	—	—	—

Bits (0-3): unused.

IL0 and **IL1** (Interrupt Level): Both bits *must* be left at 0 to initialize the Digital I/O Module for interrupt operation.

IP (Interrupt pending): when equal to "1", indicates an interrupt is pending. This is a read/write bit. You can force a hardware interrupt by setting this bit to "1" if PIEN is set to "1" and IEN is set to "1" in the Status/Control register.

PIEN (Port Interrupt enable) : when set to "1", enables interrupt. Pending or forced interrupts are ignored if set to "0".

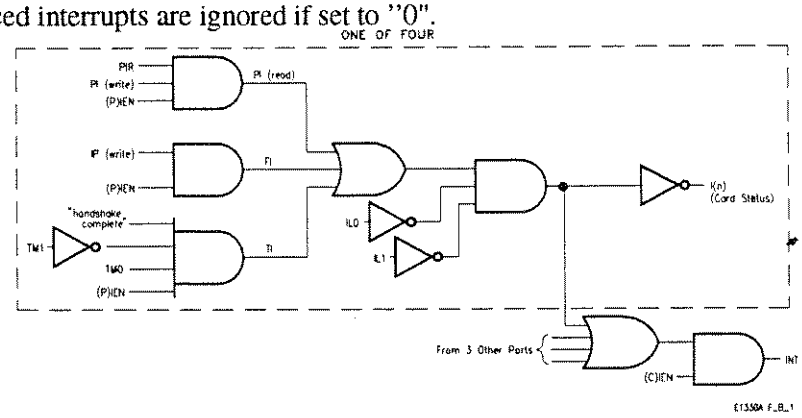


Figure B-3. Interrupt Line Logic Diagram

Port Transfer Control Register

The Port Transfer Control Register controls transfers between the mainframe and port, identifies port interrupts, and identifies forced interrupts from the controller.

Port Address (0–3)b+0Ch, b+0Dh, b+0Eh, b+0Fh							
7	6	5	4	3	2	1	0
PI	FI	TI	—	—	—	HE	DDR

DRR (Data Register Ready): a read-only bit. When set to "1" it indicates either that the Port Data Register contains valid data for the mainframe to read, or that the Port Data Register is ready for the mainframe to write a byte of data to it. When the Port Data Register is read, DRR is set to "0".

HE (Handshake Enable): when set to "1", enables handshaking for the port. You can read from or write to this bit. When the registers have been initialized, you can set this bit to "1" to enable handshaking if you are using the port handshake lines to transfer data.

Bits 2 - 4: not used.

TI (Transfer interrupt): this is a read-only bit. When set to "1", indicates a port transfer has occurred. A port transfer interrupt, if enabled, occurs on a "port data register ready" condition (when bit 0 of this register is set to "0"). To enable port transfer interrupts, specify the "interrupt driven" transfer mode of port (refer to Port Handshake Register) and set the "interrupt enable" bit (bit 7 of Interrupt Control Register) equal to "1". When the Port Data Register is read, TI is set to "0".

FI (Forced Interrupt): this is a read-only bit. When set to "1", indicates that a forced interrupt (from the mainframe) has occurred. To force an interrupt, write a "1" to bit 6 and bit 7 of the Port Interrupt Control Register and bit 6 of the status/control register.

PI (Peripheral Interrupt). Bit 7 is a read/write bit. Writing a "1" to bit 7 enables port peripheral interrupts. Writing a "0" disables port peripheral interrupts. When reading bit 7, a "1" indicates a port interrupt has occurred. When the Port Data Register is read, PI is set to "0".

Note

Port peripheral interrupts are caused by a transition in the PIR line. If bit 4 of the Port Normalization Register is "0", a rising-edge (low to high) transition caused the interrupt. If bit 4 is set to "1", a falling-edge (high to low) transition caused the interrupt. Refer to the Port Normalization Register for more information.

Port Control/Status Register

The Port Control/Status Register shows the status of STS, PIR, and FLG lines. It also directly controls the $\overline{RES}$, I/$\overline{O}$ and CTL lines.

Port Address (0–3) b+10h, b+11h, b+12h, b+13h							
7	6	5	4	3	2	1	0
CTL	I/$\overline{O}$	$\overline{RES}$	FLG	—	—	PIR	STS

STS: bit 0 is read-only bit. Read this bit to find the status of the STS line which is an input from the peripheral for the port. A "1" shows that the line is BUSY; a "0", shows that the line is READY.

PIR: bit 1 is a read-only bit. This bit shows the *normalized* state of the PIR line which is an input line from the peripheral:

- If positive-true logic is in use (bit 4 of the Port Normalization Register is equal to "0"), bit 1 is equal to 0 if the line is low; "1" if the line is high.
- If the PIR line is inverted (bit 4 of the Port Normalization Register is equal to "1"), bit 1 is equal to "0" if the line is high; "1" if the line is low.

If peripheral interrupts are not enabled, you can use the PIR line as a secondary status line. Just read bit 1 to monitor the state of the line.

If peripheral interrupts are enabled, you can still monitor the status of the PIR line by reading bit 1. However, the current status of the PIR line does not indicate whether a peripheral interrupt has occurred. Port peripheral interrupts are caused by transitions in the state of the PIR line. Read bit 7 of the Port Transfer Control Register to determine whether a port peripheral interrupt has occurred.

Bits 2 and 3: not used.

FLG: this is a read-only bit. Read this bit to find the *normalized* status of the FLG line. A "1" shows that the line is BUSY; a "0" shows that the line is READY. This bit shows the logical state (BUSY or READY) of the FLG line, regardless of the logic sense.

$\overline{\text{RES}}$: this is a read/write bit. Reading this bit shows the current state of the $\overline{\text{RES}}$ line which is an output line to the peripheral. A "1" shows that the line is high; a "0" shows that the line is low. Bit 5 is initially set to "0" by a hardware reset of the interface. This causes the $\overline{\text{RES}}$ line to go low, resetting the peripheral, if the peripheral implements the reset feature. You can control the logical state of the $\overline{\text{RES}}$ line by writing to this bit. Set bit 5 equal to "1" to change $\overline{\text{RES}}$ to the high state. The peripheral will then operate normally. To reset the peripheral, clear bit 5 to "0", putting $\overline{\text{RES}}$ in the low state.

I/$\overline{\text{O}}$: this is a read/write bit. Read this bit to find the current status of the I/$\overline{\text{O}}$ line, which is an output line to the peripheral, and the port data transceiver. If bit 6 is equal to "0", the line is FALSE and the transceiver is enabled for output. If bit 6 is equal to "1", the line is TRUE and the transceiver is enabled for input. *This bit is equal to "1" (input) after a hardware reset.* You can select input or output by changing this bit.

Note

If you are using the port handshake lines to control transfers, use the I/$\overline{\text{O}}$ line to control the direction of data transfer to your peripheral. Make sure that the peripheral is always enabled to send data during input transfers and to receive data during output transfers.

CTL: this is a read/write bit. Read this bit to find the current normalized state of the CTL line which is an output line to the peripheral. A

"1" shows the line is TRUE; a "0" shows the line is FALSE. When handshaking is enabled (bit 1 of the Port Transfer Control Register is set), the CTL line is controlled by the port controller. To prevent incorrect handshaking due to interaction with other lines, before enabling handshaking, set the control line to FALSE. If handshaking is not enabled and the handshake mode is set to NONE, you can control the logical state of the CTL line by writing to bit 7. This bit represents the logical state (TRUE or FALSE) of the CTL line, regardless of the logic sense.

Port Data Register

The Port Data Register is a read/write register. It is used for both output and input. Its operation depends on the state of the I/$\overline{O}$.

Port Address (0–3) b+14h, b+15h, b+16h, b+17h							
7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0

- If I/$\overline{O}$ is set for output (bit 6, Port Transfer Control Register = "0"), data written to the Port Data Register is latched and remains until new data is written. The current data in the Port Data Register drives the port data bus. If you read Port Data Register, the value read is the value last written to the register.

- If I/$\overline{O}$ is set for input (bit 6, Port Transfer Control Register ="1"), the data read from the Port Data Register is the data transmitted by the peripheral on the port data bus. If you write to the Port Data Register, the data is latched for output, but the data lines are not affected until I/$\overline{O}$ is again set for output.

- When the Port Data register is read the following bits are set to "0" on the Port Transfer Control Register: DRR (bit 0), TI (bit 5), and PI (bit 7).

Bits 0-7 of the Port Data Register correspond to data lines D(0-7) where bit 7 is the most significant bit.

Port Handshake Register

The Port Handshake Register determines the type of handshake protocol used for the port data transfers and how the data is transferred from the Digital I/O Module to the mainframe on the VXIbus.

Port Address (0–3) b+18h, b+19h, b+1Ah, b+1Bh							
7	6	5	4	3	2	1	0
HT2	HT1	HT0	EI	—	—	TM1	TM0

TM(0,1)(Transfer Mode): These bits control the transfer mode for the port between the Digital I/O module and the VXIbus as shown in Table B-3.

Table B-3. Transfer Mode

Transfer Mode	TM1 Bit 1	TM0 Bit 0
Flag Driven	0	0
Interrupt Driven	0	1
Fast Handshake	1	0

The three transfer modes are used to transfer data between the VXIbus and the Digital I/O Module:

- Flag Driven – the mainframe polls the Data Register Ready bit (bit 0, Port Transfer Control Register). When this bit is set, it reads data from the Port Data Register or writes data to the Port Data Register.
- Interrupt Driven – the peripheral sets bit 1 of the Port Status/Control Register and the Digital I/O Module interrupts the VXIbus for data transfer with the mainframe.
- Fast Handshake – the peripheral talks directly with the VXIbus's Data Acknowledge line to transfer data between the Port Data Registers and the VXIbus.

Bits 2 and 3: not used.

EI (Enable Inhibit): This bit, if set to "1", enables the STS line to inhibit a transfer cycle during a transfer. If bit 4 is set, the transfer is inhibited when the peripheral puts STS in the BUSY state and resumes when STS returns to the READY state.

HT(5-7)(Handshake Type): These bits determine the type of handshake for port input and output transfers as shown in Table B-4.

Table B-4. Handshake Type

Output/Input Transfer	Bit 7	Bit 6	Bit 5
No Handshake	0	0	0
Leading Edge	0	0	1
Trailing Edge	0	1	0
Pulse	0	1	1
Partial	1	0	0
Strobe	1	0	1

Port Delay Register

The Port Delay Register sets the delay time, T_d. Delay time is the time between data valid and setting the control (CTL) line TRUE. It is used with several handshake modes. You can also read this register to find the current delay time.

Port Address (0–3) b+1Ch, b+1Dh, b+1Eh, b +1Fh							
7	6	5	4	3	2	1	0
DF7	DF6	DF5	DF4	—	—	RM1	RM0

RM(0,1)(Range Multiplier): You can specify the range of delay time, T_d, by selecting the one of the range multipliers in Table B-5.

Table B-5. Range Multipliers.

Range Multiplier	RM1 Bit 1	RM0 Bit 0
1 ms	0	0
100us	0	1
10us	1	0
1us	1	1

Bits 2 and 3: not used.

DF(4-7)(Delay Factor): Regardless of the range multiplier you select, you can specify a delay factor in the range of 0 through 15 (decimal equivalent of the binary value) by setting these bits (0 specifies no delay time). For all output handshake types, the delay period T_d is equal to the range multiplier times the delay factor specified by bits 4 - 7. For example, if you write the value "00010000" to register 5, the multiplier is 1 ms and the delay factor is 1. If you write "11110010" to register 5, then the multiplier is 10 us and the delay factor is 15; hence, the delay factor is 150 us. The actual delay for a given transfer may be one count longer due to uncertainty in recognizing a transition of a handshake signal.

Note

If you are using the output strobe or pulse handshake, you can specify delay factors in the range 2 through 15, or you can specify 0 (no delay period). Thus, you can specify Td values from 2 to 15 us, from 20 to 150 us, and so forth for these handshakes.

If you are using the input strobe handshake, the delay factor specified by bits 4 through 7 is reduced by one, then multiplied by the range multiplier. For example, the register value "00100000" for an input strobe handshake specifies T_d = 1 ms. (The multiplier is 1 ms and the delay factor is 2-1= 1.) On the other hand, the value "11110010" specifies T_d = 140 µs. (The multiplier is 10 us and the delay factor is 15-1=14.)

The input strobe handshake is the only *input* handshake that uses a delay period. For the other input handshakes the value in this register has no effect.

Port Normalization Register

The Port Normalization Register allows you to normalize the port handshake and data lines to the correct logic sense for your peripheral. Positive true logic is the default. You can invert a line by setting the appropriate bit equal to "1".

Port Address (0–3) b+20h, b+21h, b+22h, b+23h							
7	6	5	4	3	2	1	0
ID	ICTL	IFLG	IPIR	—	—	—	—

Bits 0 - 3: not used.

IPIR(Invert PIR): This bit specifies the logic sense of a peripheral interrupt request. If bit 4 = "0", a rising-edge (low to high) transition of the PIR line triggers an interrupt. If bit 4 = "1", a falling-edge (high to low) transition of the PIR line triggers an interrupt. In either case, no interrupt occurs unless peripheral interrupts are enabled.

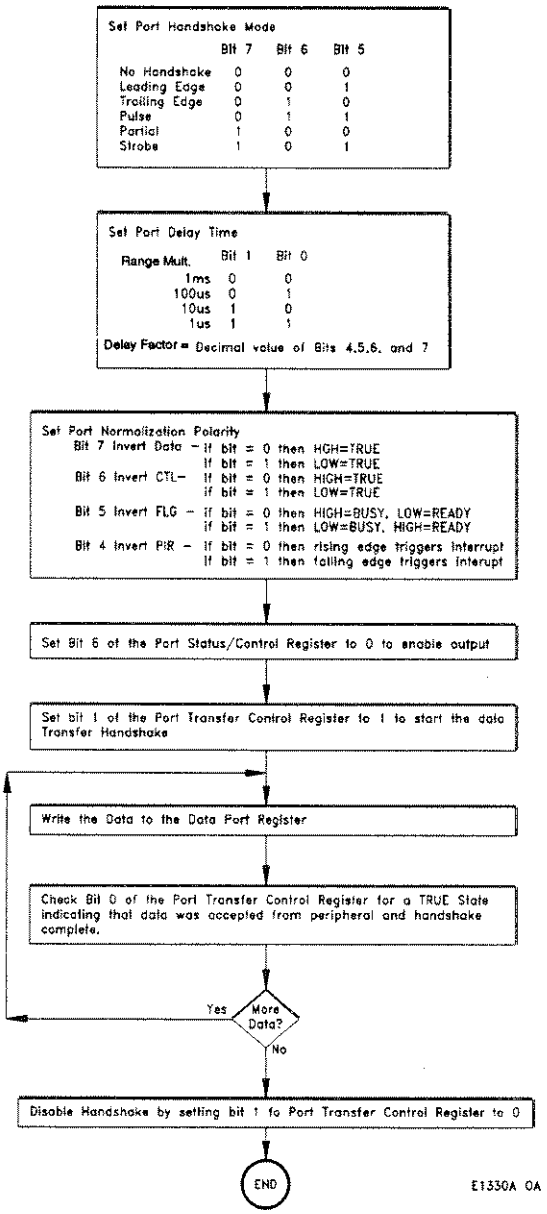
IFLG(Invert FLG): This bit specifies the logic sense of the FLG line. If bit 5 = "0", then positive-true logic is used: HIGH = BUSY, LOW = READY. IF bit 5 = "1", then negative-true logic is used: LOW = BUSY, HIGH = READY.

ICTL(Invert CTL): This bit specifies the logic sense of the CTL line. If bit 6 = "0", then positive-true logic is used: HIGH = TRUE, LOW = FALSE. IF bit 6 = "1", then negative-true logic is used: LOW = TRUE, HIGH = FALSE.

ID(Invert DATA): This bit specifies the logic sense of the port data lines. If bit 7 = "0", then positive-true logic is used: HIGH = TRUE, LOW = FALSE. If bit 7 = "1", then negative-true logic is used: LOW = TRUE, HIGH = FALSE.

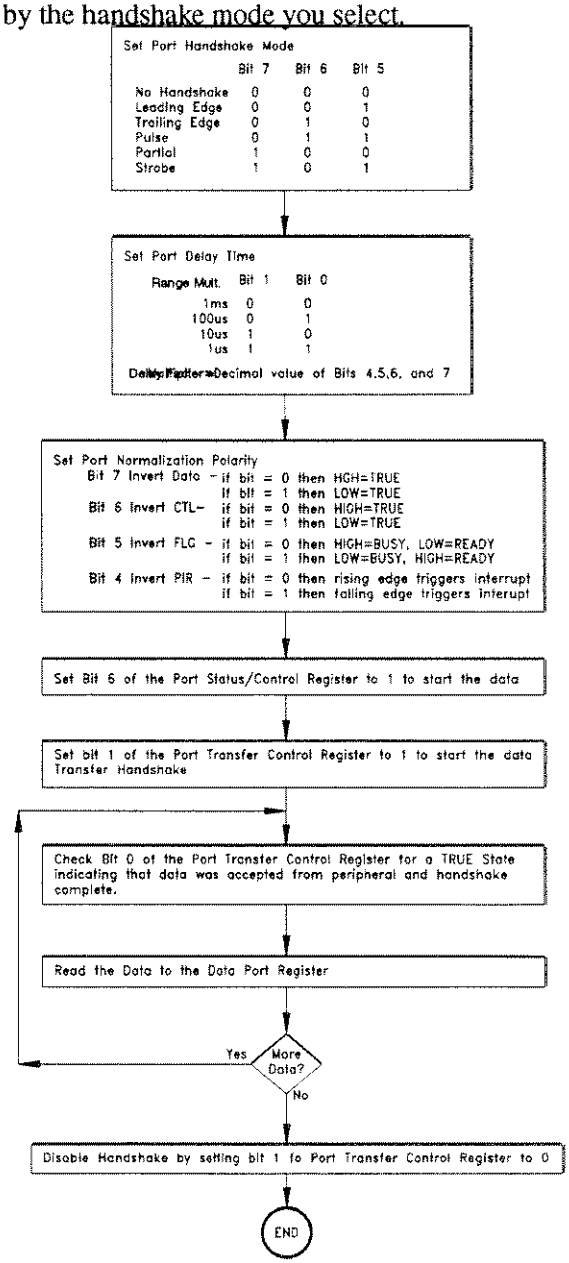
A Register-Based Output Algorithm

The following algorithm describes the procedure you would use to program the registers to transmit a byte of data to a peripheral. The algorithm follows a flag-driven output procedure initiated by the computer. The computer polls the Digital I/O Module to see if the data has been accepted by the peripheral by checking the Port Transfer/Control Register, bit 0 (referred to as the acknowledge flag - hence, the name of flag-driven). Once the flag is TRUE the computer can output new data to the port. The actual path followed by the peripheral and the Digital I/O Module to set this bit is controlled by the handshake mode you select.



A Register-Based Input Algorithm

The following algorithm describes the procedure you use to program the registers to read a byte of data from a peripheral. The algorithm follows a flag-driven input procedure initiated by the computer. The computer polls the Digital I/O Module to see if the data has been transmitted by the peripheral by checking the Port Transfer/Control Register, bit 0 (referred to as the acknowledge flag - hence, the name flag-driven). Once the flag is TRUE the computer can read new data from the port. The actual path followed by the peripheral and the Digital I/O Module to set this bit is controlled by the handshake mode you select.



Programming Examples

The examples in this section demonstrate how to program the module at the register level. The programs follow the execution and timing models covered in the previous section. The examples in this section include:

- Resetting the module
- Reading the ID, Device Type, and Status Registers
- Writing an 8-bit Byte
- Writing a 16-bit Word
- Reading an 8-bit Byte
- Reading a 16-bit word
- PIR Interrupts
- Controlling the FLG, CTL, STS, RES, and PIR lines

System Configuration

The following example programs were developed with the module at logical address 144. The HP BASIC/UX programs were developed using the HP E1300 mainframe Series B HP BASIC language. The C language programs were developed on an HP Vectra PC (IBM PC compatible) using Borland's Turbo C++® programming language.

Resetting the module

The following program resets the E1330 Digital I/O Module (Bit 6 of the Port Control/Status register set to "1" then to "0"). Reset enables all four ports for input, all other bits of other registers set to "0").

HP IBASIC Version

```
10 Base_addr = DVAL("1FE400",16)    !Logical Address 144
20 Reg_addr = 04   ! Offset for Status Control Register
30 ! write a 0 then a 1 to bit 0 of status register
40 WRITEIO 9826, Base_addr + Reg_addr; 1
50 WRITEIO 9826, Base_addr + Reg_addr; 0
60 END
```

C Version

```
#include <stdio.h>
#include <chpib.h>

#define LOG_ADDR 144
#define BASE_ADDR (long) ((0x1FC000) + (64 * LOG_ADDR))

main ()
{
        int     reg_addr;
        float   send_data[3];
        char    state[2] = {13,10};
        send_data[0] = BASE_ADDR + reg_addr;
        send_data[1] = 16;
        send_data[2] = 1;

        IOEOI (7L, 0);  IOEOL (7L, " ",0);
        IOOUTPUTS (70900L, "DIAG:POKE ",10);

        IOEOI (7L, 1);  IOEOL (7L, " state,0);
        IOOUTPUTA (70900L, send_data, 3);

        send_data[2] = 0;

        IOEOI (7L, 0);  IOEOL (7L, " ",0);
        IOOUTPUTS (70900L, "DIAG:POKE ",10);

        IOEOI (7L, 1);  IOEOL (7L, " state,0);
        IOOUTPUTA (70900L, send_data, 3);

return 0;
}
```

Reading the ID, Device Type, and Status Registers

The following examples read the module ID, DEVICE ID, and STATUS registers from the module.

HP IBASIC Version

```
10 !*************************************************************
20 !*****                    READREG                        *****
30 !*************************************************************
40 ! OPTION BASE 0 is default
50 ! Set up arrays to store register names and addresses
60 DIM Reg_name$(0:2)[32], Reg_addr(0:2)
70 !
80 ! Read register names and addresses into the arrays
90 READ Reg_name$(*)
100 READ Reg_addr(*)
110 !
120 ! Set base Address variable
130 Base_addr = DVAL ("1FE400",16)
140 !
150 !Map the A16 address space
160 !
170 !CONTROL 16,25;2  ! used only with V360 Controller
180 ! Call the subprogram Read_regs
190 Read_regs(Base_addr, Reg_name$(*),Reg_addr(*))
200 !
210 DATA Identification Register, Device Register, Status Register
220 DATA 00, 02, 04
230 END
300 ! This subprogram steps through a loop that reads each register
and prints its contents
310 !
320 SUB Read_regs(Base_addr, Reg_name$(*),Reg_addr(*))
330!
340 FOR Number = 0 to 2
350 Register = READIO(-9826,Base_addr + Reg_addr(number))
360 PRINT Reg_name$(number);" = "; IVAL$(Register, 16)
370 NEXT Number
380 SUBEND
```

This program returns:

```
Identification Register = FFFF
Device Register = FF50
Status Register = (dependent on current status, default is FFBE)
```

C Version

```
#include <stdio.h>
#include <chpib.h>
#include <cfunc.h>

#define LOG_ADDR 144
#define BASE_ADDR (long) ((0x1FC000) + (64 * LOG_ADDR))
main()
{  int  reg_addr;
   float   send_data[3], read;
   char state[2] = {13,10};

   send_data[1] = 16;
   send_data[2] = 0;
   send_data[0] = BASE_ADDR + 0;

   IOEOI (7L, 0); IOEOL (7L, " ", 0);
   IOOUTPUTS (70900L, "DIAG:PEEK? ", 11);

   IOEOI (7L, 1); IOEOL (7L, state, 2);
   IOOUTPUTA (70900L, send_data, 2);

   IOENTER(70900L, &read);
   printf("/nIdentification Register = %0x",read);

   send_data[0] = BASE_ADDR + 2;

   IOEOI (7L, 0); IOEOL (7L, " ", 0);
   IOOUTPUTS (70900L, "DIAG:PEEK? ", 11);
   IOEOI (7L, 1); IOEOL (7L, state, 2);
   IOOUTPUTA (70900L, send_data, 2);

   IOENTER(70900L, &read);
   printf("/nDevice Register = %0x",read);

   send_data[0] = BASE_ADDR + 4;

   IOEOI (7L, 0); IOEOL (7L, " ", 0);
   IOOUTPUTS (70900L, "DIAG:PEEK? ", 11);
   IOEOI (7L, 1); IOEOL (7L, state, 2);
   IOOUTPUTA (70900L, send_data, 2);

   IOENTER(70900L, &read);
   printf("/nStatus Register = %0x",read);
return o;
}
```

Writing an 8-bit Byte

Using the output algorithm described earlier, the following programs describe how to output an 8-bit byte to your peripheral device. The program use a leading edge handshake and flag-driven data transfer to send data (decimal value 255) from Port 1.

HP IBASIC Vesion

```
10 Base_addr = DVAL("1FE400",16)
    ! Logical Address 144
20  WRITEIO  9826,Base_Addr+DVAL("19",16);32
    ! Sets Port 1 Handshake Register to leading edge handshake and flag
     driven transfer
30  WRITEIO   9826,Base_Addr+DVAL("1D",16);00
    ! Sets Port 1 Delay  Register to 0
40  WRITEIO  9826 ,Base_Addr+DVAL("21",16);00
    ! Sets Port 1 Normalization Register (polarity) to positive-true (High =
true)
50  WRITEIO   9826,Base_Addr+DVAL("11",16);0
    ! Sets Port 1 Status Control bit 6 to enable output
60  WRITEIO   9826,Base_Addr+DVAL("0D",16);2
    ! Sets Port 1 Transfer Control Register bit 1 to Enable Handshake
70  WRITEIO  9826 ,Base_Addr+DVAL("15",16);255
    ! Sets Port 1 Data Register to the value to output
80 REPEAT
90          UNTIL BIT(READIO (  9826,Base_addr+DVAL("0D",16)),1)
100 ! If more data to send, repeat lines 70 - 90
110 WRITEIO   9826,Base_Addr+DVAL("0D",16);0
    ! Clears Port 1 Transfer Control Register bit 1 to Disable Handshake
120 END
```

C Version

```
/* writing an 8-bit byte */
 #include <stdio.h>
#include <chpib.h>
#define LOG_ADDR 144
#define BASE_ADDR (long) ((0x1FC000) + (64 * LOG_ADDR))
void send_info(char state[], float send_data[]);
main ()
{
        float    send_data[3], read;
        char     state[2] = {13,10};
        int      handshak_reg, delay_reg, normiz_reg,
                 statuscont_reg, transfercont_reg, data_reg;
        handshak_reg = 0x19;
        delay_reg = 0x1D;
        normiz_reg = 0x21;
        statuscont_reg = 0x11;
        transfercont_reg = 0x0D;
```

```
                data_reg = 0x15;
                send_data[1] = 16;

                send_data[0] = BASE_ADDR + handshak_reg;
                send_data[2] = 32;
                send_info(state, send_data);

                send_data[0] = BASE_ADDR + delay_reg;
                send_data[2] = 00;
                send_info(state, send_data);

                send_data[0] = BASE_ADDR + normiz_reg;
                send_data[2] = 00;
                send_info(state, send_data);

                send_data[0] = BASE_ADDR + statuscont_reg;
                send_data[2] = 00;
                send_info(state, send_data);

                send_data[0] = BASE_ADDR + transfercont_reg;
                send_data[2] = 2;
                send_info(state, send_data);

                send_data[0] = BASE_ADDR + data_reg;
                send_data[2] = 255;
                send_info(state, send_data);

return 0;
 }
void send_info(char state[], float send_data[])
{
                IOEOI (7L, 0);IOEOL (7L, " ", 0);
                IOOUTPUTS (70900L, "DIAG:POKE ", 10);
                IOEOI (7L, 1);IOEOL (7L, state, 0);
                IOOUTPUTA (70900L, send_data, 3);
}
```

Writing a 16-bit Word

Similar to the last program example, this progam outputs a 16-bit word to your peripheral device. To write a 16-bit word, two consecutive ports are required (i.e. ports 0 and 1, 1 and 2, 2 and 3, or 3 and 4). Both ports must be configured exactly the same. Configure consecutive port registers by addressing the lower port's register ands sending a 16-bit word. Handshaking is accomplished using the lower port's handshake lines.

HP BASIC Vesion

```
10 Base_addr = DVAL("1FE400",16)
    ! Logical Address 144
20  WRITEIO -9826,Base_Addr+DVAL("19",16);DVAL("3232",16)
    ! Sets Ports 0 & 1 Handshake Register to leading edge handshake and flag
     driven transfer
30  WRITEIO -9826,Base_Addr+DVAL("1D",16);DVAL("0000",16
    ! Sets Ports 0 & 1 Delay  Register to 0
40  WRITEIO -9826 ,Base_Addr+DVAL("21",16);DVAL("0000",16)
    ! Sets Ports 0 & 1 Normalization Register (polarity) to positive-true (High
= true)
50  WRITEIO  -826,Base_Addr+DVAL("11",16);DVAL("0000",16)
    ! Sets Ports 0 & 1 Status Control bit 6 to enable output
60  WRITEIO -9826,Base_Addr+DVAL("0D",16);DVAL("0202",16)
    ! Sets Ports 0 & 1 Transfer Control Register bit 1 to Enable Handshake
70  WRITEIO -9826 ,Base_Addr+DVAL("15",16);512
    ! Sets Ports 0 & 1 Data Register to the value to output
80 REPEAT
90          UNTIL BIT(READIO ( 9826,Base_addr+DVAL("0D",16)),1)
100 ! If more data to send, repeat lines 70 - 90
110 WRITEIO   9826,Base_Addr+DVAL("0D",16);DVAL("0000",16)
    ! Clears Ports 0 & 1 Transfer Control Register bit 1 to Disable Handshake
120 END
```

C Version

The C program is similar to that shown for writing an 8-bit byte except the data sent to the registers must be 16 bits.

Reading an 8-bit Byte

Using the input algorithm described earlier, the following programs describe how to input an 8-bit byte from your peripheral device. The program use a leading edge handshake and flag-driven data transfer.

HP BASIC Vesion

```
10 Base_addr = DVAL("1FE400",16)
    ! Logical Address 144
20  WRITEIO  9826,Base_Addr+DVAL("19",16);32
    ! Set Port 1 Handshake Register to leading edge handshake and flag
driven transfer
30  WRITEIO  9826,Base_Addr+DVAL("1D",16);00
    ! Set Port 1 Delay  Register to 0
40  WRITEIO  9826,Base_Addr+DVAL("21",16);00
    ! Set Port 1 Normalization Register (polarity) to positive-true (High = true)
50  WRITEIO  9826,Base_Addr+DVAL("11",16);64
```

```
    ! Set Port 1 Status Control bit 6 to enable output
60  WRITEIO  9826,Base_Addr+DVAL("0D",16);2
    ! Set Port 1 Transfer Control Register bit 1 to Enable Handshake
70  A = READIO (9826,Base_addr+DVAL("15",16))
80 Print A
90 ! If more data to send, repeat lines 70 - 80
100 WRITEIO  9826,Base_Addr+DVAL("0D",16);0
    ! Clear Port 1 Transfer Control Register bit 1 to Disable Handshake
110 END
```

C Version

```
/* reading an 8-bit byte */
#include <stdio.h>
#include <chpib.h>
#define LOG_ADDR 144
#define BASE_ADDR (long) ((0x1FC000) + (64 * LOG_ADDR))
void send_info(char state[], float send_data[]);

main ()
{
        float   send_data[3], read;
        char    state[2] = {13,10};

        int     handshak_reg, delay_reg, normiz_reg,
                statuscont_reg, transfercont_reg, data_reg;
        handshak_reg = 0x19;
        delay_reg = 0x1D;
        normiz_reg = 0x21;
        statuscont_reg = 0x11;
        transfercont_reg = 0x0D;
        data_reg = 0x15;
        send_data[1] = 16;

        send_data[0] = BASE_ADDR + handshak_reg;
        send_data[2] = 32;
        send_info(state, send_data);

        send_data[0] = BASE_ADDR + delay_reg;
        send_data[2] = 00;
        send_info(state, send_data);

        send_data[0] = BASE_ADDR + normiz_reg;
        send_data[2] = 00;
        send_info(state, send_data);

        send_data[0] = BASE_ADDR + statuscont_reg;
```

```
                send_data[2] = 00;
                send_info(state, send_data);

                send_data[0] = BASE_ADDR + transfercont_reg;
                send_data[2] = 2;
                send_info(state, send_data);

                send_data[0] = BASE_ADDR + data_reg;
                IOEOI (7L, 0); IOEOL (7L, " ",0);
                IOOUTPUTS (70900L, "DIAG:PEEK? ", 11);
                IOEOI (7L, 1);  IOEOL (7L, state, 2);
                IOOUTPUTA (70900L, send_data, 2);
                IOENTER (70900L, &read);
                printf("\nData read from module = %X",(int)read);

                send_data[0] = BASE_ADDR + transfercont_reg;
                send_data[2] = 0;
                send_info(state, send_data);

return 0;
}
void send_info(char state[], float send_data[])
{
                IOEOI (7L, 0);IOEOL (7L, " ", 0);
                IOOUTPUTS (70900L, "DIAG:POKE ", 10);
                IOEOI (7L, 1);IOEOL (7L, state, 0);
                IOOUTPUTA (70900L, send_data, 3);
}
```

Reading a 16-bit Word

To read a 16-bit word, two consecutive ports are required (i.e. ports 0 and 1, 1 and 2, 2 and 3, or 3 and 4). Both ports must be configured exactly the same. configuring consecutive port registers by addressing the lower port's register ands sending a 16-bit word. Handshaking is accomplished using the lower port's handshake lines.

PIR Interrupts on the HP E1330

The HP E1330 has four lines, PIR0 - PIR3, which are connected to the mainframe interrupt circuitry. No SCPI commands can respond to these lines. To access these lines you must use register based programming. Registers are used to enable an interrupt and program a high-to-low transition or low-to-high transition causing the interrupt. The following program demonstrates how to access this interrupt capability.

```
10    ! Digital I/O E1330A Interrupt/SRQ on PIR
20    ! Program to cause Series B to SRQ computer on receipt of a
30    ! Peripheral Interrupt Request (PIR0-PIR3) on the E1330A.
40    ! The interrupt routine determines which PIR caused the intr.
50    ! Meanwhile, digital reads and writes are occuring through
```

```
60    ! the E1330A's four data ports.
70    ! CONDITIONS UNDER WHICH THIS WILL WORK:
80    ! E1330A interrupt jumper on card is set to 2
90    ! NO HANDSHAKING for digital reads and writes, bytes only
100   ! Series B firmware must be Revision A.03.00
110   ! VITAL NOTE: Series B system instrument must NOT be talked
120   ! to after interrupts have been enabled.
130   !
140   !
150   ! initialize variables
160   !
170    COM @Sys,Ladd
180    DIM Err$[100]
190    Port=0
200    Step=1
210    Ladd=64              !dio logical address
220    Syst=70900           !system hpib address
230    ASSIGN @Sys TO Syst
240    ASSIGN @Dio TO Syst+Ladd/8
250    ON KEY 1 LABEL "STOP" GOTO Done
260   !
270   ! initial set-up for interrupt and SRQ
280   !
290    OUTPUT @Sys;"*RST"
300    WAIT .1
310    ON INTR 7,2 CALL Intr_req
320    OUTPUT @Sys;"*SRE 16"
330    OUTPUT @Sys;"DIAG:INT:SETUP2 ON"
340    CALL Setup_intr
350    ENABLE INTR 7;2
360   !
370   ! on-going digital read/writes. Every port alternating read,write.
380   ! system instrument must NOT be talked to during this.
390   !
400    LOOP
410       REPEAT
420       DISP Port
430       OUTPUT @Dio;"MEAS:DIG:DATA"&VAL$ (Port)&"?"
440       ENTER @Dio;X
450       OUTPUT @Dio;"SOUR:DIG:DATA"&VAL$(Port+Step)&" 0"
460       Port=Port+2*Step
470    UNTIL Port<0 OR Port>3
480    Step=-1*Step
490    Port=Port+Step
500 END LOOP
```

```
510 !
520 ! Executed if softkey 1 has been pushed. Disables all interrupts.
530 ! Empties error buffer if any have accumulated.
540 !
550 Done: !
560 OFF INTR 7
570 CLEAR 7
580 OUTPUT @Sys;"DIAG:INT:SETUP2 OFF"
590 OUTPUT @Dio;"*RST"
600 REPEAT
610 OUTPUT @Sys;"SYST:ERR?"
620 ENTER @Sys;Err$
630 PRINT Err$
640 UNTIL NOT VAL(Err$)
650 END
660 !
670 ! Following sub enables E1330A's interrupts, and enables system
680 ! instrument to pass on interrupt to computer through SRQ.
690 !
700 SUB Setup_intr
710 COM @Sys,Ladd
720     Vxi$="VXI:WRITE "&VAL$(Ladd)&","
730     OUTPUT @Sys;Vxi$&"4,-1"          ! all 16 bits set = 1;stat/cntl
reg
740     OUTPUT @Sys;Vxi$&"4,-2"          ! all except bit 0 set= 1
750     !   The above does a soft reset.
760     !
770     OUTPUT @Sys;Vxi$&"8, -31869"     ! bits 0,1,7,8,9,15 set=1;
intr/cntl
780     OUTPUT @Sys;Vxi$&"10,-31869"     ! same for ports 2 @ 3
790     !                    Sets PIEN (port intr enable)
800     !
810     OUTPUT @Sys;Vxi$&"12,_32640"     ! bits 7 & 15 set; Xfer Cntl
reg
820     OUTPUT @Sys;Vxi$&"14,-32640"     ! same for ports 2 & 3
830     !                Sets PI enabling peripheral interrupt line
840     !
850     OUTPUT @Sys;"DIAG:INT:WAIT?"
860         ! once this statement is set, @Sys must NOT be talked to!
870     SUBEND
880     !
890     ! Following sub services an SRQ, DETERMINES pir LINE THAT
INTERRUPTED,,
900     ! and re-enables the interrupt flow.
910     !
920     SUB Intr_req
930         COM @Sys,Ladd
```

```
940    BEEP
950    ENTER @Sys;A$
960    A=SPOLL(@Sys)
970    OUTPUT @Sys;"VXI:READ?  "&VAL$(Ladd)&",4"
980    ENTER @Sys;Intr_vector
990    FOR I=0 TO 3
1000       IF NOT BIT(Intr_vector,8+I) THEN PRINT "PIR";I
1010   NEXT I
1020   CALL Setup_intr
1030   ENABLE INTR 7;2
1040 SUBEND
```

HP E1330 Non-data Line I/O

The HP E1330 has several signal lines other than the data lines which can be individually controlled. These lines are the FLG, CTL, STS, RES, and PIR lines. The following BASIC language program demonstrates how to control these lines.

FLG0 - FLG3 are input lines Input from the peripheral to the HP E1330 module) that can be used as individual input lines when not used as handshake lines. The subroutine Ctl_flg_io demonstrates using SCPI commands to control these lines.

CTL0 - CTL3 are output lines (output from the HP E1330 module to your peripheral) which can be controlled individually when not used as handshaking lines. Subroutine Ctl_flg_io demonstrates driving these lines using SCPI programming commands.

STS0 - STS3 are input lines that can be controlled using register based programming. Subroutine Res_sts_io demonstrates using register based programming of these lines.

RES0 - RES3 are output lines that can be controlled using register based programming. Subroutines Res_sts_io, Res_pir_io, and Res_pi_io demonstrate register programming.

PIR0 - PIR3 are input lines. Subroutine Res_pir_io demonstrates directly reading these input lines. Subroutine Res_pi_io demonstrates reading a latched version of these inputs.

```
10 ! re-save "DIG_NDL"
20 ! PROGRAM TO DEMONSTRATE THE NON-DATA
LINES—FLAG/CONTROL,RES/STS, PIR
30 ! This main line code is reserved as a error handling shell
40 ! All application code must be at lower level context
50 ASSIGN @Sys TO 70900                    ! define I/O paths
60 ASSIGN @Dvm TO 70903
70 ASSIGN @Dig TO 70910
```

```
80 COM @Sys,@Dvm,@Dig '
90 ON TIMEOUT 7,3 GOTO End          !Turn TIMEOUTS to errors—this branch never taken
100 ON ERROR RECOVER Kaboom !This handles timeouts and errors not handled
110   ! at lower level contexts
120   Main                          ! Put application code in this sub
121   PRINT ""
130   E13xx_errors
140   GOTO End
150 Kaboom:PRINT ""
160   PRINT ERRM$
170   PRINT "HERE IS THE E13XX ERROR STATUS"
180   El3xx_errors
190 End:END
200   !
210   SUB E13xx_errors              !This sub reads all errors from E13xx instruments
220     COM @Sys,@Dvm,@Dig
230     DIM A$[128]
240     ABORT 7
250     CLEAR @DVM
260     REPEAT
270     OUTPUT @Dvm;"SYST:ERR?"
280     ENTER @Dvm;A,A$
290     PRINT "DVM ERROR ";A$
300     UNTIL A=0
310   !
320     CLEAR @Sys
330     REPEAT
340     OUTPUT @Sys;"SYST:ERR?"
350     ENTER @Sys;A,A$
360     PRINT "SYSTEM ERROR ";A$
370     UNTIL A=0
380   !
390     CLEAR @Dig
400     REPEAT
410     OUTPUT @Dig;"SYST:ERR?"
420     ENTER @Dig;A,A$
430     PRINT "DIG I/O ERROR ";A$
440     UNTIL A=0
450   SUBEND
460   !
470   SUB Main                      !This subroutine is treated as the main line
```

```
480       COM @Sys,@Dvm,@Dig
490    Cnt_flg_io                    ! DEMONSTRATE DRIVING
                                     CONTROL0, RECEIVING
                                     FLAG0
500    Res_sts_io                    ! DEMONSTRATE
                                     DRIVING RES0, RECEIVING
                                     STS0
510    Res_pir_io                    ! DEMONSTRATE
                                     DRIVING RES0, RECEIVING
                                     PIR0
520    Res_pi_io                     ! DEMONSTRATE
                                     DRIVING RES0, RECEIVING
                                     PI0
530    !Put Application code here
540    SUBEND
550    !
560    SUB Cnt_flg_io                ! DEMONSTRATES DRIVING
                                     CONTROL0 THEN
                                     RECEIVING Flag 0
570     COM @Sys,@Dvm,@Big           !CONNECT CONTROL 0 TO
                                     FLAG 0
580       PRINT ""
590       PRINT "SUBPROGRAM Cnt_flg_io"
600       OUTPUT @Dig;"*RST"         ! RESET TO POWER ON
                                     STATE
610       OUTPUT @Dig;"SOUR:DIG:CONT0:VAL 0" ! DRIVE
                                     CONTROL 0 TO 0
620       OUTPUT @Dig;"MEAS:DIG:FLAG0?" ! READ FLAG 0
630       ENTER @Dig;A
640       PRINT "CONTROL0 DRIVEN TO 0 AND FLAG0 RECEIVED
AS ";A
650       OUTPUT @Dig;"SOUR:DIG:CONT0:VAL 1"
                                              ! DRIVE
                                     CONTROL 0 TO 1
660       OUTPUT @Dig;"MEAS:DIG:FLAG0?"
                                              ! READ FLAG 0
670ENTER @Dig;A
680       PRINT "CONTROL0 DRIVEN TO A 1 AND FLAG0
RECEIVED AS A " ;A
690       PRINT " "
700    SUBEND
710    SUB Res_sts_io                ! DEMONSTRATES DRIVING
                                     RES0 THEN RECEIVING STS0
720      ! CONNECT RES0 TO STS0
730      !USE REGISTER PROGRAMMING TO USE RES0 & STS0
740       COM @Sys,@Dvm,@Dig
750       PRINT ""
760       PRINT "SUBPROGRAM Res_sts_io"
770    OUTPUT @Dig;"*RST"            ! RESET TO POWER ON
                                     STATE
780    Ladd=80
```

```
790   !  Base=Start of A16+Offset to VXI Reg+Offset to card Reg
800   Base=2031616+49152+(Ladd*64)
810   OUTPUT @Sys;"DIAG:POKE "&VAL$(Base+(DVAL("10",16)))
&",8,64 "   ! DRIVE RES0 TO 0
820   OUTPUT @Sys;"DIAG:PEEK? "&VAL$(Base+(DVAL("10",16)))
&",8"    !READ REG B+10H
830   ENTER @Sys;A
840   Bit0=BIT(A,0)
850   PRINT "RES0 DRIVEN TO 0, STS0 RECEIVED AS ";Bit0
860   OUTPUT @Sys;"DIAG:POKE "&VAL$(Base+(DVAL("10",16)))
&",8,96"   ! DRIVE RES0 TO 1
870   OUTPUT @Sys;"DIAG;PEEK? "&VAL$(Base+(DVAL("10",16)))
&",8"    !READ REG B+10H
880   ENTER @Sys;A
890   Bit0=BIT(A,0)
900   PRINT "RES0 DRIVEN TO 1, STS0 RECEIVED AS ";Bit0
910   SUBEND
920   SUB Res_pir_io                    ! DEMONSTRATES DRIVING
                                        RES0 THEN RECEIVING PIR0

930          !  CONNECT RES0 TO PIR0
940   !USE REGISTER  PROGRAMMING TO USE RES0  &   PIR0
950   COM @Sys,@Dvm,@Dig
960   PRINT ""
970   PRINT "SUBPROGRAM Res_pir_io"
980   OUTPUT @Dig;"*RST"                ! RESET TO POWER ON
                                        STATE
990   Ladd=80
1000  ! Base=Start of A16+Offset to VXI Reg+Offset to card Reg
1010   Base=2031616+49152+(Ladd*64)
1020   OUTPUT @Sys;"DIAG:POKE "&VAL$(Base+(DVAL("10",16)))
&",8,64"    !DRIVE RES0 TO
1030 OUTPUT @Sys;"DIAG:PEEK? "&VAL$(Base+(DVAL("10",16)))
&",8"    !READ REG B+10H
1040   ENTER @Sys;A
1050   Bit1=BIT(A,1)
1060   PRINT "RES0 DRIVEN TO 0, PIR0 RECEIVED AS ";Bit1
1070   OUTPUT @Sys;"DIAG:POKE "&VAL$(Base+(DVAL("10",16)))
&",8,96"    ! DRIVE RES0 TO 1
1080 OUTPUT @Sys;"DIAG:PEEK? "@VAL$(Base+(DVAL("10",16)))
&",8"    !READ REG B+10H
1090      ENTER @Sys;A
1100      Bit1=BIT(A,1)
1110      PRINT "RES0 DRIVEN TO 1, PIR0 RECEIVED AS ";Bit1
1120   SUBEND
1130   !
1140 SUB  Res_pi_io                   ! DEMONSTRATES DRIVING
                                      RES0 THEN RECEIVING PI
                                      WHICH IS

1141      ! LATCHED PIR0
```

```
1150    ! CONNECT RES0 TO PIR0
1170    !USE REGISTER PROGRAMMING TO USE RES0 & PIR0
1180     COM @Sys,@Dvm,@Dig
1190     PRINT ""
1200     PRINT "SUBPROGRAM Res_pi_io"
1210     OUTPUT @Dig;"*RST"          ! RESET TO POWER ON
                                     STATE
1220 Ladd=80
1230 ! Base=Start of A16+Offset to VXI Reg+Offset to card Reg
1240 Base=2031616+49152+(Ladd*64)
1270 OUTPUT @Sys;"DIAG:POKE "&VAL$(Base+(DVAL("08',16)))
&",8,131"    !SET PIEN=1
1280 OUTPUT @Sys;"DIAG:POKE "&VAL$(Base+(DVAL("0C",16)))
&",8,128"    !SET PI=1
1290 OUTPUT @Sys;"DIAG:POKE "&VAL$(Base+(DVAL("10",16)))
&",8,64"     !DRIVE RES0 TO 0
1300 OUTPUT @Sys;"DIAG;PEEK? "&VAL$(Base+(DVAL("10",16)))
&",8"     ! READ REG B+10H
1310     ENTER @Sys;A
1320     Bit1=BIT(A,1)
1330     PRINT "RES0 DRIVEN TO 0, PIR0 RECEIVED AS ";Bit1
1340     OUTPUT @Sys;"DIAG:PEEK? "&VAL$(Base+(DVAL("0C",16)))
&",8"     !READ PI
1350     ENTER @Sys;A
1370     Bit7=BIT(A,7)
1380     PRINT "PERIPHERAL INTERRUPT = ";Bit7
1390     OUTPUT @Sys;"DIAG:POKE "@VAL$(Base+(DVAL("10",16)))
&",8,96"     !DRIVE RES0 TO 1
1400     OUTPUT @Sys;"DIAG:PEEK? "&VAL$(Base+(DVAL("10",16)))
&",8"     !READ REG B+10H
1410     ENTER @Sys;A
1420     Bit1=BIT(A,1)
1430     PRINT "RES0 DRIVEN TO 1, PIR0 RECEIVED AS ";Bit1
1440     OUTPUT @Sys;"DIAG:PEEK? "&VAL$(Base+(DVAL("0C",16)))
&",8"     !READ PI
1450     ENTER @Sys;A
1470     Bit7=BIT(A,7)
1480     PRINT "PERIPHERAL INTERRUPT = ";BIT7
1490 SUBEND
```

Quad 8-bit Digital I/O Module Error Messages

Table C-1. HP E1330 Quad 8-bit Digital I/O Module Messages

Code	Message	Cause
-101	Invalid character	Unrecognized character in specified parameter
-102	Syntax error	Command is missing a space or comma between parameters.
-103	Invalid separator	Command parameters are not separated by a comma.
-104	Data type error	The wrong data type (i.e. number, character, string, expression was used when specifying a parameter.
-108	Parameter not allowed	Parameter specified in a command where none is allowed.
-109	Missing parameter	No parmeter specified when required.
-113	Undefined header	Command header was incorrectly specified.
-124	Too many digits	> 257 digits were specified for a parameter.
-128	Numeric data not allowed	A number was specified for a parameter when a letter is required.
-131	Invalid suffix	Parameter suffix incorrectly specified.
-138	Suffix not allowed	Parameter suffix is specified when one is not allowed.
-141	Invalid character data	The parameter type specified is not allowed.
-161	Invalid block data	Mismatch between character count in header and actual number of characters.
-178	Expression data not allowed	A parameter is enclosed in parentheses.
-221	Settings conflict	Digital I/O command settings are in conflict (e.g., control asserted when in a handshake mode other than NONE).
-224	Illegal parameter value	Inconsistent parameter value

Table C-1. HP E1330 Quad 8-bit Digital I/O Module Messages (continued)

Code	Message	Cause
-240	Hardware error	Hardware error detected during power-on cycle. Return Digital I/O Module to Hewlett-Packard for repair.
-410	Query interrupted	Data is not read from the output buffer before another command is issued.
-420	Query unterminated	Command which generates data not able to finish executing due to a Digital I/O Module configuration error.
-430	Query deadlocked	Command execution cannot continue since the mainframe's command input and data output buffers are full. Clearing the instrument restores control.
-1000	Out of memory	No memory available.

Index

D

F

G

H

I

L

M

O

P

Q

R

S

T

U

V