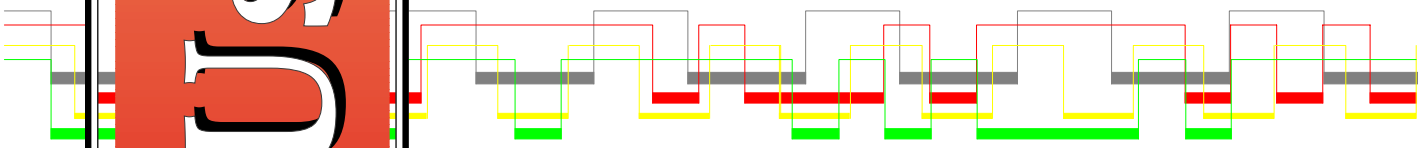
User's Manual

IO50 & IO100 Series

VXI Digital I/O Modules

Includes Coverage For:

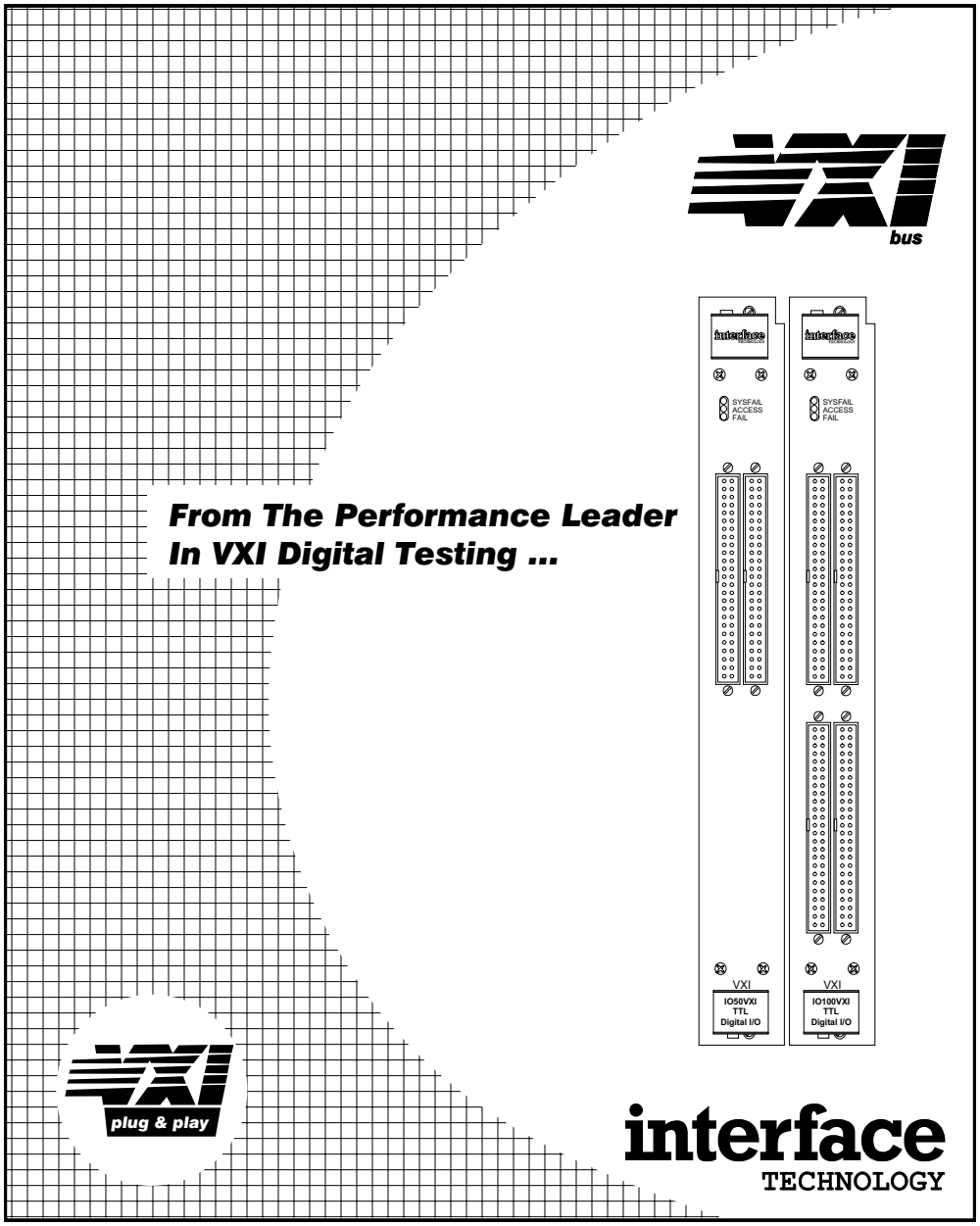
IO50
IO51
IO52
IO53
IO54
IO100
IO110
IO120
IO130
IO140



interface
TECHNOLOGY

User's Manual

IO50 / IO100 Series
Digital I/O Modules



Record of Changes

Change No.	Date of Change	Title or Brief Description	Entered By
Rev. 01	Apr, 1997	Original Issue	Factory
Rev. 02	Jan 10, 2000	Expanded coverage to include IO50 Series.	Factory
Change 1	Jan 16, 2001	Corrected driver and receiver data, Appendix A, pg A-1	Factory
Change 2	Jun 15, 2001	Added App/Tech Note section to manual	Factory
Change 3	Oct 29, 2001	Revised pg 1-4 (specifications); revised tables B-1 and B-2 of Appendix B	Factory
Change 4	Feb 17, 2004	Revised pgs 6-7 - 6-13 (Figs 6-2 - 6-8) and pg A-2, changed mating connector data.	Factory
Change 5	Apr 28, 2005	Changed pg 1-1, Table 1-1. Changed pgs 1-2 and 1-3 (IO53 and IO130), (IO54 and IO140). Changed pg 1-5, table at top of page. Changed pgs 6-9 thru 6-14 (figs 6-4 thru 6-9). Fig D-1, changed from "Connector D (IO130 only)" to "Connector D (IO130-002 only)". Corrected typo errors on pgs D-2 and D-3.	Factory

Proprietary Notice

This document, and the technical information contained herein, are proprietary of Interface Technology and shall not, without the express written permission of Interface Technology, be used in any form or part to solicit competitive quotations. The information provided herein may be used for operational purposes only, or for the purpose of incorporation into technical specifications or other documents which specify procurement from Interface Technology

DISCLAIMERS

Interface Technology, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, implied warranties or fitness for a particular use or purpose.

Interface Technology, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the performance or use of this material.

Interface Technology, Inc. reserves the right to make changes to its products and to the content of this manual without notice.

Contents

Chapter 1
General Information

About This Manual 1-1
Arrangement of Contents 1-1
Applicability 1-1
Supersedure Notice 1-1
Equipment Description 1-2
Programming Formats 1-2
Autonomous Operation 1-2
The IO130 and IO53 1-2
The IO54 and IO140 1-2
Indicators and Connectors 1-3
LEDs 1-3
Connectors 1-3
Specifications 1-4

Chapter 2
Functional Description

General 2-1
Operating Modes 2-1
Basic Input/Output Mode 2-1
Defined Test Modes 2-1
Register Access Mode 2-1
Basic Test Mode 2-2
Defined Test Mode 2-3
Block Handshake Tests 2-4
Programmed I/O Handshake Tests 2-4
Block Timed Tests 2-5
Programmed Timed Tests 2-9
Memory Emulation test 2-9
Register Access Mode 2-12
Data Organization and Memory Allocation 2-13
Vectors 2-13
Fields 2-14
Front Panel Connectors 2-15
Data Pins 2-16
External Tristate Control Pins 2-16
Request Handshake Control Pins 2-17
Acknowledge Handshake Control Pins 2-18
Power Pins 2-19
VXI/VME Connections 2-19
VME Interrupt Request Connections 2-19
TTL Trigger Connections 2-19

Chapter 3
Command Set

SCPI Command Syntax 3-1
Standard Commands for Programmable Instruments 3-3
ABORT 3-5
FIELD 3-6
:DEFINE 3-7
:PINASSIGNMENT 3-8
:NAME 3-9
:TRISTATE 3-10
:TRISTATE? 3-11
:DELETE 3-11
:CATALOG? 3-12
INITIATE 3-13
:INPUT 3-14
:OUTPUT 3-15
:BLOCK 3-16
TEST 3-17
:DEFINE 3-18
:MEMEMULATION 3-18
:SIZE 3-19
:BLKOUTHANDSHAKE 3-20
:SIZE 3-21
:BLKINHANDSHAKE 3-22
:SIZE 3-23
:BLKOUTTIMED 3-24
:SIZE 3-25
:BLKINTIMED 3-26
:SIZE 3-27
:PRGIOHANDSHAKE 3-28
:PRGIOTIMED 3-29
:NAME 3-30
:STATUS 3-31
:DELETE 3-32
:CATALOG? 3-32
:HANDSHAKE 3-33
:REQUEST 3-33
:INPUT 3-34
:INPUT? 3-35
:OUTPUT 3-36
:OUTPUT? 3-36
:ACKNOWLEDGE 3-37
:INPUT 3-38
:INPUT? 3-39
:OUTPUT 3-40
:OUTPUT? 3-41
:TIMEOUT 3-43
:INPUT 3-44

:INPUT? 3-45
:OUTPUT 3-46
:OUTPUT? 3-47
:ADDR 3-48
:ADDR? 3-48
:FREE? 3-49
VECTOR 3-50
:COUNT 3-51
:DATA 3-51
:VALUE 3-52
:VALUE? 3-53
:RADIX 3-54
:FIELD 3-55
SYSTEM 3-56
:ERROR? 3-56
:VERSION? 3-57
:FIELD 3-58
:FIELD? 3-58
:TEST 3-59
:TEST? 3-59
:LEARN 3-60
:LEARN? 3-61
:TRISTATE? 3-62
STATUS 3-63
:OPERATION 3-66
:EVENT? 3-66
:CONDITION? 3-67
:ENABLE 3-68
:ENABLE? 3-68
:TEST 3-69
:EVENT? 3-69
:CONDITION? 3-70
:ENABLE 3-71
:ENABLE? 3-71
:ISUMMARY1 3-72
:EVENT? 3-72
:CONDITION? 3-73
:ENABLE 3-74
:ENABLE? 3-74
:ISUMMARY2 3-75
:EVENT? 3-76
:CONDITION? 3-76
:ENABLE 3-77
:ENABLE? 3-77
:ISUMMARY3 3-78
:EVENT? 3-78

:CONDITION? 3-79
:ENABLE 3-80
:ENABLE? 3-80
:ISUMMARY4 3-81
:EVENT? 3-82
:CONDITION? 3-82
:ENABLE 3-83
:ENABLE? 3-83
BASICMODE 3-84
:DEFINE 3-85
:INPUT 3-86
:OUTPUT 3-87
:CATALOG? 3-88
:CLEAR 3-89
:INPUT? 3-89
:OUTPUT 3-90
:OUTPUT? 3-91
:MODE 3-92
:SLAVE 3-93
:GROUP 3-94
:MASTER 3-95
:GROUP 3-96
:STANDALONE 3-96
:MODE? 3-97

IEEE 488.2 Mandatory Commands

*CLS 3-100
*ESE 3-100
*ESE? 3-101
*ESR? 3-101
*IDN? 3-102
*OPC 3-102
*OPC? 3-103
*RCL 3-103
*RST 3-104
*SAV 3-104
*SRE 3-105
*SRE? 3-105
*STB? 3-106
*TRG 3-106
*TST? 3-107
*WAI 3-107

Chapter 4 Register Access

Register Based Operation 4-1
Register Programming Bit Definitions 4-1
Data Input and Output Registers (0x20-2F Read/Write) 4-1

Control Registers 4-3
Output Latch and Trigger 2 Generation (0x30 Write) 4-3
Input Latch and Trigger 1 Generation (0x31 Write) 4-3
Request Handshake Polarity Control (0x33 Write) 4-3
Output Acknowledge Handshake Signal Control (0x34 Write) 4-4
Output Request Handshake Status (0x34 Read) 4-4
Input Acknowledge Handshake Signal Control (0x35 Write) 4-4
Output Interrupt Mask and Inverted Acknowledge Control (0x36 Write) 4-5
Input Interrupt Mask and Inverted Acknowledge Control (0x37 Write) 4-5
Output Latch Strobe Select Control (0x38 Write) 4-6
Input Latch Strobe Select Control (0x39 Write) 4-6
Input Latch Strobe Select Control (0x39 Write) 4-6
Tristate Polarity (0x3A Write) and Output Control (0x3C Write) 4-6
Tristate Enable Readback (0x3C Read) 4-7
Trigger Input/Output Select (0x3F Write) 4-7

Chapter 5 Applications

Memory Emulation 5-1
ROM Emulation 5-2
RAM Emulation 5-4
ROM Emulation with Programmed I/O 5-6
Techniques for Clocking With Data Pins 5-7
Output and Input Testing of Memory Devices 5-7
Testing Memory Mapped I/O Control Logic 5-9
Counter or FIFO Testing 5-13

Chapter 6 Installation

Scope of Chapter 6-1
Unpacking and Inspection 6-1
Installation 6-1
Logical Addressing 6-1
Slot Dependency 6-2
Calibration 6-2
Basic Operation 6-3
Self-Test 6-3
Basicmode I/O 6-3
Running and Stopping 6-6
Pinouts 6-6

Appendix A Specifications A-1
Appendix B Drivers and Receivers B-1
Appendix C Error Codes C-1
Appendix D Programming Instructions for High Voltage Switching Option D-1

AppNotes & TechNotes

IO50/100-10 .. Using IO50 and IO100 Backplane Voltages

(THIS PAGE INTENTIONALLY LEFT BLANK)

CHAPTER 1

General Information

About This Manual

This manual provides installation and operation information for the Interface Technology IO50 series and IO100 series Digital I/O Modules. Information contained herein is intended for use by technical personnel involved in the actual installation and operation of the subject modules.

Arrangement of Contents

Information contained in this manual is arranged in six chapters, as follows:

- Chapter 1 General Information
- Chapter 2 Functional Description
- Chapter 3 Command Set
- Chapter 4 Register Access
- Chapter 5 Applications
- Chapter 6 Installation and Basic Operation

Applicability

The information contained in this manual covers ten equipment configurations, as listed in Table 1-1. Differences, if any, between this equipment and the actual equipment supplied are covered by Difference Data included at the front of this manual.

Table 1-1. List of Equipment Configurations.

Model	I/O Channels	Logic Family
IO50	64	FTTL
IO51	64	ACTTTL/CMOS
IO52	64	TTL Open Collector
IO53	30 Outputs + 32	Solid State Relay + FTTL or CMOS I/O
IO54	32 Diff. I/O	Differential TTL
IO100	128	FTTL
IO110	128	ACTTTL/CMOS
IO120	128	TTL Open Collector
IO130	30 Outputs + 96	Solid State Relay + FTTL or CMOS I/O
IO130-002	60 Outputs + 64	Solid State Relay + FTTL or CMOS I/O
IO140	32 Diff. I/O + 64	Differential TTL + TTL or CMOS
IO140-002	64 Diff. I/O	Differential TTL

Supersedure Notice

This manual supersedes *Interface Technology IO100 Digital I/O Module User's Manual* Rev. A.1.00 in its entirety.

Equipment Description

The IO50 and IO100 series of Digital Input/Output modules were developed for use in process control, microprocessor cycle emulation, bus cycle emulation, process simulation, and functional board or circuit test applications. All modules are single slot, C-size VXI modules, with the IO100 series providing up to 128 channels of digital I/O and the IO50 series providing up to 64 channels of digital I/O. Each group of 8 channels may be software configured as either input or output. Tristate control of outputs allow for emulation of bi-directional data and control busses. Four 50-pin IDC connectors are provided on the IO100 series front panel, two on the IO50 series. Each I/O connector provides 32 I/O channels, 4 I/O handshake strobes, and 4 tristate control/output enable inputs.

Programming Formats. All modules use a high level, SCPI-compatible command set for setup and control of I/O channels. They also support VME dual-ported RAM and registers. I/O pins may be programmed by sending high-level commands or with direct, high speed VME read/writes, the same as might be used for register-based instruments. Using this combination of programming formats results in the best of both worlds, high functionality and high speed.

Autonomous Operation. Memory emulation and block input or output modes allow autonomous operation from the local microprocessor. Data fields may be programmed from 1-bit to 32-bits wide. Multiple data fields may be defined, allowing I/O pins to be grouped together based on function. Double latching the outputs allows all output channels to transition at the same time, regardless of field size or the number of fields defined. Latching the inputs allows a full 128 bit wide read with a single command (64 channels for the IO50). Utilizing the VXI TTLTRG lines, modules may be linked together for even wider I/O channel groups.

The IO53 and IO130 modules provide users with 30 optically isolated solid-state relay outputs for applications requiring high voltage outputs. The IO53 provides 30 high voltage output channels and 32 TTL or CMOS I/O channels and the IO130 provides 30 high voltage output channels and 96 TTL or CMOS I/O channels. The relays allow the module to control high voltage applications up to 100 volts, AC or DC. Switched voltages can be either user supplied via the front panel, or selected from +5, ± 12 and ± 24 volts available from the VXI backplane. The TTL I/O channels are the same as the IO100 TTL channels described previously.

An option for the IO130, the IO130-002, substitutes an additional 30 high voltage output channels for the 32 TTL channels on port/connector “D”, providing a total of 60 high voltage outputs and 64 TTL or CMOS I/O channels. Both the IO53 and IO130 modules support external I/O handshaking for their TTL or CMOS I/O ports. The handshake I/O pins are used for supplying the external switched voltages on the high voltage ports.

The IO54 and IO140 modules provide users with differential TTL I/O for applications requiring greater noise immunity and longer cable runs. The IO54 provides 32 Differential TTL I/O channels, each with switchable 100 ohm termination. The IO140 also provides 32 differential I/O channels with switchable 100 ohm terminations, plus an additional 64 TTL or CMOS I/O channels. IO140 TTL channels are the same as the IO100 TTL channels described previously. An option for the IO140, IO140-002, substitutes an additional 32 differential TTL channels on port “B” for the 64 TTL or CMOS channels on ports “B” and “D”.

The differential I/O channels meet RS-422-A standard. To accommodate the greater number of signals (a positive/negative signal pair per channel), port B has been eliminated and port A signals mapped to connector A and connector B. An error will be generated if an attempt is made to define a test using port B. For option IO140–002, only ports A and C are valid. Both the IO54 and IO140 modules support external I/O handshaking for their valid ports.

Indicators and Connectors

See Fig. 1-1. All the connectors and LEDs for the I/O Module are located on the module front panel.

LEDs

There are three LEDs located at the top of the I/O Module front panel.

- **POWER** (Green) - On any time power is applied to the module.
- **SYSFAIL** (Red) - Off during normal operation. During the power-up sequence, SYSFAIL is lit until completion of internal self test. If SYSFAIL remains lit, a fault exists within the module.
- **ACCESS** (Yellow) - Lit briefly anytime the module is accessed by the VXI bus.

CONNECTORS

Four 50-pin connectors are provided on the IO100 series, two 50-pin connectors on the IO50 series. Connector location and pin numbering are described in Chapter 6 *Installation.* Each connector has 32 data pins organized as four bytes of 8-bits each. Each byte has internal and external tristate output control.

Other signals on each connector include one pair of request and acknowledge handshake strobe lines for input, another pair for output, one external tristate control for each byte, eight paired ground lines, and two fused 5 volt supply pins.

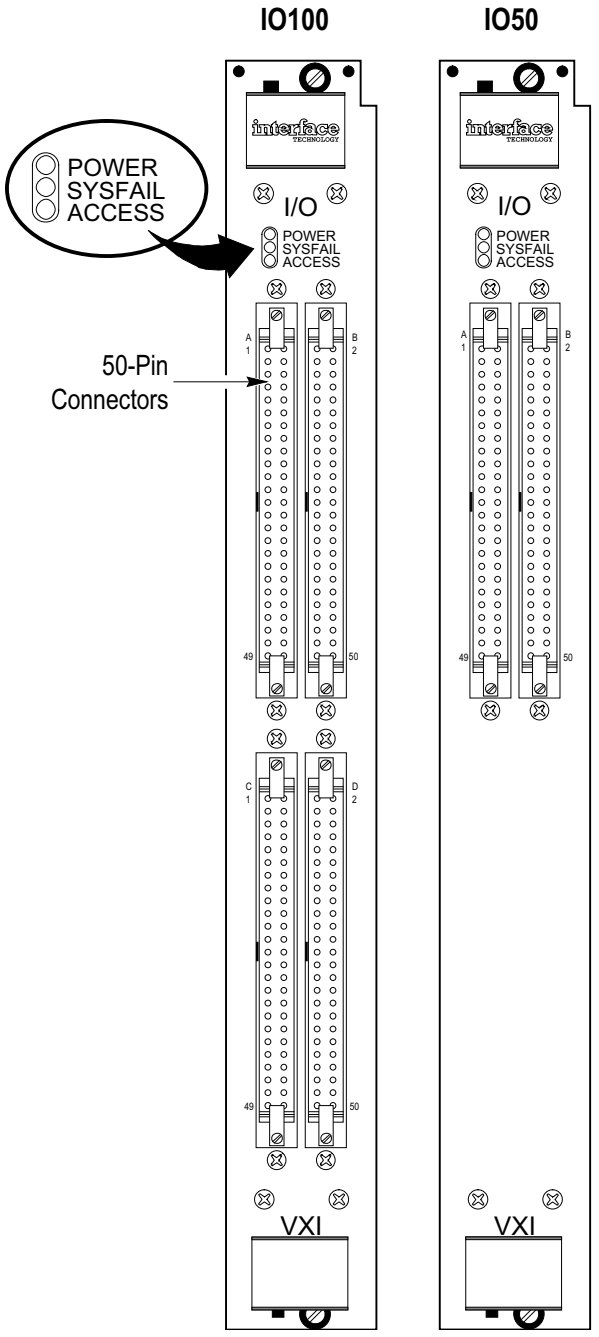


Figure 1-1.
Indicators and Connectors.

SPECIFICATIONS*

Model	I/O Channels	Logic Family
IO50	64	FTTL
IO51	64	ACTTTL/CMOS
IO52	64	TTL Open Collector
IO53	30 Outputs + 32	Solid State Relay + FTTL or CMOS I/O
IO54	32 Diff. I/O	Differential TTL
IO100	128	FTTL
IO110	128	ACTTTL/CMOS
IO120	128	TTL Open Collector
IO130	30 Outputs + 96	Solid State Relay + FTTL or CMOS I/O
IO130-002	60 Outputs + 64	Solid State Relay + FTTL or CMOS I/O
IO140	32 Diff. I/O + 64	Differential TTL + TTL or CMOS
IO140-002	64 Diff. I/O	Differential TTL

Logic Families:

FTTL	Vol*	0.55 V	Voh**	2.4 V
IO100/IO50	Vil	0.8 V	Vih	2.0 V
	Iol	64 mA	Ioh	-3 mA
Skew****	15 ns, max.			
Rise/Fall	3 ns/3 ns			
CMOS	Vol*	0.5 V	Voh**	3.7 V
IO110/IO51	Iol	24 mA	Ioh	-24 mA
Skew****	20 ns, max.			
Rise/Fall	4 ns/4 ns			
Open Collector	Vol*	0.42 V	Voh**	5.0 V
IO120/IO52	Vil	0.8 V	Vih	2.0 V
	Iol	64 mA	Ioh***	0.5 mA
Skew****	20 ns, max.			
Rise/Fall	3 ns			
Switched				
IO130/IO53				
Max Voltage	100 V peak AC/DC			
Turn on/off Time	4 ms			
On Resistance	20 ohm max.			
Isolation	Optical, 3750 V			
Carry Current	120 mA			
Differential TTL	Vol*	0.50 V	Voh**	2.5 V
IO140/IO54	Vil	0.8 V	Vih	2.0 V
	Iol	20 mA	Ioh	-20 mA (max)
Skew****	21 ns, max.			
Rise/Fall	14 ns/14 ns (typical)			

Handshake and Control:

(except Switched High Voltage Outputs)

Byte Available/Request	Per I/O connector
Data Valid/Acknowledge	Per I/O connector
Tristate Control Inputs	1 per byte (except IO120/IO52)
Output Enable Inputs	1 per byte (IO120/IO52 only)

VXI Specifications

Interface Compatibility:

Type	Message-based, servant only
VXI Revision	1.3 and 1.4
Size	C-size, single slot
Configuration	Static or Dynamic
Interrupt Level	Programmable 1-7
TTLTRG 0-7	Input or output, selectable in groups of two
Memory	A24 RAM, 256K

Power Requirements:

All modules (except IO53/130)	+5 volts, 3.2 A, 16 W typ.
IO53 / IO130	+5 volts, 3.7 A, 18 W typ., ±12, ±24 volts user selectable

Cooling Requirements:

Per-slot Average	16 W typical
Airflow	1L / sec @ 0.30 mm water pressure for 10º C temperature rise

Environmental Specifications:

Temperature	Storage = -40º C to +75º C Operating = 0º C to 45º C
Humidity	5% to 95% relative, noncondensing

Software Drivers:

National Instruments	LabWindows/CVI

*	Maximum voltage at minimum load.
**	Minimum voltage at maximum load.
***	Depends on pull-up resistor value.
****	Channel-to-channel skew. Add 50 ns for channel-to-channel skew across multiple cards.

*Specifications subject to change without notice.

(THIS PAGE LEFT BLANK INTENTIONALLY)

C H A P T E R 2

Functional Description

General

Both the IO50 and the IO100 (IO50/IO100) are general purpose, parallel, digital interfaces that provide stimulus or receive response from a variety of devices. Each data transfer (stimulus or response) is controlled by the user provided Slot-0 Controller using either message-based commands or direct shared-register access. Several modes of operation allow I/O transfers to be controlled by the local IO50/IO100 processor, or by hand-shake control signals from the UUT (Unit Under Test). The type of output driver and resistive termination can be modified by changing socketed parts within the IO50/IO100 module (refer to Chapter 5, Installation). Data signals are single ended, connected by means of ribbon cable or discrete wiring. Control lines are paired with ground lines to improve transmission reliability over cable runs of up to one meter in length.

Operating Modes

Basic Input/Output Mode. In its simplest operating mode, the IO50/IO100 serves as a parallel interface controlled from the user supplied Slot-0 Controller. Typical applications for this mode of operation include process monitoring and control involving slow operating devices such as relays, switch closures, and valve control via I/O module racks. Very wide digital test vectors and responses can be sent and received in this mode.

Defined Test Modes. A more sophisticated level of use is provided by the Defined Test modes of operation. These modes are useful in applications where a more "intelligent" device is sending or receiving data and requires strobe signals to control the transfer. Block data movement can be used to couple asynchronous systems together in the manner of FIFO buffers. Memory emulation functions are provided for aiding in test of microprocessor based devices with limited test point access.

Register Access Mode. Users with unique requirements can control all hardware aspects of the IO50/IO100 using shared access to the VXI Device Dependent registers. This mode provides those users with software development capability with the option of defining their own macro operation commands on their chosen Slot-0 Controller.

Basic Test Mode

The Basic Test Mode requires minimal setup to produce output or read input on the data pins. Data is transferred when the command is executed without regard to UUT connections to the handshake or external tristate control signals. This mode also offers the option of configuring multiple I/O modules within a single VXI chassis for simultaneous input or output.

Any number of the available data pins from 1 to 128 (1 to 64 in the case of the IO50 series) may be used for input or output. Data values transferred will be read or written simultaneously regardless of data "width". Each pin used must be defined as an input or output before data values are sent or received. Pins are defined in byte groups with the following boundaries: 32-25, 24-17, 16-9, and 8-1. The shortened references used for these groups in the command set are: 25, 17, 9, and 1. These values will always define pin groups of exactly eight bits. Pin definition commands may specify multiple bytes or may skip bytes (e.g., A25, A17, A1).

Pins defined as outputs will have their output drivers enabled and data specified in the command will appear at the outputs. Even on pins defined as outputs, the current output data may be read back for verification. This is helpful in determining if a shorted or low impedance line is being driven. Output data storage is initialized to zero by hardware reset or VXI soft reset. All data pins are affected with each BASICmode:OUTput command. Outputs which are not specifically set by a command will be zero-filled. For a data field to remain constant throughout a sequence of Basic outputs, it must be referenced in each command.

Each pin definition command for output will clear any previously defined output pins, but not affect input pin definition. Input pin definition commands will clear previously defined inputs without affecting outputs. All input and output pins are returned to undefined by the CLEar command. The CATalog? query is provided to allow readback of all currently assigned pins.

Note

Output drivers are enabled in byte-wide groups so that all pins in the same octet will have their outputs enabled. Be sure that "don't care" pins in an output group are not connected or are connected only to input devices.

An error condition will be generated if an attempt is made to define pins within the same octet as different types (input and output). The best practice in configuring test fixtures is to estimate the required number of inputs and outputs, allowing spares for future use, and round up to the nearest byte boundary before making the UUT wire harness.

Data read back from the defined input pin list will be returned in the order listed in the pin definition command. Users may take advantage of this

list order control to perform byte swapping of data read from sources with differing byte order conventions. If a data value sent is not wide enough to fill the defined output pin field, bits will be zero filled from most significant down. In this case, the pins specified last in the definition command (right most) will output the available data and the pins specified first (left most) will output zeros. Input commands will return a data value equal in width to the number of defined input pins.

The general procedure for using the Basic Input/Output Mode of operation and the programing sequence, are shown in Chapter 3, Command Set and examples are shown in Chapter 4.

Defined Test Mode

Up to four tests may be defined at any given time, and are designated by the letters A, B, C and D. The letters A, B, C, and D correspond to the independent sets of handshake control signals available on each of the four connectors. A UUT test fixture may use different control signals on each of the connectors, and switch between them by changing the active test. The first test defined after power up, or when all tests have been deleted, becomes the active test. The active test can then be changed by using the SYSTem:TEST command with the desired test letter parameter. Only the handshake signals of the current active test are enabled.

There are five Define Test types ...

1. Block In/Out Handshake
2. Block In/Out Timed
3. Programmed I/O Handshake
4. Programmed I/O Timed
5. Memory Emulation

Each defined test type provides some unique feature not provided by the other test types. Normally, only one test will be active at any given time. However, provisions have been made to allow Timed I/O Tests to be initiated after a Handshake I/O Test is already running.

The handshake signals provided for data flow control with the UUT are used in some manner by all the defined tests. The timed tests require no requesting handshake from the UUT, but toggle the acknowledge handshake with each data value transferred. This provides a strobe for UUTs that might have input or output registers or latches (FIFOs, shift registers, etc.). The handshake type tests will not transfer data until the UUT initiates a request. This allows more intelligent devices to control the flow of data at their own speed (data recorders, display devices, etc.).

Test operation is started with the INITiate command and can be stopped at any time with the ABORT command. In the case where two tests are running a one time (Memory Emulation and Programmed I/O), the

INITiate and ABORT commands act only on the active test, as selected by the SYSTem:TEST command.

Block Handshake Tests

Block Handshake type test provide the capability to move a block of data between the IO50/IO100 Shared Memory area and the UUT, with data flow controlled by the UUT. Block Handshake tests are initiated by the user supplied Slot-0 Controller, then performed autonomously by the local 68000 MPU on the IO50/IO100. Once initiated, the Slot-0 need perform no other operation until an end condition occurs (last item is transferred). All input or output operations are controlled by the UUT's use of the handshake lines (see Figs 2-1, 2-2, and 2-3). An interrupt can be sent to the Slot-0 Controller after the last data value has been transferred. The STATUS commands are used to enable this interrupt. Block Handshake tests provide a FIFO-like speed decoupling between the IO50/IO100 and the UUT.

Shared Memory locations are allocated for each block memory test, with separate areas for input and output. Each data vector allocates 16 bytes of shared RAM, allowing access to any bit in the 128 pin array (64 bit pin array in the case of IO50 series) operation, regardless of the actual number of data bits changing, or their position. The rate of transfer is, therefore, independent of the width of the data field being transferred. Shared Memory data values can be accessed by the VECTOR:DATA command or by direct access from the Slot-0 Controller. Once a Handshake Test is started, no other handshake test may be initiated. However, it is possible to start a Timed Test, even if a Handshake Test is already running.

Programmed I/O Handshake Tests

This test provides single data value input and output with the UUT controlling data flow (see Figs 2-1, 2-2, and 2-3). Data locations for input and output are separate, much like a full duplex communications peripheral. To perform output, the Slot-0 Controller will first prepare an output data vector. The VECTOR:DATA command can be used, or direct memory access to the vector location in shared memory. The INITIATE:OUT command is then sent to attempt data output. The IO50/IO100 will then wait for an output request on the handshake input from the UUT before transferring data. The TEST:NAME:STATUS? command can be used by the Slot-0 Controller to determine if the data has been transferred. Alternately, the STATUS command can be used to enable an interrupt when the UUT has received the pending data value and is ready for the next. If an attempt is made to send a second data value before the first has been transferred, an error is generated. At any time, the ABORT command may be sent to halt a pending transfer.

Input operation is similar, with the UUT sending a read request to the IO50/IO100 when it has data available. If the INITIATE:IN command has been executed, this request will cause data be read to the input vector location. If the input interrupt is enabled, a Slot-0 interrupt is sent after the data has been read. The TEST:NAME:STATUS? command can be used to determine if

any data has been transferred. The Slot-0 Controller may access the input data with the VECTOR:DATA command, or direct read of shared memory.

An acknowledge signal is sent to the UUT at the completion of each data transfer, forming an interlock handshake sequence. For output data, the acknowledge is sent when valid data is available on the IO50/IO100 output pins. For input, the acknowledge handshake is sent when the UUT data has been latched in from the IO50/IO100 input pins.

Shared Memory locations are allocated for Programmed Test with separate areas for input and output. Each data vector allocates 16 bytes of shared RAM, allowing access to any bit in the 128-pin (64-pin for IO50 Series) array. One entire data vector is transferred on each input or output operation, regardless of the actual number of data bits changing, or their position. Shared Memory data values can be accessed by the VECTOR:DATA command or by direct access from the Slot-0 Controller. Once a Handshake Test is started, no other Handshake Test may be initiated. However, it is possible to start a Timed Test even if a Handshake Test is already running.

Block Timed Tests

This test operation provides the capability to move a block of data between the IO50/IO100 shared memory area and the UUT, with data flow at a programmed rate. Block Timed Tests are initiated by the Slot-0 Controller, then performed autonomously by the local 68000 MPU in the IO50/IO100. Once initiated, the Slot-0 Controller need perform no other operation until an end condition occurs (last item is transferred). All input or output operations are controlled by instruction sequencing on the 68000 MPU (see Fig 2-4). An interrupt can be sent to the Slot-0 Controller after the last data value has been transferred. The STATUS commands are used to enable this interrupt. This test can provide output to UUT devices with clocked or latched input data requirements, but no local intelligence to control data flow. The IO50/IO100 will provide a clocking strobe with each data value output. The TEST:NAME:TIMEOUT command can be used to control the rate of data output and adjust the data setup or hold time in relation to the strobe output.

Shared Memory locations are allocated for each Block Memory Test, with separate areas for input and output. Each data vector allocates 16 bytes of shared RAM, allowing access to any bit in the 128-pin (64-pins in the case of IO50) array. One entire data vector is transferred on each input or output operation, regardless of the actual number of data bits changing, or their positions. The rate of transfer is, therefore, independent of the width of the data field being transferred. Shared Memory data values can be accessed by the VECTOR:DATA command or by direct access from the Slot-0 Controller. Once a Timed Test is started, no other commands will be executed until the test is complete. For this reason, it is not possible to

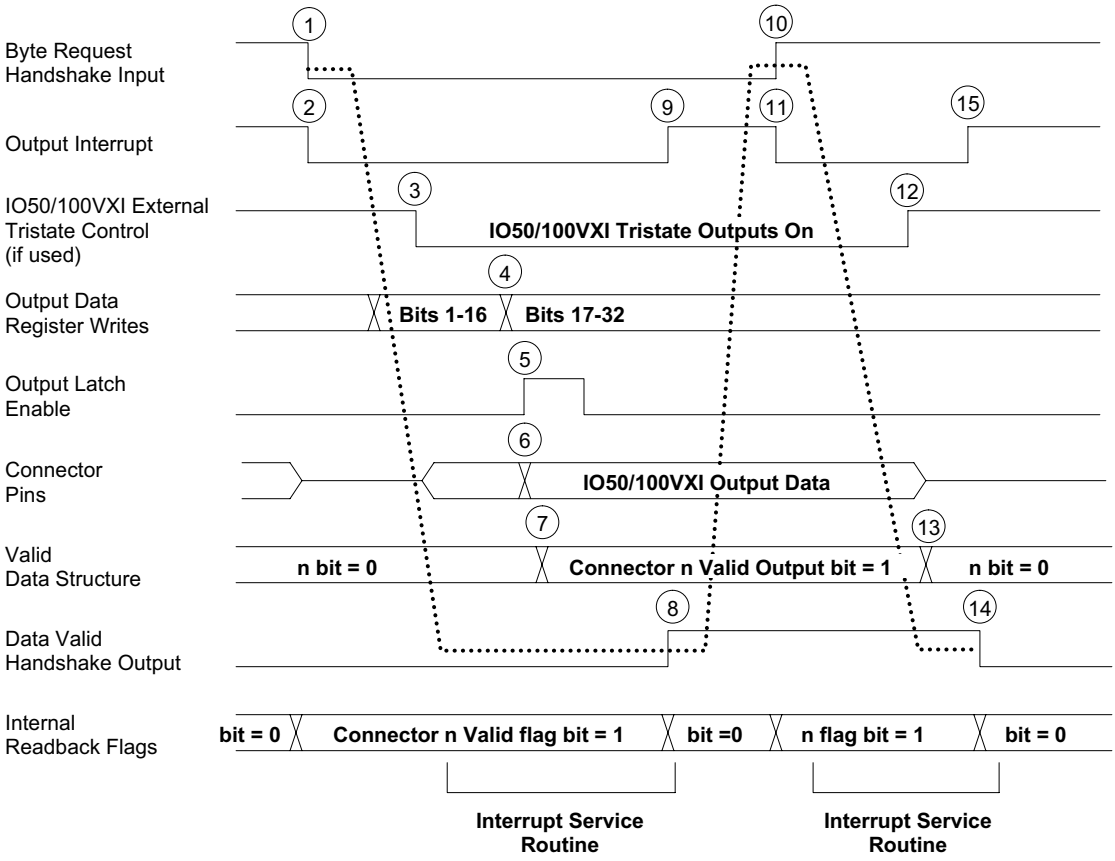


Figure 2-1.
Interlocked Output Handshake Sequence Normal Acknowledge Polarity.

1. UUT pulls Byte Request handshake line low (true).
2. Byte Request input generates interrupt to local 68000 or Resource Manager.
3. UUT turns on tristate outputs.
4. Interrupt service routine gets data from buffer or shared RAM (local 68000), or writes data to Output Registers directly (Resource Manager).
5. Interrupt service routine pulses latch enable to output all bits in parallel.
6. Output data now available to UUT.
7. Internal Valid handshake data structure updated to reflect current output.
8. Data Valid output handshake signal set true (high).
9. Valid output clears local interrupt.
10. UUT can now remove Byte Request since Data Valid if true.
11. Interrupt occurs again to signal Data Valid can now be removed.
12. External tristate enables are turned off.
13. Valid structure is referenced to determine action for handshake outputs.
14. Data Valid handshake can now be removed since byte Request is high (false).
15. Interrupt is cleared by removal of Data Valid.

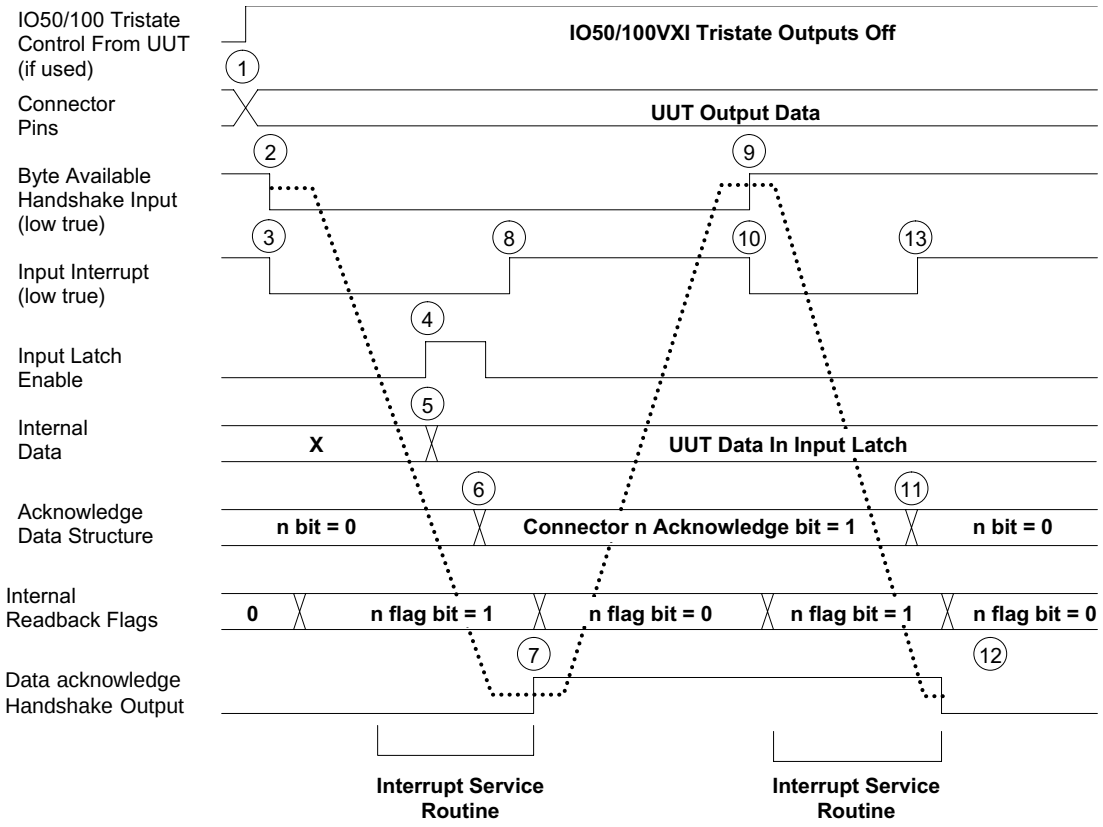


Figure 2-2.
Interlocked Input Handshake Sequence Normal Acknowledge Polarity.

1. UUT outputs data to be read by IO50/IO100.
2. UUT can then pull Byte Available handshake line low (true) since Data Acknowledge line is high (false).
3. Byte Available input generates interrupt to local 68000 or Resource Manager.
4. Interrupt service routine pulses input latch enable to capture input data on connector pins.
5. UUT data now available to local 68000 or Resource Manager via Data input registers.
6. Interrupt service routine stores data in designated buffer or shared RAM (if 68000), or reads directly (if Resource Manager). Once data is saved, internal Acknowledge Data Structure is updated to show current output status.
7. The Data Acknowledge handshake output is then written high (true).
8. Writing Data Acknowledge clears internal interrupt.
9. UUT can now remove Byte Available handshake since Data Acknowledge is present.
10. Internal interrupt occurs again to signal Data Acknowledge can now be removed.
11. Internal Acknowledge Data Structure is referenced to determine action of handshake outputs.
12. Data Acknowledge handshake can be removed since Data Available handshake is high (false).
13. Interrupt removed when Data Acknowledge goes false.

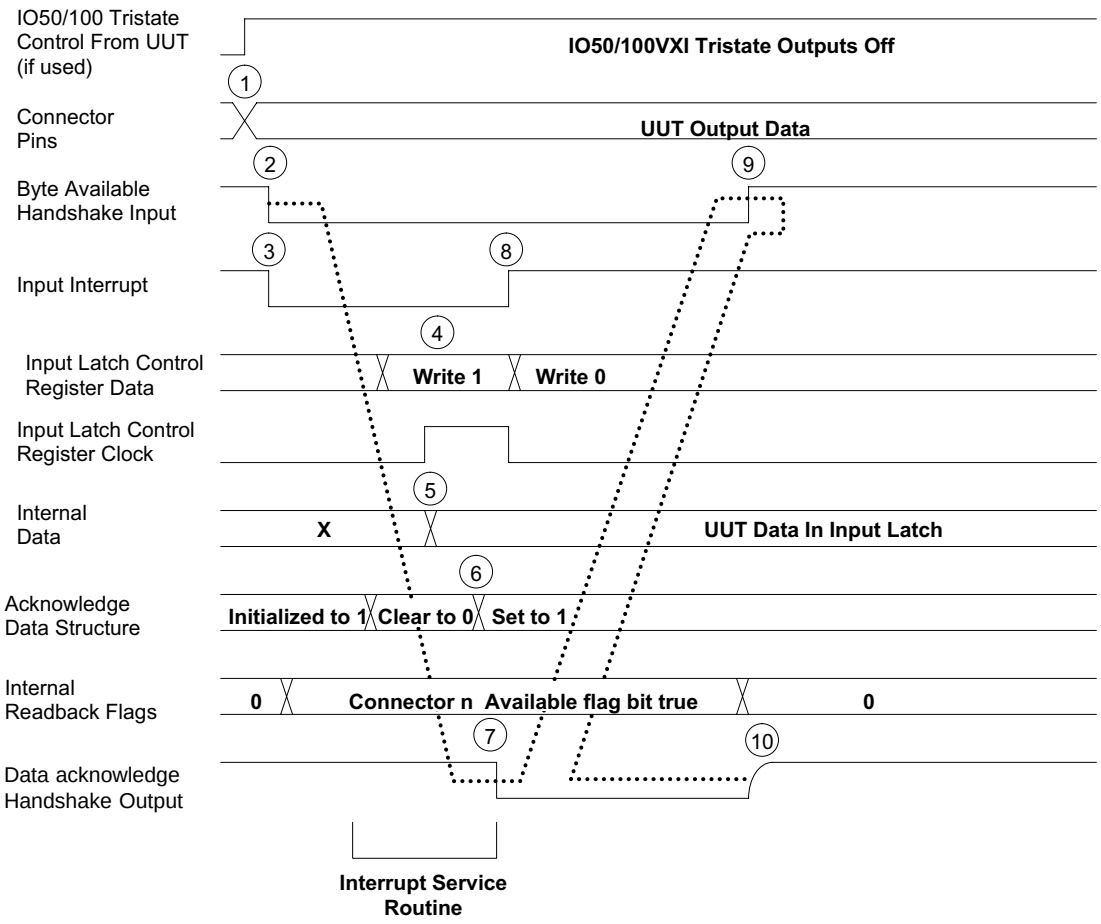


Figure 2-3.
Interlocked Input Handshake Sequence Inverted Acknowledge Polarity.

1. UUT outputs data to be read by IO50/IO100.
2. UUT can then pull Byte Available handshake line low (true) since Data Acknowledge line is high (false).
3. Byte Available input generates interrupt to local 68000 or Resource Manager.
4. Interrupt service routine pulses input latch enable to capture input data on connector pins.
5. UUT data now available to local 68000 or Resource Manager via Data input registers.
6. Interrupt service routine stores data in designated buffer or shared RAM (if 68000), or reads directly (if Resource Manager). Once data is saved, internal Acknowledge Data Structure is updated to show current output status.
7. The Data Acknowledge handshake output is then written low (true).
8. Writing Data Acknowledge clears internal interrupt.
9. UUT can now remove Byte Available handshake since Data Acknowledge is present.
10. Data Acknowledge handshake can be removed since Data Available handshake is high (false).

initiate any new test if a timed test is running.

Programmed Timed Tests

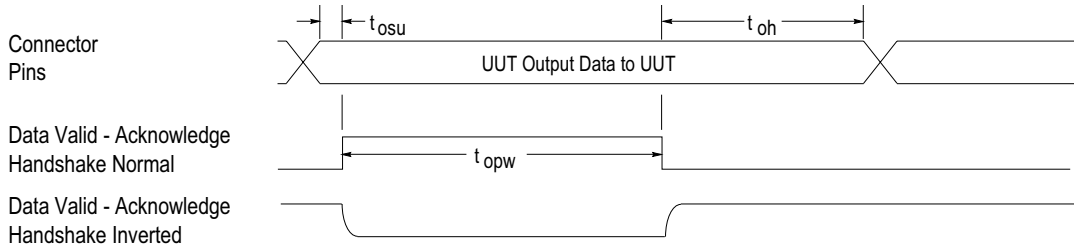
This test provides single data value input and output, with a strobe signal available to the UUT (see Fig 2-4). Data locations for input and output are separate, much like a full duplex communication peripheral. To perform output, the Slot-0 Controller will first prepare an output data vector. Either the VECTOR:DATA command can be used, or direct memory access to the vector location in shared memory. The INITIATE:OUT command is then used to send data output and set the strobe signal to the opposite state. The IO50/IO100 will then wait for a programmed time interval before returning the strobe to its initial state. The TEST:NAME:STATUS? command can be used by the Slot-0 Controller to determine if the data has been transferred. Alternately, the STATUS command can be used to enable an interrupt when the UUT has received the pending data value and is ready for the next one. The TEST:NAME:TIMEOUT command is used to set the strobe timing value.

Input operation is similar, with the INITIATE:IN command used to read data from the UUT. When this command is sent, the IO50/IO100 input strobe is set to the opposite state. After the programmed time-out value, the IO50/IO100 latches data on its input pins and sets the strobe back to its initial state. If the input interrupt is enabled, a Slot-0 interrupt is sent after the data has been read. The TEST:NAME:STATUS? command can be used to determine whether or not data has been transferred. The Slot-0 Controller can access the input data with the VECTOR:DATA command, or can directly read the shared memory.

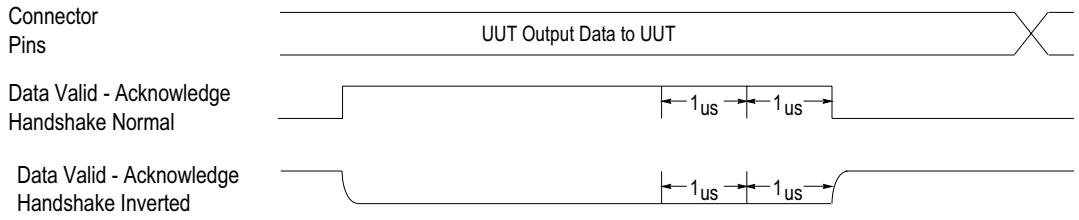
Shared memory locations are allocated for Programmed Test, with separate areas for input and output. Each data vector allocates 16-bytes of shared RAM, allowing access to any bit in the 128 pin array (64 bits in the case of the IO50 series). One entire data vector is transferred on each input or output operation, regardless of the actual number of data bits changing, or their position. Shared Memory data values can be accessed by the VECTOR:DATA command or by direct access from the Slot-0 Controller. Once a Timed Test is started, no other commands will be executed until the test is complete. For this reason, it is not possible to initiate any new test while a timed test is running.

Memory Emulation Test

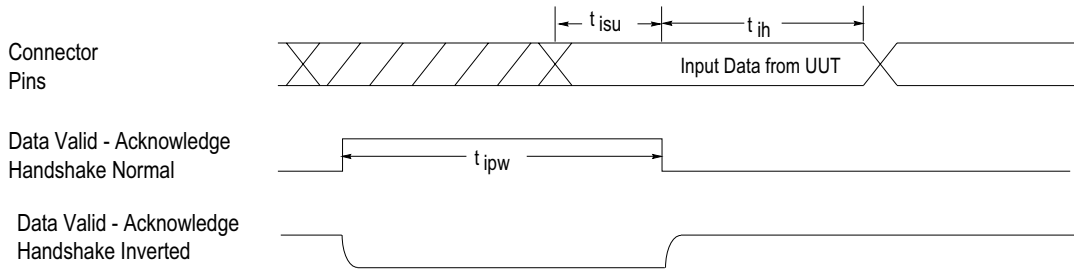
The memory emulation function is a special case of the block input and output functions. The input and output functions share a common memory, allowing RAM or ROM emulation. Data transfer must always be initiated by a UUT handshake request, and separate read and write requests are required. The UUT must also have the capability to wait several hundred microseconds for a memory acknowledge signal before continuing. Refer to *"Application Examples"* (Chapter 5) for connections for some typical applications.



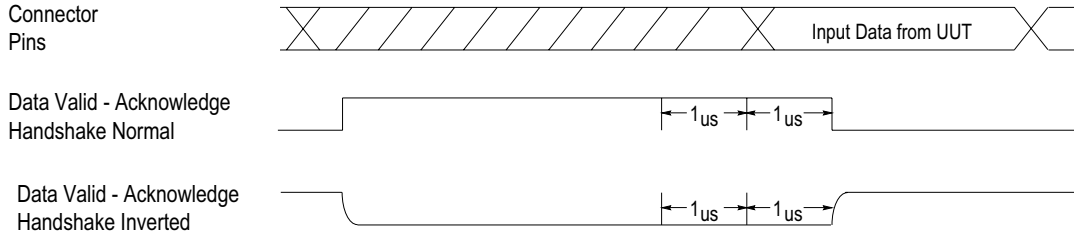
Timed O utput - No Delay



Timed Output - 2 us Delay

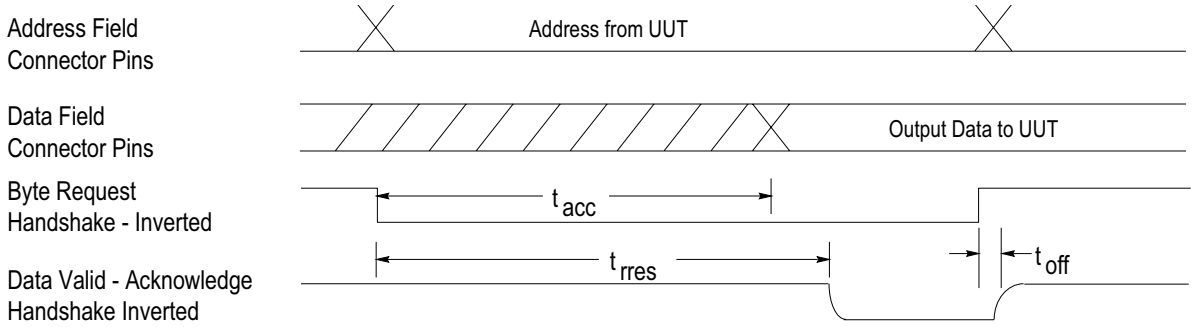


Timed Input - No Delay

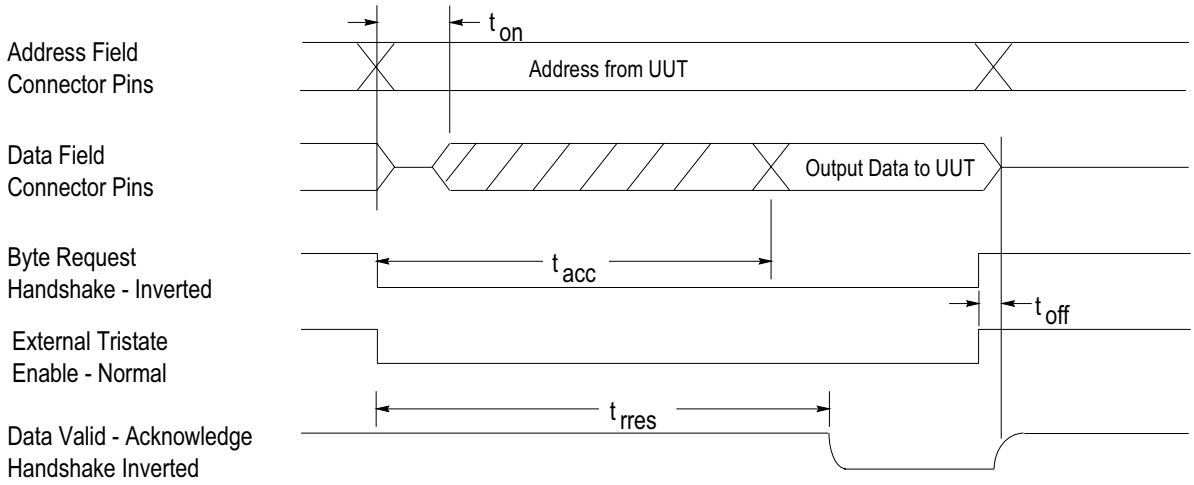


Timed Input - 2 us Delay

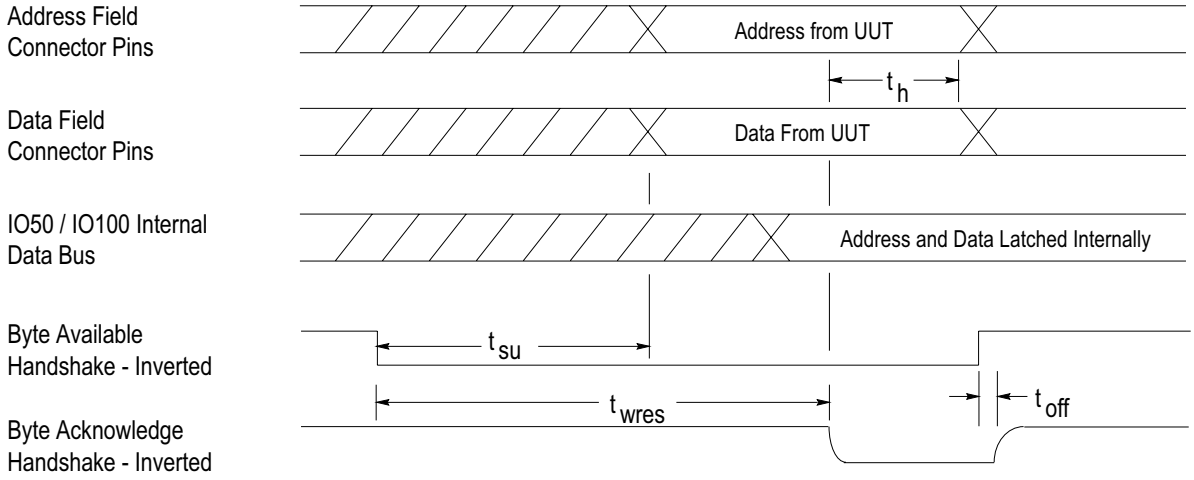
Figure 2-4.
Timed I/O Handshake Operation



Memory Emulation - Read



Memory Emulation - Read With Tristate



Memory Emulation - Write

Figure 2-5.
Memory Emulation Operation.

Memory emulation uses a UUT input field, designated by the user, as an address index into the shared memory storage area. Data will then be transferred across a designated data pin field, emulating random access memory operation. When data transfer is complete, the acknowledge handshake will be sent by the IO50/IO100. While the access and cycle time of this emulated memory is many times slower than that of a real device, it can be used to provide short initialization or troubleshooting loops in testing microprocessor based UUTs. The ability to use a Timed Test while the memory emulation function runs in the background allows changing reset or other inputs to the UUT while executing these test loops. No direct program halt control is available, but the user can store "jump to self" opcodes at the end of test loops. Memory emulation operation will continue until the Slot-0 Controller sends the ABORT command.

Register Access Mode

This method of operating the IO50/IO100 provides flexibility for users with embedded software development capability. The VXI Device Dependent Registers can be accessed directly by the Slot-0 Controller in the IO50/IO100 shared memory space. These are the same registers used by the local 68000 MPU to perform high level command operations. This means that any operation that can be performed with the command set can be performed via direct register access. Users with special requirements for higher speed data transfer on a few channels might develop special algorithms on their Slot-0 Controller to access these shared registers.

Note

Since these registers are shared, users may experience undefined operation if they attempt to use direct register control while executing high level commands or any of the defined test operations. All tests should be aborted before beginning any direct register programming operations.

Users developing code to access the shared registers should be aware that different compilers and Slot-0 hosts have differing address conventions for byte storage. The IO50/IO100 hardware register mapping corresponds to the Motorola scheme of storing the most significant part of long words at the lower of the two word address required. If byte access is used, the lowest addressed byte (even address) corresponds to bits 15-8 of the memory word location. Byte and word swapping may be required with Slot-0 hosts using other conventions. Refer to Chapter 4 for additional information on register level programming.

Data Organization and Memory Allocation

The data organization for BASIC mode operation is discussed under the description of that mode of operation. For the Defined Test operating modes, which allow field definitions, data is stored in structures called vectors. The block transfer type tests will use as many of these vectors as the user specifies (up to the limit of available memory). The Programmed I/O Test uses the same vector structure, but is limited to two vectors, one for input and one for output. Input and output data are stored separately in all cases except Memory Emulation. A memory pool of 15,360 vectors is available in the shared RAM. As tests are defined, vector storage space is allocated from this pool. If the user requests more vectors in the combined defined tests than are available, an error will be generated. If a defined test is deleted, all vectors from the remaining tests are relocated downward in RAM automatically to keep the free RAM pool in high memory unfragmented.

Note

To avoid undefined results during this memory packing operating, no tests can be executed when tests are deleted. Attempting to delete tests before the active test has completed, or aborted, will generate an error.

Vectors

Each data vector is 16 bytes (128 bits) long. This allows bit fields to be defined across any of the output pins. Data is stored in a non-inverting format and can be specified with a hex or binary radix as determined by the VECtor:DATA command. Use the VECtor:COUNt command to reserve a number of locations for these vectors in shared memory. Output vectors will then be filled from low to high memory as they are received in the VECtor:DATA command. Input vectors are filled from low to high memory as block tests execute and the IO50/IO100 reads in data from the UUT. Users may read or write vectors by direct access to shared memory. The data format orders bytes with the Motorola longword address convention.

Several commands are provided to facilitate shared memory utilization, and shared memory access to data vectors during handshake type test execution is allowed. Access to vectors during Timed Block type test execution is not possible, since no new commands are executed until the block transfer has finished. The TEST:NAME:STATus? command can be used to determine the last vector number output or input during test execution. The TEST:NAME:CATalog? command will return the offset, as a byte count, from the start of shared RAM to the address of the first vector location for the named test. This address can be combined with the base address assigned by the VXI Resource Manager to the IO50/IO100 shared memory, to address vector structures. When accessing shared RAM vector structures directly, be sure to observe the correct byte ad-

dressing format, as shown in Chapter 4 (*Register Based VXI Operation*). Also be aware that the bytes of the vector may appear in a different order than when sent as command data. This may be a result of shifting performed to meet the ordering required by the pin list in the definition command. The TEST:NAME:STATus? command also returns the total byte count of vector storage allotted for the named test.

The SYSTem:LEARn? command provides a way to save shared memory as well as configuration information for the instrument. When this command is sent, all internal configuration information is placed in shared RAM and a byte count value is returned. This byte count will include all configuration data and allocated shared memory space for defined tests. The user can read this number of bytes, incrementing from the shared memory base address, into a file to save the current instrument environment. This same file can be written to shared memory after instrument power-up to reinstall the test vector values. Sending the LEARn command will then cause the configuration data to be read into the appropriate internal registers from shared RAM, completing instrument setup.

Fields

The Defined Test Mode allows more sophisticated data manipulation, but requires more user setup than the basic mode. These tests allow bit field definitions for giving relevant names to pin groups. These fields also allow physically separated pins to be combined into logical groups so that the use may read and write them in a convenient data format. Fields may be from 1 to 32 bits wide and need not be defined on byte boundaries. If fields are not defined on byte boundaries, then input data bits will be zero-filled in the last data byte read (to the nearest hex value). Data is apportioned to the output pins in the order they were listed in the field definition pin list (from left to right). If hex output data is supplied for a field not on a byte boundary, the data will contain more bits than the pins of the output field. In this situation, the last data value sent will be truncated to the number of least significant bits required to fill out the field. Fields are "attached" to the test that is currently active when they are defined, and are cleared if that test is deleted. Field names are local to the test, and the same name may be used in multiple tests.

Fields have a tristate control parameter which provides the user the ability to select among input and output pin functions. After defining the field, the user determines whether it is to be used for input only, output only, or controlled by signals from the UUT. After power up or a field clear command, the data pins are considered undefined, and are in a high impedance state. The hardware for input and output of data pins is controlled in byte groups, so field output definitions must be consistent across byte boundaries.

Note

Error messages are generated if an attempt is made to declare pins in the same octet as different types (input and output) in two different fields. The external control of outputs may be useful in situations where the UUT is controlling data flow, and the data pins are connected to a bidirectional or multi-source bus (ROM emulation, backplane data bus emulation).

Front Panel Connectors

IO50 Series Only: Two connectors are provided, labeled A and B with pins numbered from 1 to 50.

IO 100 Series Only: Four connectors are provided, labeled A, B, C, and D with pins numbered from 1 to 50.

Connector pins are configured as two rows of 25, with 0.1 inch center spacing. Pin location and numbering are shown in Fig 1-1. Individual pin descriptions are provided in Figs 6-2 thru 6-8. Each connector has 32 data pins, organized as four bytes. Each byte has internal and external tristate output control. Therefore, pins are configured for input or output in the following octets:

32-25 24-17 16-9 8-1

Other signals on each connector include one pair of request and acknowledge handshake strobe lines for input, another pair for output, one external tristate control for each byte, eight paired ground lines, and two fused 5 volt supply pins.

Note

Handshake signals are provided with paired grounds on the cable connector to help control signal crosstalk and noise pickup common in single ended interfaces. These signals are able to maintain correct logic levels regardless of the switching transients caused by changes on the data pins. The data signals themselves experience transient spikes across logic thresholds due to adjacent channel switching. These "spikes" may be from 10 to 20 nanoseconds in duration (in cables of 1 meter or less). Data signals should, therefore, not be used as clocks or strobes for high speed logic devices and should be qualified with handshake signals if used for address decoding functions. Word generators with transmission line cabling are more suitable to providing transient-free data at high frequencies. The application information section (Chapter 5) shows typical UUT connections and some acceptable uses of data pins for clocks, when unavoidable.

Data Pins

Each of the 50 pin connectors has 32 data pins. Each data pin is capable of receiving input from the UUT or providing output to the UUT. Separate drivers and receivers are used, so output data can be read back at any time. This may be useful to determine if an output is driving a shorted line. Output data is double buffered, so that data fields wider than 16 bits (up to 128 bits) will be output simultaneously. All data pins have the resistor configurations shown in Appendix B. Several optional configurations are possible by changing socketed parts, as also shown in Appendix B. While provisions have been made for several termination schemes, signal levels will be affected by crosstalk and noise as described above. Keeping cable length to a minimum and (for extreme cases) using discrete wiring will minimize these affects.

Data pins are numbered on each connector as shown in Chapter 1, Overview. The front panel markings indicate the pinout of each connector. The command set refers to data pins with a connector letter and pin number (A1,B22,C32-1, etc.). Data is read and written without inversion. All output data bit registers are initialized to zero on power up or a VXI software reset.

External Tristate Control Pins

Each of the four data bytes on a connector has a unique external tristate control input with a paired ground. These inputs can control the output drivers in byte wide groups. The outputs are normally controlled internally, but external control may be useful when tying to a bus structure (i.e.: memory emulation). These control inputs can be inverted under software control, and read back by the Slot 0 controller. A hardware read back function gives an indication of output driver state when a UUT is externally controlling the outputs. This readback of the actual enable signals to the tristate output drivers can be performed using Register Access Mode (Chapter 4). The SYSTem :TRISTate? readback command will return the current programmed state of the internal tristate control settings. The state of output drivers is known if the internal settings are INput or OUTput, but hardware readback must be used to determine the driver status if internal settings are EXTNORMal or EXTINVerted.

Appendix B shows the receiver circuit for the external tristate control inputs. They are internally pulled up to logic high (1) if left unconnected. These external inputs are ignored after a power up condition, a hardware reset, or a VXI soft reset, to avoid spurious enabling of output drivers. They will not become effective until a field is defined with external control enabled.

Request Handshake Control Pins

Each connector has two request handshake signals. The names of these signals are assigned from the IO50/IO100 “point of view”. The Byte Request input comes from the UUT and requests the IO50/IO100 to output data (bytes). The Byte Available input from the UUT informs the IO50/IO100 that it should read in data (bytes). These inputs can be inverted under software control. The inputs are non-latching, and are meant to be used as interlocked handshakes. These handshake lines can be readback by the Slot 0 Controller in the shared registers. Each of these handshake lines can generate an interrupt under control of a separate hardware mask register. The readback function displays which connector signal is currently active. A high (1) bit in the readback register indicates an active request, a low (0) indicates an inactive connector signal. The request inputs must be received low (0) to cause an active signal status. The software inversion capability must therefore be used when UUTs request data transfers with rising edge signals.

The local 68000 MPU normally uses the interrupt capability to perform handshake operations. Program options allow interrupts to be redirected to the Slot 0 Controller for high level control. An interrupt will occur once per data transfer if the acknowledge handshake (see below) polarity is inverted. This one interrupt will occur when the request handshake is active and the acknowledge handshake has not yet been set. Setting the acknowledge in this situation will disable interrupts until the UUT removes the request. At that time, hardware gating will also remove the acknowledge handshake. Two interrupts occur per data transfer if the acknowledge handshake polarity is normal. The first interrupt occurs when the UUT request is active, and no acknowledge has been set. This interrupt will be cleared when the acknowledge handshake is set true (high), and a second interrupt then occurs when the UUT removes it’s request. The second interrupt is cleared when the acknowledge output is written false (low). This sequence provides a fully interlocked handshake with the UUT regardless of polarity. A fail safe timer is incorporated in the IO50/IO100 hardware to insure that commands (i.e.: ABORt or INITiate) can be executed even when servicing excessive UUT interrupts.

The request handshake lines each have a paired ground connection within the connector/cable interface. This ground is returned to system logic ground in non-isolated versions, and to the isolated return path in isolated versions. These paired grounds should be connected to the corresponding UUT signal. These grounds, and the physical separation from the data lines, allow the handshake signals to maintain correct logic levels despite switching of multiple data signals. Appendix B shows the receiver circuit for the request handshake inputs. They are internally pulled up to logic high (1) if left unconnected.

Acknowledge Handshake Control Pins

Each connector has a unique pair of acknowledge handshake signals that are output during both “handshake” and “timed” typed tests. These signals are named, as are the requesting handshakes, from the IO50/IO100 perspective on data transfer. The Data Valid signal is output by the IO50/IO100 to inform the UUT that data output (in response to the Byte Request signal) is now available on the data pins. The UUT would typically use Data Valid to cancel the Byte Request and strobe data into its input latching hardware. The Data Acknowledge is output from the IO50/IO100 to the UUT to indicate data sent with a Byte Available request has been strobed into the IO50/IO100 receiver latches. The UUT would normally use Data Acknowledge to cancel the Byte Available request and start generating the next output data value. These acknowledge handshake signals are output during “timed” type tests, even though no requesting handshake is required. This allows UUTs to strobe data in or out in under control of the IO50/IO100. A delay parameter is provided in these tests to stretch the width of the strobe pulse. This delay, in conjunction with the programmable polarity, can be used to adjust setup and hold times to meet UUT requirements.

The acknowledge outputs can be programmed for low or high true polarity. The high true output is initialized low (0) on power up or reset, and will go high to indicate data transfer can complete. The handshake signal output driver in this case is always enabled. When programmed for low true polarity, the acknowledge handshake output acts in an open collector fashion. In this mode, the output driver is initialized to a high impedance state. When a transfer can complete, the output driver is enabled to drive a low (0) output value. In either polarity mode, the acknowledge handshake is removed when the request is removed, as required by interlocked operation. In the inverted mode however, the acknowledge is removed much sooner, delayed only by gate propagation time. This allows memory emulation operation with devices capable of starting successive memory operations within a very short time. Figure 2-1, 2-2 and 2-3 show the details of interlocked handshake operations.

Appendix B shows the driver configuration for the acknowledge handshake outputs. The on-board pullup resistor is socketed to allow user modification. The weak pullup (10K) standard part can usually be left in parallel with existing pullup resistors on the UUT without overloading driver outputs. Rise time for operation in the inverted output mode may be excessive unless a smaller resistor is in parallel on the UUT.

The acknowledge handshake lines each have a paired ground connection within the connector/cable interface. This ground is returned to system logic ground in non-isolated versions, and to the isolated return path in isolated versions. These paired grounds should be connected to the

corresponding UUT signal. These grounds, and the physical separation from the data lines, allow the handshake signals to maintain correct logic levels despite switching of multiple data signals.

Power Pins

Each connector has two power pins available to provide limited logic power for external devices. These pins are connected directly to the VXI cardrack +5V supply, and in conjunction with the many paired ground signals mentioned above, allow operation of industry standard I/O module racks with a single cable connection. The power pins are protected with self resetting fuses (Raychem Polyswitch, RBE110A). Each connector pair (A-B, C-D) is protected by a 1 amp fuse, providing a total of 2 amps of +5V power. If an overload condition occurs, the fuses will trip within 0.05 to 10 seconds, depending on the current load, and will reset within 20 seconds after the condition is corrected.

VXI/VME Connections

VME Interrupt Request Connections

The IO50/IO100 has interrupt capability and can be programmed to drive any one of the seven backplane interrupt request lines. The assignment and programming of interrupt lines is normally handled by the Resource Manager configuration routine using VXI Word Serial commands. Refer to the Slot 0 controller documentation for details of interrupt assignments. Chapter 3, Command Set, contains detailed information about interrupt enable and status readback functions. Refer to the STATUS commands.

TTL Trigger Connections

The IO50/IO100 can be programmed to source or receive signals on any of the eight VXI TTLTRG backplane signals. These trigger lines are used to provide synchronized input and output of data across multiple boards in the same cardrack, or to an external trigger source. These trigger lines are connected for input or output in pairs, one each for input and output. The BASIC MODE command will configure signals for use between IO50/ IO100 boards acting in a master/slave arrangement. Any signal which can source the TTLTRG lines can determine input or output timing in this manner. Chapter 4 on Shared Register programming provides information on selecting the TTLTRG lines for specific applications.

(THIS PAGE INTENTIONALLY LEFT BLANK)

CHAPTER 3

Command Set

SCPI Command Syntax

The IO50/IO100 is a message-based Digital I/O instrument with a command structure patterned after the Standard Commands for Programmable Instrument (SCPI) syntax. SCPI commands are defined in a tree structure starting with a basic command function, called the command root, and expanding functions by adding additional command descriptors, called command branches, until the final control parameter, called a command leaf, is defined. In this way, commands may be logically grouped together based on function. In some instances a command branch may also be a leaf. Instances of this type will be pointed out.

The IO50/IO100 is an advanced, full feature Digital I/O module. Due to this, the standard SCPI command list is inadequate. Most of the commands for the IO50/IO100 are not defined within the SCPI document itself, or are a modification of a defined SCPI command.

A command quick reference list of the IO50/IO100 command set, and a command list key, are provided in the next few pages. Following the command key are the descriptions of the IO50/IO100 commands and their associated parameters.

Commands are grouped together based on functionality, and listed by alphabetical order or the command root. To assist in keeping track of the root command and its branches while negotiating through the various command levels, the complete command syntax up to the current level is provided in the header space of each page. Use the command quick reference found on the following pages to help identify commands by root, branch and leaf.

The command key in Table 3-1 on the following page provides a summary of the different components that make up a full command sequence. Note that command roots do not have a prefix, while branches and leaves are preceded by a colon. Except where noted, multiple commands may be sent on a single line. Also, certain commands are default commands and are optional. These commands are shown within brackets [].

Table 3-1. SCPI Command Key

ROOT	Signifies a command root. Branches and/or leaves may follow, or the root may be a solitary command, in which case a program example is provided. Root commands that are not solitary will not have examples.
:BRANCH	Signifies a command branch, preceded by a command root and followed by another command branch or a command leaf. Command branches may have modifying parameters. Examples are not provided for command branches.
:LEAF	Signifies the termination of a command sequence. Examples are always provided for command leaves.
command?	Any command followed by a question mark indicates a command query.
<required>	Required parameter.
[option]	Optional parameter.
{repeat}	repeat as many times as are needed.
(min - max)	Value entered must be within the range of min to max, inclusive.
aaa | bbb	Acceptable choices are aaa OR bbb (OR ccc OR ddd...).
response	Response from IO50/IO100.

Standard Commands for Programmable Instruments

```
ABORT
FIELD ---------- :DEFINE ---------- :PINASSIGNMENT
 ---------- :NAME ---------- :TRISTATE(?)
 ---------- :DELETE
 ---------- :CATALOG?
INITIATE -------- :INPUT
 ---------- :OUTPUT
 ---------- :BLOCK
TEST ---------- :DEFINE ---------- :MEMEMULATION ------- [:SIZE]
 ---------- :BLKOUTHANDSHAKE -- [:SIZE]
 ---------- :BLKINHANDSHAKE ----- [:SIZE]
 ---------- :BLKOUTTIMED ---------- [:SIZE]
 ---------- :BLKINTIMED ---------- [:SIZE]
 ---------- :PRGIOHANDSHAKE
 ---------- :PRGIOTIMED
 ---------- :NAME ---------- :STATUS?
 ---------- :DELETE
 ---------- :CATALOG?
 ---------- [:HANDSHAKE] --------- :REQUEST ---------- [:INPUT](?)
 ---------- :OUTPUT(?)
 ---------- [:ACKNOWLEDGE] ------- :INPUT(?)
 ---------- :OUTPUT(?)
 ---------- :TIMEOUT ---------- [:INPUT](?)
 ---------- :OUTPUT(?)
 ---------- :ADDR(?)
 ---------- :FREE?
VECTOR --------- :COUNT
 ---------- :DATA ---------- [:VALUE](?)
 ---------- :RADIX
 ---------- :FIELD
SYSTEM ---------- :ERROR?
 ---------- :VERSION?
 ---------- :FIELD(?)
 ---------- [:TEST](?)
 ---------- :LEARN(?)
 ---------- :TRISTATE?
```

Quick Reference Key	
?	Query Only
(?)	Command or Query
[]	Optional Command

Standard Commands for Programmable Instruments

```
STATUS ---------- [:OPERATION] ---------- [:EVENT]?
 ------------------------------------------------- :CONDITION?
 ------------------------------------------------- :ENABLE(?)
 ------------------------------------------------- :TEST ---------- [:EVENT]?
 ------------------------------------------------------------------------------ :CONDITION?
 ------------------------------------------------------------------------------ :ENABLE(?)
 ------------------------------------------------------------------------------ :ISUMMARY1 ---- [:EVENT]?
 ------------------------------------------------------------------------------------------ :CONDITION?
 ------------------------------------------------------------------------------------------ :ENABLE(?)
 ------------------------------------------------------------------------------ :ISUMMARY2 ---- [:EVENT]?
 ------------------------------------------------------------------------------------------ :CONDITION?
 ------------------------------------------------------------------------------------------ :ENABLE(?)
 ------------------------------------------------------------------------------ :ISUMMARY3 ---- [:EVENT]?
 ------------------------------------------------------------------------------------------ :CONDITION?
 ------------------------------------------------------------------------------------------ :ENABLE(?)
 ------------------------------------------------------------------------------ :ISUMMARY4 ---- [:EVENT]?
 ------------------------------------------------------------------------------------------ :CONDITION?
 ------------------------------------------------------------------------------------------ :ENABLE(?)
BASICMODE ---- :DEFINE ---------- :INPUT
 ------------------------------------------------- :OUTPUT
 ---------- :CATALOG?
 ---------- :CLEAR
 ---------- :INPUT?
 ---------- [:OUTPUT](?)
 ---------- :MODE(?) ---------- :SLAVE ---------- [:GROUP]
 ------------------------------------------------- :MASTER ---------- [:GROUP]
 ------------------------------------------------- :STANDALONE
```

IEEE 488.2 Common Commands

*CLS	*SAV
*ESE (?)	*SRE(?)
*ESR?	*STB?
*IDN?	*TRG
*OPC (?)	*TST?
*RCL	*WAI
*RST	

ABORT

Function: Halts currently executing handshake test.

Syntax: ABOR[T]

Remarks: This command will halt currently executing tests using handshake transfer control . These tests are MEMEMULATION, BLKOUTHANDSHAKE, BLKINHANDSHAKE and PRGIOHANDSHAKE . The ABORT command cannot stop the timed type tests since they, once started, run to completion before any new commands take effect . The TEST:NAME ALL:STATUS? command can be sent prior to ABORT to determine if any test is currently executing . The TEST:NAME ALL:STATUS? can also be sent after ABORT to determine last vector transferred before the test was halted . The ABORT command will typically be used to recover from a software timeout when a UUT fails to provide the expected handshake signals to transfer data . A fail safe hardware timer is used in the IO50/ IO100 to insure that the abort command is executed within 200 ms, even if the UUT is generating constant interrupts via the handshake request signals.

Example: ABORT

See Also: INITIATE:BLOCK, INITIATE:OUT, INITIATE:IN

FIELD

Function: Allows a name to be assigned to a logical group of connector pins and to defines that group of pins as input or output.

Syntax: FIEL[D]

Remarks: Fields are associated with whatever test is currently active when the fields are defined . There are four possible tests (A | B | C | D) . Pins assigned to fields associated with test A will have their data flow controlled by the handshake lines of connector A, and so forth for B, C, and D . Field names are local to their test, and the same name can be used in other tests with a different definition . Pins can be assigned to multiple fields, even if the fields are associated with different tests . The only restriction is that fields cannot define pins in the same byte with different input/output characteristics.

The first field defined when a particular test is active becomes the active field in that test . All subsequent field commands will act on that field . If that field is deleted, a new active field must be defined, or FIELD commands will generate a “no active field” error . The SYSTEM:FIELD command is used to change or determine the active field . The SYSTEM:TEST command is used to change or determine the active test . The FIELD:NAME:CATALOG? command will display all currently defined fields.

See Also: SYSTEM:FIELD[?], SYSTEM:TEST[?]

FIELD

:DEFINE

Function: Creates a user defined name for a logical grouping of data pins . Define is also the branch to PINASSIGNMENT.

Syntax: DEF[INE] <name>

Range: Naming string can be 1 to 8 characters . Any alpha or numeric character, or the underscore is permitted . Name may start with any of these characters.

Default: None

Remarks: A field is “attached” to whatever test is currently active when it is defined . If no test is defined, an error is generated when attempting to define a field . The first field defined after initialization becomes the active field . The active field can be changed only with the SYSTEM:FIELD command . The DEFINE command is followed by a PINASSIGNMENT command and a pin list . Once defined, a field cannot be modified by further define commands . The field must be deleted using the FIELD:NAME:DELETE command, then redefined with the desired pins.

Example: FIELD:DEFINE COUNT_4:PINASSIGNMENT C2-1,A32-24,A16-1

See Also: FIELD:NAME, SYSTEM:FIELD

FIELD:DEFINE

:PINASSIGNMENT

Function: Assign physical connector data pins to a logical field name.

Syntax: PIN[ASSIGNMENT] <pin_list>

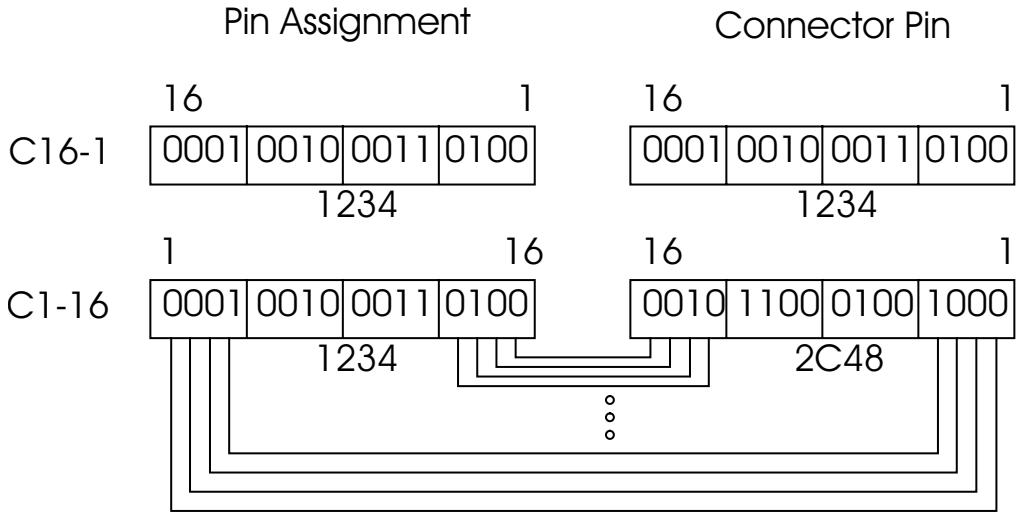
Range: Connector letters A | B | C | D, pin numbers 1 to 32.

Default: None

Remarks: Pin assignment not only attaches connector pins to a named data field, it assigns the order in which data is assigned to those pins . The order in which pins are specified in the pin list, from left to right, will be the order in which data bits are assigned, from most significant to least significant . If a pin list of C16-1 is specified, and an data vector of hexadecimal 1234 is output by that field, data pins 16-9 will present 12 and pins 8-1 will present 34 . If the field pin list specifies C1-16, and the same data vector (1234) is sent, pins 16-9 will present 2C and pins 8-1 will present 48 . See diagram below.

Pin assignment can include ranges of pins, skip pins, and include pins from more than one connector . The total number of pins that can be assigned to one field is 32 . Commas separate individual pins and hyphens indicate an inclusive range of pins.

Example: FIELD:DEFINE UART_ADD:PINASSIGNMENT C31,C20,A26-24,A8



FIELD
:NAME

Function: Used to select previously defined fields for inquiry, deletion, or modification of pin input/output control.

Syntax: NAME

Range: Any field name already defined by a FIELD:DEFINE command or ALL.

Default: None

Remarks: This branch command performs no direct action, but allows a field to be selected for action by subsequent branch commands . The name ALL can be used to effect all defined fields . To determine the existing pin assignments and field names, the FIELD:NAME ALL:CATALOG? command can be used.

Example: FIELD:NAME COUNT_4:TRISTATE OUTPUT
FIELD:NAME ALL:DELETE

FIELD:NAME
:TRISTATE

Function: Sets control of pin drivers to forced off, forced on, or external control . Allows current control setup to be cleared.

Syntax: TRIST[ATE] <control>

Range: IN[PUT] | OUT[PUT] | EXTNORM[AL] | EXTINV[ERTED] | CLE[AR]

Default: UNDEFINED

Remarks: All data pins in a defined field can be used for input or output . Separate receivers and drivers are used, so input capability is always available . The output drivers can be forced on or off internally, or placed under control of the external tristate control signals . All pins will have their output drivers forced off (undefined mode) after power up or reset . Since the pin driver hardware uses byte wide devices, each pin in an octet must be assigned the same tristate control value . Attempting to program the tristate control of a field to input will cause an error if another field has already defined pins within the same octet as output.

The octets within each connector are pins 1-8,9-16,17-24, and 25- 32 . Each of these has its own external tristate control signal with an internal pull up resistor . When outputs are programmed for EXTNORMAL, data pins will be in high impedance state when the external tristate control signals are high . Outputs will be enabled when these signals are brought low . When outputs are programmed as EXTINVERTED, the polarities are reversed . The TRISTATE? query command can be used to determine the current software settings of the tristate controls.

Example: FIELD:NAME COUNT_6:TRISTATE OUTPUT
FIELD:NAME ALL:TRISTATE CLEAR

FIELD:NAME
:TRISTATE?

Function: Query the current software settings for pin control in specified field.

Syntax: TRIST[ATE]?

Response: field_name INPut | OUTput | EXTNORMal | EXTINVerted | UNDEFINED

Remarks: None

Example: FIELD:NAME COUNT_6:TRISTATE?
OUT*put*

FIELD:NAME
:DELETE

Function: Deletes named field.

Syntax: DEL[ETE]

Remarks: The field specified by name is deleted . All pins assigned to only that field will become free, and their outputs will revert to undefined . Pins that are shared by other fields will keep their current control values . If the field deleted was the currently active field, a new active field must be specified before any VECTOR:DATA:VALUE commands can be executed . The SYSTEM:FIELD command can be used to determine and assign the active field .

Example: FIELD:NAME COUNT_4:DELETE

See Also: SYSTEM:FIELD, VECTOR:DATA:VALUE

FIELD:NAME

:CATALOG?

Function: Read information about named fields.

Syntax: CAT[ALOG]?

Response: field_name, INput | OUTput | EXTNORMal | EXTINVerted | UNDEFINED, pin_list.

Remarks: Pins are listed in the order they were specified in the PINASSIGNMENT command . If the NAME command preceding CATALOG? specified a particular field name, only data pertinent to that field is returned . If the NAME command specified ALL, data for all fields is returned, with semicolons separating field data strings . The data strings will be returned for multiple fields in the order they were defined.

Example: FIELD:NAME COUNT_4:CAT:
COUNT_4,OUTput,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B15,B15,B16

See Also: FIELD:DEFINE:PINASSIGNMENT

INITIATE

Function: Causes active test to begin data transfer operation.

Syntax: INIT[IATE]

Range: INPUT | OUTPUT | BLOCK

Default: Input

Remarks: This command will start the active test . It is required for each data transfer in the Programmed I/O test, and once for each block of data transferred in the Block Input and Output tests . The various setup and configuration commands will not actually effect any connector pins until this command is executed . Once a test has been started with the Initiate command, it will proceed to completion or until the ABORT command is sent . The Abort command will not be effective in Timed type tests, as they execute to completion before any pending commands are processed . The active test can be changed using the SYSTEM:TEST command.

In general, attempting to initiate a test while another test is already executing will generate an error . The exception to this is that Timed tests may be INITIATED if a Memory Emulation or Handshake test is already executing . This allows execution of diagnostic code loops while still providing stimulus and response capability with the UUT . Handshake and Memory Emulation tests are interrupt driven . While waiting for handshake interrupts, the microprocessor is free to process commands, including initiating a Timed test . However, Timed tests are microprocessor intensive and always run to completion before any new commands are processed . So it is not possible to initiate a Handshake or Memory Emulation test while a Timed test is executing . Also, to avoid interrupt conflicts, only one Handshake or Memory Emulation test may be executing at any time . The operating status of executing tests can be verified using the TEST:NAME ALL:STATUS? command . Data vector locations can be read and written while tests are executing using the VECTOR:DATA commands, or direct access to shared RAM . The Operation section has more information on the data format of shared RAM.

Example: INITIATE:BLOCK

See Also: TEST, TEST:NAME:STATUS?, ABORT, INITIATE:BLOCK, INITIATE:INPUT, INITIATE:OUTPUT

INITIATE

:INPUT

Function: Cause one data value input from the UUT during Programmed I/O tests.

Syntax: IN[PUT]

Remarks: The programmed I/O test uses a single input data vector location in shared RAM . The INITIATE:INPUT command will cause the IO50/IO100 to read all connector data pins, and store the result in that location . All 128 bits of data read in will stored in the 16 byte vector location . When data is read back using the VECTOR:DATA:VALUE? command, it will be formatted according to the current active field attached to the active test . An error is generated if no field has been defined . The SYSTEM:FIELD command can be used to determine and change the active field . If the Programmed I/O Handshake test is active, the data transfer will not take place until the UUT indicates it has valid output data by activating the IO50/IO100 input request signal . If the INPUT command is sent again before this occurs, an error is generated . If the programmed I/O Timed test is executing, data is input without waiting for a UUT request . Once the test has been defined, and the vector space allocated, repeated reads are performed by sending INITIATE:INPUT and VECTOR 1:DATA:VALUE? commands.

Example: INITIATE:INPUT

See Also: TEST, FIELD, VECTOR:DATA, HANDSHAKE

INITIATE

:OUTPUT

Function: Cause one data value output from the IO50/IO100 during Programmed I/O tests.

Syntax: OUT[PUT]

Remarks: The programmed I/O test uses one output data vector location in shared RAM . The INITIATE:OUTPUT command will cause the IO50/IO100 to output that data vector to the UUT . Data stored in that vector, using the VECTOR:DATA:VALUE command, will have been formatted according to field definitions . The IO50/IO100 output registers for all 128 pins are updated with the 16 byte output vector data, however only pins with output capability will reflect this update . The TRISTATE command is used to enable the pins for output . If the Programmed I/O Handshake test is active, the data transfer will not take place until the UUT indicates it is ready to receive data by activating the IO50/IO100 output request signal . If the OUTPUT command is sent again before this occurs, an error is generated . If the programmed I/O Timed test is executing, data is output without waiting for a UUT request . Once the test has been defined, and the vector space allocated, repeated writes are performed by sending VECTOR 1:DATA:VALUE xxxx, INITIATE:OUTPUT commands.

Example: INITIATE:OUTPUT

See Also: TEST, FIELD, VECTOR:DATA, HANDSHAKE

INITIATE
:BLOCK

Function: Cause a block of data values to be output by the IO50/IO100.

Syntax: BLOCK

Remarks: This command will cause the IO50/IO100 to transfer multiple data vectors to the UUT . Output of data values will be performed from the lowest vector number to the highest . Input of data values will fill shared memory vector locations in the same order . At the completion of a block transfer, the test status will change from EXECUTING to STOPPED, and no further data is transferred until the next INITIATE command.

Output data will be formatted according to the fields attached to the active test . Input is performed on all 128 connector pins and stored in the 16 byte vector location . When read back using the VECTOR:DATA:VALUE? command, it will be formatted according to field definition . An error is generated if no field has been defined . If the Block Input or Output Handshake tests are active, the data transfer will not take place until the UUT indicates it is ready by activating the IO50/IO100 request signals . The TEST:NAME:STATUS? command can be used to determine if the UUT is requesting transfer . If the UUT is not performing as expected, the ABORT command can be used to halt the test . Test execution time for block tests is dependent on the number of vectors transferred, and the UUT request rate .

If the Block Input and Output Timed tests are executing, data is output without waiting for a UUT request . Test time in this mode is dependent on the delay programmed with the TIMEOUT command and the number of vectors transferred . No status readback or abort of timed tests is possible . Data cannot be modified during timed type tests, since commands are not recognized until the test completes . Memory Emulation is a special case of block handshake testing, allowing both input an output with a common shared RAM area, and running continuously from INITIATE until an ABORT command.

Example: INITIATE:BLOCK

See Also: TEST, FIELD, VECTOR:DATA, INITIATE:IN, INITIATE:OUT, HANDSHAKE, MEMEMULATION, BLKINHANDSHAKE, BLKOUTHANDSHAKE, BLKINTIMED, BLKOUTTIMED

TEST

Function: Assigns a specific test function to one of four available connector signal groups.

Syntax: TEST

Remarks: This root command has no direct action, but used in conjunction with subsequent branch commands DEFINE and NAME, can assign or modify modes of operation . Four tests can be defined at once, but only one can be active at any time . The test names correspond to connector letters because each connector has a set of handshake signal lines . These handshake signals are used to control data flow within the various test types . The test operation defined as A will use the set of handshake signals on connector A to move data between the UUT and the IO50/IO100, however, data pins from any of the four connectors may be included under any test . The pins to be used in each test are defined in FIELD commands . Any fields defined while a given test is the active system test will be attached to that test . The first field defined under a given test will be come the active field for that test . Each test will have its own active field . The SYSTEM:TEST command can be used to set or determine the active test . The SYSTEM:FIELD command can be used to determine or set the active FIELD for the active test . The Operation section describes the theory of operation for the various test types.

Example: TEST:DEFINE A:BLKOUTHANDSHAKE:SIZE 100

See Also: FIELD

TEST

:DEFINE

Function: Assign a test operation to one of the four test control groups.

Syntax: DEFINE <test_name>

Range: A | B | C | D

Default: None

Remarks: The first test defined after power up or reset becomes the active test . When fields are defined, they are attached to the currently active test . To determine or change the currently active test, the SYSTEM:TEST command can be used . If the active test is deleted, another must be assigned before any field definition or initiate commands can be executed . Multiple tests may be defined, but the INITIATE and ABORT commands will effect only the active test.

Example: TEST:DEFINE A:PRGIOHANDSHAKE

See Also: TEST, TEST:NAME, FIELD, FIELD:DEFINE

TEST:DEFINE

:MEMEMULATION

Function: Sets defined test to perform memory emulation operation.

Syntax: MEMEMU[LATION]

Remarks: This type of test provides a random data access capability allowing the UUT to address memory locations in the IO50/IO100 . Handshake signals and tristate controls can be used to facilitate direct connection of IO50/IO100 data pins to UUT internal data bus structures . The depth of memory to be emulated can be from 1 to 8196 data values . These data values can be from 1 to 128 bits wide . The subsequent SIZE command will allocate internal storage for the user supplied data values . The test is started using the INITIATE:BLOCK command and continues to execute until the ABORT command is sent . The Programmed I/O Output Timed test operation can be performed while the Memory Emulation test is executing . In that case, the SYSTEM:TEST command is used to switch the active test before sending the INITIATE and ABORT commands.

See Also: INITIATE:BLOCK, ABORT, SYSTEM:TEST

TEST:DEFINE:MEMEMULATION

:SIZE

Function: Allocate data memory space for Memory Emulation.

Syntax: SIZE <decimal>

Range: $2^{(0\text{-}13)}$

Default: None

Remarks: Each of the defined tests require data storage in the shared RAM area . This memory is allocated from a common pool . Each block type test defined will use the amount of vector space indicated by the size parameter . Each data vector takes up 16 bytes of shared RAM, in the format shown in Section 5, Register Programming . The Memory Emulation uses the same memory locations for input and output data, giving the appearance of RAM from the UUT perspective . Once allocated, the VECTOR:DATA can be used to fill the shared memory space with “read” data for the UUT . The maximum size is 8192 data vectors for memory emulation, and the data word size is from 1 to 32 bits per field . For output words up to 128 bits, multiple fields may be defined . To delete inactive tests and return allocated memory to the free pool, use the TEST:NAME:DELETE command.

Example: TEST:DEFINE D:MEMEMULATION:SIZE 512

See Also: FIELD, VECTOR:DATA

TEST:DEFINE

:BLKOUTHANDSHAKE

Function: Sets defined test letter to block data output operation.

Syntax: BLKOUTH[ANDSHAKE]

Remarks: This test operation provides the capability to move a block of data from the IO50/IO100 shared memory area to the UUT, with data flow controlled by the UUT . An interrupt can be sent to the Slot 0 Controller after the last data value has been output . The STATUS commands are used to enable this interrupt . This test can provide a FIFO-like speed decoupling between the IO50/IO100 and some other data receiving device . The shared memory data values can be filled by DMA or direct write from the Slot 0 Controller, then passed out at an asynchronous slower rate . Another use of the block output function is to output data vectors at a higher rate than would be possible using high level commands . The Operation section has more information about the Block Out Handshake function . This operation is normally used only with UUTs containing some intelligent control over their incoming data flow.

See Also: TEST, STATUS:OPERATION, INITIATE, ABORT, TEST:NAME

TEST:DEFINE:BLKOUTHANDSHAKE
:SIZE

Function: Allocate data memory space for the block data output test function.

Syntax: SIZE <decimal>

Range: The lesser of (1-15,360 | free_memory)

Default: None

Remarks: Each of the defined tests require data storage in the shared RAM area . This memory is allocated from a common pool . Each block type test defined will use the amount of vector space indicated by the size parameter . Each data vector takes up 16 bytes of shared RAM, in the format shown in Section 5, Register Programming . Once allocated, the VECTOR:DATA can be used to fill the shared memory space with output data for the UUT . The maximum data word size is from 1 to 32 bits per field . To delete inactive tests and return allocated memory to the free pool, use the TEST:NAME:DELETE command.

Example: TEST:DEFINE A:BLKOUTHANDSHAKE:SIZE 150

See Also: TEST, FIELD, VECTOR:DATA, TEST:FREE?

TEST:DEFINE

:BLKINHANDSHAKE

Function: Sets defined test letter to perform the block data input operation.

Syntax: BLKINH[ANDSHAKE]

Remarks: This test operation provides the capability to move a block of data from the UUT to the IO50/IO100 shared memory area, with data flow controlled by the UUT . An interrupt can be sent to the Slot 0 Controller after the last data value has been input . The STATUS commands are used to enable this interrupt . This test can provide a FIFO-like speed decoupling between the IO50/IO100 and some other data transmitting device . The shared memory data values can be filled by the UUT at some rate it determines, then read by the Slot 0 using hardware DMA, direct read operations or VECTOR:DATA:VALUE? . Another use of the block input function is to transfer data vectors at a higher rate than would be possible using high level commands . This operation is normally used only with UUTs containing some intelligent control over their outgoing data flow.

See Also: STATUS:OPERATION, INITIATE, ABORT

TEST:DEFINE:BLKINHANDSHAKE
:SIZE

Function: Allocate data memory space for block input data.

Syntax: SIZE <decimal>

Range: The lesser of (1-15,360 | free_memory)

Default: None

Remarks: Each of the defined tests require data storage in the shared RAM area . This memory is allocated from a common pool . Each block type test defined will use the amount of vector space indicated by the size parameter . Each data vector takes up 16 bytes of shared RAM, in the format shown in Section 5, Register Programming . The user may use the VECTOR:DATA command to initialize the data to some other value before beginning the test operation . The maximum data word size is from 1 to 32 bits per field . For input words up to 128 bits, multiple fields may be defined . To delete inactive tests and return allocated memory to the free pool, use the TEST:NAME:DELETE command.

Example: TEST:DEFINE B:BLKINHANDSHAKE:SIZE 150

See Also: FIELD, VECTOR:DATA, TEST:FREE?

TEST:DEFINE

:BLKOUTTIMED

Function: Sets defined test to timed block data output operation.

Syntax: BLKOUTT[IMED]

Remarks: This test operation provides the capability to move a block of data from the IO50/IO100 shared memory area to the UUT, with data flow at a programmed rate . An interrupt can be sent to the Slot 0 Controller after the last data value has been output . The STATUS command is used to enable this interrupt . This test can provide output to UUT devices with clocked or latched input data requirements, but no local intelligence to control data flow . The IO50/IO100 will provide a clocking strobe with each data value output . The TEST:NAME:TIMEOUT command can be used to control the rate of data output and adjust the data setup or hold time in relation to the strobe output . The shared memory data values for output can be filled by direct write from the Slot 0 Controller, or using the VECTOR:DATA command . Once the test is started, no other commands will be executed until the entire block has been output.

See Also: TEST, STATUS:OPERATION, INITIATE, TEST:NAME

TEST:DEFINE:BLKOUTTIMED
:SIZE

Function: Allocate data memory space for block data output.

Syntax: SIZE <decimal>

Range: The lesser of (1-15,360 | free_memory)

Default: None

Remarks: Each of the defined tests require data storage in the shared RAM area . This memory is allocated from a common pool . Each block type test defined will use the amount of vector space indicated by the size parameter . Each data vector takes up 16 bytes of shared RAM, in the format shown in Section 5, Register Programming . The user may use the VECTOR:DATA command to initialize the data to some other value before beginning the test operation . The maximum data word size is from 1 to 32 bits per field . For input words up to 128 bits, multiple fields may be defined . To delete inactive tests and return allocated memory to the free pool, use the TEST:NAME:DELETE command.

Example: TEST:DEFINE A:BLKOUTTIMED:SIZE 4

See Also: TEST, FIELD, VECTOR:DATA, TEST:FREE?

TEST:DEFINE

:BLKINTIMED

Function: Sets defined test letter to block data input operation.

Syntax: BLKINT[IMED]

Remarks: This test operation provides the capability to move a block of data from the UUT to shared memory in the IO50/IO100, with data flow at a programmed rate . An interrupt can be sent to the Slot 0 Controller after the last data value has been transferred . The STATUS command is used to enable this interrupt . This test can provide input from UUT devices which must have data clocked out, but have no local intelligence to control data flow . The IO50/IO100 will provide a clocking strobe for each data value input . The TEST:NAME:TIMEOUT command can be used to control the rate of data input and adjust the data setup or hold time in relation to the read strobe . The shared memory data are filled as they are read from the UUT, and can then be read directly by the Slot 0 Controller, or using the VECTOR:DATA command . Once the test is started, no other commands will be executed until the entire block has been input.

See Also: STATUS:OPERATION, INITIATE

TEST:DEFINE:BLKINTIMED
:SIZE

Function: Allocate data memory space for block data input .

Syntax: SIZE <decimal>

Range: The lesser of (1-15,360 | free_memory)

Default: None

Remarks: Each of the defined tests require data storage in the shared RAM area . This memory is allocated from a common pool . Each block type test defined will use the amount of vector space indicated by the size parameter . Each data vector takes up 16 bytes of shared RAM, in the format shown in Section 5, Register Programming . The user may use the VECTOR:DATA command to initialize the data to some other value before beginning the test operation . The maximum data word size is from 1 to 32 bits per field . For input words up to 128 bits, multiple fields may be defined . To delete inactive tests and return allocated memory to the free pool, use the TEST:NAME:DELETE command.

Example: TEST:DEFINE A:BLKINTIMED:SIZE 4

See Also: FIELD, VECTOR:DATA, TEST:FREE?

TEST:DEFINE
:PRGIOHANDSHAKE

Function: Sets specified test for handshake controlled input/output.

Syntax: PRGIOH[ANDSHAKE]

Remarks: This test provides single data value input and output, with the UUT controlling data flow . Data locations for input and output are separate, much like a full duplex communication peripheral . To perform output, the Slot 0 Controller will first prepare an output data vector . The VECTOR:DATA command can be used, or direct memory access to the vector location in shared memory . The INITIATE:OUT command is then sent to attempt data output . The IO50/IO100 will then wait for an output request on the handshake input from the UUT before transferring data . The TEST:NAME:STATUS? command can be used by the Slot 0 Controller to determine if the data has been transferred . Alternatively, the STATUS command can be used to enable an interrupt when the UUT has received the pending data value and is ready for the next . If an attempt is made to send a second data value before the first has been transferred, an error is generated . At any time, the ABORT command may be sent to halt a pending transfer.

Input operation is similar, with the UUT sending a read request to the IO50/IO100 when it has data available . If the INITIATE:IN command has been executed, this request will cause data to be read to the input vector location . If the input interrupt is enabled, a Slot 0 interrupt is sent after the data has been read . The TEST:NAME:STATUS? command can be used to determine if any data has been transferred . The Slot 0 Controller may access the input data with the VECTOR:DATA command, or direct read of shared memory.

An acknowledge signal is sent to the UUT at the completion of each data transfer, forming an interlocked handshake sequence . For output data, the acknowledge is sent when valid data is available on the IO50/IO100 output pins . For input, the acknowledge handshake is sent when the UUT data has been latched in from the IO50/IO100 input pins . The Operation section contains more information on the Programmed I/O Handshake operation and the use of handshake signals.

Example: TEST:DEFINE C:PRGIOHANDSHAKE

See Also: INITIATE, ABORT, VECTOR

TEST:DEFINE

:PRGIOTIMED

Function: Sets specified test for timed input/output.

Syntax: PRGIOT[IMED]

Remarks: This test provides single data value input and output, with a strobe signal available to the UUT . Data locations for input and output are separate, much like a full duplex communication peripheral . To perform output, the Slot 0 Controller will first prepare an output data vector . The VECTOR:DATA command can be used, or direct memory access to the vector location in shared memory . The INITIATE:OUT command is then used to send data output and set the strobe signal to the opposite state . The IO50/IO100 will then wait for an programmed time interval before returning the strobe to its initial level . The STATUS command can be used to enable an interrupt when the UUT has received the pending data value and is ready for the next one . The TEST:NAME:TIMEOUT command is used to set the strobe timing value.

Input operation is similar, with INITIATE:IN command used to read data from the UUT . When the command is sent, the IO50/IO100 input strobe is set to the opposite state . After the programmed timeout value, the IO50/IO100 will latch data on its input pins and set the strobe back to its initial value . If the input interrupt is enabled, a Slot 0 interrupt is sent after the data has been read . The Slot 0 Controller may access the input data with the VECTOR:DATA command, or direct read of shared memory.

Example: TEST:DEFINE A:PRGIOTIMED

See Also: TEST:NAME:TIMEOUT, INITIATE, ABORT, VECTOR

TEST
:NAME

Function: Used to select previously defined tests for inquiry, deletion, or modification of configuration .

Syntax: NAME <test_name>

Range: A | B | C | D | ALL

Default: None

Remarks: This branch command performs no direct action, but allows a test to be selected for action by subsequent branch commands . The name ALL can be used to effect all defined tests . To determine the existing tests and their allocated memory, the TEST:NAME ALL:CATALOG? command can be used.

Example: TEST:NAME A:TIMEOUT:INPUT 100

TEST:NAME
:STATUS

Function: Provide execution state and data transfer information on the named test .

Syntax: STAT[US]?

Response: A | B | C | D, EXECUTING | STOPPED, output_vector, input_vector

Range: Programmed I/O is the only test which returns both an input and an output vector number . This number ranges from 0 to 1 . Memory Emulation will have the number of the last vector accessed . This number can be from 0 to the maximum allocated vector number . The other tests will have either an input or output vector number, which will increase from 0 at the start of the test, to the last (highest) vector number allocated at the end of the test.

Remarks: Once a handshake type test is started with the INITIATE command, its completion can be determined by using the STATUS? query . This command is ineffective for monitoring timed type tests, since status will not be returned until the test completes . The last accessed vector numbers can be read after terminating the test with an ABORT command, to determine how much data was transferred.

Example: TEST:NAME A:STATUS?
A,EXECUTING,1,0

See Also: TEST:DEFINE, TEST:NAME:CATALOG, INITIATE, ABORT

TEST:NAME
:DELETE

Function: Deletes named test or all tests.

Syntax: DEL[ETE]

Remarks: The test specified by name is deleted and all memory allocated to that test will become free . If the test deleted was the currently active test, a new active test must be specified before any FIELD:DEFINE commands can be executed . The SYSTEM:TEST command can be used to determine and assign the active test . Tests cannot be deleted while they are executing, the ABORT command must be sent first to stop any operations in progress . Tests need not be deleted when another test is started, they can simply remain inactive . Tests are deleted when a new function is required of a connector handshake signal set already in use, or when more shared memory area is required by another test.

Example: TEST:NAME C:DELETE

See Also: TEST, FIELD, TEST:DEFINE, TEST:NAME:CATALOG?, ABORT

TEST:NAME
:CATALOG?

Function: Read information about named test or all tests.

Syntax: CAT[ALOG]?

Response: {A | B | C | D, MEMEMULATION | BLKOUTHANDSHAKE | BLKINHANDSHAKE | BLKOUTTIMED | BLKINTIMED | PRGIOHANDSHAKE | PRGIOTIMED, byte_offset, number_vectors.}

Remarks: The byte offset can be added to the shared RAM base address to directly access data vectors . Each data vector requires 16 bytes or memory in the format shown in section 5, Register Programming . If the NAME command preceding CATALOG? specified a particular test name, only data pertinent to that test is returned . If the NAME command specified ALL, data for all tests is returned in alphabetical order.

Example: TEST:NAME A:CAT?
A,MEMULATION,128,4096

See Also: TEST:DEFINE, TEST:NAME:STATUS?, VECTOR

TEST

:HANDSHAKE

Function: Optional command used in setting polarity of request and acknowledge handshake signal outputs to match UUT requirements.

Syntax: [HAND[SHAKE]]

Remarks: This command is optional, but may be used for documentation within a test program . The purpose of the command is to provide a pathway for setting the polarity of the request handshake signals from the UUT, and the acknowledge signals from the IO50/IO100.

See Also: PRGIOHANDSHAKE, BLKOUTHANDSHAKE, BLKINHANDSHAKE, MEMEMULATION

TEST:HANDSHAKE

:REQUEST

Function: Allow setting and readback request handshake polarity input from UUT.

Syntax: REQ[UEST]

Remarks: Request signals are input from the UUT to allow external asynchronous control of data flow . Separate request lines are available on each connector for read and write . The request handshake polarity command will effect only the polarity of the signal on the currently active test connector . The active test is changed with the SYSTEM:TEST command, and determined with the SYSTEM:TEST? command . The polarity control will not take effect until the INITIATE command is sent . Separate branch commands are available to invert the input and output request signals independently . Section 3, Operation shows timing information for the handshake signals and interlocked operation requirements.

TEST:HANDSHAKE:REQUEST
:INPUT

Function: Set request signal polarity for data input from the UUT.

Syntax: [IN[PUT]] <polarity>

Range: NORM[AL] | INV[ERTED]

Default: NORMAL

Remarks: The internal interrupt logic requires a low level signal to generate an interrupt . If the UUT request signal is low when it has valid data for the IO50/IO100 to read, normal polarity should be selected . If the UUT signal is high when presenting valid data, the inverted selection should be used . The UUT should observe interlocked handshake operation with the request signals, as shown in Section 3, Operation . The request signals are not latched, and should be held until an acknowledge signal is received back from the IO50/IO100.

NOTE: The request signal inputs are pulled high by resistors on board, so if they are unconnected and programmed for inverted polarity, **spurious interrupts will occur**.

Example: TEST:HANDSHAKE:REQUEST:INPUT NORMAL

See Also: TEST, REQUEST, INITIATE, REQUEST:INPUT?

TEST:HANDSHAKE:REQUEST
:INPUT?

Function: Read back current programmed setting for polarity of the input request signal.

Syntax: INPUT?

Response: A | B | C | D), INVERTED | NORMAL.

Remarks: Returns the last programmed request polarity for only the active test . The active test is changed with the SYSTEM:TEST command, and determined with the SYSTEM:TEST? command.

Example: TEST:HANDSHAKE:REQUEST:INPUT?
A,INVERTED

See Also: TEST, REQUEST, INITIATE, REQUEST:INPUT

TEST:HANDSHAKE:REQUEST
:OUTPUT

Function: Set the request output polarity.

Syntax: OUTPUT <polarity>

Range: NORM[AL] | INV[ERTED]

Default: NORMAL

Remarks: The internal interrupt logic requires a low level signal to generate an interrupt . If the UUT request signal is low when it requests valid data from the IO50/IO100, normal polarity should be selected . If the UUT signal is high when requesting valid data, the inverted selection should be used . The UUT should observe interlocked handshake operation with the request signals, as shown in Chapter 3, Operation . The request signals are not latched, and should be held until an acknowledge signal is received back from the IO50/IO100.

NOTE: The request signal inputs are pulled high by resistors on board, so if they are unconnected and programmed for inverted polarity, **spurious interrupts will occur**.

Example: TEST:HANDSHAKE:REQUEST:OUTPUT INVERTED

TEST:HANDSHAKE:REQUEST
:OUTPUT?

Function: Read back current programmed setting for polarity of the output request signal.

Syntax: OUTPUT?

Response: A | B | C | D, INVERTED | NORMAL.

Remarks: Returns the last programmed request polarity for only the active test . The active test is changed with the SYSTEM:TEST command, and determined with the SYSTEM:TEST? command.

Example: TEST:HANDSHAKE:REQUEST:OUTPUT?
A,INVERTED

See Also: TEST, REQUEST, INITIATE, REQUEST:OUTPUT

TEST:HANDSHAKE
:ACKNOWLEDGE

Function: Sets polarity of acknowledge handshake signal outputs to match UUT requirements . Not allowed if currently active test is Memory Emulation.

Syntax: [ACK[NOWLEDGE]]

Remarks: This command is optional, but may be used for documentation within a test program . The purpose of the command is to provide a pathway for setting the polarity of the acknowledge handshake output generated by the IO50/ IO100 at the completion of data transfer . Polarity of the separate input and output handshake signals can be set independently . The Memory Emulation test functions only with inverted polarity acknowledge handshake signals, errors are generated if commands attempt to alter this polarity . There is one set of input and output handshake lines on each of the four output connectors . The acknowledge polarity command only effects signals on the active test connectors, and they are set to the programmed value of the active test when that test is initiated . Merely changing the active test will not reinitialize the polarity control . The active test is changed with the SYSTEM:TEST command, and determined with the SYSTEM:TEST? command . Test execution is started with the INITIATE command . Section 3, Operation, shows timing information for the handshake signals and interlocked operation requirements.

See Also: TEST, PRGIOHANDSHAKE, BLKOUTHANDSHAKE, BLKINHANDSHAKE, MEMEMULATION, HANDSHAKE

TEST:HANDSHAKE:ACKNOWLEDGE
:INPUT

Function: Used to set polarity of the acknowledge signal sent by the IO50/IO100 at the completion of data input from the UUT.

Syntax: INPUT

Range: NORMal | INVerted

Default: NORMal

Remarks: The acknowledge handshakes are output in both timed and handshake type tests . The normal input handshake polarity will provide a high level signal when the IO50/IO100 has received and latched UUT data . This acknowledge signal will return low when the UUT requesting handshake signal is made inactive . In this normal mode of operation, the acknowledge signal output driver is always enabled . If inverted polarity is specified, the acknowledge signal is initially high impedance pulled high by the on board or UUT pull up resistor . When data has been read, the acknowledge signal output driver is enabled driving a low level . When the requesting handshake is made false, the driver immediately returns to high impedance output . This type of output can be tied to UUT open collector acknowledge signals . During timed type tests, the acknowledge signals are removed after a programmed delay, regardless of request handshake levels.

Example: TEST:HANDSHAKE:ACKNOWLEDGE:INPUT INVERTED

See Also: TEST, ACKNOWLEDGE, INITIATE, ACKNOWLEDGE:INPUT?

TEST:HANDSHAKE:ACKNOWLEDGE
:INPUT?

Function: Used to read back current programmed setting for polarity of the acknowledge output signal.

Syntax: INPUT?

Response: A | B | C | D, INVERTED | NORMAL.

Remarks: Returns the last programmed acknowledge polarity for only the active test . The active test is changed with the SYSTEM:TEST command, and determined with the SYSTEM:TEST? command.

Example: TEST:HANDSHAKE:ACKNOWLEDGE:INPUT?
NORMAL

See Also: TEST, ACKNOWLEDGE, INITIATE, ACKNOWLEDGE:INPUT

TEST:HANDSHAKE:ACKNOWLEDGE
:OUTPUT

Function: Used to set the acknowledge output polarity.

Syntax: OUTPUT

Range: NORM[AL] | INV[ERTED]]

Default: NORMAL

Remarks: The acknowledge handshakes are output in both timed and handshake type tests . The normal output handshake polarity will provide a high level signal when the IO50/IO100 has output data to the UUT . This acknowledge signal will return low when the UUT requesting handshake signal is made inactive . In the normal mode of operation, the acknowledge signal output driver is always enabled . If inverted polarity is specified, the acknowledge signal is initially high impedance, pulled high by pull up resistors . When data has been output, the driver is enabled, driving a low level . When the requesting handshake is made false, the driver returns to the high impedance state . This type of output can be tied to UUT open collector acknowledge signals . During timed type tests, the acknowledge signals are removed after a programmed delay, regardless of request handshake levels.

Example: TEST:HANDSHAKE:ACKNOWLEDGE:OUTPUT INVERTED

See Also: TEST, ACKNOWLEDGE, INITIATE

TEST:HANDSHAKE:ACKNOWLEDGE
:OUTPUT?

Function: Read back the polarity of the output acknowledge signal.

Syntax: OUTPUT?

Response: A | B | C | D, INVERTED | NORMAL.

Remarks: Returns the last programmed acknowledge polarity for only the active test . The active test is changed with the SYSTEM:TEST command, and determined with the SYSTEM:TEST? command.

Example: TEST:HANDSHAKE:ACKNOWLEDGE:OUTPUT?
NORMAL

See Also: TEST, ACKNOWLEDGE, INITIATE, ACKNOWLEDGE:OUTPUT

THIS PAGE INTENTIONALLY LEFT BLANK

TEST
:TIMEOUT

Function: Sets time between data and acknowledge handshake signal output . Valid only for timed type tests.

Syntax: TIM[EOUT]

Remarks: The IO50/IO100 will output an acknowledge signal on data input and output for UUT devices requiring clocks to transfer data . The time between the strobe edge and the data transfer can be altered with a programmed delay . This delay, in combination with the ability to program the polarity of the acknowledge handshake signal, allows setup or hold time to be increased . See section 3, Operation, for timing information . The INPUT and OUTPUT branch commands contain independent delay values for in and out acknowledge signals . Note that the timed block move instructions run to completion once initiated, without allowing new command processing . If the delay for a timed block test is large, and the test moves several thousand data vectors, several seconds may elapse before the operation completes . The timeout commands effect the currently active test . The active test can be set with the SYSTEM:TEST command and determined with SYSTEM:TEST?.

See Also: TEST, HANDSHAKE, ACKNOWLEDGE, TIMEOUT:INPUT, TIMEOUT:OUTPUT.

TEST:TIMEOUT
:INPUT

Function: Used to set time delay from when IO50/IO100 complements the initial acknowledge handshake signal level to when UUT data is read in and latched.

Syntax: [IN[PUT]] <seconds>

Range: 0 to 1E-3, 1us resolution

Default: 0

Remarks: The default unit is seconds, but the us or ms suffix may be used, as well as scientific notation . The acknowledge handshake for input is provided to clock a new data value out of a UUT before data inputs are read . The normal sequence is to change the initial level of the acknowledge signal, then open and close the IO50/IO100 input data latch to capture data, then return the acknowledge signal to the initial level . Any timeout value programmed will increase the time between the acknowledge signal change and when the data is latched . This time can be thought of as the UUT access time from strobe to data output . The initial state of the acknowledge signal will be low if it is programmed for normal polarity, and high if programmed as inverted . For block data transfer type tests, increasing the timeout delay will decrease the data rate and increase the test completion time . Separate timeout values are retained for each defined test, and are used when that test is initiated . Programmed I/O Timed is the only test using both an input and output timeout delay . Block Input uses only the input timeout value, and Block Output only the output value . The acknowledge signals have different drive characteristics depending on the polarity programmed . The Operation section and HANDSHAKE:ACKNOWLEDGE command section have more information.

Example: TEST:TIMEOUT:INPUT .000001
TEST:TIMEOUT:INPUT 20 e-6
TEST:TIMEOUT:INPUT 10 us
TEST:TIMEOUT:INPUT .5 ms

See Also: TEST, PRGIOTIMED, BLKINTIMED, BLKOUTTIMED, INITIATE, TEST:TIMEOUT:INPUT?

TEST:TIMEOUT
:INPUT?

Function: Used to read programmed time delay.

Syntax: IN[PUT]?

Response: A | B | C | D, seconds.

Remarks: The test name will always be that of the active test, since the timeout commands effect only that test . To query values for other tests, the SYSTEM:TEST command can be used to change the active test.

Example: TEST:TIMEOUT:INPUT?
D,.0005

See Also: TEST, TEST:TIMEOUT:INPUT

TEST:TIMEOUT
:OUTPUT

Function: Used to set the time delay from when IO50/IO100 outputs data and complements the initial acknowledge handshake signal level until the acknowledge signal returns to the initial level.

Syntax: OUT[PUT] <seconds>

Range: 0 to 1E-3, 1us resolution

Default: 0

Remarks: The default unit is seconds, but the us or ms suffix may be used, as well as scientific notation . The acknowledge handshake for output is provided to clock a new data value into the UUT after IO50/IO100 output data is valid . The normal sequence is to output data, then change the initial level of the acknowledge signal, then return the acknowledge signal to the initial level . Any timeout value programmed will increase the time between the output of data and the return of the acknowledge signal to the initial value . This time can be thought of as the UUT setup time from data input to write clock . The initial state of the acknowledge signal will be low if it is programmed for normal polarity, and high if programmed as inverted . For block data transfer type tests, increasing the timeout delay will decrease the data rate and lengthen the test completion time . Separate timeout values are retained for each defined test, and are used when that test is initiated . Programmed I/O Timed is the only test using both an input and output timeout delay . Block Input uses only the input timeout value, and Block Output only the output value . The acknowledge signals have different drive characteristics depending on the polarity programmed.

Example:
```
TEST:TIMEOUT:OUTPUT .00045
TEST:TIMEOUT:OUTPUT 25 e-6
TEST:TIMEOUT:OUTPUT 250 us
TEST:TIMEOUT:OUTPUT .1ms
```

See Also: PRGIOTIMED, BLKINTIMED, BLKOUTTIMED, INITIATE, TEST:TIMEOUT:OUTPUT?, HANDSHAKE:ACKNOWLEDGE

TEST:TIMEOUT
:OUTPUT?

Function: Used to read programmed time delay.

Syntax: OUT[PUT]?

Response: A | B | C | D, seconds.

Remarks: The test name will always be that of the active test, since the timeout commands effect only that test . To query values for other tests, the SYSTEM:TEST command can be used to change the active test.

Example: TEST:TIMEOUT:OUTPUT?
C,.00035

TEST
:ADDR

Function: Designates a field as an address input in Memory Emulation.

Syntax: ADDR <field_name>

Remarks: This command is valid only in the Memory Emulation test . It specifies a input field to be used as an address into the shared memory pool for emulating ROM or RAM . The pins in this field are specified by the user when the field name is defined . The order in which the pins are listed in the definition command will be most significant first to least significant last (right to left) . The field can be no more than 13 pins wide, since only 8192 vector addresses are available for memory emulation.

Example: TEST:NAME A:ADDR ROMADDR

See Also: MEMEMULATION, FIELD

TEST
:ADDR?

Function: Queries the current specified address field for Memory Emulation.

Syntax: ADDR?

Response: field name (string of 8 or less characters).

Example: TEST:NAME A:ADDR?
ADDR_FLD

See Also: TEST, MEMEMULATION, FIELD

TEST
:FREE?

Function: Used to read remaining available data space in shared RAM.

Syntax: FREE?

Response: available_data_vectors

Remarks: Each vector location represents 16 bytes of shared RAM space in the format shown in section 5, Register Programming . The TEST:NAME ALL:CATALOG? command can be used to determine the amount of memory allocation for each test, and the address offset to that memory . Shared memory area can be freed by deleting non-active tests.

Example: TEST:FREE?
512

See Also: VECTOR:DATA, TEST:NAME:DELETE.

VECTOR

Function: Used to define or read shared memory data values .

Syntax: VEC[TOR] <data_vector>

Range: 1 to 15,360

Default: None

Remarks: Shared memory is divided into 16 byte data vectors for use by tests . This memory is allocated from a memory pool to tests in the order they are defined . The vector number assigned in this command is used to reference data locations, regardless of their location in physical memory . The memory area is “re-packed” whenever a test is deleted, keeping the free memory in a contiguous address pool at upper memory . This operation will be transparent to the user if the vector numbers are used to reference data . The VECTOR command operates on the current active field . The active field can be determined and set using the SYSTEM:FIELD command . If several fields are assigned to one test, the DATA:FIELD command provides capability to temporarily operate on non-active fields.

The Programmed I/O tests take only 2 vectors, one for input and one for output . The block type tests can use whatever memory remains unallocated . The input vector memory area must be defined for block tests using this command, and may be initialized to some value . The output vector memory must also be defined, and will be filled with user defined data . The Programmed I/O test allocates vectors automatically, and does not require definition by the VECTOR command . The TEST:FREE? query will display any remaining vector locations.

See Also: TEST, FIELD, INITIATE

VECTOR
:COUNT

Function: Allows one vector command to execute repeatedly on several vectors.

Syntax: COUN[T] <vectors>

Range: 1 to 15360 | ALL

Default: 1

Remarks: This command will allow repeated operations as a shortcut to entering a separate Vector command for each vector value . It can be used for entering several output data vectors starting at the specified vector number . It can be used to read back several input vectors starting at the specified number . The count command sets the repetition number on its own branch level, then the repeated command is executed on the same level . This requires a semicolon in the command string, as shown in the example . Data values for the successive data vectors entered under a COUNT command need only be separated by commas . If the starting vector number plus the count value exceed either the maximum vector number or the defined vector block size, then an error will be generated . The ALL parameter will calculate the correct count based on the starting vector number and the defined vector block size.

Example: VECTOR 1:COUNT 5;DATA:VALUE 1,2,3,4,5

See Also: TEST, FIELD, VECTOR:DATA

VECTOR
:DATA

Function: Allows the user to set and read data values transferred in various tests.

Syntax: DATA

Remarks: Data is stored in shared RAM as vectors for each test . The data is entered by the user for output type tests, and read by the user for input type tests . This command does not directly effect memory, but is used with branch commands to perform the set and read operations.

Example: VECTOR 2:DATA:VALUE?
DA1E

See Also: TEST, FIELD:, VECTOR

VECTOR:DATA
:VALUE

Function: Used to set data in shared RAM memory for output to UUT . Data is written to the vector number specified in the VECTOR command.

Syntax: VAL[UE] <data_list>

Range: 0-F | 0-1, depending on RADIX

Default: None.

Remarks: When a test performs an output operation, the data will come from one of the vectors allocated for that test . The VALUE command is used to write user data to these vector locations . The field which is active when this command is executed will determine which of 128 vector bits will be written, and in what order the supplied data is apportioned . Bits of data supplied will be assigned to output pins in the order given by the FIELD:DEFINE command pin list . If the number of bits supplied in the command is less than the field width, the most significant bits will be zero filled . If more data is supplied than required for the field width, the extra bits are ignored . The active field is set or determined using the SYSTEM:FIELD command . The VECTOR:DATA:FIELD command can be used to temporarily change the active field within the VECTOR:DATA command . The VECTOR:COUNT command can be used to enter several output data values without repeated VALUE commands.

Example: VECTOR 1:DATA:VALUE FFF

VECTOR 1:COUNT 5;DATA:VALUE 003,00A,555,213,955

See Also: FIELD, TEST

VECTOR:DATA
:VALUE?

Function: Query command used to read data from a shared RAM vector location . The vector number specified in the VECTOR command will be read.

Syntax: VAL[UE]?

Response: String of up to 8 hexadecimal or 32 binary digit values.

Remarks: When a test performs a read operation, the input data will fill one of the 16 byte vectors allocated for that test . The VALUE? command is used by the slot 0 controller to read these vector locations . The field which is active when this command is executed will determine which of 128 vector bits will be returned, and in what order . The number of bits returned will match the number of pins defined in the active field . Bits will be arranged as listed in the FIELD:DEFINE command pin list, with the most significant on the left and least significant on the right . If the field width is not on a hexadecimal boundary, the most significant bits will be zero filled . The active field is set or determined using the SYSTEM:FIELD command . The VECTOR:DATA:FIELD command can be used to temporarily change the active field within the VECTOR:DATA command . The VECTOR:COUNT command can be used to read several vectors without sending repeated VALUE? commands.

Example: VECTOR 1:DATA:VALUE?
DA1E

VECTOR 1:COUNT ALL;DATA:VALUE?
0,1,2,3,...511

See Also: TEST, FIELD

VECTOR:DATA
:RADIX

Function: Allows choice between hexadecimal and binary formats for data readback and write .

Syntax: RAD[IX] <format>

Range: HEX | BIN

Default: HEX

Remarks: When vector data is read back using the VECTOR:DATA:VALUE? command, it is normally formatted as hexadecimal ASCII characters . This command will change the readback format to binary ASCII characters for the duration of a particular VECTOR:DATA command . It does not permanently change the radix . The selection of binary radix also allows output data vectors to be written as a series of ASCII ones and zeros . The RADIX command is executed on the same branch level as the VALUE? command it effects, so a semicolon must be used as shown in the example.

Example: VECTOR 1:DATA:RADIX BIN;VALUE?
1101101000011110

See Also: TEST, FIELD

VECTOR:DATA
:FIELD

Function: Temporarily change the active field within the VECTOR command to allow multiple field operations.

Syntax: FIEL[D] <field_name>

Range: Any valid field name - 1 to 8 character string.

Default: None

Remarks: The VECTOR:DATA commands will normally use the currently active field to determine data format . The FIELD branch command allows data for fields other than the active field to be read and written without repeatedly executing the SYSTEM:FIELD command . The effects of the FIELD command within the VECTOR command are temporary, the systems active field is not changed . The FIELD command is executed on the same branch level as the VALUE and VALUE? commands, so a semicolon must be used as shown in the example.

Example: VECTOR 1:DATA:FIELD ROMADD;VALUE 001;FIELD TABADD;VALUE 6

See Also: VECTOR, VECTOR:DATA, TEST, FIELD

SYSTEM

Function: Provides for setting and readback of global system parameters.

Syntax: SYST[EM]

Remarks: While this command has no direct action of its own, its branch commands can be used to set or query parameters which are common to several lower level functions.

SYSTEM
:ERROR?

Function: Query the last error encountered.

Syntax: ERR[OR]?

Response: error_number, error_description

Remarks: The error query returns a text string containing an error number and an error description . A list of error codes may be found in Appendix C . The Service Request (SRQ) interrupt may be enabled to generate an SRQ on error, providing immediate feedback in the event an error condition occurs.

Example: SYSTEM:ERROR?
0,"No Error"

See Also: STATUS

SYSTEM

:VERSION?

Function: Returns the SCPI version supported.

Syntax: VERS[ION]?

Response: year.version

Remarks: Returns a formatted numeric number indicating the SCPI version number supported for which the IO50/IO100 complies . The number is returned in the format YYYY.V where YYYY represents the year-version, and V represents the revision number for that year.

Example: SYSTEM:VERSION?
1992.0

SYSTEM

:FIELD

Function: Used to set the currently active field for the active test.

Syntax: FIEL[D] <field_name>

Range: String of 1 to 8 alphanumeric characters - any valid field name defined by the user.

Default: None

Remarks: An active field parameter is kept for each of the four tests . This command will set the active field name for whichever test is currently active . The SYSTEM:TEST command can be used to determine and set the active test .

Example: SYSTEM:FIELD ROMADDR

SYSTEM

:FIELD?

Function: Used to determine the currently active field for the active test.

Syntax: FIEL[D]?

Response: String of 1 to 8 alphanumeric characters - any valid field name defined by the user.

Remarks: An active field parameter is kept for each of the four tests . This command will return the active field name for whichever test is currently active . The SYSTEM:TEST command can be used to determine and set the active test .

Example: SYSTEM:FIELD?
RAMADDR

See Also: TEST, FIELD, VECTOR

SYSTEM

:TEST

Function: Used to set the currently active test.

Syntax: [TEST] <test_name>

Range: A | B | C | D

Default: None

Remarks: Only one test can be active at a time . The letter of the active test corresponds to one of the front panel connectors . The handshake signals on that connector will be used for data flow control while that test is active . The first test defined after initialization becomes the active test . Subsequent definitions of other tests will not change the active test . If the active test is deleted, another must be defined using the SYSTEM:TEST command to avoid errors on FIELD and VECTOR commands.

Example: SYSTEM:TEST A

See Also: TEST:DEFINE, FIELD, VECTOR

SYSTEM

:TEST?

Function: Used to determine the currently active test.

Syntax: TEST?

Response: A | B | C | D

Remarks: This command will return the active test name for whichever test is currently active . This command will typically be used as a diagnostic aid in error handling routines, since the active test assignment is normally known to the Slot 0 Controller.

Example: SYSTEM:TEST?
A

See Also: TEST:DEFINE, FIELD, VECTOR

SYSTEM
:LEARN

Function: Used to set up instrument environment with previously saved data.

Syntax: LEARN

Remarks: This command is executed after downloading a file, generated using the LEARN? command, to the shared memory area . After the file has been written to the shared RAM area, executing the LEARN command will cause the local microprocessor to transfer setup and control parameters from the shared RAM to the local environment . If data vectors were included in the download file, the complete test environment is restored, and the INITIATE command can be sent to start operation . All tests must be stopped before this command can be executed.

Example: SYSTEM:LEARN

See Also: TEST:DEFINE, FIELD, VECTOR

SYSTEM

:LEARN?

Function: Used to save instrument environment for later use.

Syntax: LEAR[N]?

Response: None - Although the LEARN? command is terminated with a question mark, which usually indicates a response will be generated, a response is not generated for this command.

Remarks: This command will save all internal IO50/IO100 data structures to an area of shared RAM . These data structures contain field definition, tristate control information, and other parameters necessary to restore the instrument to its current state . Data values are not modified by this command . Once this command has been executed, the Slot 0 Controller can upload the appropriate shared memory area to a file, and save the existing test environment . The shared memory area to be uploaded will start at the base address of shared RAM, as assigned by the Slot 0 Controller . The data from the base address through base address+4000 (hexadecimal) will contain internal setup information . Data in increasing addresses after that will be the input and output data vector values defined in the various tests . The slot 0 controller must save the full 256K byte shared memory in order to insure that all setup and data are properly saved . Each vector count represents 16 bytes of shared RAM . All tests must be stopped before this command can be executed.

Example: SYSTEM:LEARN?

See Also: TEST, FIELD, VECTOR

SYSTEM

:TRISTATE?

Function: Used to read current input/output configuration for all data pins as defined in software . This command is not to be confused with the FIELD:NAME:TRISTATE? query which returns configuration by field name only . This command does not read back actual signal levels of tristate driver control signals . That operation is performed as described in section 5, Register Programming.

Syntax: TRIST[ATE]?

Response: String containing a designator for all four octets of each connector, and an I/O configuration description.

Remarks: The designators for octets use the following shorthand notation; 1 for pins 1-8, 9 for 9-16, 17 for 17-24, and 25 for 25-32 . These octets are shown for each connector, listed in alphabetical order starting with A . The I/O configuration is described as :INPUT, OUTPUT, EXTNORMAL, or EXTINVERTED . These are the attributes assigned with the FIELD:NAME:TRISTATE command . The condition for all data pins after power up or reset is INPUT . If pins are assigned to only one field and that field is deleted, they retain their last configuration.

Example: SYSTEM:TRISTATE?
A1,OUTput,A9,OUTput,A17,OUTput,A25,OUTput,B1,OUTput, B9,OUTput,B17,OUTput,B25,OUTput,C1,OUTput,C9,OUTput, C17,OUTput,C25,OUTput,D1,OUTput,D9,OUTput,D17,OUTput, D25,OUTput,

See Also: FIELD:DEFINE, FIELD:NAME

STATUS

Function: Root command for setting status interrupts and accessing the various status registers.

Syntax: STAT[US]

Remarks: The IO50/IO100 utilizes the 488.2 and SCPI Status Reporting structure for reading instrument status . Refer to Figure 3-1 for supported status events (registers/bits) and Figure 3-2 for condition status . The structure for Event and Condition registers is identical, the difference being that Event Status is latched (static) while Condition Status is constantly changing as test conditions change (dynamic) . Each event register indicated in the Figure 3-1 is actually two registers, the status register itself and the corresponding status enable register . For clarity, only the status registers are shown . Figure 3-3 in the IEEE 488.2 Register section contains more detail on the relationship between status registers and status enable registers . The status register may be queried at any time to determine operational status of the IO50/IO100, even if event interrupts are not enabled . By setting the appropriate enable bits in the status enable registers, a Service Request (SRQ) interrupt may be generated when the enabled event occurs . This provides immediate feedback to the user.

The structure of the event status registers are several layers deep, with the IEEE 488.2 registers being the lowest layer and the IO50/IO100 event registers being the highest . The enable register functions as a mask to the event status register . To enable a certain status event to generate an SRQ, set the events corresponding enable bit to a "1" . Setting the enable bit to "0" disables SRQ generation for that event only . The device dependent (high level) events may be enabled as desired, and then disabled as a group by one of the lower layered registers . For example, all of the Test Operation Status Registers may be disabled and enabled as a group by disabling and enabling bit 8 of the Operation Status Register.

SCPI defines two reporting register pairs, Operation Status and Questionable Status . Likewise, IEEE 488.2 defines two register pairs, Standard Event Status and Status Byte . In both SCPI and 488.2 cases, only the supported bits in each register are shown in Figures 4-1 and 4-2 . And, since the SCPI Questionable Status is not supported, the Questionable Status register is not shown.

See Also: IEEE Register Configuration

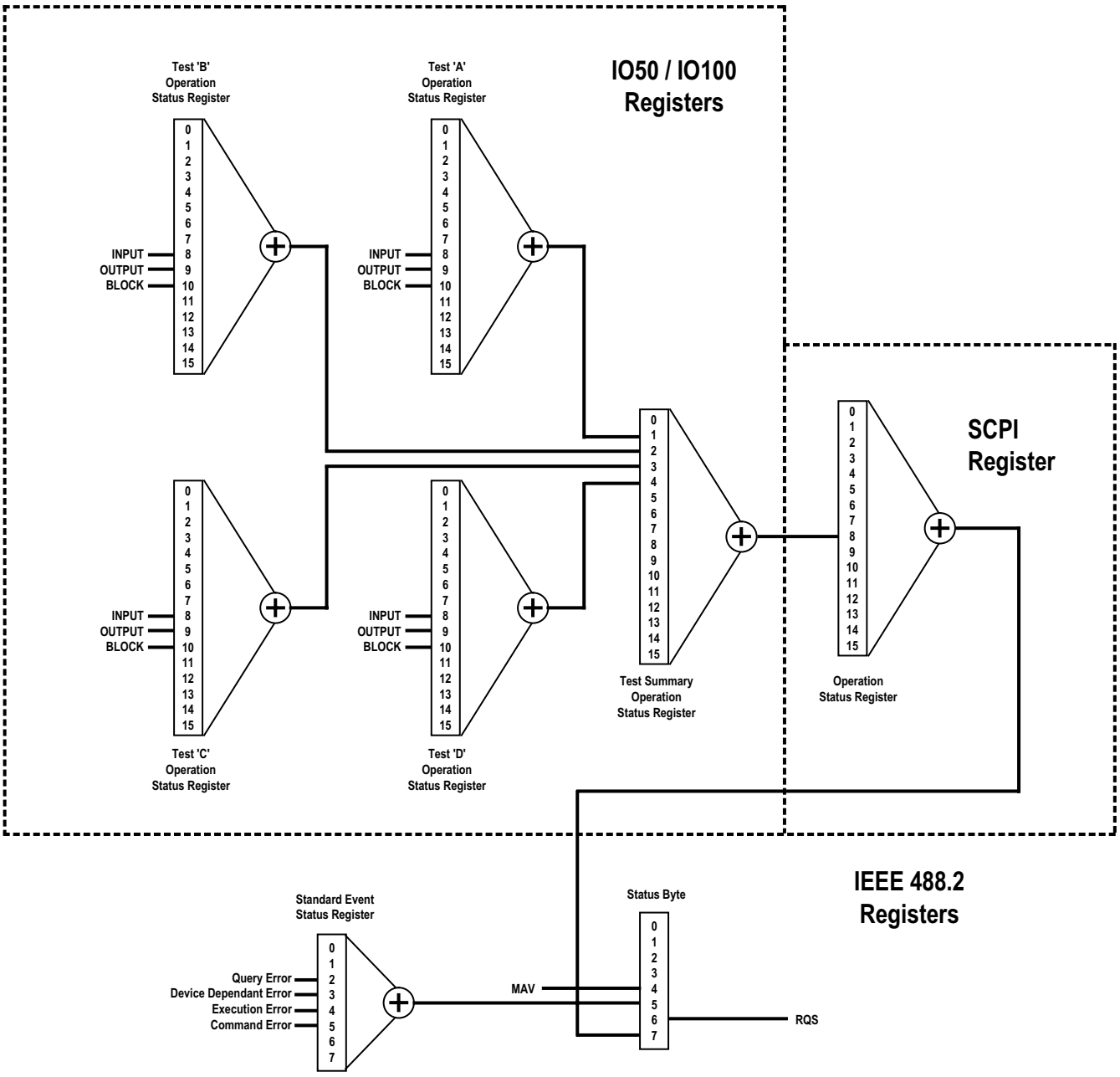


Figure 3-1.
IO50 / IO100 Event Status Structure.

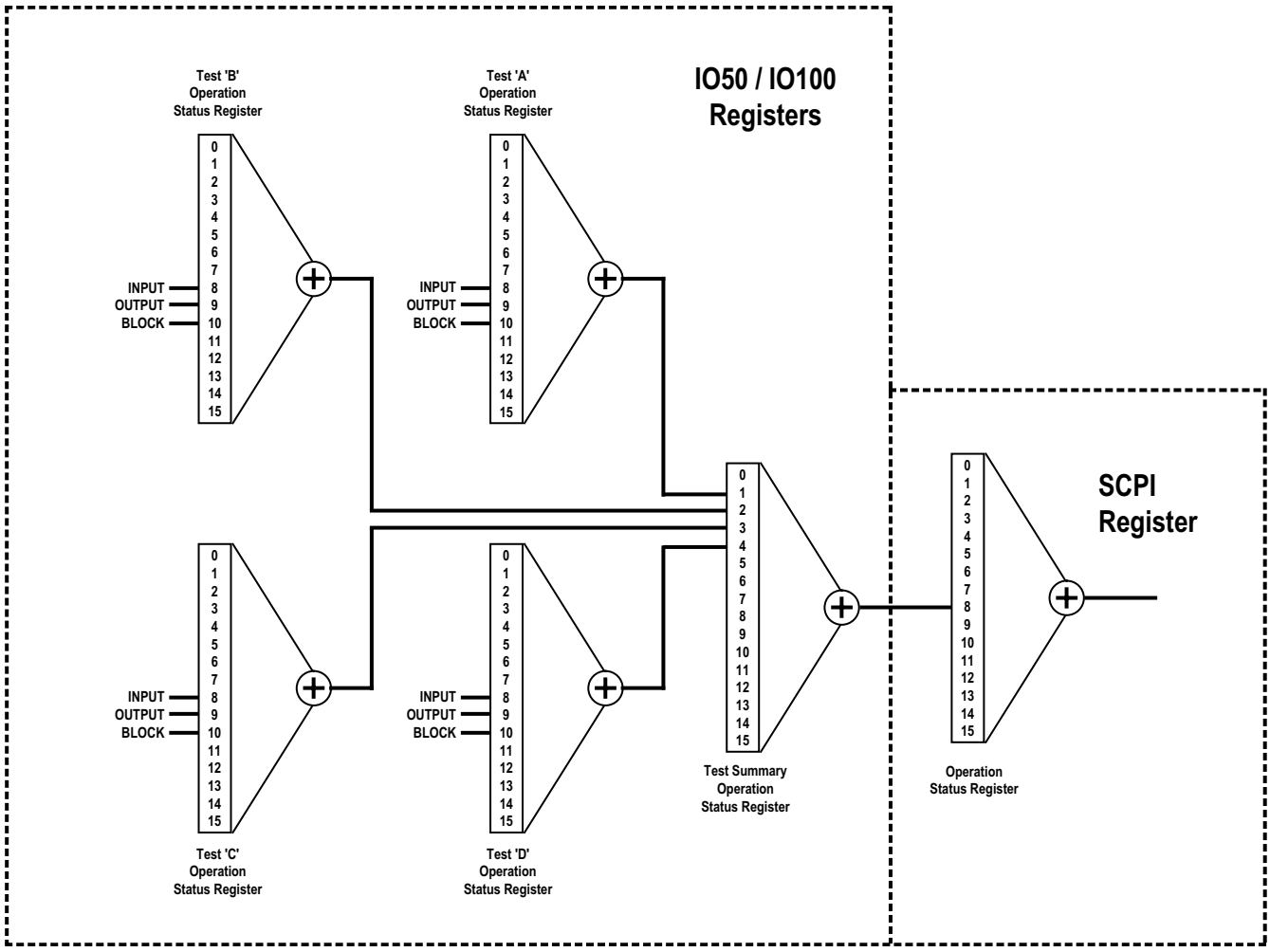


Figure 3-2.
IO50 / IO100 Condition Status Structure.

STATUS

:OPERATION

Function: Provides access to the IO50/IO100's normal operating conditions.

Syntax: OPER[ATION]

Remarks: The only operation event supported is bit 8, which is "available to designer" according to SCPI, Volume 1, Syntax and Style. In the IO50/IO100, bit 8 is used to enable the Test Operation Summary event. Operational status of any defined test is available via the Test Operation Status Registers, which, as a group, are enabled via bit 8 of the Operation Status register.

STATUS:OPERATION

:EVENT?

Function: Query the contents of the Operation Status Event register.

Syntax: [EVENT]?

Response: 0 | 256

Remarks: Reading this register clears it. If the returned value is 0, then no test summary events are set. If the returned value is 256, then at least one test summary event is set and the Status Operation Test register should be queried to determine which test bits are set.

Example:
```
STATUS:OPERATION:EVENT?
256
```

See Also: STATUS:OPERATION:TEST, STATUS:OPERATION:TEST:ISUMMARY

STATUS:OPERATION
:CONDITION?

Function: Query the contents of the Operation Condition register.

Syntax: COND[ITION]?

Response: 0 | 256

Remarks: The Operation Condition register returns the current status of bit 8 of the Operation Status register . This register (bit) will reflect whether any test is complete, or all tests are incomplete, providing a dynamic poll status . The Condition register is dynamic and is updated immediately as the test status changes . The Event Status registers provide similar information, however, the information is latched in the Event register and cleared only when the Event register is read . The Condition register also differs from the Event register in that it does not generate SRQ interrupts when test status changes . Refer to Figure 4-2 for Condition register configurations.

Example: STATUS:OPERATION:CONDITION?
256

STATUS:OPERATION

:ENABLE

Function: Set the Status Operation enable mask.

Syntax: ENAB[LE] <mask>

Range: (0 - 32767)

Remarks: While any combination of bit patterns may be set, only bit 8 has any significance . Writing a value of 256 will enable the test summary register for SRQ generation.

Example: STATUS:OPERATION:EVENT 256

STATUS:OPERATION

:ENABLE?

Function: Query the contents of the Status Operation enable mask.

Syntax: ENAB[LE]?

Response: 0 - 32767

Remarks: A query of this register will return the last value written to it . The query is non-destructive.

Example: STATUS:OPERATION:ENABLE?
256

STATUS:OPERATION

:TEST

Function: Provides access to the Operation Test Summary status register.

Syntax: TEST

Remarks: The Operation Test Summary status register combines the results of the individual Test Operation Status registers to provide a common control/ query path . Bit "1" indicates the one or more of the enabled events for "A" has occurred . Bit "2" is used for test "B", bit "3" for test "C" and bit "4" for test "D".

STATUS:OPERATION:TEST

:EVENT?

Function: Query the contents of the Operation Status Test Event register.

Syntax: [EVENT]?

Response: 0 - 30 inclusive, even values only.

Remarks: Reading this register clears it . If the returned value is 0, then no individual test events are set . If the value is other than 0, then the bit position indicates which individual tests have events set . The appropriate Status Operation Test Isummary register(s) should be queried to determine which test events are set.

Example: STATUS:OPERATION:TEST:EVENT?
6

See Also: STATUS:OPERATION:TEST:ISUMMARY

STATUS:OPERATION:TEST
:CONDITION?

Function: Query the contents of the Operation Test Condition register.

Syntax: COND[ITION]?

Response: 0

Remarks: The Test Condition register returns a summary of the current status of tests A, B, C, and D . This register will reflect whether any of these tests are complete or incomplete . The Condition register is dynamic and is updated immediately as test status changes . The Event Status registers provide similar information, however, the information is latched in the Event register and cleared only when the Event register is read . The Condition register also differs from the Event register in that it does not generate SRQ interrupts when test status changes . Refer to Figure 4-2 for Condition register configurations.

Example: STATUS:OPERATION:TEST:CONDITION?
6

STATUS:OPERATION:TEST

:ENABLE

Function: Set the Status Operation Test enable mask.

Syntax: ENAB[LE] <mask>

Range: (0 - 32767)

Remarks: While any combination of bit patterns may be set, only bits 1, 2, 3 and 4 have any significance . Writing a value of 12 will enable the SRQ generation for events in test "B" and test "C", if the individual events in those registers are enabled.

Example: STATUS:OPERATION:TEST:EVENT 12

STATUS:OPERATION:TEST

:ENABLE?

Function: Query the contents of the Status Operation enable mask.

Syntax: ENAB[LE]?

Response: 0 - 32767

Remarks: A query of this register will return the last value written to it . The query is non-destructive.

Example: STATUS:OPERATION:TEST:ENABLE?
256

STATUS:OPERATION:TEST

:ISUMMARY1

Function: Provide access to the various test mode status bits for test "A" only.

Syntax: ISUM[MARY1]

Remarks: There are 4 basic test modes, Input, Output, Block and Memory Emulation . The Memory Emulation test does not have a logical completion . Memory Emulation test will only stop executing when ABORTed . The other test modes do have logical completion, i.e . when the contents of the defined vector memory have been sent or filled (output or input) . The ISUMMARY branch to the STATUS:OPERATION:TEST command allows an SRQ to be generated when the Input, Output or Block test has reached its logical conclusion . There is a separate Event Status register for each test, A, B, C, and D . Each has its own enable as well . Status may be checked, regardless of whether an SRQ was generated or not, to determine test completion . The SRQ interrupt provides immediate notification of test completion.

See Also: ISUMMARY2, ISUMMARY3, ISUMMARY4

STATUS:OPERATION:TEST:ISUMMARY1

:EVENT?

Function: Query the contents of the Operation Status Test Isummary1 Event register.

Syntax: [EVENT]?

Response: 256 | 512 | 1024

Remarks: Reading this register clears it . If the returned value is 0, then the test is not complete . If the value is other than 0, then the bit position indicates which test type is complete . Refer to Figure 4-1.

Example: STATUS:OPERATION:TEST:ISUMMARY1:EVENT?
256

STATUS:OPERATION:TEST:ISUMMARY1
:CONDITION?

Function: Query the contents of the Operation Test Isummary1 Condition register.

Syntax: COND[ITION]?

Response: 0

Remarks: The Isummary1 Condition register returns the current status of test A . This register will reflect whether test A is complete or incomplete . The Condition register is dynamic and is updated immediately as the test status changes . The Event Status registers provide similar information, however, the information is latched in the Event register and cleared only when the Event register is read . The Condition register also differs from the Event register in that it does not generate SRQ interrupts when test status changes . Refer to Figure 4-2 for Condition register configurations.

Example: STATUS:OPERATION:TEST:ISUMMARY1:CONDITION?
0

STATUS:OPERATION:TEST:ISUMMARY1
:ENABLE

Function: Set the Status Operation Test Isummary1 enable mask.

Syntax: ENAB[LE] <mask>

Range: (0 - 32767)

Remarks: While any combination of bit patterns may be set, only bits 8, 9 and 10 have any significance . Setting bit 8 will cause bit 8 in the Event register to be set when test "A" completes a Input test . Bit 9 provides a similar function when a Output test completes . Bit 10 is associated with completion of a Block test.

Example: STATUS:OPERATION:TEST:ISUMMARY1:ENABLE 12

STATUS:OPERATION:TEST:ISUMMARY1
:ENABLE?

Function: Query the contents of the Status Operation enable mask.

Syntax: ENAB[LE]?

Response: 0 - 32767

Remarks: A query of this register will return the last value written to it . The query is non-destructive.

Example: STATUS:OPERATION:TEST:ISUMMARY1:ENABLE?
256

STATUS:OPERATION:TEST
:ISUMMARY2

Function: Provide access to the various test mode status bits for test "B" only.

Syntax: ISUM[MARY2]

Remarks: There are 4 basic test modes, Input, Output, Block and Memory Emulation . The Memory Emulation test does not have a logical completion . Memory Emulation test will only stop executing when ABORTed . The other test modes do have logical completion, i.e . when the contents of the defined vector memory have been sent or filled (output or input) . The ISUMMARY branch to the STATUS:OPERATION:TEST command allows an SRQ to be generated when the Input, Output or Block test has reached its logical conclusion . There is a separate Event status register for each test, A, B, C, and D . Each has its own enable as well . Status may be checked, regardless of whether an SRQ was generated or not, to determine test completion . The SRQ interrupt provides immediate notification of test completion.

See Also: ISUMMARY1, ISUMMARY3, ISUMMARY4

STATUS:OPERATION:TEST:ISUMMARY2
:EVENT?

Function: Query the contents of the Operation Status Test Isummary2 Event register.

Syntax: [EVENT]?

Response: 256 | 512 | 1024

Remarks: Reading this register clears it . If the returned value is 0, then the test is not complete . If the value is other than 0, then the bit position indicates which test type is complete . Refer to Figure 4-1.

Example: STATUS:OPERATION:TEST:ISUMMARY2:EVENT?
256

STATUS:OPERATION:TEST:ISUMMARY2
:CONDITION?

Function: Query the contents of the Operation Test Isummary2 Condition register.

Syntax: COND[ITION]?

Response: 0

Remarks: The Isummary2 Condition register returns the current status of test B . This register will reflect whether test B is complete or incomplete . The Condition register is dynamic and is updated immediately as the test status changes . The Event Status registers provide similar information, however, the information is latched in the Event register and cleared only when the Event register is read . The Condition register also differs from the Event register in that it does not generate SRQ interrupts when test status changes . Refer to Figure 4-2 for Condition register configurations.

Example: STATUS:OPERATION:TEST:ISUMMARY2:CONDITION?
0

STATUS:OPERATION:TEST:ISUMMARY2
:ENABLE

Function: Set the Status Operation Test Isummary2 enable mask.

Syntax: ENAB[LE] <mask>

Range: (0 - 32767)

Remarks: While any combination of bit patterns may be set, only bits 8, 9 and 10 have any significance . Setting bit 8 will cause bit 8 in the Event register to be set when test "A" completes a Input test . Bit 9 provides a similar function when a Output test completes . Bit 10 is associated with completion of a Block test.

Example: STATUS:OPERATION:TEST:ISUMMARY2:ENABLE 12

STATUS:OPERATION:TEST:ISUMMARY2
:ENABLE?

Function: Query the contents of the Status Operation Test Isummary2 enable mask.

Syntax: ENAB[LE]?

Response: 0 - 32767

Remarks: A query of this register will return the last value written to it . The query is non-destructive.

Example: STATUS:OPERATION:TEST:ISUMMARY2:ENABLE?
256

STATUS:OPERATION:TEST

:ISUMMARY3

Function: Provide access to the various test mode status bits for test "C" only.

Syntax: ISUM[MARY3]

Remarks: There are four basic test modes, Input, Output, Block and Memory Emulation . The Memory Emulation test does not have a logical completion . Memory Emulation test will only stop executing when ABORTed . The other test modes do have logical completion, i.e . when the contents of the defined vector memory have been sent or filled (output or input) . The ISUMMARY branch to the STATUS:OPERATION:TEST command allows an SRQ to be generated when the Input, Output or Block test has reached its logical conclusion . There is a separate Event status register for each test, A, B, C, and D . Each has its own enable as well . Status may be checked, regardless of whether an SRQ was generated or not, to determine test completion . The SRQ interrupt provides immediate notification of test completion.

See Also: ISUMMARY1, ISUMMARY2, ISUMMARY4

STATUS:OPERATION:TEST:ISUMMARY3

:EVENT?

Function: Query the contents of the Operation Status Test Isummary3 Event register.

Syntax: [EVENT]?

Response: 256 | 512 | 1024

Remarks: Reading this register clears it . If the returned value is 0, then the test is not complete . If the value is other than 0, then the bit position indicates which test type is complete . Refer to Figure 4-1.

Example: STATUS:OPERATION:TEST:ISUMMARY3:EVENT?
256

STATUS:OPERATION:TEST:ISUMMARY3

:CONDITION?

Function: Query the contents of the Operation Test Isummary3 Condition register.

Syntax: COND[ITION]?

Response: 0

Remarks: The Isummary3 Condition register returns the current status of test C . This register will reflect whether test C is complete or incomplete . The Condition register is dynamic and is updated immediately as the test status changes . The Event Status registers provide similar information, however, the information is latched in the Event register and cleared only when the Event register is read . The Condition register also differs from the Event register in that it does not generate SRQ interrupts when test status changes . Refer to Figure 4-2 for Condition register configurations.

Example: STATUS:OPERATION:TEST:ISUMMARY3:CONDITION?
0

STATUS:OPERATION:TEST:ISUMMARY3
:ENABLE

Function: Set the Status Operation Test Isummary3 enable mask.

Syntax: ENAB[LE] <mask>

Range: (0 - 32767)

Remarks: While any combination of bit patterns may be set, only bits 8, 9 and 10 have any significance . Setting bit 8 will cause bit 8 in the Event register to be set when test "A" completes a Input test . Bit 9 provides a similar function when a Output test completes . Bit 10 is associated with completion of a Block test.

Example: STATUS:OPERATION:TEST:SUMMARY3:ENABLE 12

STATUS:OPERATION:TEST:ISUMMARY3
:ENABLE?

Function: Query the contents of the Status Operation Test Isummary3 enable mask.

Syntax: ENAB[LE]?

Response: 0 - 32767

Remarks: A query of this register will return the last value written to it . The query is non-destructive.

Example: STATUS:OPERATION:ENABLE?
256:

STATUS:OPERATION:TEST
:ISUMMARY4

Function: Provide access to the various test mode status bits for test "D" only.

Syntax: ISUM[MARY4]

Remarks: There are 4 basic test modes, Input, Output, Block and Memory Emulation . The Memory Emulation test does not have a logical completion . Memory Emulation test will only stop executing when ABORTed . The other test modes do have logical completion, i.e . when the contents of the defined vector memory have been sent or filled (output or input) . The ISUMMARY branch to the STATUS:OPERATION:TEST command allows an SRQ to be generated when the Input, Output or Block test has reached its logical conclusion . There is a separate Event status register for each test, A, B, C, and D . Each has its own enable as well . Status may be checked, regardless of whether an SRQ was generated or not, to determine test completion . The SRQ interrupt provides immediate notification of test completion.

See Also: ISUMMARY1, ISUMMARY2, ISUMMARY3

STATUS:OPERATION:TEST:ISUMMARY4
:EVENT?

Function: Query the contents of the Operation Status Test Isummary4 Event register.

Syntax: [EVENT]?

Response: 256 | 512 | 1024

Remarks: Reading this register clears it . If the returned value is 0, then the test is not complete . If the value is other than 0, then the bit position indicates which test type is complete . Refer to Figure 4-1.

Example: STATUS:OPERATION:TEST:ISUMMARY4:EVENT?
256

See Also: ISUMMARY1, ISUMMARY2, ISUMMARY3

STATUS:OPERATION:TEST:ISUMMARY4
:CONDITION?

Function: Query the contents of the Operation Test Isummary4 Condition register.

Syntax: COND[ITION]?

Response: 0

Remarks: The Isummary4 Condition register returns the current status of test D . This register will reflect whether test D is complete or incomplete . The Condition register is dynamic and is updated immediately as the test status changes . The Event Status registers provide similar information, however, the information is latched in the Event register and cleared only when the Event register is read . The Condition register also differs from the Event register in that it does not generate SRQ interrupts when test status changes . Refer to Figure 4-2 for Condition register configurations.

Example: STATUS:OPERATION:TEST:ISUMMARY4:CONDITION?
0

STATUS:OPERATION:TEST:ISUMMARY4
:ENABLE

Function: Set the Status Operation Test Isummary4 enable mask.

Syntax: ENAB[LE] <mask>

Range: (0 - 32767)

Remarks: While any combination of bit patterns may be set, only bits 8, 9 and 10 have any significance . Setting bit 8 will cause bit 8 in the Event register to be set when test "A" completes a Input test . Bit 9 provides a similar function when a Output test completes . Bit 10 is associated with completion of a Block test.

Example: STATUS:OPERATION:TEST:ISUMMARY4:ENABLE 12

STATUS:OPERATION:TEST:ISUMMARY4
:ENABLE?

Function: Query the contents of the Status Operation enable mask.

Syntax: ENAB[LE]?

Response: 0 - 32767

Remarks: A query of this register will return the last value written to it . The query is non-destructive.

Example: STATUS:OPERATION:ENABLE?
256

BASICMODE

Function: Used to perform simple input and output operations with a minimum of programming effort.

Syntax: BASIC[MODE]

Remarks: This root command selects a simple set of branch commands for sending or receiving data to devices which require no data flow control or external control of output drivers . Basic formatting of data to output pins is provided . A Master/Slave configuration can be used to synchronize input or output across multiple boards in a chassis.

BASICMODE
:DEFINE

Function: Used to assign input or output function to data pins.

Syntax: DEF[INE]

Range: INPUT | OUTPUT

Default: None

Remarks: The data pins are internally configured for input or output in basic mode, the external tristate control signals are ignored . Separate driver and receiver hardware is used for each set of 8 data pins, so input is always available . If a pin is defined as input only, the output driver is disabled . If a pin is defined as output, the output driver is forced on when the BASIC:OUT command is executed . On board pullup resistors on the IO50/IO100 will return all undriven signals to a high state . The output drivers are implemented with byte wide devices, requiring all pins in an octet to be identically configured for either input or output.

Example: BASICMODE:DEFINE:INPUT C25

See Also: BASICMODE

BASICMODE:DEFINE
:INPUT

Function: Defines data pin as input only.

Syntax: IN[PUT]

Range: A25,A17,A9,A1,B25,B17,B9,B1,C25,C17,C9,C1,D25,D17,D9,D1

Default: None

Remarks: Pins are defined as input or output in byte wide groups . The shorthand notation for these groups in the command language is; A25 for pins A32-A25, A17 for pins A24-A17, A9 for pins A16- A9, A1 for pins A8-A1, and so forth for connectors B, C, and D . Attempting to define pin octets as inputs, which have previously been defined as outputs, will generate an error . The order in which the pins are listed in this Define command will determine the order that input data is returned . In this definition pin list, the most significant byte is assumed on the left, and least significant on the right . When data is read in from this pin field using the INPUT? command, it will returned most significant byte first, least significant last . Some minimal formatting of data, such as byte swapping, can be performed by listing the input groups in non-sequential order in the pin list . The amount of data returned will match the number of input only pins defined, two hex characters for each octet . All 128 bits may be read simultaneously . Each time a new DEFINE:INPUT command is executed, all previous input pin definition is cleared . Input pins have an internal pullup resistor on the IO50/IO100, and will be in a high state when not driven by the UUT.

Example: BASICMODE:DEFINE:INPUT C17,C25,C1,C9,B1,A1

See Also: BASICMODE

BASICMODE:DEFINE
:OUTPUT

Function: Defines data pin as output only.

Syntax: OUT[PUT]

Range: A25,A17,A9,A1,B25,B17,B9,B1,C25,C17,C9,C1,D25,D17,D9,D1

Default: None

Remarks: Pins are defined as input or output in byte wide groups . The shorthand notation for these groups in the command language is; A25 for pins A32-A25, A17 for pins A24-A17, A9 for pins A16- A9, A1 for pins A8-A1, and so forth for connectors B,C, and D . Attempting to define pin octets as outputs, which have previously been defined as inputs, will generate an error . The order in which the pins are listed in this Define command will determine the order that output data is apportioned . In this definition pin list, the most significant byte is assumed on the left, and least significant on the right . When data is sent for output using the OUTPUT command, the first byte sent will be assigned to the most significant pin octet, the last byte to the least significant . If data in the OUTPUT command is not sufficient to fill the width of the defined output pin list, the most significant bits will be zero filled . If data in excess of the output pin width is sent, the more significant bits will be ignored . Some minimal formatting of data, such as byte swapping, can be performed by listing the output groups in non-sequential order in the pin list . Each time a new DEFINE:OUTPUT command is executed, all previous output pin definition are cleared . If pins that were previously outputs are cleared back to an undefined state, their outputs will be disabled . On-board pullup resistors on the IO50/IO100 will then return all undriven signals to a high state.

Example: BASICMODE:DEFINE:OUTPUT C17,C25,C1,C9,B1,A1

See Also: BASICMODE

BASICMODE
:CATALOG?

Function: Used to determine the current input and output pin assignments.

Syntax: CAT[ALOG]?

Response: IN followed by the input pin octet list ; OUT followed by the output pin octet list.

Remarks: The shorthand notation for pin octets used in the BASICMODE:DEFINITION :INPUT and :OUTPUT commands is used here . If no pins are defined, the return string is IN ;OUT . Input or output pin definition can be modified only by using the :DEFINE pin to overwrite the existing configuration . All pin definition can be cleared with the CLEAR command.

Example: BASICMODE:CATALOG?
IN,A25,A17,B1,C25;OUT,A1,B25,A9

See Also: BASICMODE:DEFINE, CLEAR

BASICMODE
:CLEAR

Function: Removes all existing pin input and output configuration, returning all pins to undefined.

Syntax: CLE[AR]

Remarks: Used to return all 128 data pins to an undefined state . This command will disable all output drivers when executed . Any outputs which were driven low will then be returned to a high state by on board pullup resistors on the IO50/IO100 . Individual pin octets can be returned to undefined by executing BASICMODE:DEFINE commands which do not make reference to those octets . The CLEAR command will not initialize data.

Example: BASICMODE:CLEAR

See Also: BASICMODE:DEFINE

BASICMODE
:INPUT?

Function: Used to read data from the UUT on the defined input pins.

Syntax: IN[PUT]?

Response: #h, followed by character string with two hexadecimal values for each pin octet defined as input.

Remarks: All input data is latched simultaneously, whether 1 or 128 bits . Data is returned in hexadecimal format, the most significant byte first and the least significant last . No data inversion or masking is performed . The ordering of bytes is determined in the BASICMODE:DEFINE:INPUT octet pin list . The leftmost pin octet on the command line will be the most significant, the rightmost will be least significant . The number of hexadecimal characters returned will correspond to the number of pins defined as inputs.

Example: BASICMODE:INPUT?
#hDA1E

See Also: BASICMODE, BASICMODE:DEFINE:INPUT

BASICMODE
:OUTPUT

Function: Writes data to UUT on defined output pins .

Syntax: [OUT[PUT]]

Range: 0 to 32 hexadecimal characters.

Default: Zero

Remarks: The hexadecimal data provided is output on the pin octets listed in the BASICMODE:DEFINE:OUTPUT command when this command is executed . No data inversion or masking is performed . The order in which the pins are listed in the Define command will determine the order that output data is apportioned . In the definition octet pin list, the most significant byte is assumed on the left, and least significant on the right . When data is sent using the OUTPUT command, the first byte sent will be assigned to the most significant pin octet, the last byte to the least significant . If data in the OUTPUT command is not sufficient to fill the width of the defined output pin list, the most significant bits will be zero filled . Sending the command without any data will cause all outputs to be driven low . If data in excess of the output pin width is sent, the more significant bits will be ignored . Some minimal formatting of data, such as byte swapping, can be performed by listing the output groups in non-sequential order in the pin list.

Example: BASICMODE:OUTPUT F455ACD

See Also: BASICMODE, BASICMODE:DEFINE:OUTPUT

BASICMODE
:OUTPUT?

Function: Used to read data from the UUT on the defined output pins.

Syntax: OUT[PUT]?

Response: #h, followed by character string with two hexadecimal values for each pin octet defined as output.

Remarks: Same operation as the BASICMODE:INPUT command, but reads data from pins defined as outputs in BASICMODE:DEFINE command . Normally, this data will match the value last sent with the BASICMODE;OUTPUT command . If the IO50/IO100 is attempting to drive a shorted or grounded signal, the data written and read back will differ . If all signals connected to the data pins of the IO50/IO100 are high impedance (or open), this command can be used with the OUTPUT command to perform wrap-around internal data path testing.

Example: BASICMODE:OUTPUT?
#h1234

See Also: BASICMODE, BASICMODE:INPUT, BASICMODE:OUTPUT, BASICMODE:DEFINE:OUTPUT

BASICMODE
:MODE

Function: Selects optional configurations for synchronizing data input and output across multiple IO50/IO100 boards.

Syntax: MODE

Range: SLAVE | MASTER | STANDALONE

Default: STANDALONE

Remarks: The VXI system architecture provides user selectable trigger signals for synchronizing events across instruments within a chassis . The branch commands of the MODE command are used to configure multiple IO50/ IO100 boards to use these signals for latching in, or clocking out data . Normally, the IO50/IO100 operates in the Standalone mode, where data input and output are controlled internally in response to the INPUT? and OUTPUT commands . Boards in this mode ignore the VXI trigger signals . If an IO50/IO100 is set to Master mode, it will send a VXI trigger signal whenever it inputs or outputs data in response to the INPUT? and OUTPUT commands . Boards set to Slave mode will input and output data under control of the VXI trigger signals, not the INPUT? and OUTPUT commands they execute.

To insure simultaneous data input and output on a Master board and all of its Slaves, the following sequence rules must be observed . When performing input, the INPUT? command is sent to the Master board first (causing it and all slaves to latch input pin data) . The INPUT? command can then be sent to Slaves to read their data in turn . When performing output, the OUTPUT command is sent to the Slave boards first (setting up all data in the first stage of the output pipeline) . The OUTPUT? command can then be sent to the Master, which will cause it, and all Slave boards, to open their second stage pipeline latch and present data on the output pins.

Note that Master and Slave board sets must be configured to use the same VXI trigger signals using the :GROUP command . Several sets of independent Master/Slave groups can be configured by using the available VXI trigger signal pairs . If other VXI devices use these signals, high level software may have to allocate them at run time .

Example: BASICMODE:MODE:SLAVE:GROUP TTLT0

See Also: BASICMODE:MODE:SLAVE, BASICMODE:MODE:MASTER

BASICMODE:MODE

:SLAVE

Function: Sets IO50/IO100 input/output control to VXI trigger operation.

Syntax: SLAV[E]

Range: See branch command

Default: None

Remarks: The Slave command will configure the IO50/IO100 board for external control of its input and output latch strobe signals by VXI triggers . It also provides a means of selecting one of four available pairs of VXI trigger signals via its GROUP branch command . Once an IO50/IO100 has been put in Slave mode, it will not be able to perform input or output without the use of VXI trigger signals . A hardware or software reset, or a MODE:STANDALONE command, will return it to normal operation.

Example: BASICMODE:MODE:SLAVE:GROUP TTLT2

See Also: BASICMODE:MODE

BASICMODE:MODE:SLAVE
:GROUP

Function: Selects VXI TTLTRG signal pair to receive

Syntax: [GROU[P]]

Range: TTLT0 | TTLT2 | TTLT4 | TTLT6

Default: None

Remarks: This command is optional, but may be used to improve test documentation . The VXI trigger signals are allocated in pairs for synchronizing data flow between Master/Slave boards within a chassis . One signal is used for data input latching, one for data output . The shorthand notation for the signal pairs is; TTLT0 for TTLTRG0 and TTLTRG1, TTLT2 for TTLTRG2 and TTLTRG3, TTLT4 for TTLTRG4 and TTLTRG5, TTLT6 for TTLTRG6 and TTLTRG7 . The Slave board receives the signals from the Master board (or any other VXI source), on the pair selected by this GROUP command . The pair selected here must match the pair selected for output on the Master board .

Example: BASIC:MODE:SLAVE:GROUP TTLT6

See Also: BASICMODE:MODE:SLAVE, BASICMODE:MODE:MASTER

BASICMODE:MODE
:MASTER

Function: Sets IO50/IO100 to output VXI trigger signals whenever performing input or output.

Syntax: MAST[ER]

Range: See branch command

Default: None

Remarks: The Master command will configure the IO50/IO100 to provide VXI trigger signals for Slave boards . These signals will be generated whenever the Master board opens its input or output data latch devices in response to INPUT? or OUTPUT commands . The Master board operates normally in all other respects . The branch command GROUP is used to select one of four available pairs of VXI trigger signals to be driven . Once an IO50/ IO100 has been put in Master mode, it will continue to send VXI triggers until a hardware or software reset, or receiving a MODE:STANDALONE.

Example: BASICMODE:MODE:MASTER:GROUP TTLT2

See Also: BASICMODE:MODE

BASICMODE:MODE:MASTER
:GROUP

Function: Selects VXI TTLTRG signal pair for output.

Syntax: [GROU[P]]

Range: TTLT0 | TTLT2 | TTLT4 | TTLT6

Default: None

Remarks: Provides the same function as :MODE:SLAVE:GROUP, except the signal pair selected here will be driven by the Master IO50/IO100 for use by the Slave board (or any other VXI device) . The pair selected here must match the pair selected for input on the Slave board.

Example: BASIC:MODE:MASTER:GROUP TTLT6

See Also: BASICMODE:MODE, BASICMODE:MODE:SLAVE, BASICMODE:MODE:MASTER

BASICMODE:MODE
:STANDALONE

Function: Sets IO50/IO100 to normal internal input/output control.

Syntax: STAN[DALONE]

Remarks: This is the normal operating mode of the IO50/IO100 after power up or reset . Data input and output are controlled internally in response to IN-PUT? and OUTPUT commands . This command would typically be used only to return a board in Master or Standalone operation to normal.

Example: BASICMODE:MODE:STANDALONE

See Also: BASICMODE:MODE

BASICMODE
:MODE?

Function: Used to read back the current operating mode of an IO50/IO100 board.

Syntax: MODE?

Response: MASTER | SLAVE | STANDALONE, TTL0 | TTL2 | TTL4 | TTL6

Remarks: Returns the current mode of operation for an IO50/IO100 board as a character string . The mode names correspond to those used in the MODE command . If the mode is other than standalone, the VXI TTLTRG signal selection is also returned.

Example: BASICMODE:MODE?
STANDALONE

See Also: BASICMODE:MODE:SLAVE, BASICMODE:MODE:MASTER

IEEE 488.2 MANDATORY COMMANDS

The IO50/IO100 support the mandatory commands set forth in the IEEE 488.2, 1987 specification . The bulk of these commands utilize a four register set for passing operational information to the system . These registers are the Standard Event Status Register, Standard Event Status Enable Register, Status Byte Register and the Service Request Enable Register . Together, these register allow certain conditions to generate a service request to the system Slot 0 Controller, in much the same way that GPIB supports the Service Request (SRQ) function . Many of the commands on the following pages make use of these four registers, so an understanding of the working relationship of these registers is required . For this reason, a functional diagram of the registers is shown in Figure 3-3 . It is also recommended that the user refer to the IEEE 488.2, 1987 manual for further information.

The Standard Event Status Enable register and Service Request Enable register are used to enable potential interrupt events . The enable bits and the event bits are logically ANDed together, and then logically ORed with all other enable/event pairs to produce a flag bit which is fed into bit 5 (Event Status Byte) of the Status Byte Register . If the ESB bit is enabled, by writing a 1 to bit 5 of the Status Request Enable register, then any enabled Standard Event will generate an Service Request (SRQ) to the Slot 0 Controller.

The Standard Event interrupts supported by the IO50/IO100 are Query Error (bit 2), Device Dependent Error (bit 3), Execution Error (bit 4) and Command Error (bit 5) . The Status Byte interrupt supported are the Operation Event Status (bit 7), Event Status Byte (bit 5) and Message Available (bit 4).

Some of the 488.2 mandatory commands have parameters associated with them . In all cases the parameters may be entered in either decimal (default format), hexadecimal (#h prefix) or binary (#b prefix) formats.

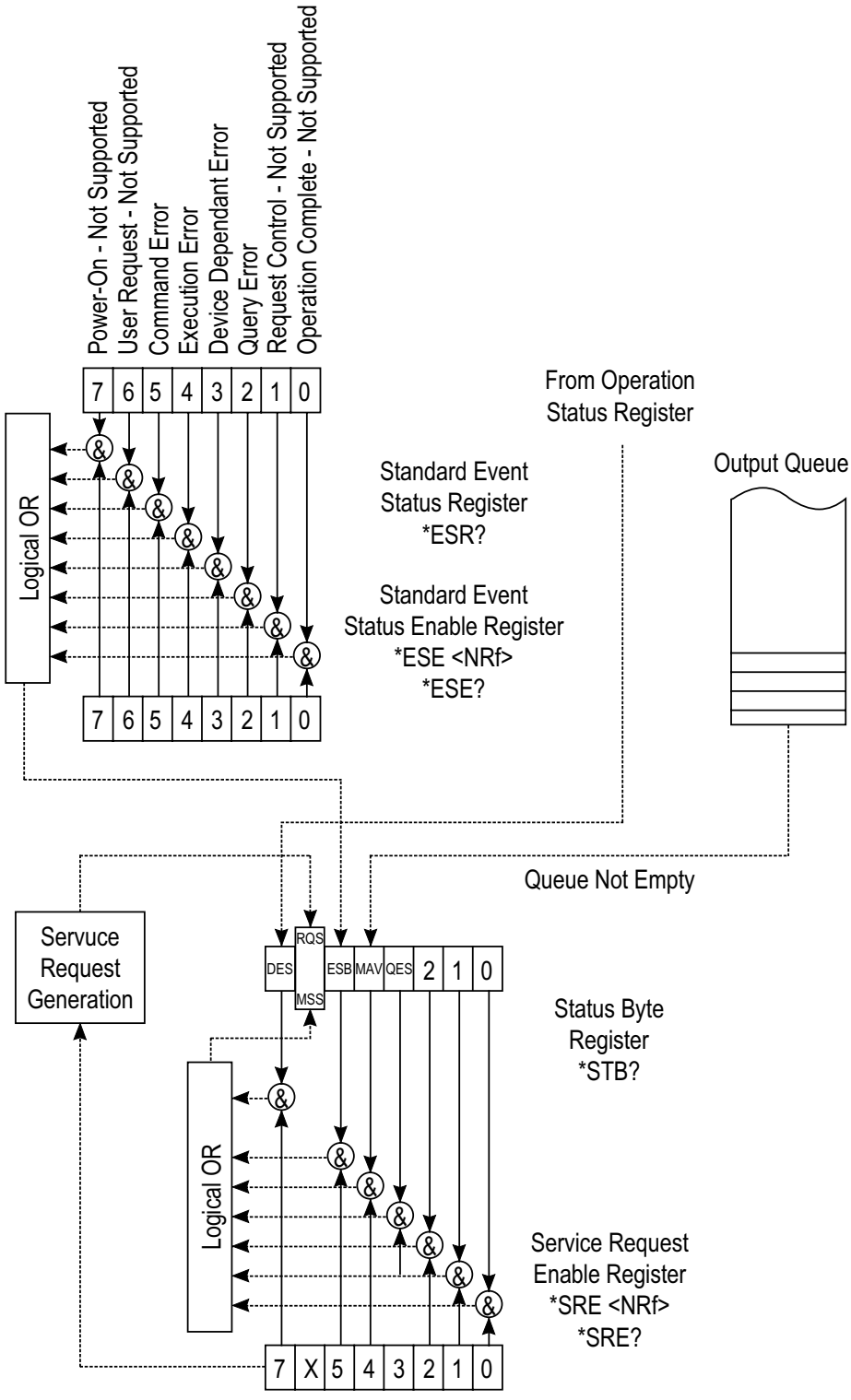


Figure 3-3.
IEEE 488.2 Register Configuration.

*CLS

Function: Clear Status Command.

Syntax: *CLS

Remarks: Clears all Event Registers in the status byte register . All queues, except the output queue, in the status byte are emptied . The device is forced into the operation complete command idle state and operation complete query idle state.

Example: *CLS

*ESE

Function: Standard Event Status Enable Command.

Syntax: *ESE <number>

Range: 0 to 255

Default: 0

Remarks: A "1" in the bit position enables the corresponding bit of the standard event status enable register . Refer to Figure 4-3 for the meaning of each bit.

Example: *ESE #b00000001
**ESE 255*

*ESE?

Function: Standard Event Status Enable Query.

Syntax: *ESE?

Response: decimal

Remarks: Returns an integer (0-255) which is the value of the standard event status enable register . Refer to Figure 4-3 for the meaning of each bit.

Example: *ESE?
255

*ESR?

Function: Standard Event Status Register Query.

Syntax: *ESR?

Response: decimal

Remarks: Returns an integer (0-255) which is the value of the standard event status register . Refer to Figure 4-3 for the meaning of each bit.

Example: *ESR?
32

*IDN?

Function: Identification Query.

Syntax: *IDN?

Response: INTERFACE TECHNOLOGY,IO50/IO100,0,REV (n)

Remarks: Returns the unique identification of the IO50/IO100, which is defined in four fields separated by commas (Manufacturer, Model, Serial number., Firmware level) . The serial number field, which is not used, will always default to 0 . The firmware level may be different than shown.

Example: *IDN?
INTERFACE TECHNOLOGY,IO100,0,1.0

*OPC

Function: Operation Complete.

Syntax: *OPC

Remarks: The IO50/IO100 has no application for the Operation Complete status and overlap commands . Therefore, *OPC is parsed only; no operation is performed.

Example: None

*OPC?

Function: Operation Complete Query.

Syntax: *OPC?

Response: None

Remarks: The IO50/IO100 has no application for the Operation Complete status and overlap commands . Therefore, *OPC? is parsed only; no operation is performed.

Example: None

*RCL

Function: Recall.

Syntax: *RCL

Remarks: The IO50/IO100 has no application for the Recall Setup command . Therefore, *RCL is parsed only; no operation is performed . The LEARN command provides a function similar to the intended use of *RCL.

Example: None

*RST

Function: Reset.

Syntax: *RST

Remarks: Resets the IO50/IO100 to the power-up state (setting all parameters to their default values) . Aborts all tests in progress.

Sets all parameters to their default values . Clears Errors . Does not change IEEE 488.2 Registers.

Example: *RST

*SAV

Function: Save.

Syntax: *SAV <setup number>

Remarks: The IO50/IO100 has no application for the Save Setup command . Therefore, *SAV is parsed only; no operation is performed . The LEARN? command provides a function similar to the intended use of *SAV.

Example: None

*SRE

Function: Sets the bits of the Service Request Enable register.

Syntax: *SRE <number>

Default: 0

Remarks: This eight bit register is used to enable what event will cause generation of a Service Request (SRQ) to the Slot 0 Controller . For example, if you wanted to generate a SRQ when the output queue has a message available, you would write a 16 to the SRE register . Refer to Figure 3-3 for the bit definitions to this register.

Example: *SRE 16
**SRE #b00010000*

*SRE?

Function: Service Request Enable Query.

Syntax: *SRE?

Response: decimal

Remarks: Returns an integer (0-255) which is the value of the Service Request Enable register . Refer to Figure 3-3 for the bit definitions to this register.

Example: *SRE?
112

*STB?

Function: Read Status Byte Query.

Syntax: *STB?

Response: decimal

Remarks: Returns an integer (0-255) which is the value of the status byte register.

Example: *STB?
64

*TRG

Function: Trigger.

Syntax: *TRG

Remarks: The IO50/IO100 has no application for the Trigger command . Therefore, *TRG is parsed only; no operation is performed.

Example: None

*TST?

Function: Self Test Query.

Syntax: *TST?

Response: error,message

Remarks: Forces the IO50/IO100 to reset state . All tests will be deleted . Sending the *TST? command is like sending *RST.

Example: *TST?
0, "No Error"

*WAI

Function: Wait.

Syntax: *WAI

Remarks: The IO50/IO100 has no application for the Wait command . Therefore, *WAI is parsed only; no operation is performed.

Example: None

(THIS PAGE INTENTIONALLY LEFT BLANK)

CHAPTER 4

Register Access

Register Based Operation

The IO50/IO100 provides shared memory access to the Device Dependent Registers for users wishing to implement special functions not provided by the command set. These registers are the same ones accessed by the local 68000 MPU when executing the high level word serial command operations. To avoid conflicts with internal processor operation, only one method of programming the IO50/IO100 should be used at a time.

The VXI Specification dictates that Device Dependent Registers are addressed at the Base Address of the device Configuration Registers + 20 hexadecimal. This base address may be set by the Logical Address Switch, or dynamically configured by the Slot 0 Resource Manager. Most Slot 0 Controller software packages have some utility function for determining the base addresses of devices installed in the card rack.

The registers are byte, word, and longword addressable in the Motorola data format. Intel based Slot 0 Controllers may have to perform byte swapping operations for direct access to bytes and longwords. Write operations to the data input and output registers can use longword access freely. Longword write operations to the control registers are not recommended, as the sequence of control may vary between Slot 0 Controller vendors and language compilers.

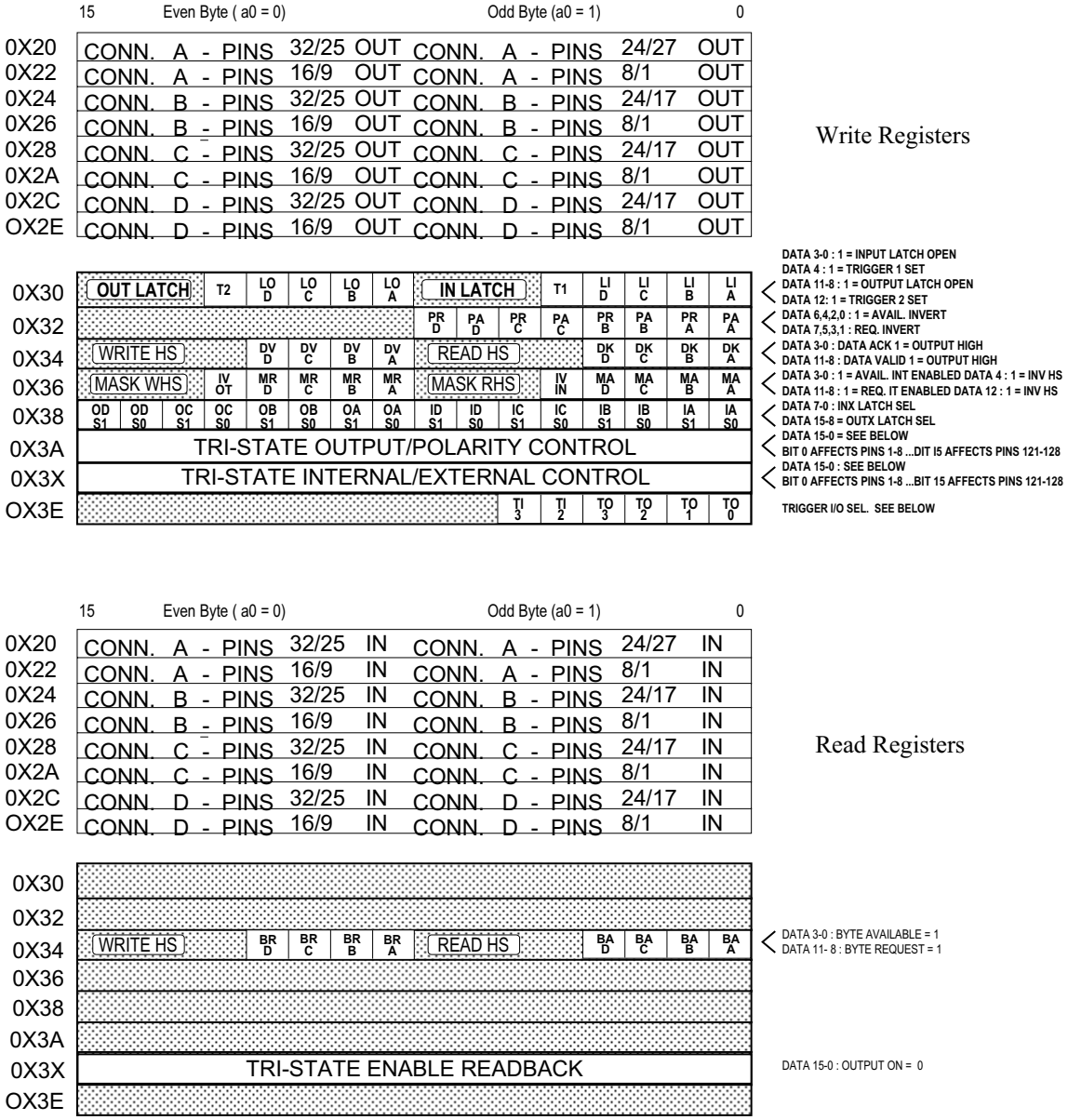
Register based programming may offer significant data throughput advantages for users with the required software development resources.

Register Programming Bit Definitions

Refer to the Device Dependent Register bit map, Figure 4-1, for the following register descriptions.

Data Input and Output Registers (0x20-2F Read/Write)

The bits in these registers correspond to the indicated data pin on the front panel connectors. Data is passed without inversion. Input data is captured in byte wide latches which are enabled onto the data bus when the addresses from 0x20 to 0x2F are read. Note that the read operation only enables the latch outputs, the latch hold control (0x31), must be pulsed or held open to pass data directly from the pins. Output data is output in two stages, first to byte wide registers, and finally through output latches. The first stage registers are loaded by write operations to addresses 0x20 to 0x2E. The output latches are used to allow all 128 bits to be output at the same time, and their hold control (0x30) must be pulsed or held open to output data directly to the pins.



OUTPUT LATCH SEL	INPUT LATCH SEL	TRI-STATE INT/EXT	OUT/POL CONTROL	TRIGGER IN SEL	TRIGGER OUT SEL
00 = LATCH OUT REG. BIT	00 = LATCH IN REG. BIT	0	0 = OUTPUTS HI Z	00 = TTLTRG 0 & 1	0X1 = TTLTRG 0 & 1
01 = TRIGGER 2	01 = TRIGGER 1	0	1 = OUTPUTS ENABLED	01 = TTLTRG 2 & 3	0X2 = TTLTRG 2 & 3
00 = BYTE REQ. HS	00 = BYTE REQ. HS	1	0 = EXT. 0 = ENABLED	10 = TTLTRG 4 & 5	0X4 = TTLTRG 4 & 5
11 = SPARE	11 = SPARE	1	1 = EXT 1 = ENABLED	11 = TTLTRG 6 & 7	0X8 = TTLTRG 6 & 7

Figure 4-1.
Device Dependent Registers.

The Motorola byte addressing scheme is used, pin connection is defined as LS Bit to lowest pin number, MS Bit to highest pin number within any given byte. Data values are stored in this same configuration in the Shared RAM area for use by the various defined test operations. Each 128 bit data vector is stored or read using the high level VECTOR command.

Control Registers

Output Latch and Trigger 2 Generation (0x30 Write) A one bit written to the bits 0-3 of byte 0x30 (8-11 of word 0x30) will open the 32 bit wide output latch for connector A,B,C, or D as indicated. This will allow data from the first stage data output registers (0x20-2F) to pass to the output pins. When a zero is written to these bits, the current value in the first stage data output registers is held on the output pins.

A one bit written to bit 4 of byte 0x30 (bit 12 of word 0x30) will cause a low level on the higher numbered VXI TTLTRG signal of the two selected in the Trigger Control register (0x3E) below. A zero in this position causes the trigger level to be high (inactive).

A write to the output latch address will cause an acknowledge handshake signal to transition low under certain conditions. See the Interrupt Mask Control (0x36) register section below.

Input Latch and Trigger 1 Generation (0x31 Write). A one bit written to the bits 0-3 of byte 0x31 (word 0x30) will open the 32 bit wide input latch for connector A,B,C, or D as indicated. This will allow data from the connector pins to pass through tristate latches enabled at addresses 0x20-2F. When a zero is written to these bits, the current value on the data pins is held.

A one bit written to bit 4 of byte 31 (word 0x30) will cause a low level on the lower numbered VXI TTLTRG signal of the two selected in the Trigger Control register (0x3F) below. A zero in this position causes the trigger level to be high (inactive).

A write to the input latch will cause an acknowledge handshake signal to transition low under certain conditions. See the Interrupt Mask Control (0x36) register section below.

Request Handshake Polarity Control (0x33 Write). A one bit written to bits 0,2,4, or 6, of byte 0x33 (word 0x32) will invert the Data Available input request handshake signal from the indicated. The interrupt logic requires a low level on this signal to cause an interrupt. Request signals from the UUT must be inverted if they indicate readiness for data transfer by transition to a high level. A zero bit in these positions will allow the request handshake to pass without inversion. The non-inverted polarity is

required when UUT request handshake signal transitions low to indicate that the UUT has data ready for input to the IO50/IO100.

Bits 1,3,5,7 perform the same function for the Byte Request output request handshake signals. The UUT will cause these signals to transition when it wants to receive another data value from the IO50/IO100.

Output Acknowledge Handshake Signal Control (0x34 Write). A one bit written to bits 3-0 of byte 0x34 (bits 11-8 of word 0x34) will appear as a high level output on the Data Valid acknowledge handshake signals. A zero bit will cause a low level output. Exceptions to this operation occur when using inverted acknowledge handshake operation. See the Interrupt Mask Control (0x36) register section below.

The Data Valid handshake may be set true after a new value has been passed to the connector pins by the IO50/IO100, to provide an input latch strobe for the UUT.

Output Request Handshake Status (0x34 Read). A one bit read from bits 3-0 of byte 0x34 (bits 11-8 of word 0x34) indicates a handshake request has been received from the indicated connector. The value appearing in this status register is inverted from the signal level being input. Each one bit in this register can cause an interrupt, if enabled by the Interrupt Mask register (0x36). The status read here is not affected by interrupt mask bit settings. The bits here are not latched, but follow the level of the request input signal.

When operating with normal acknowledge handshake polarity, the status bits will be one under two conditions; 1. When the UUT has requested output and the IO50/IO100 has not set the Data Valid acknowledge handshake true (high), 2. When the IO50/IO100 has set Data Valid high and the UUT has removed the requesting handshake.

When operating with inverted acknowledge handshake polarity, the status bit is high whenever the UUT has set the request handshake line to its active state.

Input Acknowledge Handshake Signal Control (0x35 Write). A one bit written to bits 3-0 of byte 0x34 (bits 11-8 of word 0x34) will appear as a high level output on the Data Acknowledge handshake signals. A zero bit will cause a low level output. Exceptions to this operation occur when using inverted acknowledge handshake operation. See the Interrupt Mask Control (0x36) register section below.

The Data Acknowledge handshake may be set true after a new value has been latched from the connector pins by the IO50/IO100 to provide an UUT output clock to change to the next data value.

Output Interrupt Mask and Inverted Acknowledge Control (0x36 Write). Writing a one bit to bits 3-0 of byte 0x36 (bits 11-8 of word 0x36) will enable the Byte Request handshake signal to generate a local interrupt to the 68000 MPU. The STATUS commands can be used to program the 68000 MPU to pass the local interrupt on to the Slot 0 Controller. A zero bit will inhibit interrupts from the indicated connector signal. In normal operation, only one connector request signal will be enabled at a time.

A one bit at bit 4 of byte 0x36 (bit 12 of word 0x36) will enable the Inverted Output Acknowledge handshake mode of operation. This is used to provide an open collector type acknowledge handshake signal. This is accomplished by controlling the Data Valid acknowledge signal output driver tristate enable. When the inverted mode of operation is selected, the driver output is initially turned off. Writing to the Output Latch control register (0x30) with a given Data Valid handshake bit (0x34) low, and the Interrupt Mask control bit high (in the matching bit position), will enable the output driver. At that time, the connector handshake signals with 0 bit values in the Output Acknowledge register (0x34) will go from pulled-up to low. The acknowledge handshake output driver will go back to high impedance output as soon as the UUT makes the Byte Request handshake signal inactive. If a local output type interrupt is pending, it is disabled when the acknowledge output driver is enabled.

Input Interrupt Mask and Inverted Acknowledge Control (0x37 Write), Writing a one bit to bits 3-0 of byte 0x37 (word 0x36) will enable the Byte Available handshake signal to generate a local interrupt to the 68000 MPU. The STATUS commands can be used to program the 68000 MPU to pass the local interrupt on to the Slot 0 Controller. A zero bit will inhibit interrupts from the indicated connector signal. In normal operation, only one connector request signal will be enabled at a time.

A one bit at bit 4 of byte 0x36 (bit 12 of word 0x36) will enable the Inverted Input Acknowledge handshake mode of operation. This is used to provide an open collector type acknowledge handshake signal. This is accomplished by controlling the Data Acknowledge signal output driver tristate enable. When the inverted mode of operation is selected, the driver output is initially turned off. Writing to the Input Latch control register (0x31) with a given Data Acknowledge (0x35) handshake bit low, and the Interrupt Mask control bit high (in the matching bit position), will enable the output driver. At that time, the connector handshake signals with 0 bit values in the Data Acknowledge register (0x35) will go from pulled-up to low. The acknowledge handshake output driver will go back to high impedance output as soon as the UUT makes the Byte Available handshake signal inactive. If a local input type interrupt is pending, it is disabled when the acknowledge output driver is enabled.

Output Latch Strobe Select Control (0x38 Write). There are four pairs of bits in this control register, one pair for each connector. The function selected will determine when the output latch opens to pass the data from the first stage register to the connector pins. There are four possible functions selected by the two bits, decoded as follows;

0 = Latch strobe controlled by Output Latch Control register. 1 = Latch strobe controlled by VXI TTLTRG2 signal. 2 = Latch strobe controlled by Byte Request handshake. 3 = Latch forced open.

The normal mode of operation uses decode 0. When one IO50/IO100 is slaved to another, decode 1 is used. The remaining two may meet specific user requirements.

Bits 7 and 6 in byte 0x38 (bits 15 and 14 in word 0x38) will control the 32 bit latch for connector D. The other bit pairs, in descending order, control the C, B, and A connector latches.

Input Latch Strobe Select Control (0x39 Write). There are four pairs of bits in this control register, one pair for each connector. The function selected will determine when the input latch opens to pass the data from the connector pins into the internal data bus. There are four possible functions selected by the two bits, decoded as follows;

0 = Latch strobe controlled by Input Latch Control register. 1 = Latch strobe controlled by VXI TTLTRG1 signal. 2 = Latch strobe controlled by Byte Available handshake. 3 = Latch forced open. The normal mode of operation uses decode 0. When one IO50/IO100 is slaved to another, decode 1 is used. The remaining two may meet specific user requirements. Bits 7 and 6 in byte 0x39 (word 0x38) will control the 32 bit latch for connector D. The other bit pairs, in descending order, control the C, B, and A connector latches.

Tristate Polarity (0x3A Write) and Output Control (0x3C Write). The bits in these two registers control the output pin tristate drivers on a byte wide basis. One pair of bits is provided for each of the 16 bytes. The control function is encoded as follows:

Table 4-1.
Tristate Polarity and Output Control Encoding.

Int/Ext Control Bit (0x3C)	Enable/Polarity Bit (0x3A)	Function
0	0	Outputs tristated (pulled up)
0	1	Outputs enabled
1	0	External control (0 = enabled; 1 = tristate)
1	1	External control (0 = tristate; 1 = enable)

The bit control is apportioned to the connector pins as follows:

Table 4-2.
Tristate Polarity and Output Control Encoding.

Bit	Connector Pins	Bit	Connector Pins
0	A8-1	4	B8-1
1	A16-9	5	B16-9
2	A24-17	6	B24-17
3	A32-25	7	B32-25
Bit	**Connector Pins**	**Bit**	**Connector Pins**
8	C8-1	12	D8-1
9	C16-9	13	D16-9
10	C24-17	14	D24-17
11	C32-25	15	D32-25

Note
Grayed items apply to I/O100 series only. Not used with I/O 50 series.

The control is initialized to decode 0 on power up so that all output pin drivers are at high impedance.

Tristate Enable Readback (0x3C Read). Bits read back from this address indicate the state of the tristate enables for the output pin drivers. Bits which are read as low indicate that the driver for the corresponding byte is enabled. Bits which are read high indicate a disabled driver. These bits are read from the actual device enable signals, and reflect all external and internal control inputs.

The bit positions correspond to the connector bytes in the same way shown above under Tristate Polarity and Output Control.

Trigger Input/Output Select (0x3F Write). The bits in this register are used to select among the VXI TTLTRG signals for input and output functions. The IO50/IO100 is capable of generating two independent output trigger signals and receiving two independent signals from another VXI device. The two separate trigger signals allow control of input and output operations independently. Trigger generation is performed by a “master” IO50/IO100 card to control data flow on a “slave” board. Trigger reception is used on a slave board to open and close data latches. Trigger signals to be received are selected by bits 5 and 4 of byte 0x3F (word 0x3E) as follows:

Table 4-3.
Receive Trigger Pairs.

Bit 5	Bit 4	Receive Trigger Pair
0	0	TTLTRG0 for input, TTLTRG1 for output
0	1	TTLTRG2 for input, TTLTRG3 for output
1	0	TTLTRG4 for input, TTLTRG5 for output
1	1	TTLTRG6 for input, TTLTRG7 for output

Trigger signals are output on lines selected as follows:

Table 4-4.
Send Trigger Pairs.

Bit 3210	Send Trigger Pair
0000	None
0001	TTLTRG0 for input, TTLTRG1 for output
0010	TTLTRG2 for input, TTLTRG3 for output
0100	TTLTRG4 for input, TTLTRG5 for output
1000	TTLTRG6 for input, TTLTRG7 for output

The initial state of the output circuit on power up is 0000.

CHAPTER 5

Applications

Memory Emulation

The ROM and RAM Emulation Connection figures (Figs 5-1 and 5-2) show ways of providing limited support hardware on the UUT to facilitate a data bus type interface to the IO50/IO100. If address and data signals are already available at a UUT connector, the few remaining signals could be provided on a small connector, or any remaining spare pins. Using a connector harness which plugs into an actual memory device socket is also possible, but will usually not be rugged enough for the test environment. The configuration shown requires an open collector type of acknowledge signal, and assumes a pullup resistor is provided on the UUT. The IO50/IO100 does provide a 10K pullup resistor on each acknowledge signal which will be in parallel with the UUT pullup device. To satisfy signal rise time requirements, UUT pullups of 1K or less are often used.

The test cable should include paired ground wire connections on all handshake and tristate control signals used to minimize crosstalk. If ribbon cable is used, extra ground pins on the UUT test connector may ease cable assembly.

Many microprocessor architectures provide an access strobe/acknowledge transfer type of handshake to memory devices. An access error timer is often used to indicate a non-existent memory or bus error condition is the strobe remains on too long without an acknowledge. To avoid timeout conditions when emulating ROM operation, this timer duration should be set to 300 us or more. This value could be constant, or made shorter during non-test operation, if system performance might suffer. For RAM emulation, separate read and write strobes are required. If the MPU architecture on the UUT uses a single read/write line, the decoding logic shown will be required. The logic shown is meant to be symbolic, and can be implemented in other device types, or with programmable logic.

For the examples shown in the figure, the active test operation is Memory Emulation. This operation has an inverted acknowledge handshake strobe (negative true), with an open collector type characteristic. The request handshake signals may be either normal (low for request) or inverted (high for request) for Memory Emulation. This figure shows normal polarity request signals being generated by the UUT. It also shows normal tristate control polarity (low enabled).

ROM Emulation

(See Fig 5-1) The following program example shows the commands required to emulate a EPROM used to provide 16 bit data. Only 16 locations of the EPROM are used, to provide a simple boot up and jump to self operation. The user would connect the lower 4 address bits to the I/O pins defined as ROM_ADDR, and the data pins to ROM_DATA. These pins can be located on any connector, and in non-sequential order. The field definition commands can shuffle the pins connected to any desired order to correct for cabling changes. Note that the address field definition will assign significance to the pins in the order listed. The most significant address bit will be the leftmost listed in the field pin assignment string. The least significant bit will be the rightmost pin in the string. Note also that vectors are numbered with decimal values starting at 1 not 0. This difference will cause the test vector number to be one higher than the ROM address as seen by the UUT.

```
{Setup}

    TEST:DEF A:MEMEMULATION:SIZE 16          ;Define and reserve memory
    FIELD:DEF ROM_ADDR:PIN A1,A2,A4,A3       ;Define address field on A
    FIELD:NAME ROM_ADDR:TRIST INPUT          ;Make input only field
    FIELD:DEF ROM_DATA:PINS B16-1            ;Define data field on B
    FIELD:NAME ROM_DATA:TRIST EXTNORM        ;Data field tristate control
    SYST:FIELD ROM_DATA                      ;Change active field to data
    VECTOR 1:COUNT 8;DATA 0000,0008,0000,0010,4E71,4E71,4E71,4E71
    VECTOR 9:COUNT 8;DATA 4E71,4E71,4E71,4E71,4E71,4E71,4E71,4E7E
                                             ;Data in “ROM” address 0-F

{Specify address for Memory Emulation}

    TEST:ADDR ROM_ADDR                       ;Declare field to use for address

{Start test}

    INITIALIZE:BLOCK                         ;Now UUT controls flow

{Check operation status}

    TEST:NAME ALL:STATUS?                    ;Check last vector addressed

{Returns Test letter, EXECUTING status, Number of last vector read}

{End test}

    ABORT

{Delete all tests before continuing to next}

    TEST:NAME ALL:DEL
```

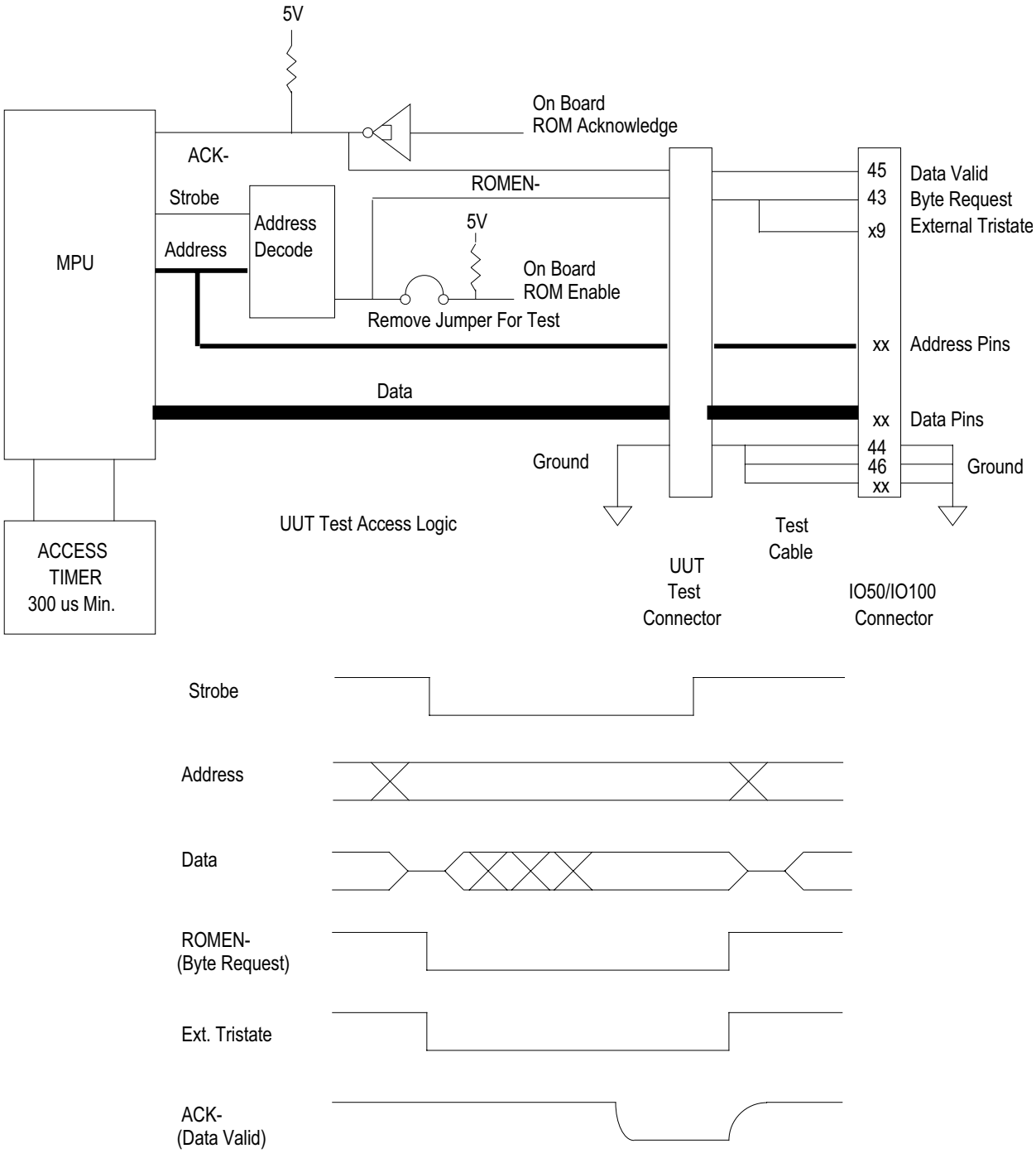


Figure 5-1.
R0M Emulation Example.

RAM Emulation

(See Fig 5-2) The following program example shows the commands required to emulate a 32-word by 16-bit RAM used for stack and scratchpad operation by the UUT. The UUT memory map normally uses high RAM for stack storage, and low RAM for scratchpad. The emulation operation will provide 16 bytes starting at address 0x0000 and 16 bytes starting at address 0x8000. The memory split is performed by assignment of UUT address bit 15 to address bit 4 position in the address field. The RAM is initialized with 0 in the stack area and an 0xFF pattern in the scratchpad area. One scratchpad location is changed while the test is running to cause an action by the UUT firmware. This action will modify the stack area, which may be examined while the test executes.

```
{Setup}

    TEST:DEF D:MEMEMULATION:SIZE 32              ;Define and reserve memory
    FIELD:DEF RAM_ADDR:PIN A15,A4-1              ;Define address field on A
    FIELD:NAME RAM_ADDR:TRIST INPUT              ;Make input only field
    FIELD:DEF RAM_DATA:PINS B8-1                 ;Define data field on B
    FIELD:NAME RAM_DATA:TRIST EXTNORM            ;Data field tristate control
    VECTOR 1:COUNT 8;DATA:FIELD RAM_DATA;VALUE FF,FF,FF,FF,FF,FF,FF,FF
    VECTOR 9:COUNT 8;DATA:FIELD RAM_DATA;VALUE FF,FF,FF,FF,FF,FF,FF,FF
                                                 ;Data in scratchpad address 0-F
    VECTOR 17:COUNT 8;DATA:FIELD RAM_DATA;VALUE 00,00,00,00,00,00,00,00
    VECTOR 25:COUNT 8;DATA:FIELD RAM_DATA;VALUE 00,00,00,00,00,00,00,00
                                                 ;Data in stack address 10-1F

{Specify address for Memory Emulation}

    TEST:ADDR RAM_ADDR                           ;Declare field to use for address

{Start test}

    INITIALIZE:BLOCK                             ;Now UUT controls flow

{Check operation status}

    TEST:NAME ALL:STATUS?                        ;Check last vector addressed

{Returns Test letter, EXECUTING status, Number of last vector read}

{Change scratchpad location 0xF}

    SYST:FIELD RAM_DATA                          ;Select data as active field
    VECTOR 16:DATA 55                            ;Write flag to scratchpad

{Check for UUT change of stack location 0}
```

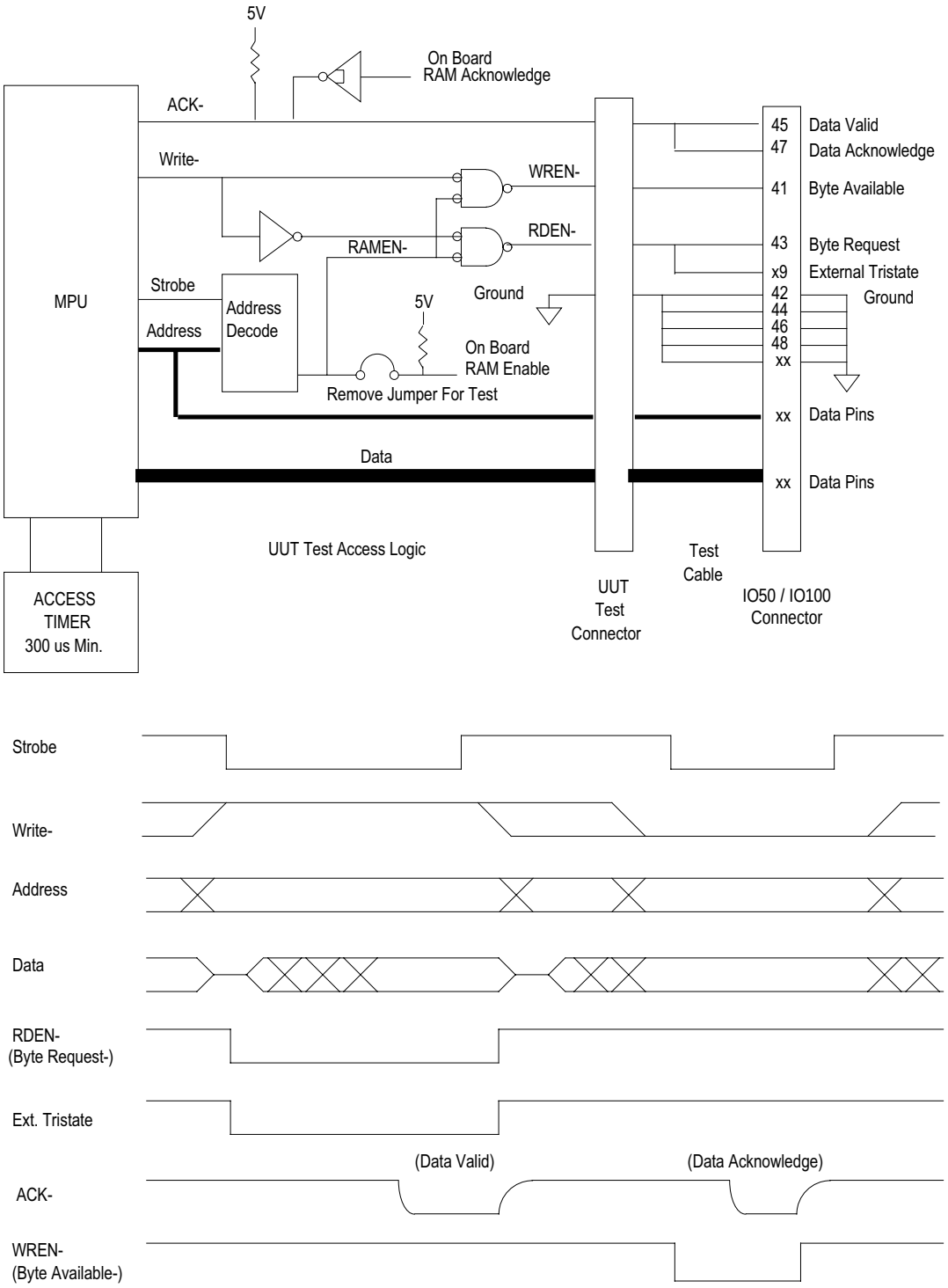


Figure 5-2.
RAM Emulation Example.

```
    VECTOR 17:COUNT ALL;DATA:VAL?
```

{Returns 16 byte hex value stored at UUT address 0x8000-800F}

{End test}

```
    ABORT
```

{Delete all tests before continuing to next}

```
    TEST:NAME ALL:DEL
```

ROM Emulation with Programmed I/O

The ability to force a UUT MPU to execute instruction test loops while examining output and input signals can be a valuable test aid. The ROM Emulation, once started, will run until an ABORT command is issued while it is the active test. The Programmed I/O Timed test can be used during ROM emulation (without handshake signals) to force input signals or examine outputs.

This example shows a test loop that causes the UUT MPU to perform a simple power up reset sequence, then write a pattern to a peripheral port. After writing to this location, the MPU will enter and endless wait loop. The Programmed I/O test can then read in and verify the pattern latched in the peripheral port.

{Setup}

```
    TEST:DEF A:MEMEMULATION:SIZE 16          ;Define and reserve memory
    FIELD:DEF ROM_ADDR:PIN A1,A2,A4,A3       ;Define address field on A
    FIELD:NAME ROM_ADDR:TRIST INPUT          ;Make input only field
    FIELD:DEF ROM_DATA:PINS B16-1            ;Define data field on B
    FIELD:NAME ROM_DATA:TRIST EXTNORM        ;Data field tristate control
    SYST:FIELD ROM_DATA                      ;Change active field to data
    VECTOR 1:COUNT 8;DATA 0000,0008,0000,0010,80FA,0010,CC00,0000
    VECTOR 9:COUNT 8;DATA A5A5,4E71,4E71,4E71,4E71,4E71,4E71,4E7E
                                             ;Data in “ROM” address 0-F
```

{Specify address for Memory Emulation}

```
    TEST:ADDR ROM_ADDR                       ;Declare field to use for address
    TEST:DEF B:PRGIOT                        ;Define another test on B
    SYST:TEST B                              ;Make it the active test
```

```
    FIELD:DEF PORT_DAT:PIN B32-25          ;Define field to read port
    FIELD:NAME PORT_DAT:TRISTINPUT         ;Tristate input
    SYST:TEST A                            ;Active test to start

{Start test}

    INITIALIZE:BLOCK                       ;Now ROM Emulation starts
    SYST:TEST B
    INIT:IN                                ;Read from UUT port latch
    VECT 1:DATA:FIELD PORT_DAT;VAL?        ;Get data to verify

{Returns eight bit value read in from pins}

    ABORT                                  ;Halt Programmed I/O test
    SYST:TEST A ABORT                      ;Halt ROM Emulation test

{End test}
```

Techniques for Clocking With Data Pins

The IO50/IO100 is a general purpose digital interface for low to medium speed devices. The data signals are single ended and are carried on ribbon cable type connectors, making them susceptible to crosstalk. When adjacent signals switch, static data signals may “glitch” across logic levels for tens of nanoseconds. The handshake signals provided on each connector have paired grounds and are positioned to minimize crosstalk. These signals should be used whenever clocking fast logic devices, but many users have requirements for other clock signals. Data pins may be used in these cases if the following guidelines are used. Two data pins on each side of the clock data pin, and within the same octet, should be configured as output only and left at a low level. Performance may be improved by replacing the standard pullup resistor pack on the IO50/IO100 with a pullup/pulldown type, replacing the series termination resistor pack with a shorting pack, and using a matching termination at the signal destination on the UUT. An open collector driver device may be used with this type of transmission line termination to provide increased sink current.

Output and Input Testing of Memory Devices

(See Fig 5-3). The timed block output test can be used to fill memory devices with a data pattern, then programmed I/O can read back and verify correct storage. The throughput rate may not be sufficient for large memory arrays, but small devices, or blocks of large devices, can be tested in a reasonable time. A simple walking one pattern is used in this example. The UUT memory must be filled from low to high, then read back in the same order. In this fashion, any shorted address lines will be exposed by locations containing a higher value than expected. If larger memory devices are to be tested, the Slot 0 Controller could fill the shared RAM area with the pattern for block output. This would decrease test time by eliminating all the data vector commands.

Data is first written using the Block Output Timed test with an inverted handshake to match the UUT write strobe requirement. The inverted acknowledge handshake has an open collector output characteristic. The timeout value of the block output test can be increased to provide a longer write pulse if required.

Data is read using the Programmed I/O Timed test with an inverted strobe to match the UUT output enable requirements. This test must be run under the name of a different connector than the Block Output. This is necessary because the Data Valid output handshake signal from the first connector is connected to the memory write line. This line must not move when the address is output during the readback operation, so another connector control group is used. The Data Valid handshake signal of this second group remains unconnected. Note that both tests use the same data pins, even though they are in different connector control groups.

The data comparison and verification is performed on the Slot 0 Controller after read back.

{Setup}

```
TEST:DEF A:BLKOUTH:SIZE 16                          ;Define test for fill
FIELD:DEF ADDR:PIN A16-1                            ;Memory address field
FIELD:DEF DATA:PIN A32-17                           ;Memory data field
FIELD:NAME ADDR:TRIST OUT                           ;Address outputs on
FIELD:NAME DATA:TRIST OUT                           ;Data outputs on for write
VECT 1:DATA:FIELD ADDR;VAL 0;FIELD DATA;VAL 1       ;Walking 1 data
VECT 2:DATA:FIELD ADDR;VAL 1;FIELD DATA;VAL 2
VECT 3:DATA:FIELD ADDR;VAL 2;FIELD DATA;VAL 4
VECT 4:DATA:FIELD ADDR;VAL 3;FIELD DATA;VAL 8
VECT 5:DATA:FIELD ADDR;VAL 4;FIELD DATA;VAL 10
VECT 6:DATA:FIELD ADDR;VAL 5;FIELD DATA;VAL 20
VECT 7:DATA:FIELD ADDR;VAL 6;FIELD DATA;VAL 40
VECT 8:DATA:FIELD ADDR;VAL 7;FIELD DATA;VAL 80
VECT 9:DATA:FIELD ADDR;VAL 8;FIELD DATA;VAL 100
VECT 10:DATA:FIELD ADDR;VAL 9;FIELD DATA;VAL 200
VECT 11:DATA:FIELD ADDR;VAL A;FIELD DATA;VAL 400
VECT 12:DATA:FIELD ADDR;VAL B;FIELD DATA;VAL 800
VECT 13:DATA:FIELD ADDR;VAL C;FIELD DATA;VAL 1000
VECT 14:DATA:FIELD ADDR;VAL D;FIELD DATA;VAL 2000
VECT 15:DATA:FIELD ADDR;VAL E;FIELD DATA;VAL 4000
VECT 16:DATA:FIELD ADDR;VAL F;FIELD DATA;VAL 8000
```

{Start test}

```
INIT:BLOCK
```

{Check for test completion}

```
TEST:NAME ALL:STATUS?
```

{Returns Stopped when done filling memory}

```
ABORT                                   ;Stop when done
FIELD:NAME DATA:TRIST CLEAR             ;Free data pins for input
TEST:DEF B:PRGIOT                       ;Verify test on conn. B
SYST:TEST B                             ;Make active test for field def's.
FIELD:DEF ADDR:PIN A16-1                ;Address field same as fill
FIELD:DEF DATA:PIN A32-17               ;Data field same as fill
FIELD:NAME ADDR:TRIST OUT               ;Address out as before, no con-
flict
FIELD:NAME DATA:TRIST IN                ;Data pins are now inputs
VECT 1:DATA:FIELD ADDR;VAL 0            ;Set address out for readback
INIT:OUT                                ;Output address for setup
INIT:IN                                 ;Read in data
VECT 1:DATA:FIELD DATA;VAL ?            ;Return data to Slot 0
```

{Slot 0 verifies returned data with expected data}

```
VECT 1:DATA:FIELD ADDR;VAL 1            ;Repeat for 16 bit walking one
INIT:OUT INIT:IN VECT 1:DATA:FIELD DATA;VAL?
VECT 1:DATA:FIELD ADDR;VAL 2 INIT:OUT INIT:IN VECT
1:DATA:FIELD:DATA;VAL?
```

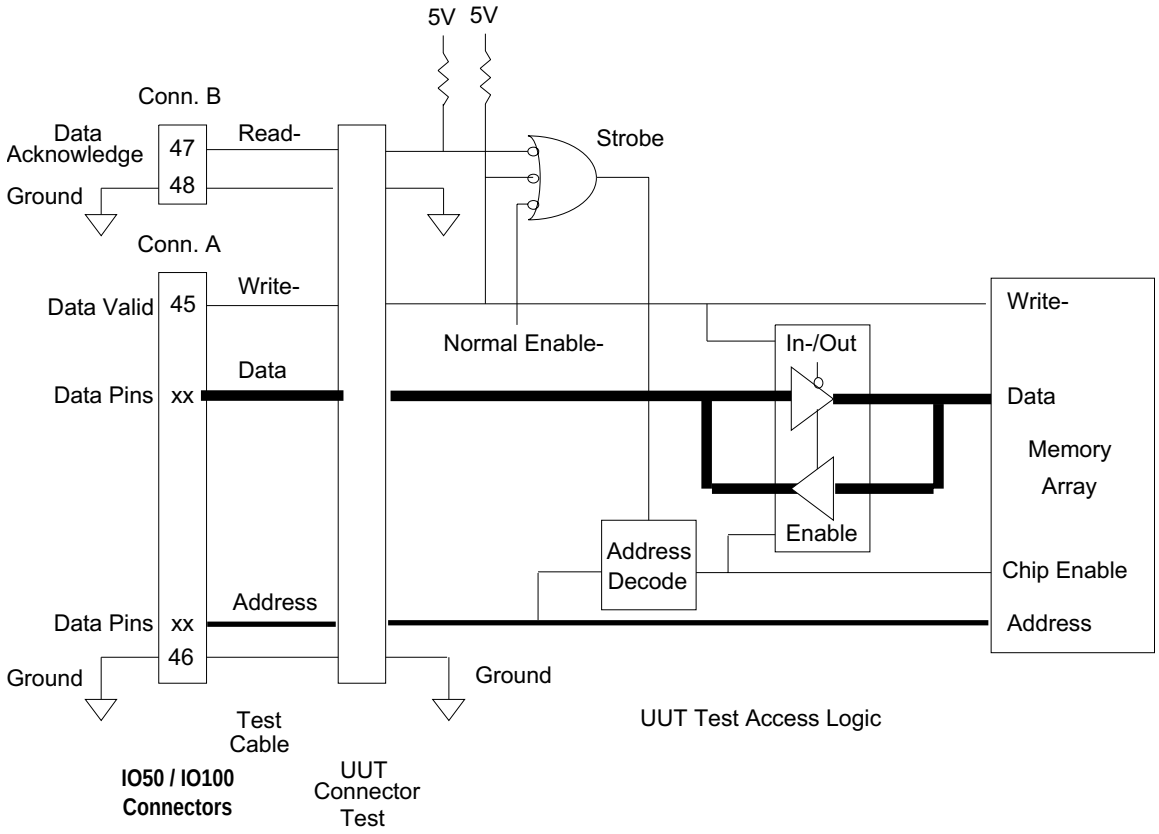
{Continues for remaining pattern values}

```
ABORT TEST:NAME ALL:DEL
```

Testing Memory Mapped I/O Control Logic

(See Fig 5-4). Many UUT devices that are normally controlled by a microprocessor elsewhere in the system, require a control sequence to enable their functional operation for testing. The IO50/IO100 can provide the address, data, and strobe sequences required to set, and read back, UUT control registers and status ports.

In this example, read and write address decoders on the UUT are strobed by the IO50/IO100 while data is provided or read through the normal UUT microprocessor path. The Programmed I/O Timed operation is used with normal (non-inverted) handshake as might be used by a dedicated microcontroller. The data bus is a tristate interface with data flow controlled by the



Memory Array Testing

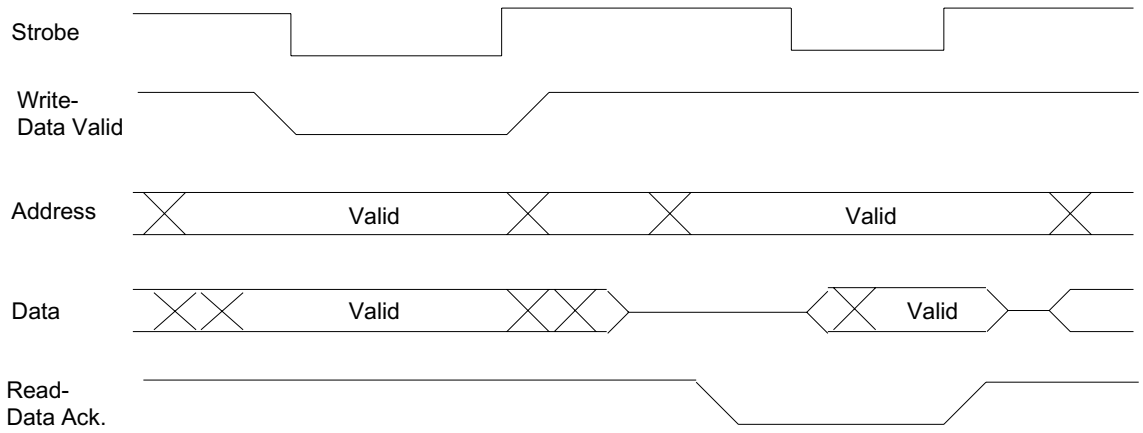
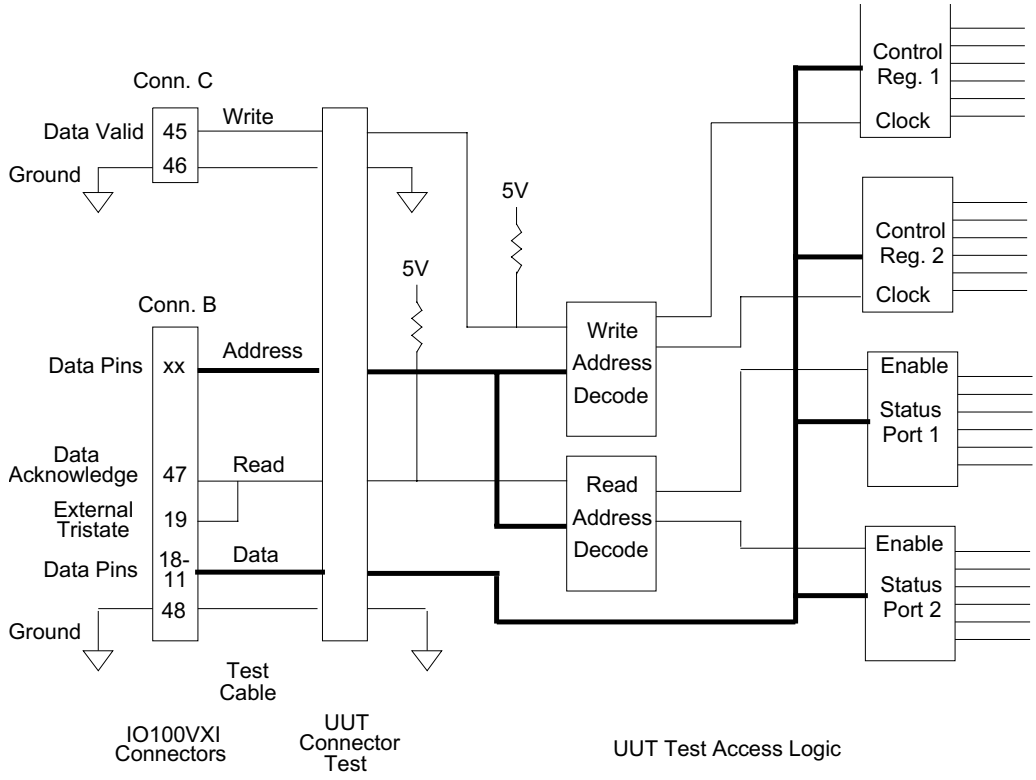


Figure 5-3.
Memory Array Test Example.

IO50/IO100. Instead of reprogramming the internal data pin tristate control for each change from write to read, the external tristate controls are used in a feed back fashion to switch data direction.

The test will set control patterns in registers 1 and 2 first, then read status ports 1 and 2 to determine correct operation. The read back operation must be performed on a different connector from the write operation, for the same reason given in the Memory I/O Test example above.

```
{Setup}

    TEST:DEF C:PRGIOT                       ;       Define write test
    FIELD:DEF CTL_ADDR:PIN B32,B2,B1        ;Define address field
    FIELD:DEF CTL_DATA:PIN B15-8            ;Define data field
    FIELD:NAME CTL_ADDR:TRIST OUT           ;Address field outputs on
    FIELD:NAME CTL_DATA:TRIST EXTNORM       ;Data field outputs on

{Send control data pattern to first register}

    VECT 1:DATA:FIELD CTL_ADDR;VAL 0;FIELD CTL_DATA;VAL 4C
    INIT:OUT

{Send control data pattern to second register}

    VECT 1:DATA:FIELD CTL_ADDR;VAL 1;FIELD CTL_DATA;VAL 22
    INIT:OUT
    ABORT

{Now verify status resulting from control outputs}

    TEST:DEF B:PRGIOT                       ;Define read test
    SYST:TEST B                             ;Make it the active test
    FIELD:DEF CTL_ADDR:PIN B32,B2,B1        ;Same field as write
    FIELD:DEF CTL_DATA:PIN B15-8            ;Same field as write
    FIELD:NAME CTL_ADDR:TRIST OUT
    FIELD:NAME CTL_DATA:TRIST EXTNORM
    SYST:FIELD CTL_DATA                     ;Change active field for all reads
    VECT 1:DATA:FIELD CTL_ADDR;VAL 3        ;Set read address
    INIT:IN                                 ;Read in status
    VECT 1:DATA:VAL?                        ;Return to Slot 0
```



Memory Mapped Control Testing

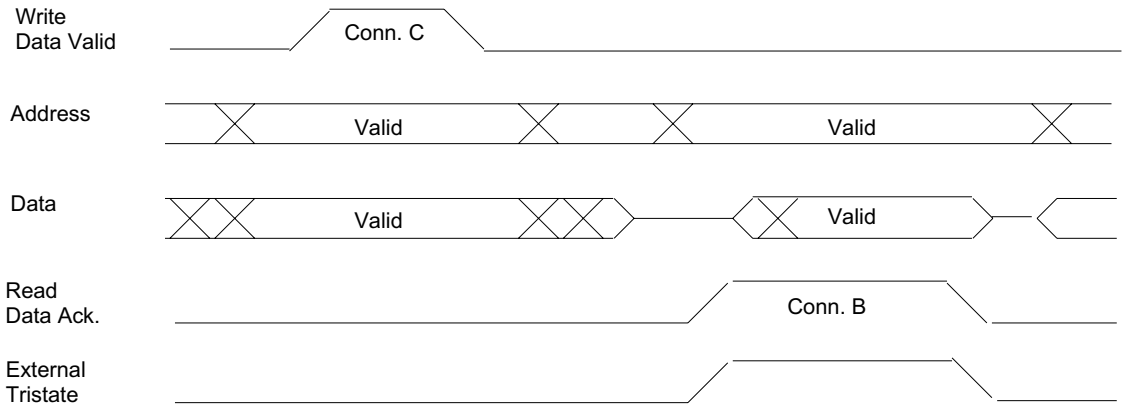


Figure 5-4.
Memory Mapped Control Test Example.

{Slot 0 verifies first status result}

```
VECT 1:DATA:FIELD CTL_ADDR;VAL 4        ;Next read address
INIT:IN VECT 1:DATA:VAL?
```

{Slot 0 verifies second status result}

```
ABORT TEST:NAME ALL:DEL
```

Counter or FIFO Testing

See Fig 5-5. A UUT device which outputs changing data in response to a strobe input can be tested with the Block Input Timed test. The example below shows data read back using the VECTOR commands. The UUT output data could be read directly from the shared memory locations for minimum test times. In this case a TEST:NAME ALL:CAT? command would be used to determine the start address of the shared memory vector area.

This example shows an 12 bit counter chain tested with an inverted handshake output. The counter increments on the rising clock edge, so the first state read should be 0. The timeout value of the block input test may be increased to allow for longer counter propagation delays.

A single data bit is used to reset the counter before testing. This pin is used with adjacent guard pins as recommended in the application section on clocking with data pins. The reset is a low true signal. The reset signal is output in another connector control group so that it will remain static during the block test operation.

{Setup}

```
TEST:DEF B:PRGIOT                       ;Define reset test
FIELD:DEF RESET:PIN A2                  ;Define reset pin
FIELD:DEF GUARD:PIN A4,A3,A1            ;Define constant level pins
FIELD:NAME RESET:TRIST OUT              ;Turn on reset pin
FIELD:NAME GUARD:TRIST OUT              ;Force guards low
```

{Reset counter before test}

```
VECT 1:DATA:FIELD RESET;VAL 0;FIELD GUARD;VAL 0 INIT:OUT
VECT 1:DATA:FIELD RESET;VAL 1;FIELD GUARD;VAL 0 INIT:OUT
```

{Reset line stays high throughout remainder of test}

```
TEST:NAME ALL:STAT?
```

{Poll for completion before continuing}

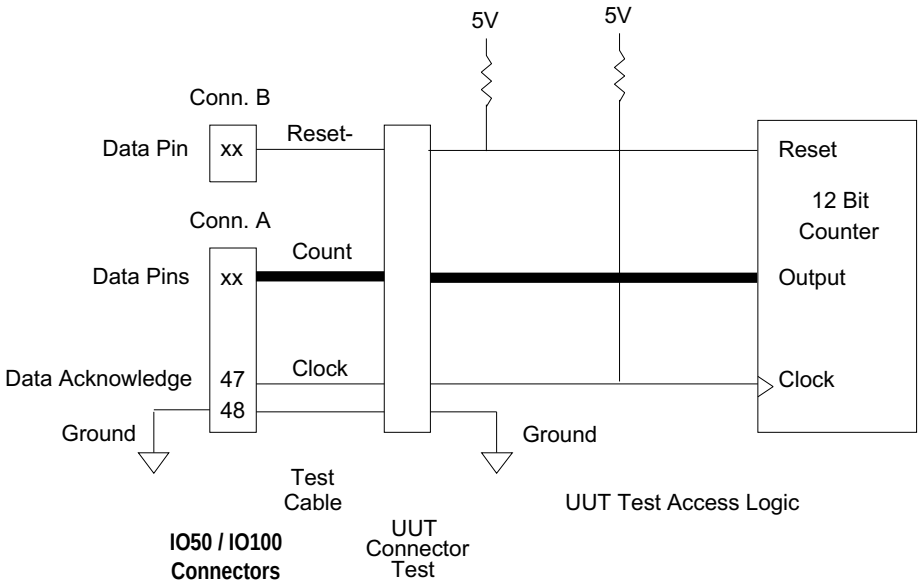
```
TEST:DEF A:BLKINT:SIZE 4096          ;Define block input test
SYST:TEST A                          ;Make the active test
FIELD:DEF CNT_DATA:PIN D12-1         ;Define field in new test
FIELD:NAME CNT_DATA:TRIST IN         ;Make inputs bidirectional
INIT:BLOCK                           ;Start block read
```

{Poll for completion before continuing}

```
TEST:NAME A:STAT?                    ;Return stopped when done
VECT 1:COUNT 8;DATA:VAL?             ;Read data from active field
VECT 9:COUNT 8;DATA:VAL? VECT 17:COUNT 8;DATA:VAL?
```

{Continue for remaining data verification}

```
ABORT TEST:NAME ALL:DEL
```



Counter Pattern Testing

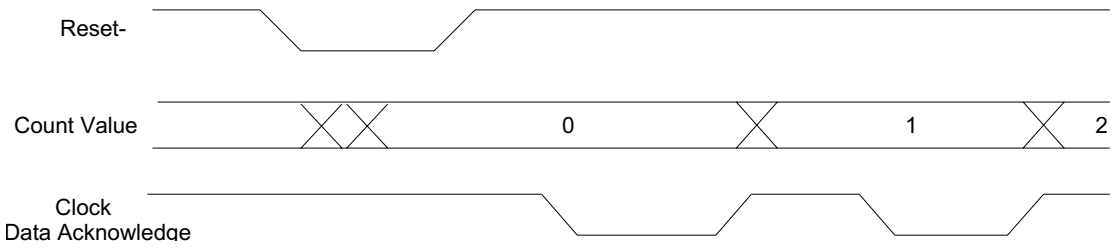


Figure 5-5.
Counter Pattern Test Example.

(THIS PAGE INTENTIONALLY LEFT BLANK)

CHAPTER 6

Installation and Basic Operation

Scope of Chapter

This chapter contains instructions for unpacking, inspecting, installing, and checking out the IO50 / IO100 Series Digital I/O Modules.

Unpacking and Inspection

Your IO50 / IO100 was thoroughly inspected and tested before shipment from the factory and is ready for immediate operation once all installation procedures have been completed. Carefully remove the instrument from its shipping carton and check for any obvious damage that may have occurred during shipment. If damage is found, report it to the freight carrier immediately. Interface Technology is not liable for damage that may have occurred during transit. Save the shipping carton and all packing material for possible future use.

Installation

Logical Addressing

Before installation, it is necessary to set the logical address of the IO50 / IO100. The address switches should be set according to the requirements of your Slot-0 Controller and system configuration. Each VXI instrument must have a unique address. The location of the address switch is shown in Fig 6-1. Switch positions are marked from 1 to 8, with 1 corresponding to the least significant bit of the logical address. Switches in the "on" (closed) position set their corresponding address bit to 0. Conversely, switches in the "off" (open) position set their corresponding bit to 1. To

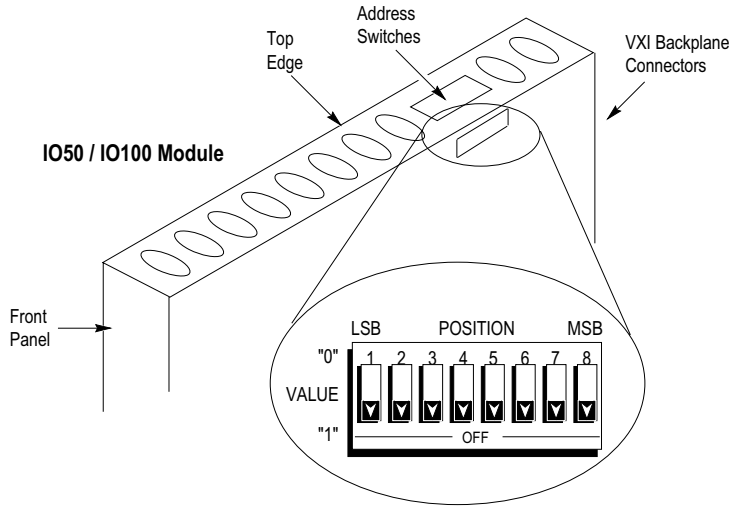


Figure 6-1.
Address Switches Set to
Dynamic Configuration Position (all 1's).

allow your Slot-0 Controller to perform dynamic configuration, set all switches to "off" (open). This corresponds to logical address 255 (FFh).

Interrupter devices such as the IO50 / IO100 are required to place their logical address value on the lower byte of the data bus in response to bus Interrupt Acknowledge cycles. The Slot-0 Controller uses this value as an index to its internal exception handling vector table. Be sure that the address set on the logical address switch does not conflict with other vector table locations reserved by system software. Refer to your Slot-0 documentation for the type of calculation used to convert the address switch value to a table address. If you use dynamic configuration, the Resource Manager software must assign the logical address to a board, being careful to protect its own reserved vector table locations. If you do not intend to enable IO50 / IO100 VME interrupts, these precautions can be ignored.

Note

Some VXI backplanes provide bypass jumpers or switches for each empty slot to pass the interrupt and bus acquisition control signals. The IO50 / IO100 does not use the bus acquisition signals but does use the interrupt priority signal. Any backplane jumpers in the slot position where the IO50 / IO100 is installed should therefore be removed. The IO50 / IO100 provides internal pass through connection of the bus acquisition signals.

The output signals of the IO50 / IO100 are source by various driver ICs, depending on the model configuration. The selection of output series resistance and pullup resistance has been chosen to meet the majority of user requirements. Appendix B shows the standard configuration of the input and output logic connected to each pin and the locations of socketed parts on the IO50 / IO100. Appendix B also makes some recommendations for special requirements. If socketed parts are to be changed, the top shield cover must be removed, parts exchanged, and the cover replaced. Anti-static precautions should be observed when changing any part.

After the logical address has been set and any socketed parts replacement completed, the IO50 / IO100 can be installed in the VXI chassis. Once the IO50 / IO100 module is seated in the chassis, tighten the retaining screws at the top and bottom of the module front panel.

Slot Dependency

No VXI Local Bus backplane lines are connected, so the IO50 / IO100 may be installed adjacent to any other VXI device. It is not necessary to place IO50 / IO100 modules in adjacent slots if they are being slaved together since the VXI TTLTRG lines used for this operation connect to all backplane slots.

Calibration

The IO50 / IO100 does not require calibration.

Basic Operation

Self-Test

When the VXI card cage is powered up, or after a hardware reset, the IO50 / IO100 will perform a self test operation. If this test completes successfully, the red SYSFAIL LED will be extinguished within 5 seconds. The test will check operation of the local and shared static RAM, operation of the local timer, the ability to read and write the shared device dependent registers, and the ability to read and write the VXI communication registers. The outputs are not tested in a wrap-around fashion because the UUT signal connections cannot be predicted.

If the SYSFAIL LED does not turn off in 5 seconds, an error has been detected, and the backplane SYSFAIL signal will be driven in accordance with the VXI specification. The Slot 0 Resource Manager then has the option of placing the IO50 / IO100 in safe state and aborting the test sequence, or continuing with other tests. Possible causes of failure, not including component failures, are bent pins on the VXI rear connectors, or failure to fully seat the board into the chassis and mate with the backplane connectors.

If the self test fails, the local processor will attempt to put limited diagnostic information in the VXI Data Low Data Register. This can be read by the Slot 0 Controller using the Byte Request VXI command. Table 2-1 on the following page provides a list of the self-test error codes and their descriptions.

Basicmode I/O

Before executing example commands, some mention of the VXI message based protocol implementation of the IO50 / IO100 is required. Most Slot 0 Controllers offer some form of message based communication utility which will probably meet the IO50 / IO100 requirements.

NOTE:
Proprietary Slot 0 Controllers may need the following information.

The IO50 / IO100 will not put data into its output buffer until a command requesting output is executed. The Response Register Read Ready bit will be set to one at that time. When data is read back, the End message bit will be set true in the last data byte. All data up to and including the end message must be read. Each command executed will update the error status message, which is available via the SYSTem:ERRor? command. During initial development and debug, it is a good idea to send the SYSTem:ERRor? command after every command, and verify a No Error return status.

The Basicmode function is a simplified version of the programmed input/output function. Data is not defined in fields with representative names or

Table 2-1.

Code (hex)	Description
100	Ram Test Error in Program RAM
101	Ram Test Error in Program RAM
200	Ram Test Error in Shared RAM
201	Ram Test Error in Shared RAM
300	VXI ASIC Read/Write Acknowledge Test Failure
301	VXI ASIC Read/Write Pattern Test Failure
400	Shadow RAM Test Failure
600	ROM Checksum Test Failure

transferred under any handshake signal or timed interval control. In Basicmode, pins are defined in output and input lists, then data values are transferred. The output enables are forced on or off by the command execution, no control or sequencing is expected (or possible) by the UUT.

Outputs and inputs are defined as pin groups on byte boundaries for each connector (32-25, 24-17, 16-9, 8-1). The shorthand reference used for these groups in the command set is 25, 17, 9, and 1. These values will always define pin groups of exactly eight bits. Pin definition commands may specify multiple bytes or may skip bytes (i.e.: A25, A17, A1).

Once the output pin list has been defined, data sent is apportioned to the output pins on a byte basis in the order they were listed in the pin definition command (from left to right). Data read back from the defined input pin list will be returned in the order listed in the pin definition command. Users may take advantage of this listing order control to perform byte swapping of data read from sources with differing byte order conventions. If a data value sent is not wide enough to fill the defined output pin field, bits will be zero filled from most significant down. In this case, the pins specified last in the definition command (rightmost), will output the available data, and the pins specified first (leftmost), will output zeros. Input commands will return a data value equal in width to the number of defined input pins.

Data values transferred may be from one bit up to 128 bits, all will be read or written simultaneously. The data value output may be read back for verification. Five examples of Basicmode programming are provided on the following pages.

1. **Output TTL high level on Connector A pins 31,29,..3,1 (odd pins), and low level on Connector A pins 32,30,..4,2 (even pins).**

```
BASIC:DEF:OUT A25,A17,A9,A1     ;define forced output pins
BASIC:OUT 55555555              ;define data for output
```

Input signals on Connector A pins 32-1 (readback data just output) and signals on Connector B pins 16-1 (all high - pullup resistors on open pins).

```
BASIC:OUT?                      ; readback data just output
BASIC:DEF:IN B9,B1              ; define input only pin field
BASIC:IN?                       ; read in unconnected pins
```

2. **To output an incrementing pattern on pins 10 and 9 of connector C, the following commands are executed.**

```
BASIC:DEF:OUT C9,C1             ;note list from MS on left to LS on right-byte
                                 boundary
BASIC:OUT 0000                  ;output values in hex radix
BASIC 100                       ;OUT operator is optional
BASIC 200
BASIC 300
BASIC:OUT?                      ;check last output for shorts
```

3. **Determine current input and output pin assignments.**

```
BASIC:CATALOG?                  ;returns list of unused pins
BASIC:CLEAR                     ;all pins back to undefined
```

4. To output a 16 bit "address" on connector A and 16 bit "data" on connector B, and one bit "write" strobe on connector C pin 1.

```
BASIC:DEF:OUT C1,B9,B1,A9,A1      ;define output pins
BASIC:OUT 000000FE00              ;address :FE00, data 0
BASIC:OUT 001234FE00              ;data now :1234, write low
BASIC:OUT 011234FE00              ;data :1234, write high (true)
BASIC:OUT 001234FE00              ;write low - addr & data held
```

5. To input a 16 bit "data" field on connector B resulting from "address" on connector A pins 16-1 and "read" strobe on connector C pin 1.

```
BASIC:DEF:OUT C1,A9,A1            ;define output pins
BASIC:DEF:IN B9,B1                ;define input pins
BASIC:OUT 00FE00                  ;address :FE00, read low
BASIC:OUT 01FE00                  ;read now high (true)
BASIC:IN?                         ;read in data
BASIC:OUT 00FE00                  ;read low - address held
```

These examples are all done in the simple Basicmode. If the above operations are performed in the more complex modes, pin fields may be given relevant names and manipulated individually, without requiring static data values to be rewritten for each output. Some modes of operation can perform entire input/output sequences under control of the local processor, often more quickly than would be possible using the command language. For examples of this type of programming refer to Chapter 5, Application Examples.

Running and Stopping

The Basicmode of operation on a IO50 / IO100 configured for Master or Standalone operation requires only execution of IN and OUT commands to initiate and complete data transfer. If the IO50 / IO100 is configured for Slave operation, a VXI TTLTRG signal is required to initiate data transfer. This trigger event can come from another IO50 / IO100 board in master mode, or any other VXI trigger generation hardware in the system. This trigger event will input or output a single value. Master/slave operation is described in the Basic Mode programming section.

Pinouts

Connector and pin numbering of all the various I/O modules are indicated on the following pages. See Figs 6-2 through 6-8.

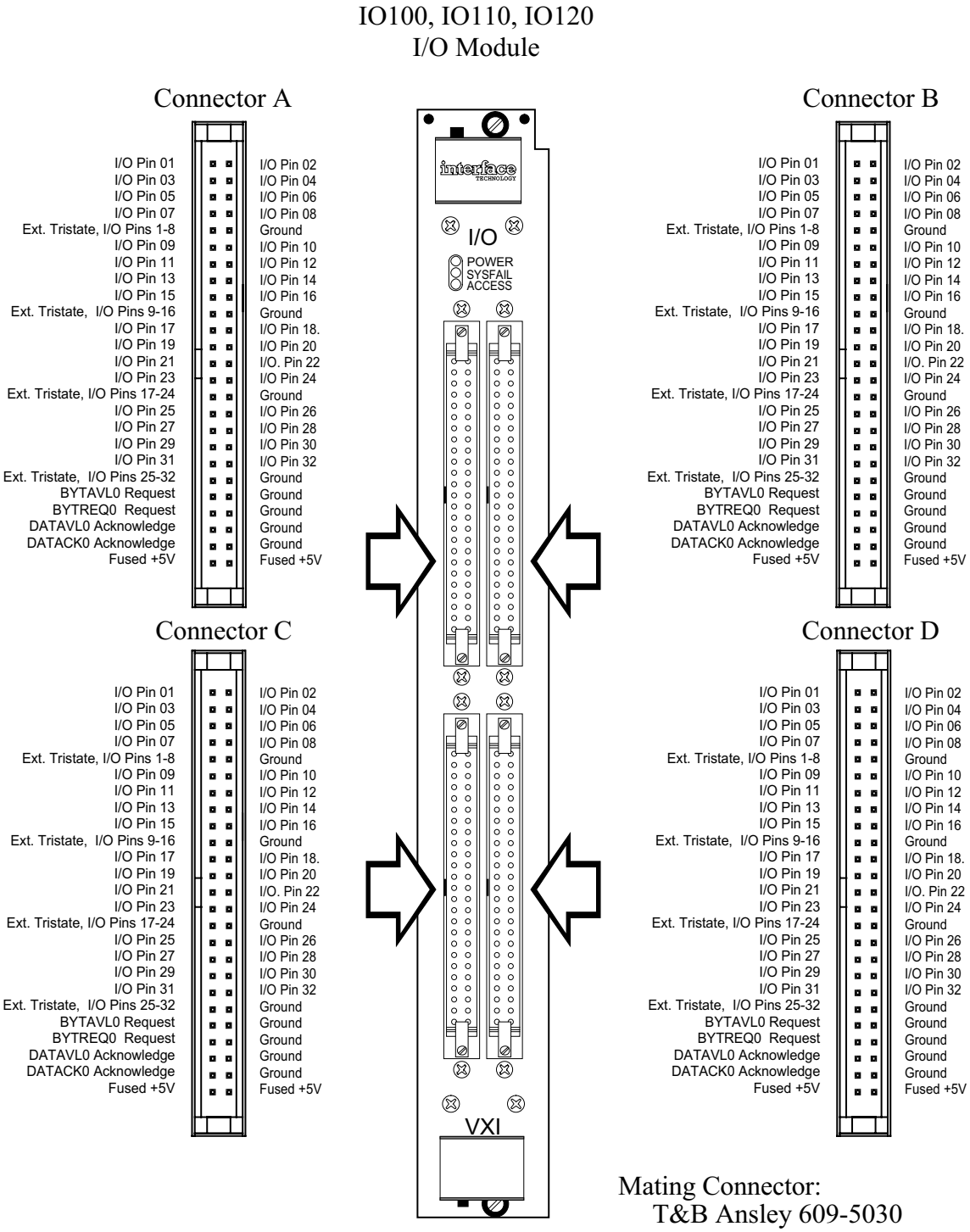
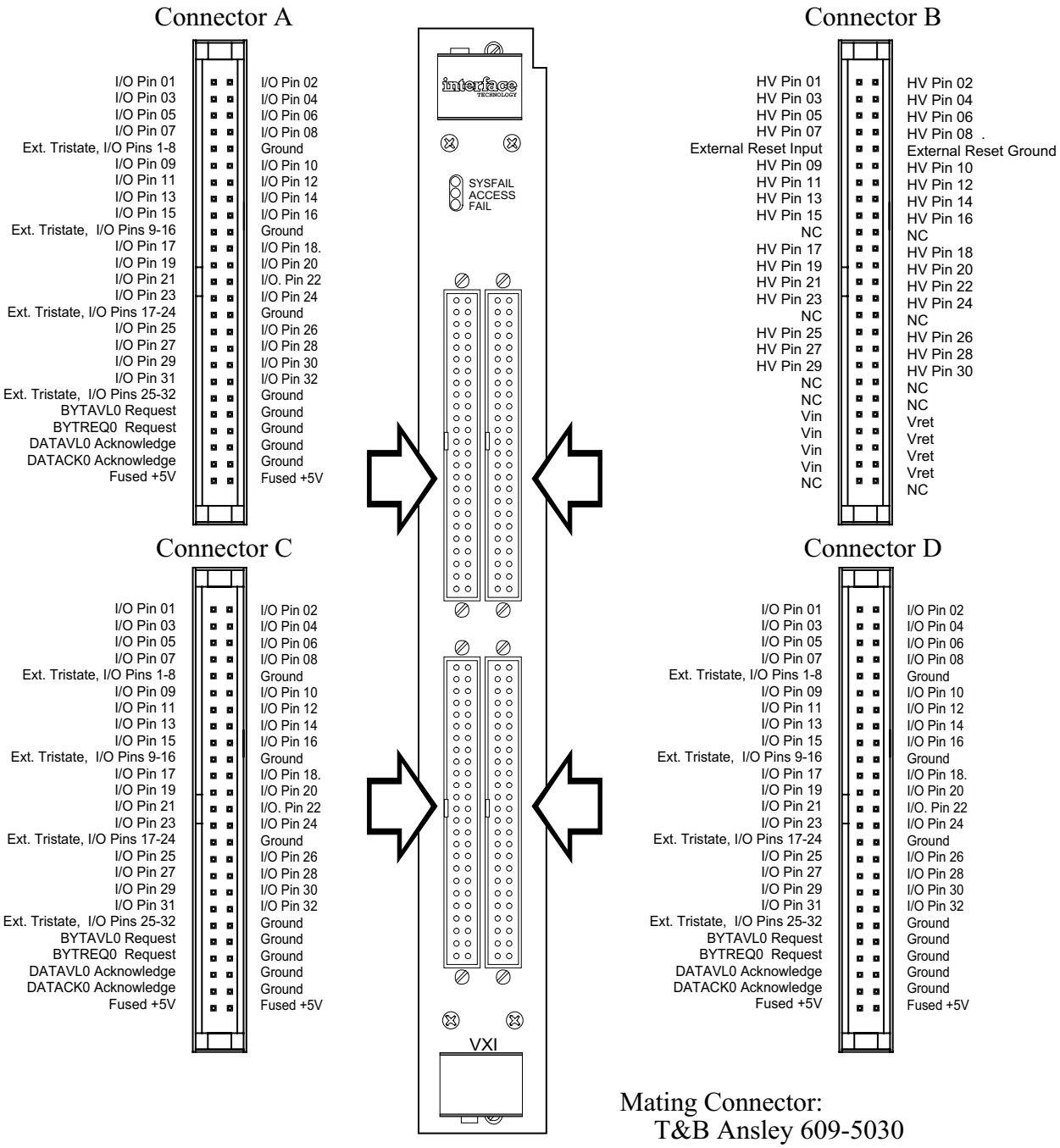


Figure 6-2.
Pinouts for IO100, IO110, IO120.

IO130
I/O Module

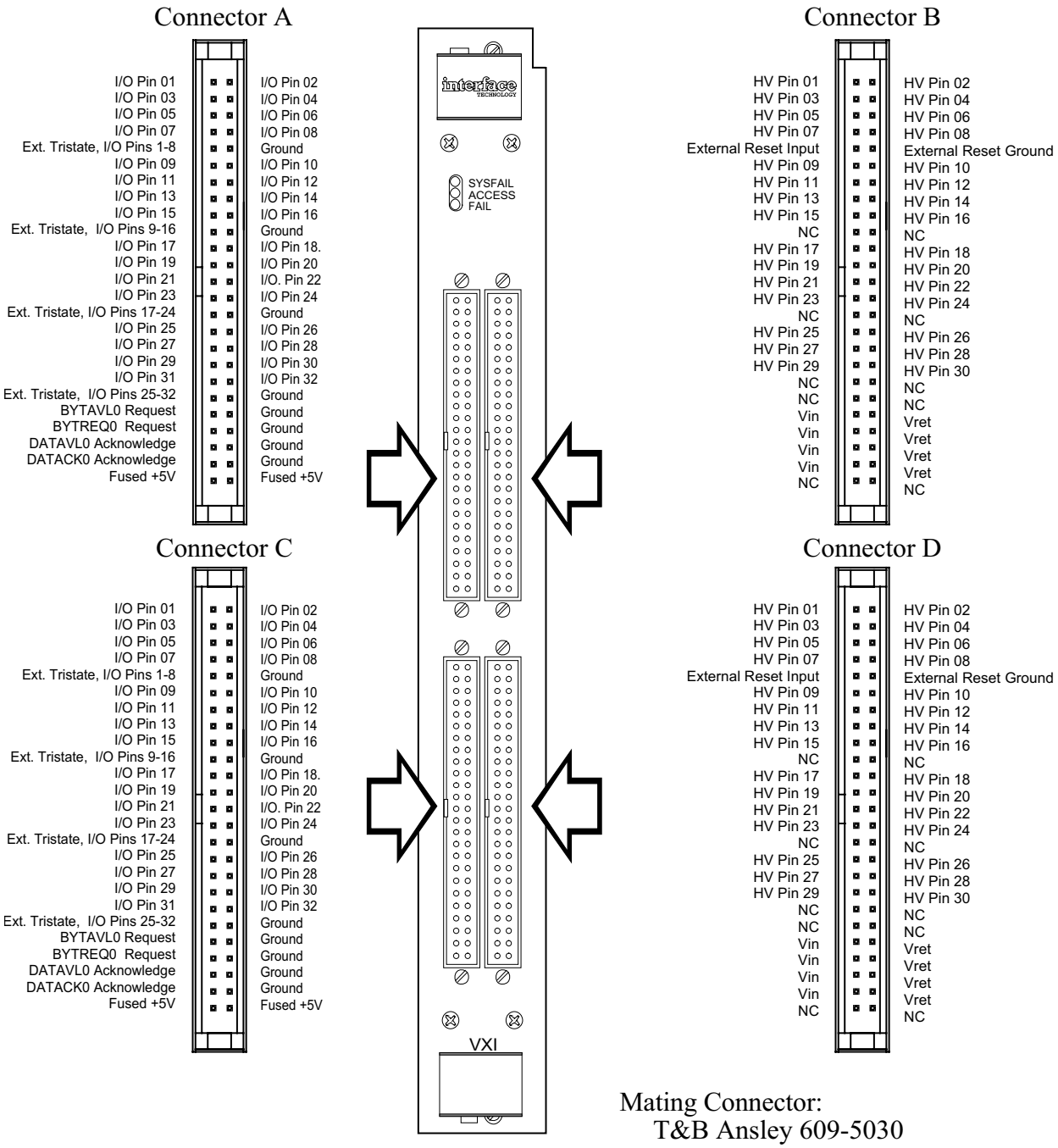


Mating Connector:
T&B Ansley 609-5030

See Appendix D
For Programming Instructions

Figure 6-3.
Pinouts for IO130.

IO130-002
I/O Module



Mating Connector:
T&B Ansley 609-5030

See Appendix D
For Programming Instructions

Figure 6-4.
Pinouts for IO130-002.

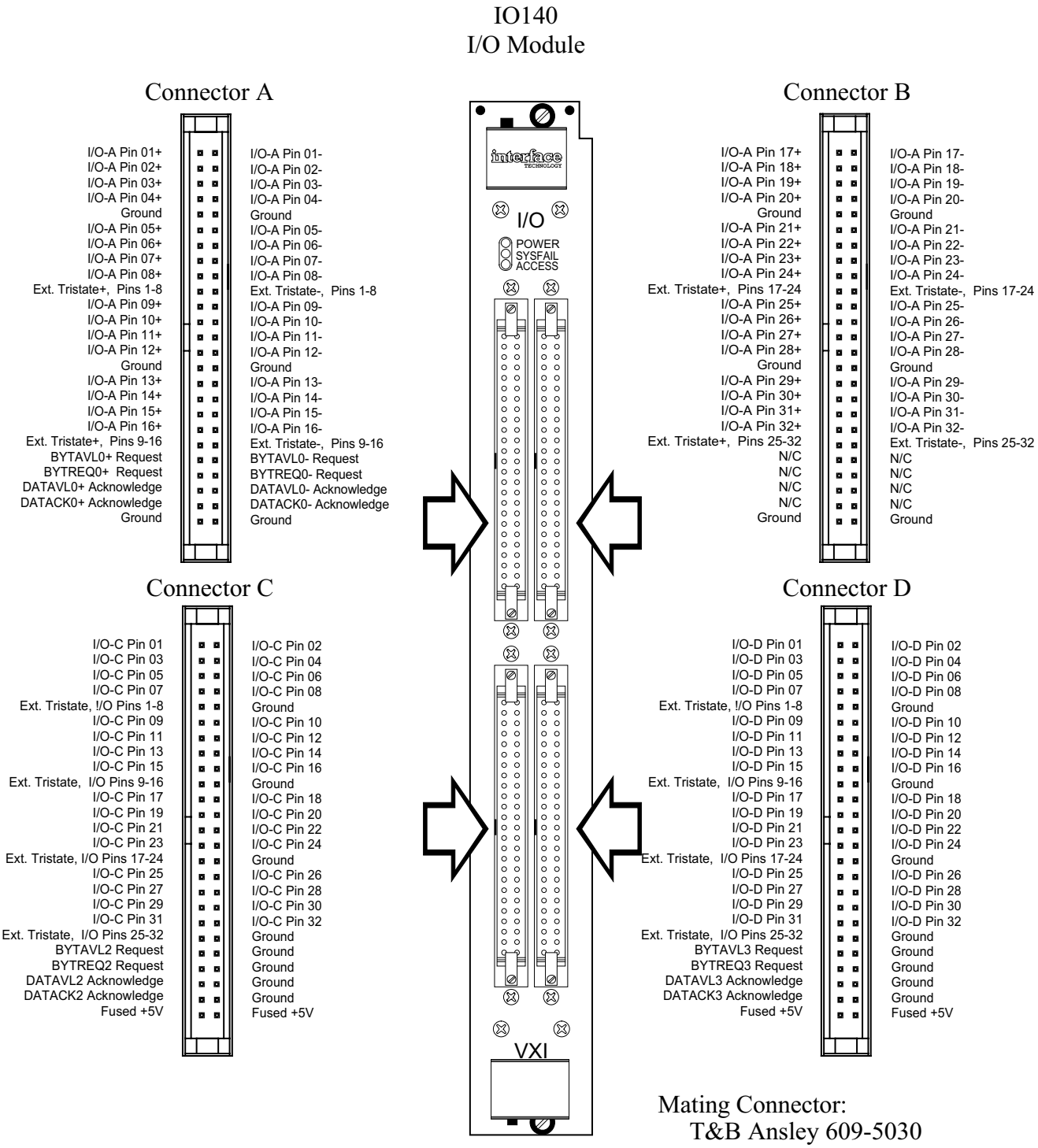


Figure 6-5.
Pinouts for IO140.

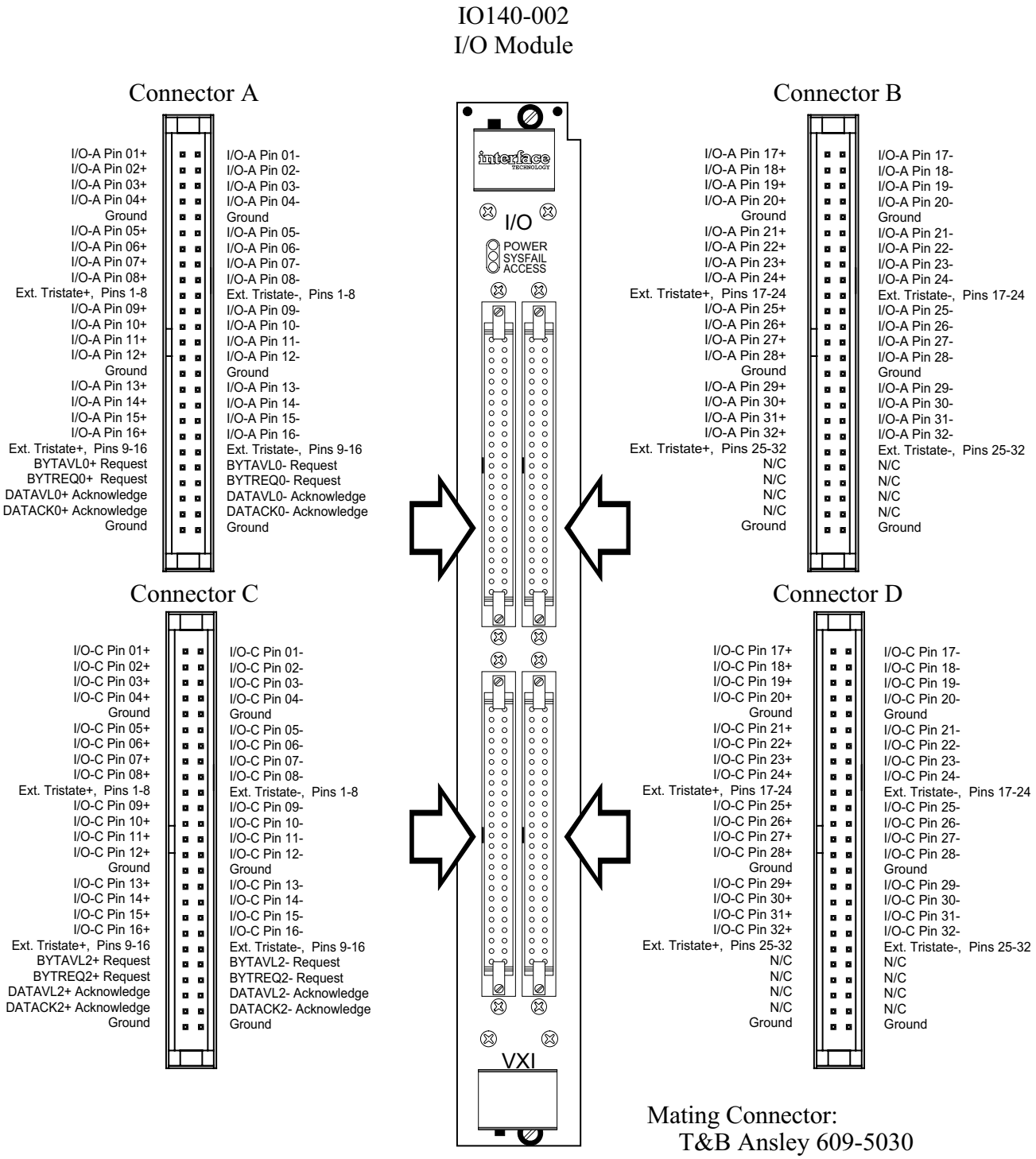


Figure 6-6.
Pinouts for IO140-002.

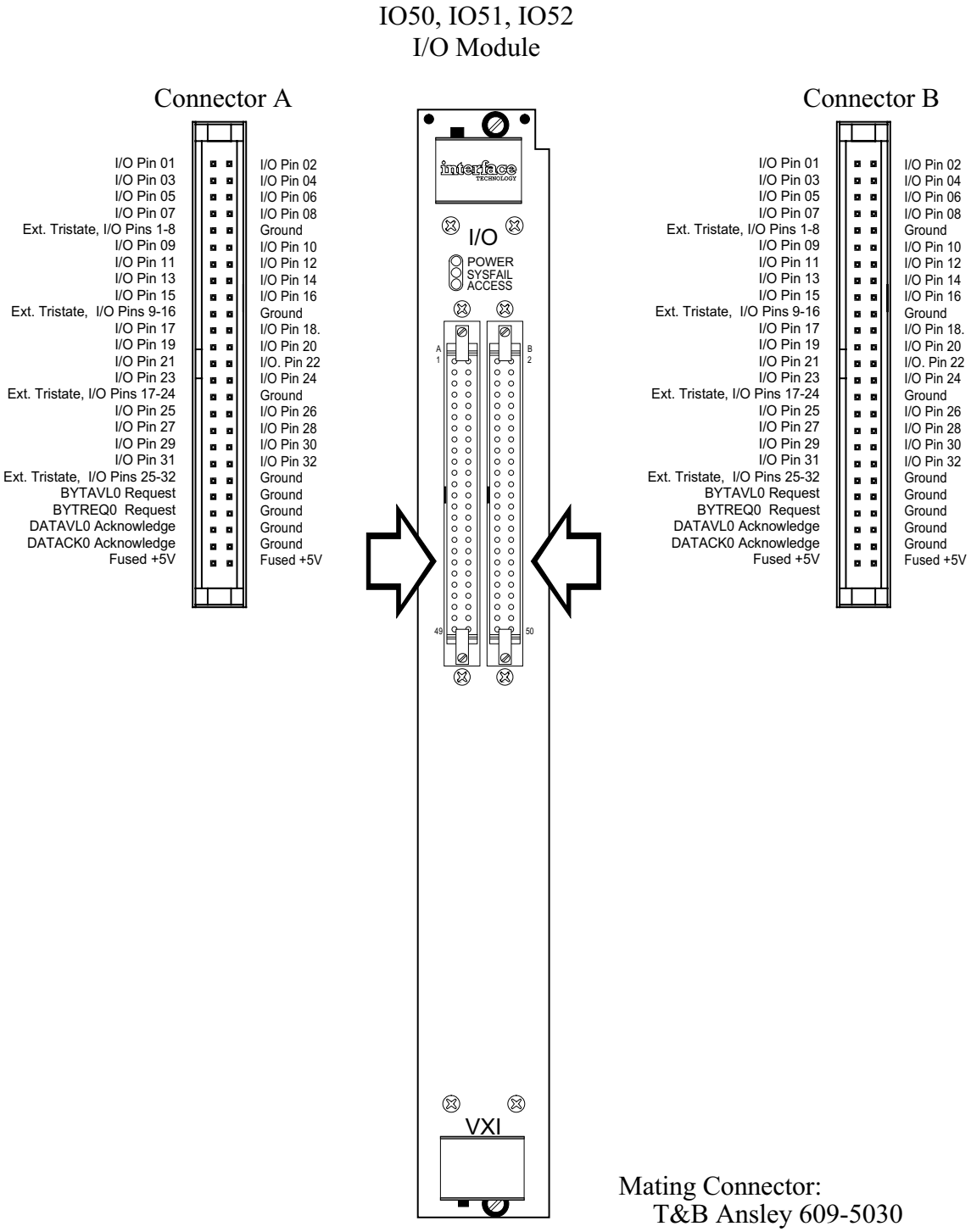
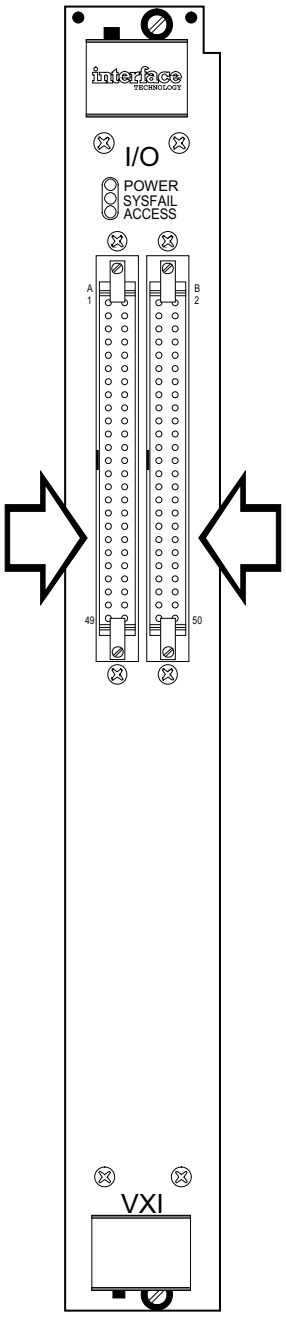


Figure 6-7.
Pinouts for IO50, IO51, IO52.

IO53
I/O Module

Connector A

I/O Pin 01	I/O Pin 02
I/O Pin 03	I/O Pin 04
I/O Pin 05	I/O Pin 06
I/O Pin 07	I/O Pin 08
Ext. Tristate, I/O Pins 1-8	Ground
I/O Pin 09	I/O Pin 10
I/O Pin 11	I/O Pin 12
I/O Pin 13	I/O Pin 14
I/O Pin 15	I/O Pin 16
Ext. Tristate, I/O Pins 9-16	Ground
I/O Pin 17	I/O Pin 18.
I/O Pin 19	I/O Pin 20
I/O Pin 21	I/O. Pin 22
I/O Pin 23	I/O Pin 24
Ext. Tristate, I/O Pins 17-24	Ground
I/O Pin 25	I/O Pin 26
I/O Pin 27	I/O Pin 28
I/O Pin 29	I/O Pin 30
I/O Pin 31	I/O Pin 32
Ext. Tristate, I/O Pins 25-32	Ground
BYTAVL0 Request	Ground
BYTREQ0 Request	Ground
DATAVL0 Acknowledge	Ground
DATACK0 Acknowledge	Ground
Fused +5V	Fused +5V



Connector B

HV Pin 01	HV Pin 02
HV Pin 03	HV Pin 04
HV Pin 05	HV Pin 06
HV Pin 07	HV Pin 08 .
External Reset Input	External Reset Ground
HV Pin 09	HV Pin 10
HV Pin 11	HV Pin 12
HV Pin 13	HV Pin 14
HV Pin 15	HV Pin 16
NC	NC
HV Pin 17	HV Pin 18
HV Pin 19	HV Pin 20
HV Pin 21	HV Pin 22
HV Pin 23	HV Pin 24
NC	NC
HV Pin 25	HV Pin 26
HV Pin 27	HV Pin 28
HV Pin 29	HV Pin 30
NC	NC
NC	NC
Vin	Vret
Vin	Vret
Vin	Vret
Vin	Vret
NC	NC

Mating Connector:
T&B Ansley 609-5030

Figure 6-8.
Pinouts for IO53.

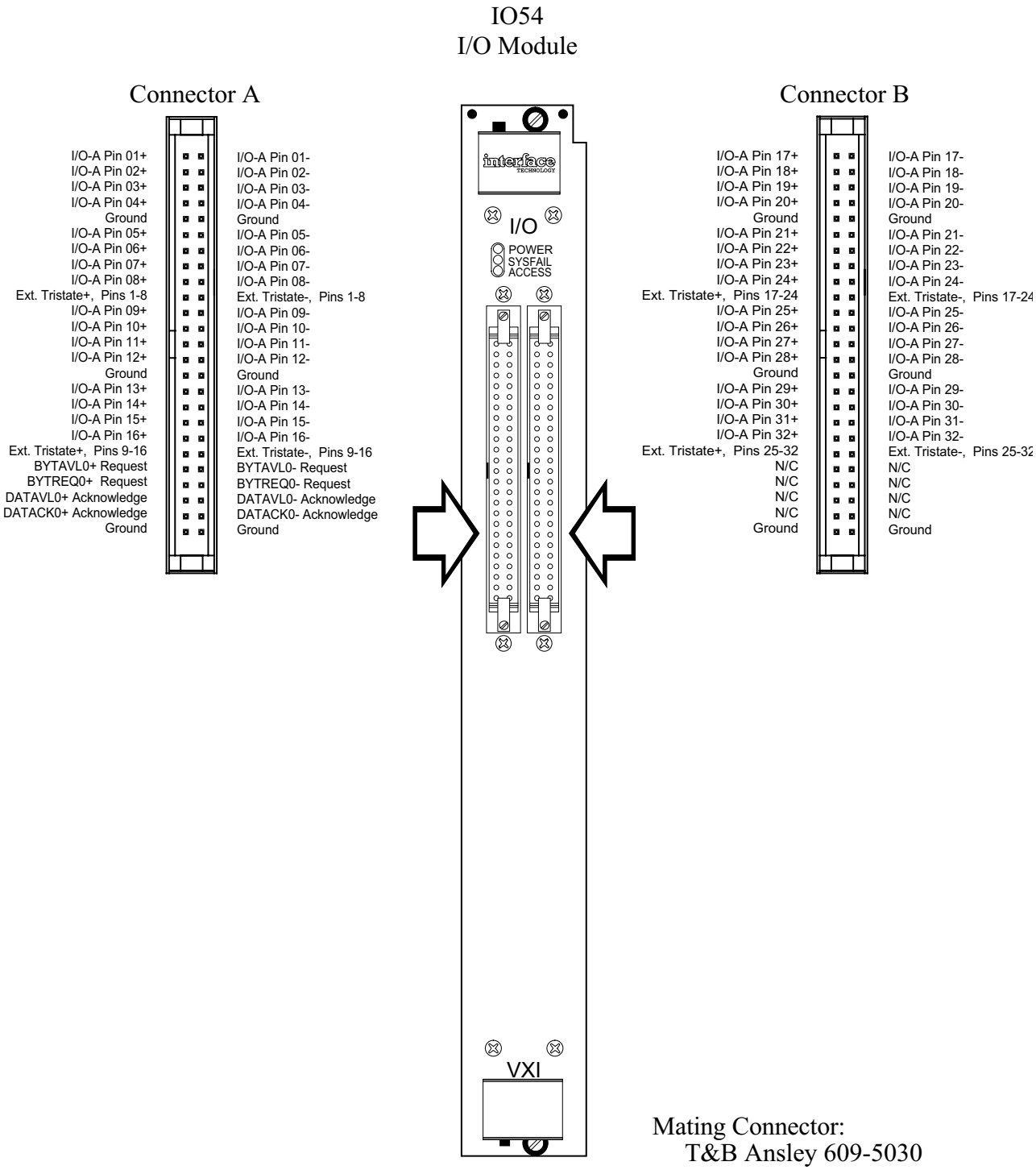


Figure 6-9.
Pinouts for IO54.

A.1 SPECIFICATIONS

Timing:

Timed Block I/O, 0 delay

Output: ..... 20 KHz, Typ.
Input: ..... 32 KHz, Typ.

Handshake I/O, Timed Block I/O Inactive

Byte Request to Data Valid: ..... 83us, Typ.
Byte Available to Data Acknowledge: ..... 43us, Typ.

Memory Emulation, Timed Block I/O Inactive

Address in to Data Valid (read cycle): ..... 230us, Typ.
Address in to Data Latched (write cycle): ..... 50us, Typ.

Direct Register Access Cycle Time: ..... 420ns, Min.

External Tristate Control (IO50/IO100, IO51/IO110)

Enable: ..... 36ns, Max.
Disable: ..... 36ns, Max.

External Output Enable Control (IO52/IO120)

Enable: ..... 36ns, Max
Disable: ..... 40ns, Max.

Skew

Channel-to-channel: ..... 20ns, Max.
Card-to-card (Slave Mode, TTLTRG): ..... 52ns, Max.

Drivers:

IO50 / IO100: 74F244
IO51 / IO110: 74ACT244
IO52 / IO120: 74AS760
IO53 / IO130: LH1505AB
IO54 / IO140: AM26LS31CD

Receivers:

IO50 / IO100: 74HCTC7CD
IO51 / IO110: 74HCTC7CD
IO52 / IO120: 74HCTC7CD
IO53 / IO130: ..... N/A
IO54 / IO140: AM26LS32ACD

Handshake & Control

Input Strobe: ..... 2 per connector
Output Strobe: ..... 2 per connector
Tri-state Control Inputs: .....4 per connector (IO100 & IO110)
Output Enable Inputs: ..... 4 per connector (IO120)

VXI Interface

Size: .....C-Size, Single Slot
Type: .....Message-based, Servant only
Configuration: ..... Static or Dynamic
Interrupt Level: .....Programmable
TTLTRG: ..... Input or output, selectable in groups of two
Dual-access RAM: 256KB
Other: ..... A24/D16 Only, Direct Register Access

Power and Temp.

+5.0V: ..... 3.2A, Typ.
+5.0V Outputs: ..... 0.5A per connector, Max.

Storage Temp.: .....-40°C to +75°C
Specified Operating Temp.: ..... +25°C, ±10°C
Maximum Operating Temp.: ..... 0°C to 50°C

A.2 CONNECTORS & FUSES

Mating Connectors

T&B Ansley: 609-5030

Fuses

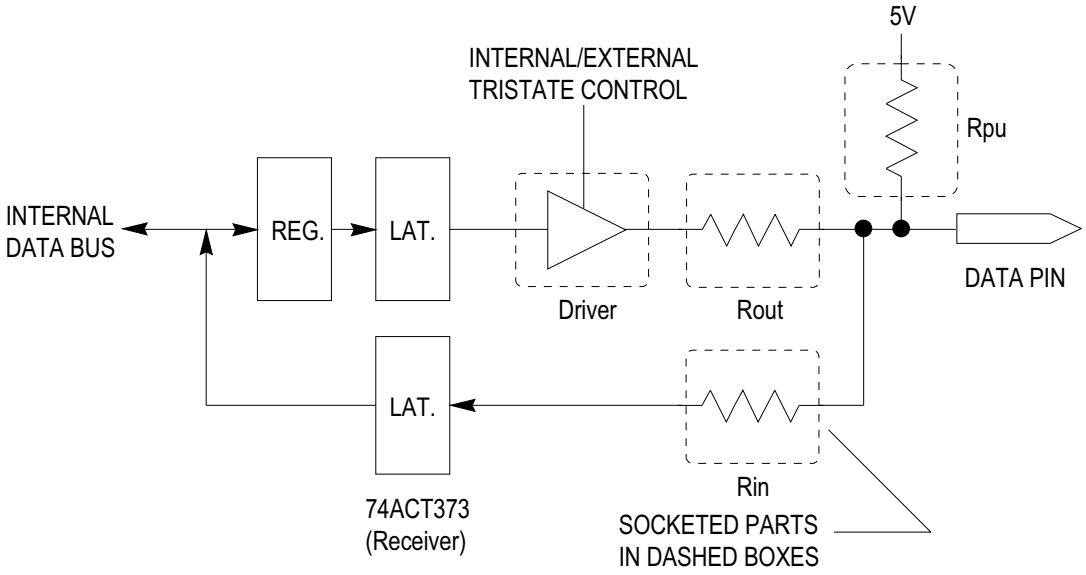
VXI Module: .....Little Fuse, 252-007
+5.0V Output Pins: .....Raychem, RBE110A

B.1 DRIVER/RECEIVER CONFIGURATIONS

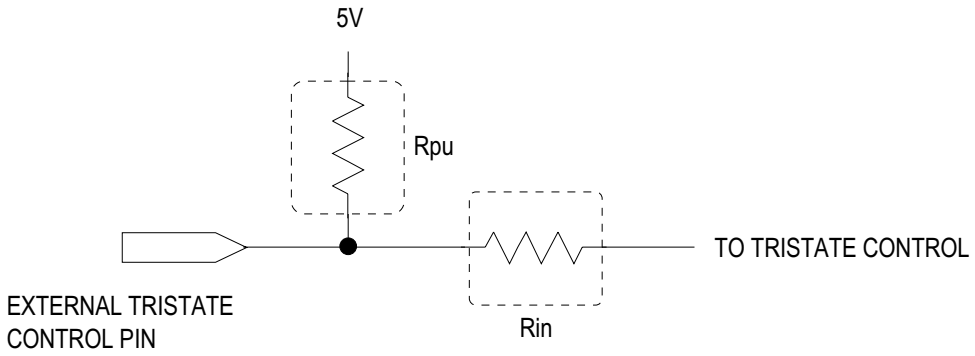
The IO50 / IO100 I/O modules support various input and output termination schemes. The diagrams on the following pages detail standard data pin configurations, tri-state control/output enable pin configurations, request handshake pin configurations and acknowledge handshake pin configurations. In addition to the standard pin configurations, suggested pin configurations for a terminated bus interface and +15V receiving inputs are also shown. The components shown in dashed boxes are socketed, customer replaceable parts. Tables B-1 and B-2 indicate the socketed component locations and part numbers for the IO50 / IO100 modules.

Table B-1.
Data Input/Output Pin.

	IO50 IO100	IO51 IO110	IO52 IO120	IO53 IO130	IO54 IO140
Driver	74F244	74ACT244	74F760	LH1505AB	74ACT244
Rout	898-3-R22 (22 Ohm)	898-3-R22 (22 Ohm)	898-3-R22 (22 Ohm)	N/A	RS-422 Compliant
Rin	898-3-R820 (820 Ohm)	898-3-R820 (820 Ohm)	898-3-R820 (820 Ohm)	N/A	RS-422 Compliant
Rpu	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	RS-422 Compliant

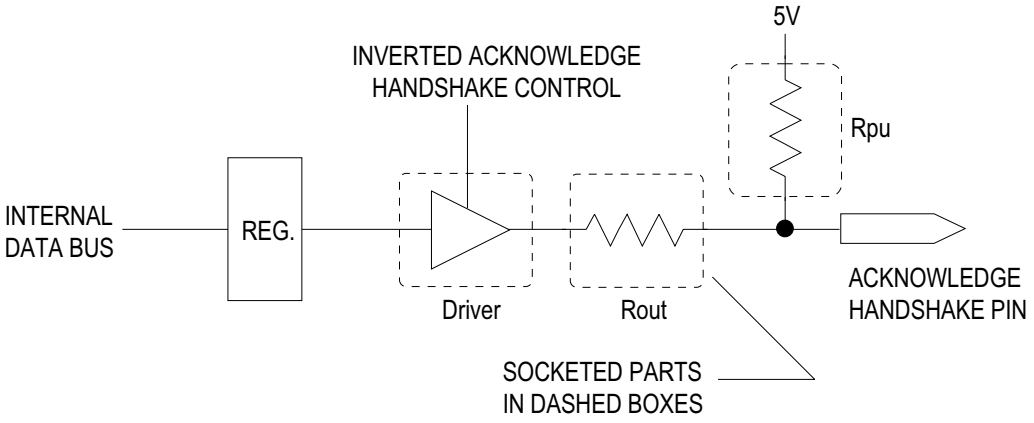


STANDARD DATA PIN CONFIGURATION

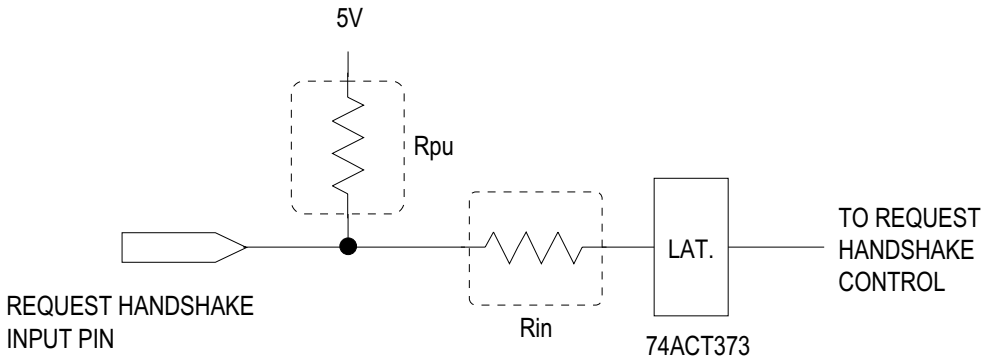


STANDARD EXTERNAL TRISTATE CONTROL PIN CONFIGURATION

Figure B-1.
Standard Data/Tristate Configurations.



STANDARD ACKNOWLEDGE HANDSHAKE PIN CONFIGURATION

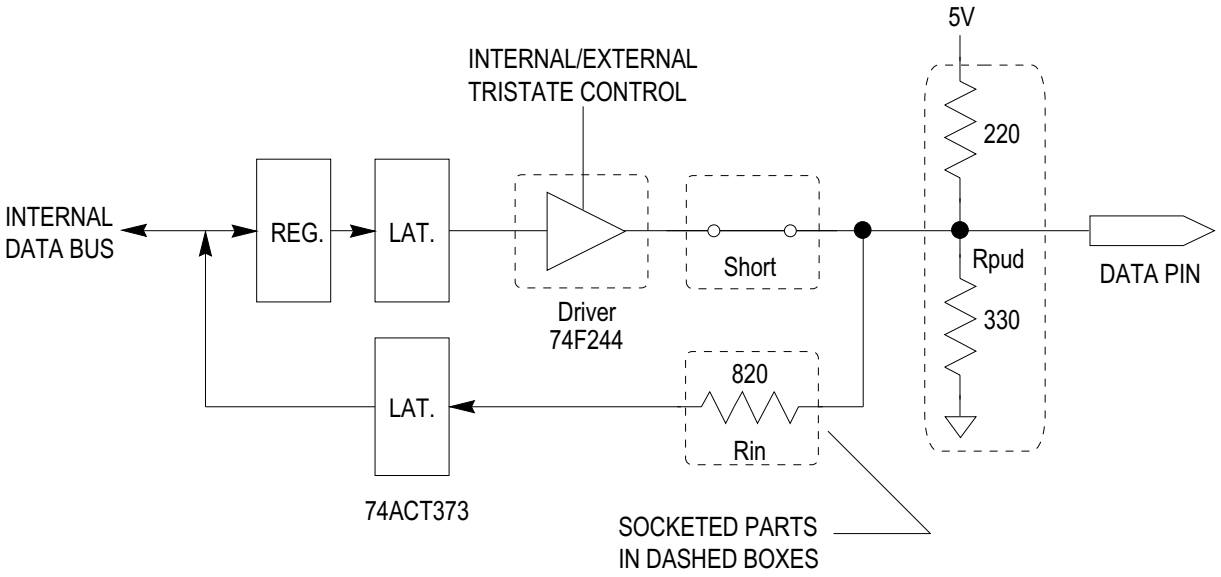


STANDARD REQUEST HANDSHAKE PIN CONFIGURATION

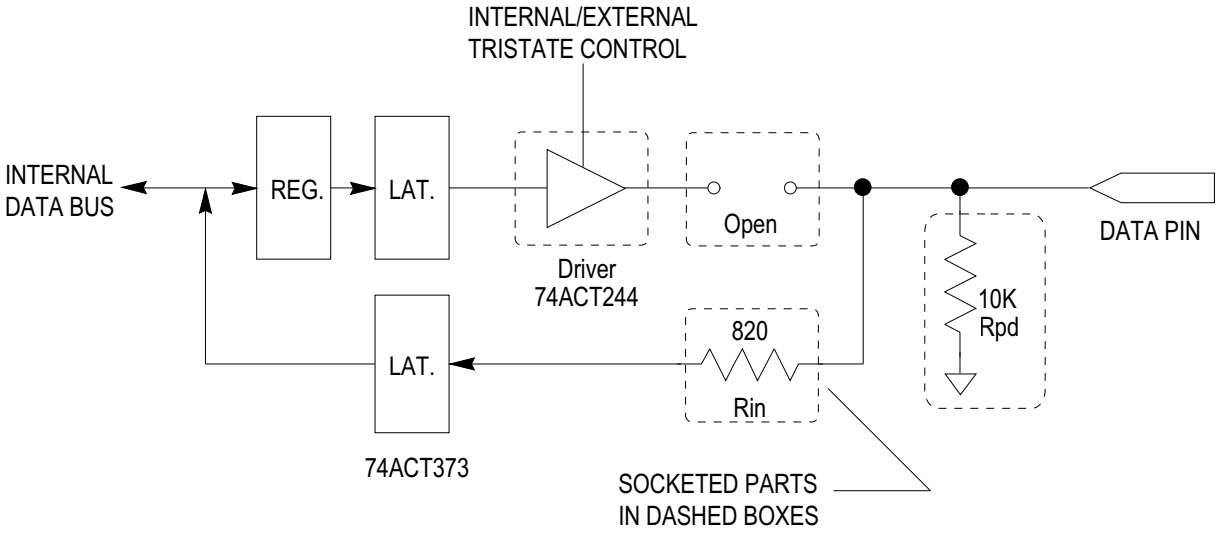
Figure B-2.
Standard Handshake Configurations.

Table B-2.

	IO50 IO100	IO51 IO110	IO52 IO120	IO53 IO130	IO54 IO140
External Tristate Control/Output Enable Pin					
Rpu	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	RS-422 Compliant
Rin	898-3-R22 (22 Ohm)	898-3-R22 (22 Ohm)	898-3-R22 (22 Ohm))	N/A	RS-422 Compliant
Acknowledge Handshake Pin					
Driver	74F244	74ACT244	74F760	LH1505AB	74ACT244
Rout	898-3-R22 (22 Ohm)	898-3-R22 (22 Ohm)	898-3-R22 (22 Ohm)	N/A	RS-422 Compliant
Rpu	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	RS-422 Compliant
Request Handshake Pin					
Rpu	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	4310R-101-103 (10k Ohm)	RS-422 Compliant
Rin	898-3-R820 (820 Ohm)	898-3-R820 (820 Ohm)	898-3-R820 (820 Ohm)	N/A	RS-422 Compliant



SUGGESTED DATA PIN CONFIGURATION FOR TERMINATED BUS INTERFACE



SUGGESTED DATA PIN CONFIGURATION FOR RECEIVING INPUTS TO +15V

Figure B-3.
Suggested Alternate Configurations.

COMMAND ERRORS

0,"No Error"
-101,"Invalid character;Semicolon can’t start command"
-103,"Invalid separator;Semicolon or colon expected"
-101,"Invalid character;Syntax error at second colon"
-101,"Invalid character;Syntax error at semicolon following colon"
-101,"Invalid character;Double semicolons not allowed"
-103,"Invalid separator;Asterisk found instead of separator"
-111,"Header separator error;Alpha after 488.2 common cmd invalid"
-101,"Invalid character;Double colons not allowed"
-101,"Invalid character;Colon found but no commands at a lower level"
-101,"Invalid character;Unknown in this context"
-101,"Invalid character;Double semicolons not allowed"
-103,"Invalid separator;Asterisk found instead of separator"
-103,"Invalid separator;Alpha found instead of separator"
-101,"Invalid character;Asterisk found instead of separator"
-158,"String data not allowed;No match found for parameter string"
-113,"Undefined header;A 488.2 common command was expected"
-114,"Header suffix out of range;Number after 488.2 cmd not allowed"
-113,"Undefined header;No match found for command"
-131,"Invalid suffix;Suffix not appropriate"
-113,"Undefined header;Question mark expected"
-101,"Invalid character;Unexpected character found after header"
-113,"Undefined header;Number attached to header not allowed"
-111,"Header separator error;A space separator was expected"
-131,"Invalid suffix;Suffix not appropriate for command"
-144,"Character data too long;Name is maximum of 8 chars"
-103,"Invalid Separator;Comma not found as expected"
-104,"Data type error;PIN LIST syntax <A|B|C|D><n>[,<A|B|C|D><n>] not found"
-104,"Data type error;Syntax error on number list parameter"
-104,"Data type error;Syntax error on data list parameter"

EXECUTION ERRORS

-224,"Illegal parameter value;Invalid conversion"
-224,"Illegal parameter value;Invalid base value"
-224,"Illegal parameter value;Undefined parameter"
-224,"Illegal parameter value;Invalid data type"
-222,"Data out of range;Value out of current radix bounds"
-222,"Data out of range;Baud rate not supported"

-222,"Data out of range;Data bits must be 7 or 8"
-222,"Data out of range;Stop bits must be 1 to 2"
-222,"Data out of range;Parity type not supported"
-241,"Hardware missing;Address generates bus/addr exception"
-224,"Undefined field name"
-224,"Undefined test name"
-225,"All available fields have been defined"
-225,"All available tests have been defined"
-225,"Not enough Free IO vectors available for allocation"
-222,"Data out of range;Valid connector names are A, B, C, & D"
-222,"Data out of range;Valid pin numbers are 1-32"
-222,"Data out of range;Valid channels are 1-32"
-224,"Invalid field name, only 8 char."
-224,"No default field is defined"
-224,"No working test is defined"
-224,"Invalid vector number"
-224,"Invalid test name, only 8 char."
-224,"Invalid test name"
-221,"Test name is already defined"
-221,"Field name is already defined"
-222,"Data out of range;Valid trace word numbers are 1-8"
-220,"Not enough data value were provided base on the count value"
-222,"Data out of range;Statement’s parameter must be 1 to 15360"
-241,"Invalid operation; The shared memory option is not installed"
-230,"Invaid Learn format; Learn encountered invalid format, Learn aboorted"
-230,"Incorrect I/O Card; Different I/O Card was used on Learn? command"
-221,"Invalid Learn record header"
-221,"Learn command requires additional blocks of data to complete Learn session"
-221,"Setting conflict;Tristate settings conflicts with other field’s Tristate settings"
-221,"Invalid setting;Tristate must be EXTNORMal or EXTINVerted for current test"
-221,"Invalid setting;Tristate must be INput for current test"
-221,"Invalid setting;Tristate must be OUTput for current test"
-221,"Invalid setting;Invalid test type"
-221,"Invalid I/O vector number"
-222,"Data out of range;Valid pin group numbers are 1,9,17 or 25"
-222,"Data out of range;A maximum of 16 items allowed on PIN_GROUP_LIST"
-221,"Settings conflict;BASICmode pin group already defined"
-221,"Settings conflict;BASICmode pin group not defined"
-221,"Test definition;No test MUST be defined when learning from LEARN"
-213,"System state;No test can be executing when using BASICmode"

-213,"System state;Only one timed and one handshake test can be executing at any given time"
-221,"Test type;Only PRGIOTimed and PRGIOHandshake may use INITate:INput or INITiate:OUTput"
-221,"Test type;PRGIOTimed and PRGIOHandshake cannot use INITate:BLOCk"
-221,"Address Field:BUSEMULation must have an Address field define to execute"
-221,"System state;Stop any executing test to use this command"
-222,"Data out of range;Maximum timeout allowed is 1ms. Minimum is 0"
-221,"System state;Stop executing the current test to use this command"
-223,"Number of data values exceeded the provided count value"
-221,"System state:System must be in STANDalone to execute in block mode"
-222,"Invalid vector or count;Vector and Count value of one is expected for program I/O"
-222,"Invalid count;Must be greater that zero"
-222,"Invalid size;Must be a power of two for MEMEMUlation"
-222,"Invalid field;Number of field bits exceeds the size of test"
-222,"Undefined Tristate;Tristate setting MUST be defined for ALL fields"
-221,"Invalid setting;Tristate must NOT be INput for current test"
-221,"Invalid setting;Acknowledge is fixed to inverted for MEMEMUlation"

DEVICE DEPENDANT ERRORS

-350,"Queue overflow;Tail of output string is lost"
-316,"Out of DRAM memory"
-310,"System error;Software bug - error number is out of range"

QUERY ERRORS

-410,"Query INTERRUPTED;Previous query output within string was overwritten"
-410,"Query INTERRUPTED;Previous query output lost"
-420,"Query UNTERMINATED;Output buffer was empty"

(THIS PAGE INTENTIONALLY LEFT BLANK)

APPENDIX D

Programming Instructions for High Voltage Switching Option

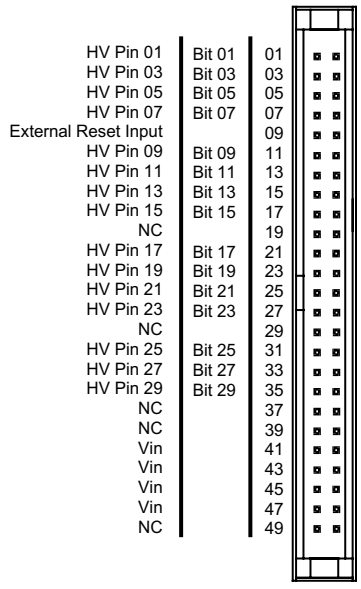
Applicability

Note

The information contained in this appendix applies to the IO53 and IO130 modules only.

The high voltage relay card is programmed with the Connector D data output pins. The upper two pins (pin 127 and 128) are used as enable strobes to write control data to latches on the option board. The remaining 30 data pins from Connector D are ued to provide "on/off" and "high/low" control bits for the 30 high voltage switching pins.

The pinout for the 50-pin Connector D with the high voltage option are as shown in Fig D-1



Connector B (IO53 and IO130)
Connector D (IO130-002 only)

Figure D-1.
Pinout for Connectors B and D with HV option.

Register-Level Bit Programming.

The register level bit programming definitions are shown below. For these bit patterns to perform the action described, other conditions must be true. First, the data pins for connector D must be defined as outputs so their tristate drivers are turned on. If register-based programming is used, the second stage latch for connector D data should be forced open. Each time the data patterns below are output, the Data Output Valid stribe for connector D must be written high and then low again to clock the data into the pin control latches. All of these operations are automatically performed if the message-based SCPI command set is used, as shown in the example below.

The register level bit programming definitions are:

Bit 128	Bit 127	Bit 126, 125 . . . . . 97, 96
0	0	Don't care - outputs retain present state.
0	1	1 = connect pin to Vin input level. 0 = connect pin to Vrtn input level. - Bit 126 controls HV Pin 30. - Bit 125 controls HV Pin 29. : : - Bit 97 controls HV Pin 2. - Bit 96 controls HV Pin 1.
1	0	1 = enable output pin for voltage output. 0 = disable pin output (tristate). - Bit 126 controls HV Pin 30. - Bit 125 controls HV Pin 29. : : - Bit 97 controls HV Pin 2. - Bit 96 controls HV Pin 1.
1	1	Enable control and data together (not recommended).

Note

See Figure D1 for diagram showing relationship between above bit numbers and Connector D pin numbers.

SCPI PROGRAMMING.

Note that when using the field definition capability of the IO130 VXI software that data pin outputs are only changed when the field name for those pins is assigned a new vector. If you set a field bit with a command, then send several other commands that do not change that field, the output is not changed.

Note that special field names associated with UUT control functions can be defined for bits 126-96, if desired.

Initialize connector D to timed output mode for high voltage option control.

```
TEST:DEF D:PRGIOTIMED
SYST:TEST D
TEST:HANDSHAKE:ACKNOWLEDGE:OUTPUT NORMAL
```

Name the two control pins CTLEN and make them outputs. 1(OXO1) is HIGH/LOW WRITE and 2(0X10) is tristate control.

```
FIELD:DEFINE CTLEN:PIN D32-31
FIELD:NAME CTLEN:TRISTATE OUT
```

Name the remaining 30 pins PINCTL and make them outputs.

```
FIELD:DEFINE PINCTL:PIN D30-1
FIELD:NAME PINCTL:TRISTATE OUT
```

The following assumes that -48V connects to Vin and Gnd connects to Vrtn. First set pin level before enabling tristate. Set odd pins to -48 and even pins to gnd.

```
VECT1:DATA:FIELD CTLEN;VAL 1;FIELD PINCTL;VAL 15555555
INIT:OUT
```

Turn on pins 1-16.

```
VECT 1:DATA:FIELD CTLEN;VAL 2;FIELD PINCTL;VAL FFFF
INIT:OUT
```

This may be unnecessary, but it incures that no spurious clocking will occur if inverted polarity tests are un on connectors A-C.

```
VECT 1:DATA:FIELD CTLEN;VAL 0
INIT:OUT
```

Here the active test could be changed to perform other operations using digital I/O on pins 1-95. Any operations, including basic mode I/O, can be performed as long as bits 128 and 127 remain 0, the high voltage outputs remain static.

Turn off pins 16-1, turn on pins 17-30.

```
VECT 1:DATA:FIELD CTLEN;VAL 1;FIELD PINCTL;VAL 3FFF0000
INIT:OUT
```

Now turn off all pins to change data.

```
VECT 1:DAT:FIELD CTLEN;VAL 2;FIELD PINCTL;VAL 0
INIT:OUT
```

Now set all pins to -48V.

```
VECT 1:DATA:FIELD CTLEN;VAL 1;FIELD PINCTL;VAL 3FFFFFFF
INIT:OUT
```

Now enable all pin outputs.

```
VECT 1:DATA:FIELD CTLEN;VAL 2;FIELD PINCTL;VAL 3FFFFFFF
INIT:OUT
```

Now switch pin outputs while leaving tristates enabled. Toggle all bits to make a square wave output.

```
VECT 1:DATA:FIELD CTLEN;VAL 1;FIELD PINCTL;VAL 15555555
INIT:OUT
```

Note that CTLEN retains last value, so there is no need to re-send.

```
VECT 1:DATA:FIELD PINCTL;VAL 2AAAAAAA
INIT:OUT
VECT 1:DATA:FIELD PINCTL;VAL 15555555
INIT:OUT
VECT 1:DATA:FIELD PINCTL;VAL 2AAAAAAA
INIT:OUT
```

Set HV pins 16 and 2 to -48V, set all others to ground.

```
VECT 1:DATA:FIELD PINCTL;VAL 8002
INIT:OUT
```

REGISTER -BASED PROGRAMMING INSTRUCTIONS.

The following instructions refer to Figure 5-1 in the IO100VXI manual, and the Unique Bit Functions figure for the High Voltage switch card. The control registers are byte-wide, write only devices. Since bits cannot be individually manipulated in the hardware, a common practice is to keep register "images" in RAM. The masking and "ORing" operations required for changing single bits are then performed in RAM, and then the control word is written to the hardware.

Register addresses will be determined by the VXI Resource Manager at power up configuration. The VXI registers shown in the example will be offset by some value for 0XC000 to 0XFFC0. The offset must be determined using functions provided with the Resource Manager software. In the example, "ba" is used to represent the base address offset.

The following example shows the programming steps for a High Voltage switch card on connector D. No other connectors are assumed active, so their control bits are all zero-filled.

The IO130VXI output function uses a two stage latch to provide for simultaneous update across 128 pins. The first stage is a set of byte addressable registers, the final stage is a transparent latch. For the example shown, the second stage latches are left open. All 30 pins will be switched or enabled at the same time anyway, by virtue of the clock strobe signal used to program pin control.

A strobe signal is used to clock the control bits from the connector B data field into the pin control registers on the high voltage switch card. This clocking is performed by writing the Data Valid strobe for connector D high, and then low, after each change in the pin control data field. This separate strobe insures that setup and hold times are met for the enable and control bits. Note that the pin control registers on the HV switch card are reset by a hardware reset, a VXI Control Register soft reset, or the user provided External Reset input from the front panel connector.

The HV switch card relies on the IO130VXI connector D digital pins for programming information. Since the normal function of these pins (without the HV switch option installed) can be programmed for input or output, they must first be programmed for output only.

Example: The High Voltage pins 25-1 are switched on and off in a "walking one" pattern in this example. It is assumed that Vin is connected to +15V and Vrtn is connected to the +15V source ground (0V). The example is not written in any real programming language, but is meant only to be illustrative of bit use.

Program Example .. Enter From Initialization or Reset.

```
write 0XF000     address ba+0X3A   /* Conn D outputs on */
```

```
write 0X0        address ba+0X3C   /* Conn D internal control */
write 0X800      address ba+0X3C   /* Open Conn D output latch */
write 0X4000     address ba+0X2C   /* Switch strobe on and pins low */
write 0X0        address ba+0X2E   /* Set all pins low before enable */
call cntrl_clk                     /* Call function to clock registers */
write 0X81FF     address ba+0X2C   /* Enable strobe on and pins 25-17 */
write 0XFFFF     address ba+0X2E   /* Enable pins 16-1 */
call cntrl_clk                     /* Call function to clock registers */
                                   /* Output pins now on - all at 0 volts */
                                   /* Now start walking a one from pin 1 */
                                   /* to pin 25 */
write 0X4000     address ba+0X2C   /* turn on switch control */
set variable i = 0X1               /* Pin 1 gets 15V first, then 2-25 */
for i = 1 to 0X8000
     write i to address ba+0X2E    /* Need write only lower 16 pins */
     call cntrl_clk                /* Call function to clock registers */
     i = i*2
next i
write 0X0        address ba+0X2E   /* Zero pin 16 */
                                   /* Finished lower 16 - do upper with */
                                   /* variable j */
for i = 1 to 0X10
     j = i .or. 0x4000             /* Leave switch enable on */
     write j to address ba+0X2C    /* Need write only upper 14 pins */
     call cntrl_clk                /* Call function to clock registers */
     i = i*2
next i
                                   /* Now leave pin 25 on while going to */
                                   /* do some digital I/O */
j = j .and. 0X3FFF                 /* Zero enable bits */
write j to address ba+0X2C         /* Always leave all enables low */
                                   /* when going off to other programming */
                                   /* operations */
done
function cntrl_clk
write 0X800      address ba+0X34   /* Set Conn d Data Valid high */
write 0X0        address ba+0X34   /* Return Data Valid low */
return
```

AppNotes & TechNotes

AppNotes & TechNotes

Note:
This section contains Application Notes and Technical Notes describing the technical details and applications of the subject equimment.

(THIS PAGE LEFT BLANK INTENTIONALLY)

App/Tech Note

Using IO50 and IO130 Backplane Voltages

IO50/100-01

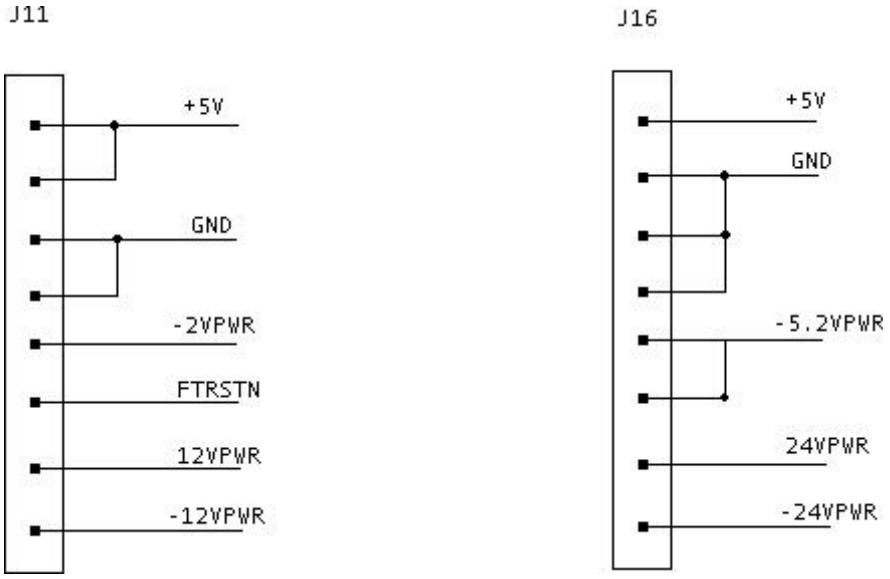
Purpose

This tech note provides instructions for accessing VXI backplane voltages to be used by Interface Technology's IO53/130 High Voltage Switching Option. The following information assumes that the user is familiar with message-based instruments and SCPI programming of those instruments.

The IO130 and IO53 use 32 output channels to drive 30 optically isolated solid state relays. This allows the modules to control high voltage applications up to 100 volts. Switched voltages can be either user supplied, or selected from +5, ± 12 and ± 24 volts available from the VXI backplane. Both modules use a daughter board to provide the switched high voltage outputs.

On the IO130 and IO53 daughter cards there are (2) jumper connectors J4 (pins 1-5) and J3 (pins 1-5) (see figure 1). Also, on the IO130 and IO53 digital IO mother board are (2) jumpers labeled J11 and J16. These are provided to jumper voltages from the backplane instead of supplying voltages to Vin/Vrtn pins on the module's front panel.

The pinouts for the jumper locations on the motherboard are listed below.



On the High Voltage daughter cards, VXI voltage input and pass through pins are available at jumper locations J3 and J4. Pins 1 and 2 (for *both* jumper locations J3 and J4) are connected to Vin. Pins 3 and 4 (for *both* jumper locations J3 and J4) are connected to Vrtn. Thus, by connecting the appropriate jumpers from the daughter card to the motherboard, the user can supply VXI backplane voltages from +5, ± 12 and ± 24 volts.

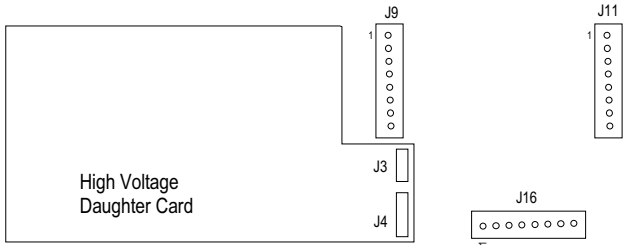


Figure 1.

Note:
When connecting voltages from the VXI backplane, there should not be any power sources connected to Vin or Vrtn at the front panel.

Example:

The High Voltage pins for the IO130 are located on connector D. In order to utilize the option you must program connector D to the desired specifications. The IO130 is a message based instrument capable of being programmed via SCPI commands. This example will use SCPI programming to set up the parameters and run the test.

First, the user will open the top cover of the IO130 to view the daughter card and mother board as shown in Figure 1. Next, wire the IO130 as directed below (refer to Figure 2).

Connect J4 pin 2 *to* J11 pin 7. This will tie Vin to +12V on the VXI backplane.

Connect J4 pin 4 *to* J11 pin 3. This will tie Vrtn to GND.

Define a test, test type, and test size (number of vectors to be used to store data). Initialize connector D to block out timed mode for high voltage output control.

```
TEST:DEF D:BLKOUTTIMED:SIZE 20
SYST:TEST D
TEST:HANDSHAKE:ACKNOWLEDGE:OUTPUT NORMAL
```

****Set timeout parameter to 1 ms for each data vector output.****

```
TEST:TIME:OUT 0.001000
```

Name the two control pins CTLEN and make them outputs. 1 (#h01) is High/Low Write and 2 (#h02) is tristate control.

```
FIELD:DEF CTLEN:PIN D32,D31
FIELD:NAME CTLEN:TRISTATE OUTPUT
```

Name the remaining 30 pins PINCTL and make them outputs.

```
FIELD:DEF PINCTL:PIN D30-1
FIELD:NAME PINCTL:TRISTATE OUTPUT
```

At this point the IO130 should be connected as shown in figure 2. Where Vin connects to +12 V and Vrtn connects to GND. Now, determine the pin levels before enabling tristate. Set the odd pins to +12V and even to GND.

```
SYST:FIEL CTLEN
VEC 1:COUNT 10;DATA 1,1,1,1,1,1,1,1,1,1
VEC 11:COUNT 10;DATA 1,1,1,1,1,1,1,1,1,1
SYST:FIEL PINCTL
VEC 1:COUNT 4;DATA 15555555,15555555,15555555,15555555
VEC 5:COUNT 4;DATA 15555555,15555555,15555555,15555555
VEC 9:COUNT 4;DATA 15555555,15555555,15555555,15555555
VEC 13:COUNT 4;DATA 15555555,15555555,15555555,15555555
VEC 17:COUNT 4;DATA 15555555,15555555,15555555,15555555

INIT:BLOCK
```

Enable pins for output (turn on).*

```
SYST:FIEL CTLEN
VEC 1:COUNT 10;DATA 2,2,2,2,2,2,2,2,2,2
VEC 11:COUNT 10;DATA 2,2,2,2,2,2,2,2,2,2
SYST:FIEL PINCTL
VEC 1:COUNT 4;DATA 3FFFFFFF,3FFFFFFF,3FFFFFFF,3FFFFFFF
VEC 5:COUNT 4;DATA 3FFFFFFF,3FFFFFFF,3FFFFFFF,3FFFFFFF
VEC 9:COUNT 4;DATA 3FFFFFFF,3FFFFFFF,3FFFFFFF,3FFFFFFF
VEC 13:COUNT 4;DATA 3FFFFFFF,3FFFFFFF,3FFFFFFF,3FFFFFFF
VEC 17:COUNT 4;DATA 3FFFFFFF,3FFFFFFF,3FFFFFFF,3FFFFFFF

INIT:BLOCK
```

Odd pins should measure +12V and even pins should measure at GND level.

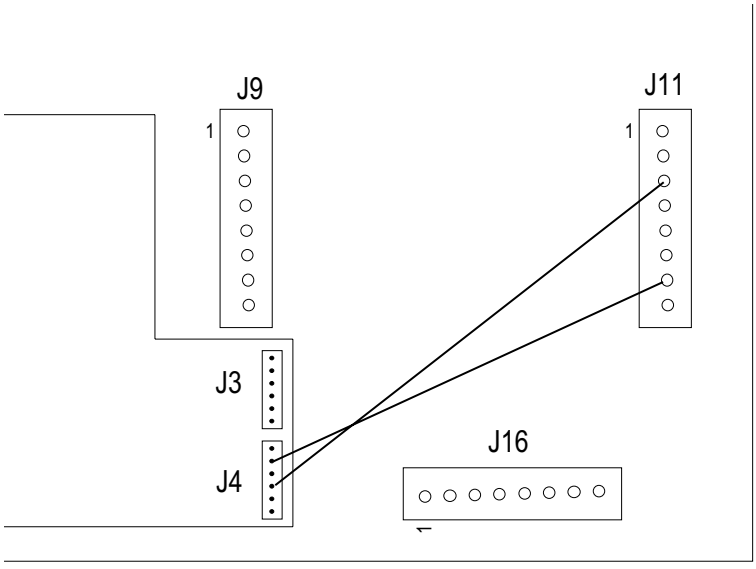


Figure 2.

Conclusion

Considering the list of availble power supplies on the VXI bus, a multitude of applications can be accommodated by tapping into the proper resources. Most logic families, TTL and CMOS use +5 Vdc as their main power source e.g., microprocessors and memory chips. RS-232C transmitters and receivers, op-amps, D/A and A/D converters are applications that may require ±12 Vdc. Higher voltage applications can use the ±24 V levels from the VXI backplane. The flexibility of the IO130 and IO53 high voltage modules give the user choices of implementation to supply user defined voltage or use resouces already present on the VXI backplane.

(THIS PAGE INTENTIONALLY LEFT BLANK)

Interface Technology, Inc.
300 S. Lemon Creek Dr., Walnut, CA 91789
Tel: 909/595-6030 - Fax: 909/595-7177
e-mail: info@interfacetech.com - Internet: www.interfacetech.com