



**ICS
ELECTRONICS**

a division of Systems West Inc.

MODEL 8003/8013

Ethernet↔Parallel Interface

Instruction Manual

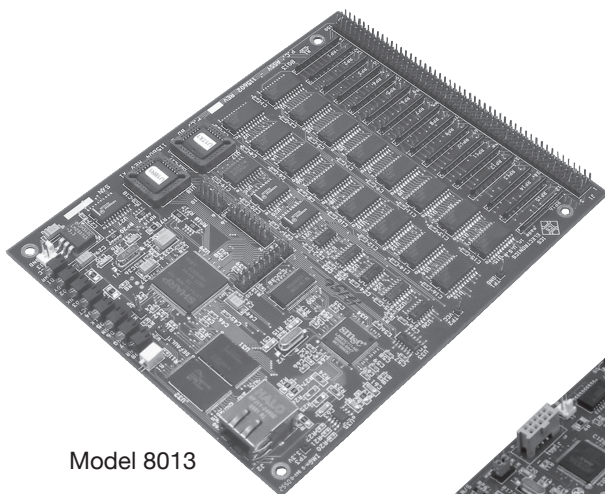
A large, light grey arrow pointing upwards, with a thick black outline. The arrow is positioned on the right side of the page. Inside the arrow, the text 'Parallel I/O' is written vertically in a large, bold, black, sans-serif font.

Parallel I/O

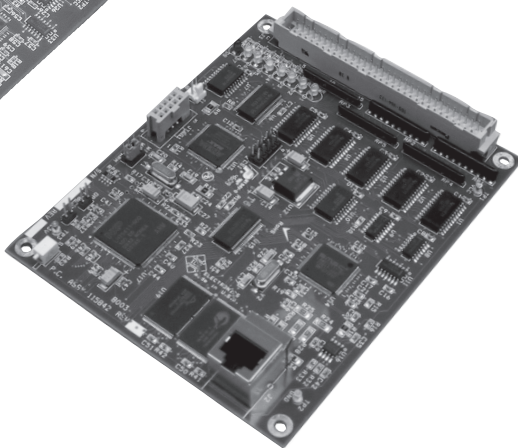
MODEL 8003/8013

Ethernet↔Parallel Interface

Instruction Manual



Model 8013



Model 8003



**ICS
ELECTRONICS**

division of Systems West Inc.

7034 Commerce Circle, Pleasanton, CA 94588
Phone 925.416.1000, Fax 925.416.0105
Web Site <http://www.icselect.com>

Publication Number 120187
June 2018 Edition Rev 7

LIMITED WARRANTY

Within 12 months of delivery, ICS Electronics will repair or replace this product, at our option, if any part is found to be defective in materials or workmanship (labor is included). Return this product to ICS Electronics, or other designated repair station, freight prepaid, for prompt repair or replacement. Contact ICS for a return material authorization (RMA) number prior to returning the product for repair.

CERTIFICATION

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

EMI/RFI WARNING

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause interference to radio communications. The Model 8013 and 8003 have not been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of the FCC Rules and to comply with the EEC Standards EN 55022 and EN 50082-1, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

TRADEMARKS

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

VEE is a trademark of Agilent, Palo Alto, CA

LabVIEW is a Trademark of National Instruments, Austin, TX

ICS and GPIB AnyWhere are trademarks of ICS Electronics, Pleasanton, CA

Contents

General Information

Product Description, Model Numbers, VXI-11 Conformance, Ethernet Interface, Digital Interface, Configurable Functions and Default Settings, Indicators, Physical Specifications, Certifications and Accessories.

Installation

Shipment Verification, Installation Guide, Configuration Instructions, Resetting Defaults, Jumper Settings, Digital I/O Connections, Signal-Pin Tables, Digital I/O Design Guide and Digital I/O Example.

Operation

Digital Interface Operation, Status Reporting Structure, IEEE-488.2 and SCPI Conformance, SCPI Commands, Programming Guidelines, VXI-11 Keyboard, Error Logger Utility and OEM Documentation.

Theory of Operation

Block Diagram Description

Maintenance, Troubleshooting and Repair

Maintenance, Troubleshooting Guide, Selftest Error Codes, Updating Firmware, Reverting to Factory Firmware and Repair Information

Appendices

- A1 IEEE-488.1, IEEE-488.2 and SCPI Descriptions
- A2 VXI-11 Concept
- A3 VXI-11 RPCgen Information
- A4 ICS RPC Configuration Commands

Index

1

2

3

4

5

A

I

This page intentionally left blank

General Information

1.1 INTRODUCTION

This section provides a description and specifications for ICS's Model 8003 and 8013 Ethernet to Parallel Interface Boards. All specifications and functional descriptions apply to all units unless otherwise stated.

1.2 DESCRIPTION

The Model 8003 and 8013 Ethernet Parallel Interface Boards are VXI-11.3 compliant and Raw Socket capable interfaces that provide digital signals to drive external devices and to input digital data from external devices. They perform these operation in response to commands and data received through the Ethernet interface.

The Model 8003 is a 4.5 x 5.5 inch PC board with 40 programmable I/O lines. The Model 8013 is a 5 x 7 inch PC board with 128 programmable I/O lines. The I/O lines are heavy-duty TTL signals for controlling devices or for inputting or outputting digital data. Both digital interfaces are user configurable into gated inputs and/or latched outputs in increments of eight bits. As inputs, each data line has a pullup resistor for sensing contact closures or for interfacing with CMOS signals. As outputs, each line can source 24 mA and sink up to 48 mA. The interfaces can also monitor 15 of the digital lines and report any changes to the system controller. Applications include interfacing devices with parallel digital interfaces to an Ethernet network, controlling discrete devices, reading the state of external switches or signals or monitoring digital events.

Handshake lines are provided for transferring data to external devices and for inputting data. Other control outputs include a Remote line for enabling external controls, a Data Stable line for enabling external devices, Trigger, Clear and Reset pulse outputs.

Both boards are VXI-11.3 compliant instruments which makes them easily controllable from virtually any computer with network access. One programming method is to make program calls to a VISA or SICL library which can communicate with VXI-11.3 instruments. LabView and VEE are graphical applications that can make VISA calls. SICL or VISA calls are recommended for Visual Basic, C and other program languages that can call any library. Another programming technique is to use the RPC protocol to communicate with the boards. The RPC protocol makes it easy to control the 8003 or 8013 from any Linux/UNIX like environment.

The boards contain two instrument personalities. *inst0* is an IEEE-488.2 compatible instrument and lets the user access the internal parser and transfer data or control the digital IO lines with commands. *inst1* allows transparent data transfer. *inst0*'s operation and digital IO configuration settings are completely programmable by SCPI commands from its Ethernet interface. This programming capability makes the unit easy to use in a system application since its functions can be changed at any time or stored in its internal nonvolatile memory. The Ethernet IP settings can be accessed by a web browser or by a Visual Basic utility program. A 'LAN Reset' button allows the user to return the board to its default IP settings at any time.

At power turn-on, the boot up and internal selftest process typically takes approximately 4 seconds. At the end of the selftest, the RDY LED turns on if the test was successful. The LAN and ACT LEDs show the status of the network connection. The TALK, LSTN and SRQ LEDs show the current address status and if it has asserted a Service Request. The ERR LED is momentarily illuminated when the board senses a soft error condition or has a problem with a command that it received.

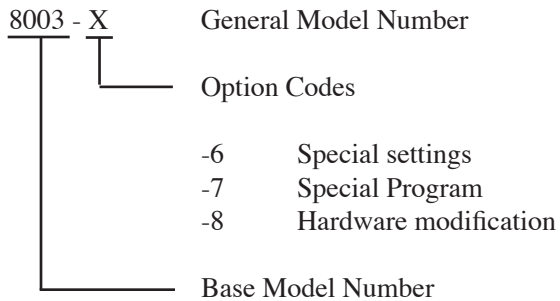
The 8003 is physically interchangeable with ICS's 4803 GPIB-to-Digital, 2303 Serial-to-Digital and 2003 USB-to-Digital Interface boards and has the same digital interface and mounting dimensions. The 8003 is also compatible with the xx03DVR Relay Driver board and other 4803 accessory items.

The 8013 is interchangeable with ICS's 4813 GPIB-to-Digital and 2313 Serial-to-Digital Interfaces and has their same digital interface and board outline dimensions. The 8013 is also compatible with the xx13DVR Relay Driver boards and other 4813 accessory items.

Both boards support LXI's VXI-11 Discovery Method and will coexist in systems with LXI instruments. Both boards include a WebServer that serves user-configurable HTML pages for displaying and configuring the board's network settings and User Description.

1.3 MODEL SPECIFICATIONS

The following specifications apply to all 8003 and 8013 models. Options for your unit may be found by comparing the list below to those listed on the program label on your unit.



8003 for 40 line Parallel Interface with a vertical RJ-45
8003H for 40 line Parallel Interface with a horizontal RJ-45
8013 for 128 line Parallel Interface with a horizontal RJ45
8013V for 128 line Parallel Interface with a vertical RJ45

1.4. VXI-11 CONFORMANCE

The 8003 and 8013 are fully compliant with the VXI-11 and VXI-11.3 Specifications.

1.4.1 RPC Protocol

The RPC protocol conforms to ONC RPC Version 2.

1.4.2 Sockets

The board's VXI-11 service supports 15 TCP/IP sockets for client communication. The sockets are normally opened and closed by the clients. The unit will close the socket and release all resources if a broken connection is detected or when the link count goes to zero if Auto Disconnect is enabled.

There is a separate socket for UDP RPC Port Mapper communication.

1.4.3 Channels

Supports Core, Abort and Interrupt channels. Core and Abort channels each use a socket connection. Core channels support up to 64 device links and locks. A reverse Interrupt channel is a TCP/IP socket connection that does not count against the 15 client communication sockets limit.

1.4.4 Device Links and Locks

The boards support a maximum of 64 device links and 64 locks that can be used over multiple Core channels by one or more clients.

1.4.5 VXI-11 Interface Name

Any 8 character string whose default value is 'inst'. The Interface Name should not be changed unless the user understands its affect on his programs.

1.4.6 VXI-11.3 Supported Functions

The boards support all VXI-11.3 functions including:

create_link	destroy_link	create_intr_channel	destroy_intr_channel
device_lock	device_unlock	device_abort	
device_read	device_write	device_clear	device_trigger
device_remote	device_local	device_readstb	create_intr_channel
device_intr_SRQ	device_enable_SRQ		

1.5 ETHERNET INTERFACE

1.5.1 Type

IEEE-802.3 Compliant

1.5.2 Speed

Auto speed sensing, 10 Mbs with 10BaseT and 100 Mbs with 100BaseT

1.5.3 Network Address

Static: IP Address, Subnet Mask, and Gateway IPv4 values are user set from 0.0.0.0 to 255.255.255.255. Default values are listed in Table 1-3.

DHCP: Accepts an IPv4 address from a DHCP Server.

1.5.4 KeepAlive Message

User enabled. Message sent if no activity for 120 minutes.

1.5.5 COMM Timeout

User set period, 0 to 2³² seconds, to release VXI-11 socket resources if no activity.

1.5.6 Port Usage

TABLE 1-1 8003 PORT USAGE

Port	Usage	Protocols	Notes
23	Raw Socket	Telnet	Web Browser access
80	Internal WebServer	TCP	
111	RPC Port Mapper	UDP, TCP	
5555	Core Channel	TCP	Assigned when opened Defined by client
2000-2999	Abort Channel	TCP	
xxxx	Reverse Notification	TCP	
5556	Configuration Port, Error Logger	TCP	

1.5.7 Protocols

TCP/IP for VXI-11, HTTP and RPC communication.

UPD and TCP/IP for RPC Port Mapper commands.

Raw Socket for ASCII commands.

1.5.8 Raw Socket

The 80x3 Raw Socket configuration is set to the following values:

Port	23
Sockets	4
Echo	Off
Prompt	None
Timeout	120 seconds fixed
Logon message	Message equal to *IDN? response.
Terminator	Linefeed character
Carriage Returns	Client generated carriage returns are ignored.
Backspace	Prior character deleted up to the start of the buffer.
Echo On command	Cntl-E
Echo off command	Cntl-F

The Raw Socket connection will be closed if there is no communication for 120 seconds. To prevent the Raw Socket from timing out and disconnecting, the client can issue a no change message like space-backspace or a Cntl-E-Cntl-F sequence, on an occasional basis, to restart the timeout counter.

If Echo was enabled, all client character will be echoed back to the client. The echo sequence for a backspace is a BS-space-BS to visually wipe the deleted character off of the telnet client screen.

1

1.6 INTERNAL WEB SERVER

The internal WebServer provides HTML web pages to W3C compliant browsers.

1.6.1 HTML Pages

The standard HTML pages conform to HTML version 4.01 or XHTML version 1.0. The required pages are needed for correct WebServer operation. User can redefine the other page names. The WebServer serves the stored pages after substituting values for the variable placeholders. The standard html pages are:

404.html	404 Error Page (required page)
501.html	501 Error Page (required page)
index.html	Welcome Page (required page)
config.html	Configuration Page
confirm.html	Confirmation Page
reboot.html	Reboot Page

1.6.3 Graphics

The standard graphics file is:

ICS-Logo.gif ICS Logo

1.6.4 User Configurability

The user can replace the standard HTML pages and image files with modified pages or add additional pages and images to the board using ICS's VXI11_html utility program. User is responsible for assuring that any substituted HTML pages conform to HTML version 4.01 or XHTML version 1.0. Guidelines for modifying the pages are described in Application Bulletin AB80-5.

File supported by VXI11_html	css, html, gif, jpg, png, hgl and xsd
Number of files	32 maximum
File size	63 kbytes maximum for all files
	32 kbytes maximum for a single file
File name size	27 characters

This page left intentionally blank

1

1.7 DIGITAL INTERFACE SPECIFICATIONS

The Digital I/O signals have the following specifications:

1.7.1 Data Lines

Model	8003	40 lines
	8013	128 lines
Pullups	33 Kohm pullups to + 5 Vdc on all lines	
Input Levels	High	≥ 2.4 Vdc or open circuit
		5.5 Vdc maximum input
	Low	< 0.5 Vdc at 200 μ A
Output Levels	High	> 3.0 Vdc with 3 mA source
		> 2.0 Vdc with 24 mA source
	Low	< 0.55 Vdc with 48 mA sink

1.7.2 Data Input, EDR Input and Inhibit Output

Data may be read after receipt of an External Data Ready signal (EDR) if Talk handshaking is enabled or anytime if Talk handshaking is disabled. Inhibit responds $< 0.1 \mu$ sec. after EDR edge. Figure 1-1 shows data loading and output times for 6 HEX characters. Active EDR edge and Inhibit signal polarities are selected by CONFIGure commands. The EDR F/F is reset when the data is read, by the SENSE:RESET:EDR command or by a Device Clear. Standard units only use EDR #1 input. Both EDR inputs have 33 Kohm pullups to + 5 Vdc. Input timing shown in Figure 1-1.

EDR Input	Same as data lines in 1.7.1	
Inhibit Output	High	> 2.4 Vdc with 4 mA source
	Low	< 0.55 Vdc with 16 mA sink

1.7.3 Output Data and Data Strobe

Data may be placed in the output latches by a bit, port or string command or the output lines may be pulsed. A strobe pulse is automatically generated after data is placed in the output latches by a data string or in response to a separate STrobe command. Output timing is shown in Figure 1-2. If data is outputted with a bit or port command, data polarity is set by a port polarity command. If data is outputted with a string command or transparently, port numbers and data polarity are set by the CONFIGure commands. Strobe polarity is set by the CONFIGure commands.

1.7.4 Trigger Output

Trigger pulse is generated by a 488.2 *TRG or a *device_trigger* RPC (GET) commands. Pulse polarity is defined by a CONFIGure command. Logic levels are defined in paragraph 1.7.1.

1.7.5 Reset Output

Reset output is pulsed by 488.2 *RST or *RCL command and when the unit is powered on or reset. Pulse is on while the output is being configured. Pulse polarity is defined by a CONFIGure command. Logic levels are defined in paragraph 1.7.1.

1.7.6 Remote Output

Level output that is asserted when the unit is in the Remote state in response to the *device_remote* RPC command. Signal polarity is defined by a CONFIGure command. Logic levels are defined in paragraph 1.7.1.

1.7.7 Status Inputs

Two external device status inputs with 33 Kohm pullups to + 5 Vdc. Inputs sampled at a 1 kHz or faster rate and the states placed in the Operational Register in the IEEE-488.2 Status Reporting Structure. Status logic polarity is defined by a CONFIGure command. Status A may be used for output data handshaking. Input logic levels are defined in paragraph 1.7.1.

1.7.8 Monitored Digital Inputs

When the first fifteen Digital I/O lines (CH1-11 and 13-16) are used as inputs, they are also sampled at an approximate 1 kHz rate and the values placed in the Questionable Register in the board's IEEE-488.2 Status Reporting Structure. Changes may be used to generate a Service Request. The digital input lines are reported at the following bits in the Questionable Register:

CH#	-	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Bit		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

The lines may also be read like normal input lines.

1.7.9 Stable Output

The Stable output is a high true output signal that can be used to enable external logic or apply power to external relays to prevent erroneous settings during the power turn-on time. It is hardware set to low at power turn-on or when the unit reset. Stable goes high after the Digital I/O lines have been configured. See Figure 1-3 for the Stable signal timing.

1.7.10 External Reset Input

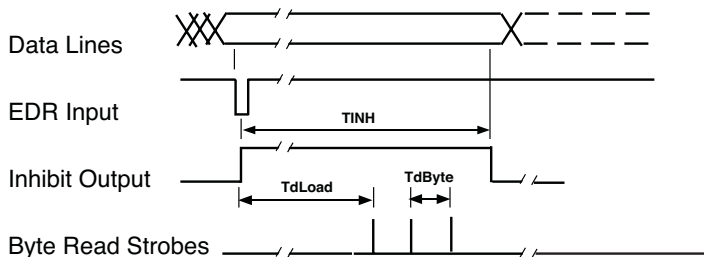
The -ExtRst is a low true input with a 33 Kohm pullup that is used to reset the 8003. The 8003 is held in the reset state while -ExtRst is low.

1.7.11 LED Driver Outputs

Five, low true driver outputs for driving remote LEDs. Maximum sink current is 3.2 mA at 0.5 Vdc.

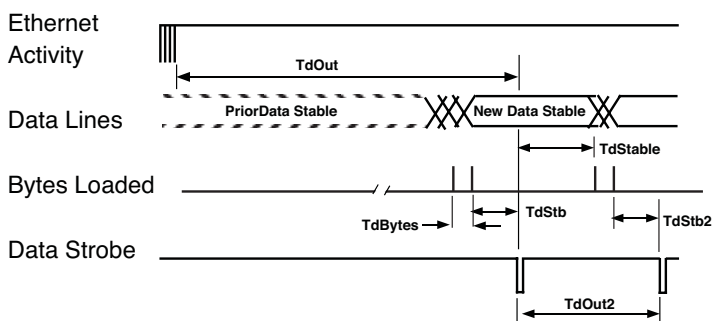
1.7.12 Timing Diagrams

The timing diagram in Figure 1-1 shows input data handshake timing from receipt of the EDR pulse. Figure 1-2 shows data output from command or data packet. Times are given in Table 1-2 for hexadecimal and binary data formats. Figure 1-3 shows how the digital I/O lines are configured after a power turn-on or when the board is reset.



Notes: TdLoad is time to load data after EDR set
Data must remain stable until Inhibit goes false

Figure 1-1 Input Data Timing



Notes: TdOut is delay from command terminator to first data strobe
TdStb is the delay from last byte output to the data strobe for string or binary data transfers.
TdOut2 is the time to the next data strobe for multiple blocks of data in the same command.

Figure 1-2 Output Data Timing

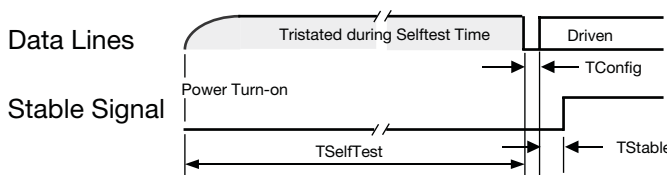


Figure 1-3 Data Outputs After Power Turn-on

1.7.13 Timing Chart

TABLE 1-2 TIMING VALUES

Symbol	Description/Notes	8003	8013
TINH	Inhibit on time to load bytes 1-3	350 μ s	484 μ s
TdLoad	Loading delay time to byte 1	326 μ s	434 μ s
TdByte	Time to next byte	6.0 μ s	6.0 μ s
TdTRd	Delay to Response packet	1-11 ms	1-11 ms
TdOut	Comd packet (hex data) to Data output (16 bit output) "SOURCE:DATA hhhh" ":DATA hhhh" "PO hhhh" "hhhh" (sent to inst1)	4-23 ms 4-22 ms 3-23 ms 3-18 ms	4-23 ms 4-22 ms 3-23 ms 3-18 ms
TdStb	Last data byte output to first strobe pulse	15 μ sec	15 μ s
Strobe	Data Strobe Pulse Width	5 μ sec	5 μ s
TdStable	Delay before next output byte changed	38 μ s	38 μ s
TdStb2	Last data byte to next strobe pulse	32 μ s	32 μ s
TdOut2	Time between strobe pulses	109 μ s	109 μ s
TdOut	Data packet (binary data) to Data output (2 bytes) "bb" (sent to inst1)	3-18 ms	3-18 ms
TdStb	Last data byte output to first strobe pulse	32 μ sec	70 μ s
Strobe	Data Strobe Pulse Width	5 μ sec	5 μ s
TdStable	Delay before next output byte changed	16 μ s	16 μ s
TdStb2	Last data byte to next strobe pulse	12 μ s	12 μ s
TdOut2	Time between strobe pulses	39 μ s	39 μ s
TdSTB	Delay from comd packet to Data Strobe "SOUR:DATA:STR" ":DATA:STR" "SP"	3-24 ms 3-24 ms 2-23 ms	3-24 ms 3-24 ms 2-23 ms
Reset	Reset pulse width (min value)	30 μ sec	76 μ s
TdTRG	Delay from command packet to Trigger pulse "*TRG"	3-22 ms	3-22 ms
Trigger	Device Trigger (GET) command Trigger pulse width (min value)	3-20 ms 5 μ sec	3-20 ms 5 μ s
TSelfTest	Power on to Digital I/O lines configured *	4.3 s	4.3 s
TConf	Byte configuration time	20 μ s	163 μ s
TStable	Last byte configured to Stable high	185 μ s	189 μ s

Notes:

*For Static IP addresses. DHCP startup times vary with server response times.

1. All times are typical and vary due to the multithreaded operating system.
2. Measurements made from receipt of the Ethernet packets.
3. All data sent to inst0 unless otherwise noted.

1.8 CONFIGURABLE FUNCTIONS

Table 1-3 lists the configurable parameters and their factory settings.

TABLE 1-3 DEFAULT SETTINGS

Command	Function	Factory Setting
IP Address Mode	Selects Static or DHCP mode	Static
IP Address	0.0.0.0 to 255.255.255.255	192.168.0.254
Net Mask	0.0.0.0 to 255.255.255.0	255.255.255.0
Gateway IP	0.0.0.0 to 255.255.255.255	192.168.0.1
COMM Timeout	Sets socket timeout	120 sec.
IP KeepAlive	Enables 8013 Keepalive messages	On
Auto Disconnect	Aborts sockets if link count goes to 0	Off
Raw Socket Enable	Enables Raw Socket	Off
Raw Socket Port#	Sets port number	23
Raw Socket Echo	Enables raw socket echo	Off
Raw Socket Timo	Sets raw socket timeout in seconds	120 sec
:INPut	Sets number of Talk bytes	5 or 16 #
:POLarity	Sets Input polarity	1 #
:HANDshake	Enables Input Handshaking	ON #
:OUTput	Sets number of Listen bytes	0 #
:POLarity	Sets Output polarity	1
:HANDshake	Enables Output Handshaking	OFF #
:EDR	Sets EDR input active level	0 #
:INHibit	Sets Inhibit output active level	1 #
:REMOte	Sets Remote output active level	0 #
:RESet	Sets Reset pulse output active level	0 #
:STRobe	Sets Data Strobe pulse active level	0 #
:TRIGger	Sets Trigger pulse active level	0 #
:ASTATus	Sets Status A input active level	1 #
:BSTATus	Sets Status B input active level	0 #
:TALK	Selects Input Format	HEX #
:TRANSlation	Sets input conversion table for BCD	
	HEX characters	see note #
:LISTen	Sets Output Format	HEX #
:WIDTh	Sets Pulse width in ms	50
:IDN	Sets user's IDN message	ICS IDN msg #
:LOCK	Blocks # items from changes	Off
:QUES:ENABle	Enables digital signal changes	0 *
:PTRANSISTION	Enables positive going inputs	All 1s *
:NTRANSISTION	Enables negative going inputs	All 0s *
*ESE	Enables events to generate an SRQ	0
*SRE	Enables Summary bits to generate a service Request	0

* Indicates parameters that cannot be saved in the unit's non-volatile memory.

Indicates parameters that may be blocked by the LOCK command.

1.9 INDICATORS

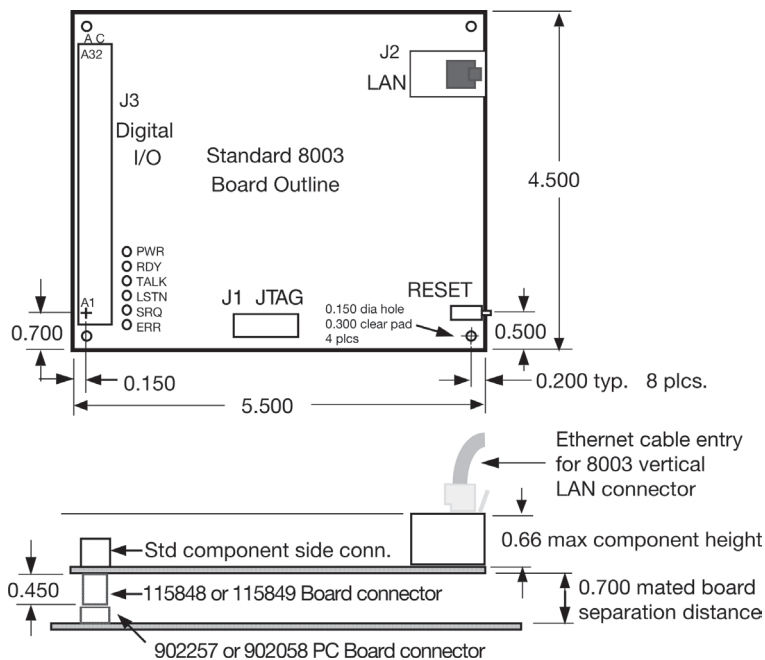
Eight LEDs that normally display the following conditions:

PWR	Indicates power on
LAN	Indicates that the unit is ready and is connected to an active LAN. Blinks at user request to identify the unit.
ACT	Indicates messages are being transferred between the unit and the LAN.
RDY	Indicates the unit has passed self test. Blinks when all 8013 sockets are used and the unit cannot open a new socket or link.
TALK	Indicates the unit was sent a device_read command
LSTN	Indicates the unit was sent a device_write command.
SRQ	On when the board is requesting service. When a reverse Interrupt channel is established and Service requests are enabled, the SRQ LED will blink momentarily to indicate that the board has sent an service request message to the host application.
ERR	Blinks on when the unit has detected a soft error condition such as a command error, device error or a communication problem. Steady on when any of the ESR Register error bits 2 thru 5 are set.

When the board is turned on, it performs an internal selftest and startup which takes about 4 seconds. Only the PWR LED is on during the self test-startup time.

At the end of a successful selftest, the board turns the RDY LED on. At this time the LAN and ACT LEDs display the board's network status. LAN communication is immediate for static IP addresses. DHCP IP address assignment times add to the LAN startup time.

If the board detects a hard self test error, it blinks the error code on its front panel LEDs. Refer to paragraph 6.4 for a description of the selftest errors and their possible causes.



Notes: 1. J2 LAN connector shield is connected to the adjacent mounting hole

Figure 1-4a 8003 Outline Drawing (Vertical RJ45)

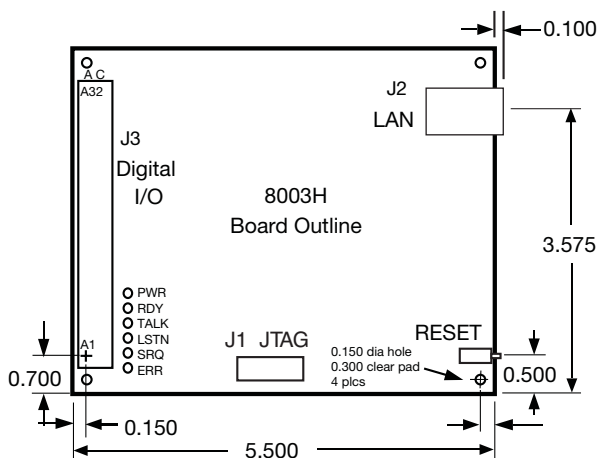


Figure 1-4b 8003H Outline Drawing (Horizontal RJ45)

1.10 PHYSICAL

1.10.1 8003 Physical

Size	5.5" L x 4.5" W x 0.7" H (13.97 cm L x 11.43 cm W x 1.78 cm H) (See Figure 1-4)	
Material	PC Board - FR406 Flame resistant Fiberglass Components - RoHS compliant	
Construction	Lead free, RoHS compliant	
Weight	0.3 lbs (0.136 kg)	
Temperature	Operating	-10 °C to +55 °C
	Storage	-40 °C to +85 °C
Humidity	0-90% RH without condensation	
Power	5 ± 0.2 Vdc at 450 mA	
Connectors	LAN - RJ-45 Orientation - Model 8003 vertical; 8003H horizontal Digital Interface - component side versions 96 pin right-angle female DIN connector shell with male pins in rows A and C. Mates to ICS P/N 902023, 902067 and 902124 Digital Interface - circuit side versions 96 pin male DIN connector shell with female sockets. Mates to ICS P/N 902257	

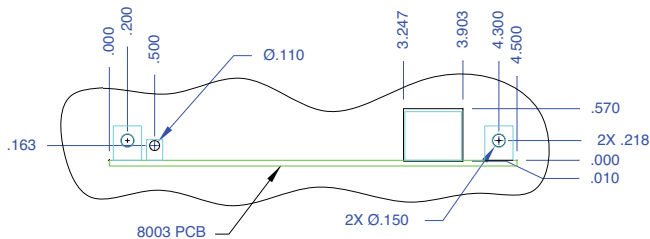


Figure 1-5 8003H Rear Panel Mounting Dimensions

Figure 1-6 8013 Outline Drawing

1.10.1 8013 Physical

Size	7.0" L x 5.5" W x 0.65" H (17.78 cm L x 13.97 cm W x 1.65 cm H)
Material	PC Board - FR406 Flame resistant Fiberglass Components - RoHS compliant
Construction	Lead Free, RoHS compliant
Weight	0.3 lbs (0.136 kg)
Temperature	
Operating	-10 °C to +55 °C
Storage	-40 °C to +85 °C
Humidity	0-90% RH without condensation
Power -	5 ± 0.2 Vdc @ 500 mA
Connectors	
Ethernet	RJ-45 Orientation 8013 Horizontal; 8013V vertical
Digital	Component side versions 150 pin vertical header (3 rows x 50 pins on 0.1 in centers) Pin height 0.430 inch. Samtec P/N TSW-150-07-G-T Note: Contact Samtec for information about mating connectors and their part numbers. Part numbers will vary depending upon mating connector pin heights. A suggested part is SSW-150-21-G-T Circuit side version 150-pin female header (3 rows x 50 pins on 0.1 in centers) Samtec P/N SSW-150-21-G-T

1.11 CERTIFICATIONS OR APPROVALS

None required. Approval is by part of the host chassis.

1.12 INCLUDED ACCESSORIES

120187	8003/8013 Instruction Manual
123038	Support CD-ROM with Configuration Program, Documentation, Sample Programs and Utilities.
895011	Ethernet Crossover Cable
902323	8013 Power Plug (mates to J3)

1.13 OPTIONAL ACCESSORIES

120187	8003/8013 Instruction Manual
115606-01	Wiring Kit with 1 foot Ethernet cable
115606-02	Wiring kit with 2 foot Ethernet cable
115490	xx03 Relay Driver Board (for 8003 conn down bd)
115640	xx13 Relay Driver Board (8013 only)
115650	xx13 Connector Board (8013 only)
895011	Ethernet Crossover Cable (5 feet long)
902023	8003 DIN mating connector, female, solder eyelet
902025	8003 DIN mating connector, female, solder pins
902058	8003 DIN mating connector, male, solder pins for connector down boards
902308	8013 Digital I/O mating connector, solder tails
902323	8013 Power Plug (mates to J3)
902329	Ethernet shielded bulkhead connector

Installation

2.1 INTRODUCTION

This section provides the user with directions for shipment verification, for configuring the Parallel Interface Boards for operation on the network, for installing the boards and for connecting to their digital interface.

2.2 UNPACKING

When unpacking, check the unit for signs of shipping damage (damaged box, scratches, dents, etc.) If the unit is damaged or fails to meet specifications, notify ICS Electronics or your local sales representative immediately. Also, call the carrier immediately and retain the shipping carton and packing material for the carrier's inspection. ICS will make arrangements for the unit to be repaired or replaced without waiting for the claim against the carrier to be settled.

2.3 SHIPMENT VERIFICATION

Take a moment to verify that the following items were included with your unit:

- (1) Model 8003 or 8013 Ethernet to Parallel Interface Board
- (1) Instruction Manual
- (1) Support CD-ROM
- (1) Ethernet Crossover Cable

2.4 QUICK INSTALLATION GUIDE

The following steps should be used as to set up and use your board:

IMPORTANT: A new board's network parameters must be configured before being connected to your test system or company network. Follow the directions in Paragraph 2.5 to configure the board's network parameters before connecting it to any network.

1. Plan the board's mounting location. Refer to paragraph 2.7 for suggestions on mounting the board in a chassis or on a PC board.
2. Design the power connections to the board. The 8003 and 8013 require 500 mA of regulated 5 volts power. Add additional current capacity for any loads that the board will be sourcing. Loads the board will be sinking do not require any additional current from the board. Power can be applied through the Digital I/O Connector or through Power Connector J3 on the 8013 (See Figures 2-9 and 2-10). **CAUTION:** Do not power the boards from a power supply that will be powering noisy devices like relays.
3. Design the digital connections between the board and the device(s) it will connect to as directed in Paragraphs 2.8 - 2.10. This will provide you with the signal-pin assignments for wiring the connector and the digital I/O configuration values.
4. Install the board in the location where it will be used. Connect the board to a power source and complete the Digital I/O connections.
5. Use a web browser or ICS's VXi-11 Configuration Utility to configure the board for your network by following the directions in Paragraph 2.5.
6. Use ICS's VXi-11 Keyboard Utility or your own program to test the board's digital I/O connections. When satisfied that all connections are correct, set the board's digital I/O lines and configuration parameters to their desired power turn-on state. Use the *SAV 0 command to save the current state and polarities as the new power turn-on state.
7. New users should read Section 3 before attempting to write a program to control the interface board.

2.5 NETWORK SETTINGS

This paragraph configures the 8003 or 8013 for operation on your network. The board's digital interface is configured later by sending commands as outlined in Section 3.

When shipped, the boards are configured as shown in Table 1-3. Review the Table with your network administrator and decide on which settings, if any, that need to be changed. Table 2-1 provides additional information about each network setting to help you with your decisions. A minimum change is to set a static IP address so your PC can communicate with the board.

The network configuration can be changed and the board's MAC Address can be read with a web browser, by running ICS's VXI-11 Configuration Utility on a WIN32 or WIN98 PC, or with the RPC configuration commands listed in Appendix 3.

2.5.1 Web Browser Configuration Method

This method uses a standard browser such as Firefox, Internet Explorer or Safari to view and change the current network settings.

1. Connect the 8003 or 8013 directly to the computer running the browser. Temporarily disconnect the computer from the company network and use the supplied Ethernet Crossover Cable to connect the computer to the board as shown in Figure 2-1. This will eliminate any potential network conflicts while configuring the board.

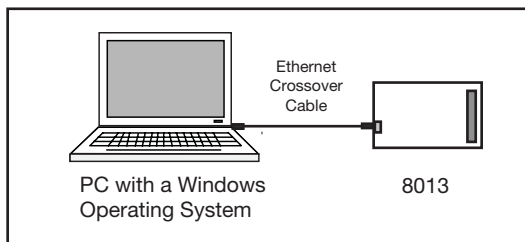


Figure 2-1 8013 Connected to computer with a Crossover Cable

An alternate connection is to use a standard Ethernet Cable to connect the board to the same hub or switch that the computer running the browser is connected to as shown in Figure 2-2. CAUTION: Temporarily disconnect the local network connection to avoid network conflicts until the board is configured.

TABLE 2-1 8003/8013 CONFIGURATION SETTINGS

Setting	Choices	Comments
IP Address Mode	Static	Static lets the user set the unit's IP Address, Net Mask and Gateway IP values.
	Dynamic	Dynamic enables the unit to accept the IP, net mask and gateway values supplied by a DHCP Server.
IP Address	Any	Any valid IP address setting. Must be four groups of numbers between 0.0.0.0 and 255.255.255.255.
Net Mask	Any	Same range as the IP Address
Gateway IP	Any	Same range as the IP Address.
COMM_Timeout	0 to 2 ³² -1 seconds	Sets the COMM timeout value. If the unit senses no client communication for more than the COMM_Timeout setting, the unit will close the socket, release any locks and reclaim all associated instrument resources. Use a short setting of 3-5 minutes when debugging programs to recover broken links faster and a longer setting of 10 minutes or more for debugged applications. A value of 0 disables COMM_Timeout. Value is 0 or 1 to 2 ³² -1.
IP KeepAlive	On or Off	Enables the unit's socket layer to send the client's side socket layer a short test message once every 120 minutes if no messages have been sent. If the other socket fails to reply, the unit will close the socket, release any locks and reclaim all associated instrument links. Do not enable Keep Alive if the target socket does not support Keep Alive messages. The recommended setting is On.
Interface Name	Any	Sets the name used by the VXI-11 Create Link command when linking to a GPIB device. The default names is: inst CAUTION: The Interface Name is not the Host Name. Changing the Interface Name may make the board difficult to link to and your programs unusable. The recommendataion is to use the default value.
Auto Disconnect	On or Off	Closed a socket when link count goes to zero. Only use with programs that exhaust the 8003's sockets. See paragraph A2.2 for details. The default is Off.

Notes: Table continued on page 2-6. Default values are listed in Table 1-3

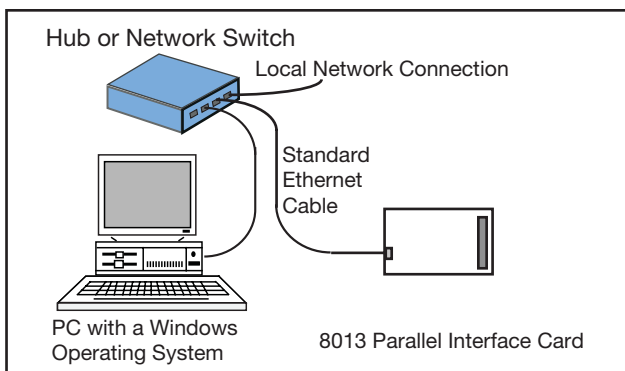


Figure 2-2 8013 Connected to the local hub

2. Apply power to the board and verify that it passes its selftest routine and that the PWR, RDY and LAN LEDs are on.
3. Check your computer's network settings to be sure its IP address is in the 192.168.0.xxx range so it can communicate with the board's default IP address. If it is not, it must be set before proceeding. Use the values shown below. For Windows PCs, right-click on My Network Places and click on Properties. Right-click on Local Area Connection and click Properties. Highlight Internet Protocol (TCP/IP) and click Properties. If your PC's IP address is in a different range, record the current settings and temporarily set the following network values:

Check	'Use the following IP Address'
IP Address	192.168.0.2
Subnet mask	255.255.255.0
4. Open the browser and enter the default IP address of 192.168.0.254 or your last set IP address in the browser address window. A Welcome Page similar to the one shown Figure 2-3 should appear in your browser.
5. If you want to change any of the settings, press the 'Update Configuration' button. A Configuration Page similar to the one shown in Figure 2-4 should appear in your browser.

**TABLE 2-1 NETWORK CONFIGURATION SETTINGS
CONTINUED**

Setting	Choices	Comments
Raw Socket Enable	On or Off	Enables Raw Socket protocol. Default is Off.
Raw Socket Port#	0-65535	Instrument raw socket port number, 0 to 65535. Default is port 23.
Raw Socket Echo Enable	On or Off	Enables raw socket commands to be echoed back to the sender. Default is Off.
Raw Socket	0-2 ³²	Sets the time the 8064 will go without any communication on a raw socket before closing the socket and restoring all resources to the system. Default is 120 seconds.

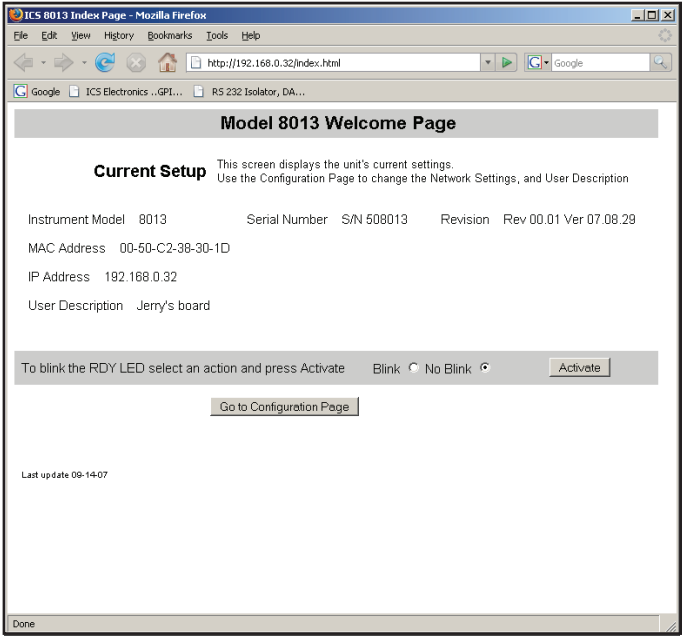


Figure 2-3 8013 Welcome Page

**ICS
ELECTRONICS**

Model 8013 Configuration Page

Instructions

This screen contains the unit's configurable network and pulse width settings.
Enter any new settings and press the Update Flash button to save the new settings.

TCP/IP Mode: STATIC ☒ DHCP ☐

IP Address:

Net Mask:

Gateway Address:

User Description:

Comm Timeout: in seconds

IP KeepAlive: On ☒ Off ☐

Auto Disconnect Sockets: On ☐ Off ☒ [what's this?](#)

Raw Socket Enable: On ☐ Off ☒

Raw Socket Port#:

Raw Socket Echo Enable: On ☐ Off ☒

Raw Socket Timeout: in seconds

Update Flash

Clear

Return to the Welcome Page

Last update 12-20-2015

Figure 2-4 8013 Configuration Page

6. Enter the new settings as desired. If you select DHCP for the TCP/IP Mode, the page blanks out the IP, Net and Gateway addresses as they will be supplied by your DHCP server. Check the entered values carefully as the unit's webserver does minimal error checking. Press the 'Update Flash' button when done. A Confirmation Page similar to the one shown in Figure 2-5 will appear in your browser.

7. Your new settings have been saved in the board's flash memory. You have to reboot the unit or power cycle it for the changes to take affect. Press the 'Reboot' button to reboot the unit now or the 'Return to the Configuration Page' button to revisit the new settings.

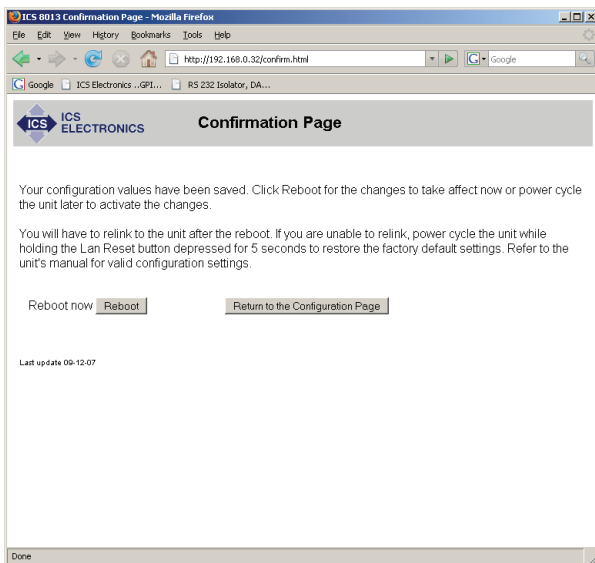


Figure 2-5 8013 Confirmation Page

2.5.2 VXI-11 Configuration Utility Method

The VXI-11 Configuration Utility program, VXI11_config, can view and change the current network and serial configuration settings but cannot configure any Raw Socket settings. VXI11_config runs in WIN32 PC (Windows 98, Me, 2K, XP and 2003 operating systems and is included on the Support CD-ROM supplied with the unit. Select the Install VXI-11 Support option on the Support CD's Main Page to install VXI11_config and VXI11_kybd on your computer. VXI11_config can be run from Window's Start menu by pointing to Programs and then to ICS_Electronics. Select VXI11_config from the submenu.

1. Connect the board directly to the WIN32 PC that will be running the Configuration Utility. Disconnect the PC from the company network and use the supplied Ethernet Crossover Cable to connect the PC to the board as shown in Figure 2-6. This will eliminate any potential network conflicts while configuring the board.

Alternately, use a standard Ethernet patch cable to connect the 8013 to the same hub or switch that the PC running the Configuration Utility is connected to as shown in Figure 2-7. Temporarily disconnect the local network connection to avoid network conflicts until the board is configured.

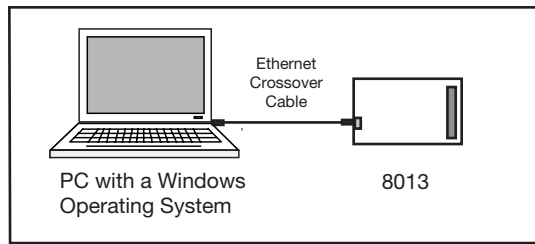


Figure 2-6 8013 Connected to PC with a Crossover Cable

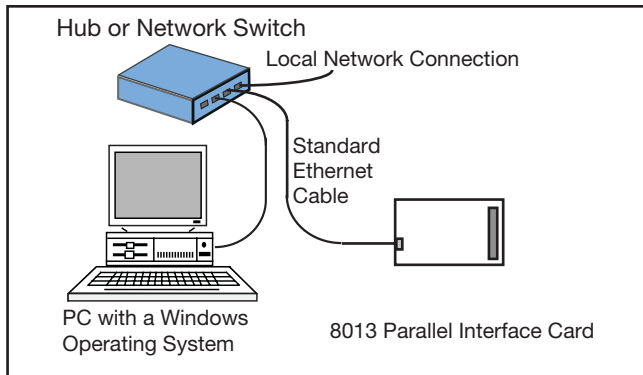


Figure 2-7 8013 Connected to the local hub

2. Apply power to the board and verify that it passes its selftest routine and that the PWR, RDY and LAN LEDs are on.
3. Check your PC's network settings to be sure its IP address is in the 192.168.0.xxx range so it can communicate with the board's default IP address. To check, right-click on My Network Places and click on Properties. Right-click on Local Area Connection and click Properties. Highlight Internet Protocol (TCP/IP) and click Properties. If your PC's IP address is in a different range, record the current settings and temporarily set the following network values:

Check	'Use the following IP Address'
IP Address	192.168.0.2
Subnet mask	255.255.255.0

4. Run the VXI11_config program. The Configuration Utility opens a window as shown in Figure 2-8. Initially only the Find Server, Help and Exit buttons are enabled on the program window. The other buttons will be enabled as you advance through the program.

5. Click on the **Find Server** button. The program scans for all VXI-11 Services connected to the local LAN or to your PC. (The 8003 and 8013 are RPC servers which provide a VXI-11 Service) The results are displayed in the Results box.
6. When the servers(s) have been found, use the pulldown arrow in the Found Servers box to view the Found Server addresses. The board's default address is 192.168.0.254. Highlight the board's IP address and click the **Create Link** button. If the server is not found, you can enter the default IP address (192.168.0.254) in the Found Servers box. Click the Create Link button.

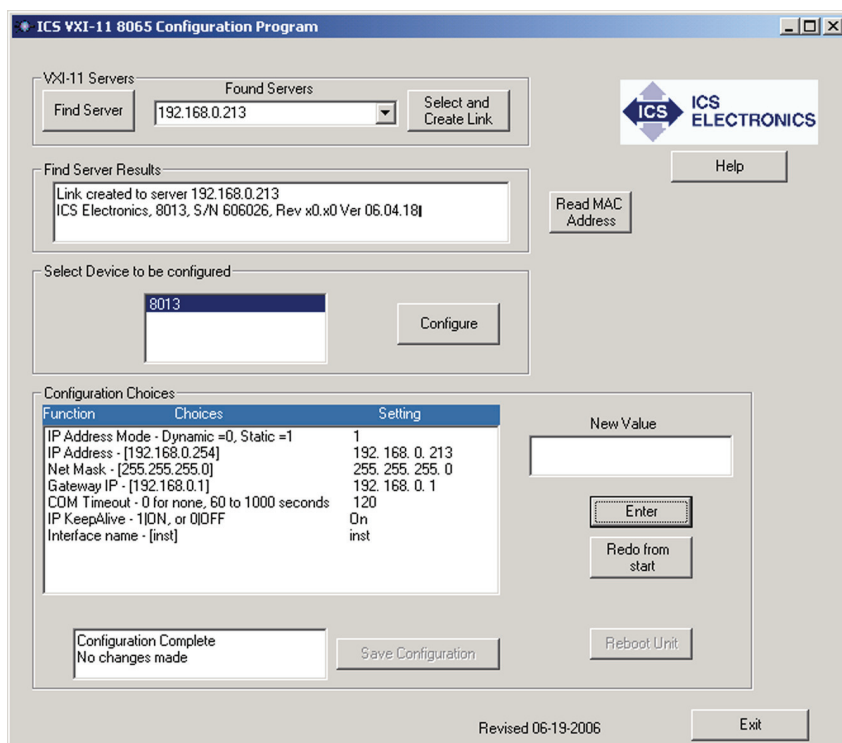


Figure 2-8 VXI-11 Configuration Utility
(Showing all configuration choices with no changes)

7. When the link has been created, device model number(s) will appear in the 'Select Device to be Configured' box. Highlight the desired model number and click the **Configure** button to start the configuration process.

8. The Configuration Choices box displays only one line with the first parameter to be changed and its current setting. If you like the current setting, click **Enter** to advance to the next parameter. If you want to change the setting, type a new value in the New Value box and click **Enter**. The program will send your setting to the board and read back the new setting. Repeat as needed to make another change or click **Enter** again to advance to the next parameter.
9. Repeat step 8 for each configuration parameter. Figure 2-8 shows the VXI-11 Configuration Utility after all parameters have been entered for a Model 8013. Click the **Redo From Start** button if you need to start over or if you want to change any of the prior settings
10. When done, the **Save Configuration** button is enabled if you changed any settings. Click the **Save Configuration** button to save the values in the board's flash memory.

If you did not make any changes you can just exit the program.
11. The board has to be power cycled or rebooted before the configuration changes take affect. Click the **Reboot** button to reboot the board and use the new settings.
12. Press the **Exit** button to quit the VXI11_config program.
13. If the IP address was changed to an address outside the 192.168.0.xxx range in step 3, your PC's network settings will have to be changed to communicate with the board. Exit the VXI11_config program and restore the PC's network settings.

RESETTING DEFAULT NETWORK SETTINGS

The board can be reset to the default network settings listed in Table 1-3 at any time by holding the LAN Reset Button in for 5 seconds while applying power to the board. The TALK, LSTN and SRQ LEDs all blink on momentarily when the board is changed back to its default settings. The Digital I/O configuration values are not affected by the LAN Reset operation.

2.6 JUMPER SETTINGS

The 8003 and 8013 have the following jumper positions as shown in Tables 2-2 and 2-3. Jumper locations are shown in Figures 2-9 and 2-10 below.

TABLE 2-2 8003 JUMPERS

Jumper	Function	Factory Setting
W1	Factory PLD-ARM programming jumper	ARM
W2	Factory PLD-ARM programming jumper	Open
W3	Option Jumper. Not used in standard firmware	Open
W4	Restores I/O default settings. See paragraph 5.6	Open

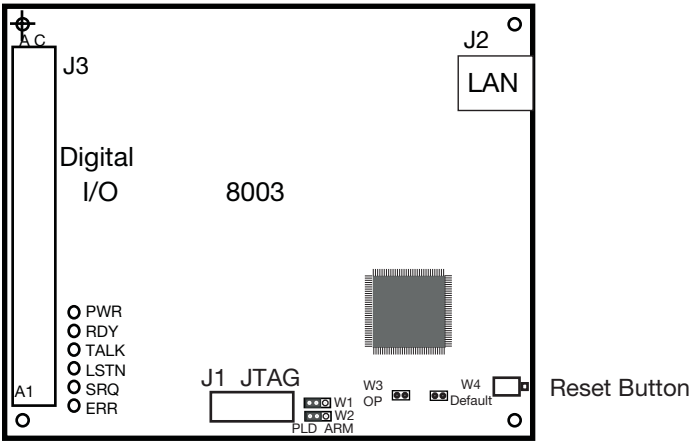


Figure 2-9 8003 Jumper Locations

TABLE 2-3 8013 JUMPERS

Jumper	Function	Factory Setting
W1	Restores I/O default settings. See paragraph 5.6	Open
W2	Option Jumper. Not used in standard firmware	Open
W3	8013 only. EDR #2 Input or LAN LED Output select.	EDR #2
W4	8013 only. INH #2 Output or ACT LED Output select	INH #2

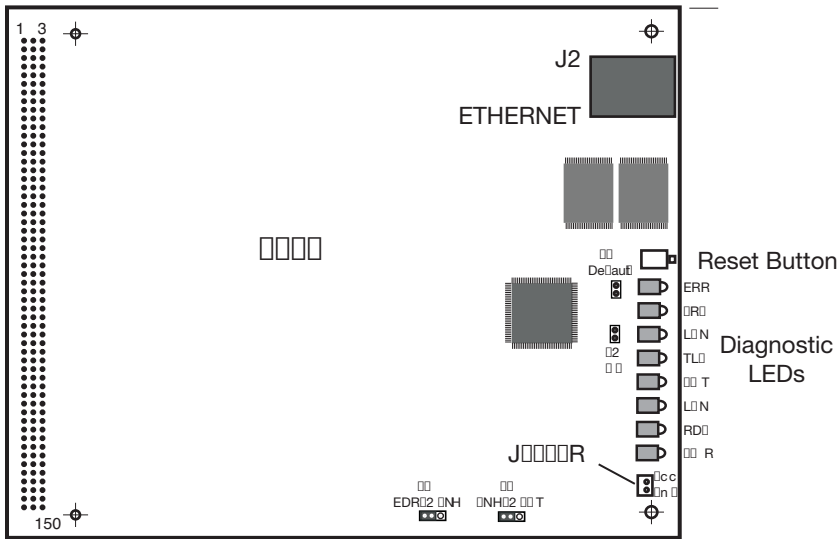


Figure 2-10 8013 Jumper Locations

2.7 MOUNTING INSTRUCTIONS

Both boards may be mounted inside a host chassis as described below. Avoid areas of high heat or strong electrical fields. Use a Ethernet extension cable from the board to a bulkhead adapter on the rear panel. ICS's 115606 Wiring Kits include short Ethernet cables and a shielded Ethernet Bulkhead Adapter (P/N 902329) for panels from 0.040 to 0.062 inches thick. Figure 2-11 shows the Bulkhead Adapter cutout dimensions.

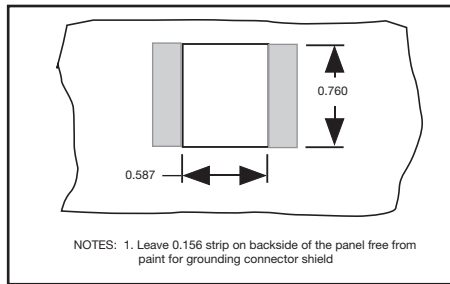


Figure 2-11 Cutout for 902329 Bulkhead Adapter

2.7.1 8003

The 8003 has four mounting holes as shown in Figure 1-4 and can be mounted against any panel or floor of the host chassis. Minimum spacing to any metal surface is 0.125 inches (3.2 mm). The recommendation is to connect the mounting pad adjacent to J2 to chassis ground. Mating DIN connectors for PC boards for wires or for flat-ribbon cables can be obtained from any DIN connector supplier or from ICS Electronics. Refer to the mating connector list in paragraph 1.10.1.

2.7.1.1 Mounting to another PC Board

The 8003 is available with a DIN connector on the circuit-side for mounting piggy-back style on top of another PC board as shown in Figure 1-4. The circuit-side connector has female pins. Use a DIN connector with male pins on the motherboard like ICS P/N 902058 (96 pins) or 902257 (64 pins). The mated separation distance between the two boards is 0.700 inches. The connector layout on the motherboard should be for a 3-row, 96-pin DIN 41612 connector. Label the pads to match the row and pin orientation shown in Figure 1-4a so the pin numbers match those on the 8003 board.

2.7.1.2 Rear Panel Mounting

Mounting the 8003 against the rear panel provides the end user with access to the 8003's Ethernet connector, to the IP Reset push-button, and eliminates an Ethernet extension cable. Figure 2-12 shows the recommended rear panel cutouts and openings for rear panel mounting. Use the optional mounting blocks to hold the 8003 against the rear panel and to ground the connector shield to the chassis.

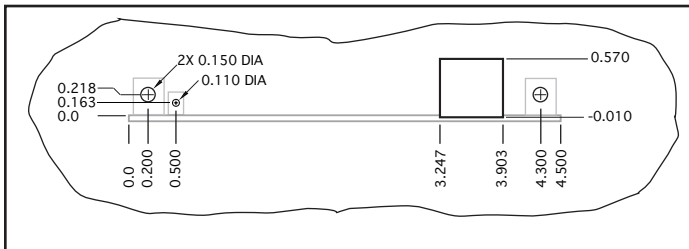


Figure 2-12 8003 Rear Panel Mounting Dimensions.

2.7.2 8013

The 8013 has four mounting holes as shown in Figure 1-5 and can be mounted against any panel or floor of the host chassis. Minimum spacing to any metal surface is 0.125 inches (3.2 mm). The recommendation is to connect the mounting pad adjacent to J2 to chassis ground.

Mating Digital I/O Connectors for J1 can be obtained from Samtec or from ICS Electronics as P/N 902308. Mating Power Connectors with lock tabs for connector J3 can be obtained from Amp or from ICS Electronics as P/N 902323.

2.7.2.1 Mounting to another PC Board

The 8013 can be mounted on another PC board (piggy-back style) or have a board mounted on it as shown in Figure 1-5. The component height on the 8013 is less than the mated separation distance between the two boards so there are no interferences around connector J1. However the mating PC board cannot extend over the 8013's Ethernet connector, J2, since J2's height of 0.545 inches exceeds the mated separation distance.

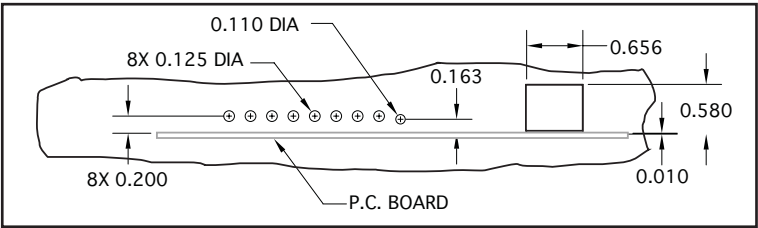
The 8013 is available with a female connector mounted on the circuit side. Use a 3-row, 150 pin male connector, 902307, on the motherboard. Copy the pin numbers from the 8013 down to the pad layout on the motherboard so they match those on the 8013. Support the front of the 8013 at the front mounting holes with 0.435 long spacers.

2.7.2.2 Connector Board

The 115650 Connector Board Assembly plugs on top of the 8013 and provides the user with four 36-pin flat-ribbon headers for users who want several small connectors in stead of the 8013's 150-pin connector. Each header contains 32 digital I/O signals, ground pins, a +5 Vdc line and an unassigned signal line that the user can jumper to any of the 8013 handshake lines.

2.7.2.3 Rear Panel Mounting

Mounting the 8013 against the rear panel provides the end user with access to the 8013's Ethernet connector, to the IP Rest push-button, to the diagnostic LEDs and also eliminates an Ethernet extension cable. Figure 2-13 shows the recommended rear panel cutouts and openings for rear panel mounting. Connect the mounting hole adjacent to J2 to chassis to ground the connector shield.



Note: Use horizontal dimensions shown in Figure 1.5 to complete the cutout.

Figure 2-13 8013 Rear Panel Mounting Dimensions

2.8 DIGITAL I/O CONNECTIONS

New users should read all of Section 2.8 to familiarize themselves with the board's digital interface and how it operates before laying out the connections to the external devices.

2.8.1 Digital I/O Overview

The digital I/O lines are controlled by 8-bit bidirectional latches. The 8 bits or lines in a latch are referred to as a byte or port. Data direction is output when the digital lines output data received from the network. Data direction is input when data is read in from the digital lines and is outputted to the network. All bits in a byte have the same data direction.

There are three methods to transfer data to and from the bidirectional latches. The user can use a combination of methods that is best for the application.

The simplest data transfers are with the port type commands that address a specific byte or with bit commands that set/reset/pulse or read a specific bit in a byte. All bits in the byte have the same data direction. Data polarity in the byte can be set on a bit-by-bit basis for the port and bit commands. Bytes controlled by the port commands are automatically set as inputs or outputs when the command is first executed and **are not configured** with the configuration commands.

String commands or transparent data strings can be used to transfer data in multiple byte wide words. Bytes used by the string commands **must be configured** into input or output strings by the configuration commands. Data polarity is also set by the configuration commands. Depending upon the selected data format, each eight-bit byte can be considered as two 4-bit nibbles where two BCD or HEX coded digits are encoded into a single character. Two BCD or HEX characters make up a byte. Other formats let the user express a byte as a decimal number (0 to 255) or as a single binary byte.

The user can save the current Digital I/O line output states, polarity and data direction at any time with the 488.2 *SAV 0 command. The board will use the saved configuration to initialize the Digital I/O lines when next powered on or reset.

2.8.2 Digital I/O Lines

In the Signal-Pin Assignment Tables, the Digital I/O lines are grouped by bytes and by 4-bit nibbles. Each byte has eight bits, numbered 0 (LSB) thru

7 (MSB), or two nibbles. The bytes are numbered 1 thru n in ascending order. This is the order the 8003 inputs or outputs multi-byte wide data. The corresponding 4-bit nibbles are numbered Most Significant to Least Significant Nibble in descending order. This is the same way as you would write a string of characters from left to right, i.e.

12345 where 1 is the MSD and 5 is the MSD-4.

Data transfer for byte or bit commands is to or from the specified byte. i.e.

Command 'BO1 13' 'writes 00001101 to byte 1

Command 'Close 1,0' 'sets bit 0 in byte 1 on

Data transfer for string commands is in the MSN to LSN nibble direction which is the same as the ascending byte count direction. The user can assign one or multiple bytes into Talk (input) or Listen (output) strings with the configuration commands. Bytes assigned to strings must be assigned in numerical order and can start with any byte number. The data polarity, conversion format and data transfer handshaking settings apply to all of the bytes configured in the same input or output string. Only one input string and one output string can be configured on a board. Unassigned bytes can be used by the port and bit commands or monitored for changes.

When the first two bytes are used as input bytes, 15 data lines in the first two bytes are sampled at a > 1 kHz rate and their values saved in the Questionable Register. Sampling may be interrupted or slowed when the processor is transferring data. Changes in any of the 15 data lines can be used to generate an SRQ or service request message. In the Signal-Pin Assignment Tables these signals are shown shaded to denote their dual use. The user can select bit transition directions to detect signal changes and enable individual bits to generate a Service Request when an enabled bit is set. See Section 3.4 for more information about the board's Status Reporting Structure. To enable monitoring, the first one or two bytes must be used for inputs and not used as outputs.

Once the board's Digital I/O lines have been set to their desired state, the state can be saved and recalled at power turn-on. At power turn-on, the Digital I/O lines are initially tristated and pulled up to V_{cc} by 33 Kohm pullup resistors. This tristated time is the gray shaded area in Figure 1-3. The Digital I/O lines are not set to their saved configured state until the board boots up and completes its selftest. This takes approximately 4 seconds. Low true output lines will have a 30 μ sec glitch when they are switched from the tristated state to the configured driven state. The Stable output signal is asserted after the I/O lines are configured and should be used to enable external devices that cannot stand a 4 second boot up time.

2.8.3 Input Handshaking

When input handshaking is enabled, the External Data Ready (EDR) signal tells the board that the external data is valid and can be read. The data must be steady from the leading edge of the EDR signal to the trailing edge of the Inhibit (INH) signal. The user can select positive or negative going EDR edge, high or low true Inhibit signal and several modes of inputting the data.

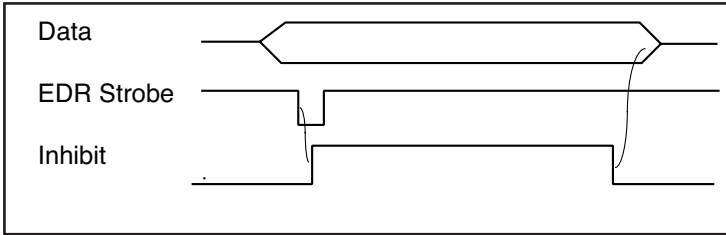


Figure 2-14 EDR Signal Timing

2.8.4 Status Inputs

Two input signals (Status A and Status B) which can be used to monitor the external device's condition. The Status signals are inputted thru the Operational Register in the board's 488.2 Status Reporting Structure so that changes to the status inputs can be used to generate a Service Request and interrupt the Bus Controller. The user can program the Status Signal polarities, monitor conditions and enable SRQ generation. Status A is also used for output handshaking.

2.8.5 Output Data Strobe and Handshaking

The Output Data Strobe is pulsed after data is outputted with a string command or by executing the STRobe command. If Listen handshaking is enabled, the Strobe line is only pulsed when the Status A input is in its logic 1 state. Holding the Status A input in the logic 0 state inhibits data transfer when Output Handshaking is enabled. Use the CONFigure commands to set the Strobe polarity and enable/disable Output Handshaking.

2.8.6 Trigger Output

The Trigger line is pulsed when the board receives a valid trigger command such as the 488.2 *TRG command to initiate an external action. Use the CONFigure commands to set the Trigger polarity.

2.8.7 Reset Output

The Reset line is pulsed at power turn-on or when the board receives the 488.2 *RST or *RCL 0 command. Use the Reset output to initialize an external device. Use the CONFIgure commands to set the Reset polarity.

2.8.8 Remote Output

The Remote output signal can be used to enable the user's front panel controls. The board assumes the remote state and sets Remote true when it receives a network command or when it receives the *device_remote* RPC command. The board assumes the local state and sets Remote false when it receives the *device_local* RPC command. Use the CONFIgure commands to set the logic true state of the Remote Output signal.

2.8.9 Stable Output

The Stable output is held low at power turn-on time or when the board is reset. The Stable output is set high when the Digital I/O lines have been set to their saved configuration at the end of the power turn-on sequence. Use the Stable signal to hold off any external devices that are bothered by glitches in the digital I/O lines before they are configured. Figure 1-3 shows the power turn-on timing diagram. Timing values are listed in Table 1-2.

2.8.10 External Reset Input

The board can be reset by pulling the -ExtRst input low. The digital I/O lines are tristated while the -ExtRst input is low. Although the board has an internal supervisor circuit that resets the board at power turn-on time, the user may want to reset the board when the host chassis is reset.

2.8.11 LED Driver Outputs

Low true driver outputs are provided for driving seven remote LEDs: LAN, ACT, RDY, TALK, LSTN, SRQ and ERR. The LED drivers sink 3.2 mA. The user should use low current LEDs and a pullup resistor to + Vcc that limits the current to 3 mA or use a non-inverting driver like a 74xx126 to increase the LED drive current. 8013 LAN and ACT LEDs share pins with the EDR#2 input and INH#2 output signals. Signal selection is made by setting jumpers on the 8013 board. See Figure 2-10 for jumper locations.

TABLE 2-4 8003 DIGITAL SIGNAL PIN ASSIGNMENTS

Signal	Signal Weighting		Pin	Wire* Color	User Signals	
	Binary	BCD/HEX			Pin	Signal
CH 8	Byte 1 Bit 7	MSN Bit 8	C16	BRN 4		
CH 7	Byte 1 Bit 6	MSN Bit 4	A16	RED 4		
CH 6	Byte 1 Bit 5	MSN Bit 2	C15	WHT 3		
CH 5	Byte 1 Bit 4	MSN Bit 1	A15	BLK 3		
CH 4	Byte 1 Bit 3	MSN-1 Bit 8	C14	VIO 3		
CH 3	Byte 1 Bit 2	MSN-1 Bit 4	A14	GRY 3		
CH 2	Byte 1 Bit 1	MSN-1 Bit 2	C13	GRN 3		
CH 1	Byte 1 Bit 0	MSN-1 Bit 1	A13	BLU 3		
CH 16	Byte 2 Bit 7	MSN-2 Bit 8	C20	WHT 4		
CH 15	Byte 2 Bit 6	MSN-2 Bit 4	A20	BLK 4		
CH 14	Byte 2 Bit 5	MSN-2 Bit 2	C19	VIO 4		
CH 13	Byte 2 Bit 4	MSN-2 Bit 1	A19	GRY 4		
CH 12	Byte 2 Bit 3	MSN-3 Bit 8	C18	GRN 4		
CH 11	Byte 2 Bit 2	MSN-3 Bit 4	A18	BLU 4		
CH 10	Byte 2 Bit 1	MSN-3 Bit 2	C17	ORG 4		
CH 9	Byte 2 Bit 0	MSN-3 Bit 1	A17	YEL 4		
CH 24	Byte 3 Bit 7	MSN-4 Bit 8	C24	VIO 5		
CH 23	Byte 3 Bit 6	MSN-4 Bit 4	A24	GRY 5		
CH 22	Byte 3 Bit 5	MSN-4 Bit 2	C23	GRN 5		
CH 21	Byte 3 Bit 4	MSN-4 Bit 1	A23	BLU 5		
CH 20	Byte 3 Bit 3	MSN-5 Bit 8	C22	ORG 5		
CH 19	Byte 3 Bit 2	MSN-5 Bit 4	A22	YEL 5		
CH 18	Byte 3 Bit 1	MSN-5 Bit 2	C21	BRN 5		
CH 17	Byte 3 Bit 0	MSN-5 Bit 1	A21	RED 5		
CH 32	Byte 4-bit 7	MSN-6 Bit 8	C28	GRN 6		
CH 31	Byte 4-bit 6	MSN-6 Bit 4	A28	BLU 6		
CH 30	Byte 4-bit 5	MSN-6 Bit 2	C27	ORG 6		
CH 29	Byte 4-bit 4	MSN-6 Bit 1	A27	YEL 6		
CH 28	Byte 4-bit 3	MSN-7 Bit 8	C26	BRN 6		
CH 27	Byte 4-bit 2	MSN-7 Bit 4	A26	RED 6		
CH 26	Byte 4-bit 1	MSN-7 Bit 2	C25	WHT 5		
CH 25	Byte 4-bit 0	MSN-7 Bit 1	A25	BLK 5		


 indicates signals also used as the Questionable Register inputs

TABLE 2-4 8003 DIGITAL SIGNAL PIN ASSIGNMENTS

Signal	Signal Weighting		Pin	Wire* Color	User Signals	
	Binary	BCD/HEX			Pin	Signal
CH 40	Byte 5 Bit 7	MSN-8 Bit 8	C32	ORG 7		
CH 39	Byte 5 Bit 6	MSN-8 Bit 4	A32	YEL 7		
CH 38	Byte 5 Bit 5	MSN-8 Bit 2	C31	BRN 7		
CH 37	Byte 5 Bit 4	MSN-8 Bit 1	A31	RED 7		
CH 36	Byte5 Bit 3	MSN-9 Bit 8	C30	WHT 6		
CH 35	Byte 5 Bit 2	MSN-9 Bit 4	A30	BLK 6		
CH 34	Byte 5 Bit 1	MSN-9 Bit 2	C29	VIO 6		
CH 33	Byte 5 Bit 0	MSN-9 Bit 1	A29	GRY 6		
Signal	Function		Pin	Color		
EDR#1	EDR #1 Input		A7	YEL 2		
INH#1	Inhibit Signal Output #1		A8	BLU 2		
EDR#2	EDR #2 Input		C7	ORG 2		
INH#2	Inhibit Signal Output #2		C8	GRN 2		
Stat A	Status A/EDR #3 Input		C6	BRN 2		
Stat B	Status B/EDR #4 Input		A6	RED 2		
Trigger	Trigger Output		C11	BRN 3		
Remote	Remote State Output		A11	RED 3		
Reset	Reset Output		C12	ORG 3		
Strobe	Data Strobe Output		A12	YEL 3		
Stable	Digital I/O Configured Output		C10	WHT 2		
-ExtRst	External Reset Input		A10	BLK 2		
Vcc	+5 Vdc Input		A1	RED 1		
			C1	BRN 1		
Gnd	Signal Ground		A2	YEL 1		
			C2	ORG 1		
-RDY	RDY LED Output		C4	VIO 1		
-TALK	TALK LED Output		C3	GRN 1		
-LSTN	LISTEN LED Output		A5	BLK 1		
-SRQ	SRQ LED Output		A4	GRY 1		
-ERR	ERR LED Output		A3	BLU 1		
-LAN	LAN LED Output		C9	VIO 2		
-ACT	LAN ACTIVITY LED Output		A9	GRY 2		
n/c	not used		C5	WHT 1		

* Colors shown for 4803 Open End Flat Ribon Cable, P/N 112343.

TABLE 2-5 8013 DIGITAL SIGNAL-PIN ASSIGNMENTS

Signal	Signal Weighting		Pin	User Signals	
	Binary	BCD/HEX		Pin	Signal
CH 4	Byte 1 Bit 7	MSN Bit 8	130		
CH 3	Byte 1 Bit 6	MSN Bit 4	131		
CH 2	Byte 1 Bit 5	MSN Bit 2	132		
CH 1	Byte 1 Bit 4	MSN Bit 1	133		
CH 8	Byte 1 Bit 3	MSN-1 Bit 8	126		
CH 7	Byte 1 Bit 2	MSN-1 Bit 4	127		
CH 6	Byte 1 Bit 1	MSN-1 Bit 2	128		
CH 5	Byte 1 Bit 0	MSN-1 Bit 1	129		
CH 12	Byte 2 Bit 7	MSN-2 Bit 8	122		
CH 11	Byte 2 Bit 6	MSN-2 Bit 4	123		
CH 10	Byte 2 Bit 5	MSN-2 Bit 2	124		
CH 9	Byte 2 Bit 4	MSN-2 Bit 1	125		
CH 16	Byte 2 Bit 3	MSN-3 Bit 8	118		
CH 15	Byte 2 Bit 2	MSN-3 Bit 4	119		
CH 14	Byte 2 Bit 1	MSN-3 Bit 2	120		
CH 13	Byte 2 Bit 0	MSN-3 Bit 1	121		
CH 20	Byte 3 Bit 7	MSN-4 Bit 8	114		
CH 19	Byte 3 Bit 6	MSN-4 Bit 4	115		
CH 18	Byte 3 Bit 5	MSN-4 Bit 2	116		
CH 17	Byte 3 Bit 4	MSN-4 Bit 1	117		
CH 24	Byte 3 Bit 3	MSN-5 Bit 8	110		
CH 23	Byte 3 Bit 2	MSN-5 Bit 4	111		
CH 22	Byte 3 Bit 1	MSN-5 Bit 2	112		
CH 21	Byte 3 Bit 0	MSN-5 Bit 1	113		
CH 28	Byte 4-bit 7	MSN-6 Bit 8	106		
CH 27	Byte 4-bit 6	MSN-6 Bit 4	107		
CH 26	Byte 4-bit 5	MSN-6 Bit 2	108		
CH 25	Byte 4-bit 4	MSN-6 Bit 1	109		
CH 32	Byte 4-bit 3	MSN-7 Bit 8	102		
CH 31	Byte 4-bit 2	MSN-7 Bit 4	103		
CH 30	Byte 4-bit 1	MSN-7 Bit 2	104		
CH 29	Byte 4-bit 0	MSN-7 Bit 1	105		



Indicates signals also used as the Questionable Register inputs

TABLE 2-5 8013 DIGITAL SIGNAL-PIN ASSIGNMENTS

Signal	Signal Weighting		Pin	User Signals	
	Binary	BCD/HEX		Pin	Signal
CH 36	Byte 5 Bit 7	MSN-8 Bit 8	96		
CH 35	Byte 5 Bit 6	MSN-8 Bit 4	97		
CH 34	Byte 5 Bit 5	MSN-8 Bit 2	98		
CH 33	Byte 5 Bit 4	MSN-8 Bit 1	99		
CH 40	Byte 5 Bit 3	MSN-9 Bit 8	92		
CH 39	Byte 5 Bit 2	MSN-9 Bit 4	93		
CH 38	Byte 5 Bit 1	MSN-9 Bit 2	94		
CH 37	Byte 5 Bit 0	MSN-9 Bit 1	95		
CH 44	Byte 6 Bit 7	MSN-10 Bit 8	88		
CH 43	Byte 6 Bit 6	MSN-10 Bit 4	89		
CH 42	Byte 6 Bit 5	MSN-10 Bit 2	90		
CH 41	Byte 6 Bit 4	MSN-10 Bit 1	91		
CH 48	Byte 6 Bit 3	MSN-11 Bit 8	84		
CH 47	Byte 6 Bit 2	MSN-11 Bit 4	85		
CH 46	Byte 6 Bit 1	MSN-11 Bit 2	86		
CH 45	Byte 6 Bit 0	MSN-11 Bit 1	87		
CH 52	Byte 7 Bit 7	MSN-12 Bit 8	80		
CH 51	Byte 7 Bit 6	MSN-12 Bit 4	81		
CH 50	Byte 7 Bit 5	MSN-12 Bit 2	82		
CH 49	Byte 7 Bit 4	MSN-12 Bit 1	83		
CH 56	Byte 7 Bit 3	MSN-13 Bit 8	76		
CH 55	Byte 7 Bit 2	MSN-13 Bit 4	77		
CH 54	Byte 7 Bit 1	MSN-13 Bit 2	78		
CH 53	Byte 7 Bit 0	MSN-13 Bit 1	79		
CH 60	Byte 8 Bit 7	MSN-14 Bit 8	72		
CH 59	Byte 8 Bit 6	MSN-14 Bit 4	73		
CH 58	Byte 8 Bit 5	MSN-14 Bit 2	74		
CH 57	Byte 8 Bit 4	MSN-14 Bit 1	75		
CH 64	Byte 8 Bit 3	MSN-15 Bit 8	68		
CH 63	Byte 8 Bit 2	MSN-15 Bit 4	69		
CH 62	Byte 8 Bit 1	MSN-15 Bit 2	70		
CH 61	Byte 8 Bit 0	MSN-15 Bit 1	71		

TABLE 2-5 8013 DIGITAL SIGNAL-PIN ASSIGNMENTS

Signal	Signal Weighting		Pin	User Signals	
	Binary	BCD/HEX		Pin	Signal
CH 68	Byte 9 Bit 7	MSN-16 Bit 8	64		
CH 67	Byte 9 Bit 6	MSN-16 Bit 4	65		
CH 66	Byte 9 Bit 5	MSN-16 Bit 2	66		
CH 65	Byte 9 Bit 4	MSN-16 Bit 1	67		
CH 72	Byte 9 Bit 3	MSN-17 Bit 8	60		
CH 71	Byte 9 Bit 2	MSN-17 Bit 4	61		
CH 70	Byte 9 Bit 1	MSN-17 Bit 2	62		
CH 69	Byte 9 Bit 0	MSN-17 Bit 1	63		
CH 76	Byte 10 Bit 7	MSN-18 Bit 8	56		
CH 75	Byte 10 Bit 6	MSN-18 Bit 4	57		
CH 74	Byte 10 Bit 5	MSN-18 Bit 2	58		
CH 73	Byte 10 Bit 4	MSN-18 Bit 1	59		
CH 80	Byte 10 Bit 3	MSN-19 Bit 8	52		
CH 79	Byte 10 Bit 2	MSN-19 Bit 4	53		
CH 78	Byte 10 Bit 1	MSN-19 Bit 2	54		
CH 77	Byte 10 Bit 0	MSN-19 Bit 1	55		
CH 84	Byte 11 Bit 7	MSN-20 Bit 8	46		
CH 83	Byte 11 Bit 6	MSN-20 Bit 4	47		
CH 82	Byte 11 Bit 5	MSN-20 Bit 2	48		
CH 81	Byte 11 Bit 4	MSN-20 Bit 1	49		
CH 88	Byte 11 Bit 3	MSN-21 Bit 8	42		
CH 87	Byte 11 Bit 2	MSN-21 Bit 4	43		
CH 86	Byte 11 Bit 1	MSN-21 Bit 2	44		
CH 85	Byte 11 Bit 0	MSN-21 Bit 1	45		
CH 92	Byte 12 Bit 7	MSN-22 Bit 8	38		
CH 91	Byte 12 Bit 6	MSN-22 Bit 4	39		
CH 90	Byte 12 Bit 5	MSN-22 Bit 2	40		
CH 89	Byte 12 Bit 4	MSN-22 Bit 1	41		
CH 96	Byte 12 Bit 3	MSN-23 Bit 8	34		
CH 95	Byte 12 Bit 2	MSN-23 Bit 4	35		
CH 94	Byte 12 Bit 1	MSN-23 Bit 2	36		
CH 93	Byte 12 Bit 0	MSN-23 Bit 1	37		

TABLE 2-5 8013 DIGITAL SIGNAL-PIN ASSIGNMENTS

Signal	Signal Weighting		Pin	User Signals	
	Binary	BCD/HEX		Pin	Signal
CH 100	Byte 13 Bit 7	MSN-24 Bit 8	30		
CH 99	Byte 13 Bit 6	MSN-24 Bit 4	31		
CH 98	Byte 13 Bit 5	MSN-24 Bit 2	32		
CH 97	Byte 13 Bit 4	MSN-24 Bit 1	33		
CH 104	Byte 13 Bit 3	MSN-25 Bit 8	26		
CH 103	Byte 13 Bit 2	MSN-25 Bit 4	27		
CH 102	Byte 13 Bit 1	MSN-25 Bit 2	28		
CH 101	Byte 13 Bit 0	MSN-25 Bit 1	29		
CH 108	Byte 14 Bit 7	MSN-26 Bit 8	22		
CH 107	Byte 14 Bit 6	MSN-26 Bit 4	23		
CH 106	Byte 14 Bit 5	MSN-26 Bit 2	24		
CH 105	Byte 14 Bit 4	MSN-26 Bit 1	25		
CH 112	Byte 14 Bit 3	MSN-27 Bit 8	18		
CH 111	Byte 14 Bit 2	MSN-27 Bit 4	19		
CH 110	Byte 14 Bit 1	MSN-27 Bit 2	20		
CH 109	Byte 14 Bit 0	MSN-27 Bit 1	21		
CH 116	Byte 15 Bit 7	MSN-28 Bit 8	14		
CH 115	Byte 15 Bit 6	MSN-28 Bit 4	15		
CH 114	Byte 15 Bit 5	MSN-28 Bit 2	16		
CH 113	Byte 15 Bit 4	MSN-28 Bit 1	17		
CH 120	Byte 15 Bit 3	MSN-29 Bit 8	10		
CH 119	Byte 15 Bit 2	MSN-29 Bit 4	11		
CH 118	Byte 15 Bit 1	MSN-29 Bit 2	12		
CH 117	Byte 15 Bit 0	MSN-29 Bit 1	13		
CH 124	Byte 16 Bit 7	MSN-30 Bit 8	6		
CH 123	Byte 16 Bit 6	MSN-30 Bit 4	7		
CH 122	Byte 16 Bit 5	MSN-30 Bit 2	8		
CH 121	Byte 16 Bit 4	MSN-30 Bit 1	9		
CH 128	Byte 16 Bit 3	MSN-31 Bit 8	2		
CH 127	Byte 16 Bit 2	MSN-31 Bit 4	3		
CH 126	Byte 16 Bit 1	MSN-31 Bit 2	4		
CH 125	Byte 16 Bit 0	MSN-31 Bit 1	5		

TABLE 2-5 8013 DIGITAL SIGNAL-PIN ASSIGNMENTS

Signal	Function	Pin	User Signals	
			Pin	Signal
EDR#1	EDR #1 Input	137		
INH#1	Inhibit Signal Output #1	139		
EDR#2	EDR #2 Input or /LAN LED Output	136 *		
INH#2	Inhibit Signal Output #2 or /ACT LED Output	138 *		
Stat A	Status A Input	135		
Stat B	Status B Input	134		
Trigger	Trigger Output	148		
Remote	Remote State Output	149		
Reset	Reset/Clear Outputs	150		
Strobe	Data Strobe Output	147		
Stable	Digital I/O Configured	146		
-ExtRst	External Reset Input	140		
Vcc	+5 Vdc Input	100 50		
Gnd	Signal Ground	101 51 1		
-RDY	RDY LED Output	141		
-TALK	TALK LED Output	142		
-LSTN	LISTEN LED Output	143		
-SRQ	SRQ LED Output	144		
-ERR	ERR LED Output	145		

Notes: * Signal selection made by jumpers on the 8013 PCB.

TABLE 2-6 I/O CONFIGURATION CHART

Parameter	Function	New Setting
Talk String		
:INPut	Sets number of Talk bytes	
:POLarity	Sets Input data polarity	
:HANDshake	Enables Input Handshaking	
:EDR	Sets input polarity of edge	
:INH	Sets inhibit output polarity	
:TALK	Selects Input String Format	
:TRANSLation	Sets input conversion table if needed	
:EOM	Sets End-of-message string	
Listen String		
:OUTput	Sets number of Listen bytes	
:POLarity	Sets Output polarity	
:HANDshake	Enables Output Handshaking	
:STRobe	Sets Output Strobe polarity	
:LISTen	Sets Output Format	
:ASTATus	Sets Input Polarity	
:BSTATus	Sets Input Polarity	
:REMOte	Sets Output Polarity	
:RESet	Sets Output Polarity	
:TRIGger	Sets Output Polarity	
Byte Transfer		
:POLarity	Sets Byte 1 polarity	
:POLarity	Sets Byte 2 polarity	
:POLarity	Sets Byte 3 polarity	
:POLarity	Sets Byte 4 polarity	
:POLarity	Sets Byte 5 polarity	
:POLarity	Sets Byte 6 polarity	
:POLarity	Sets Byte 7 polarity	
:POLarity	Sets Byte 8 polarity	
:POLarity	Sets Byte 9 polarity	
:POLarity	Sets Byte 10 polarity	
:POLarity	Sets Byte 11 polarity	
:POLarity	Sets Byte 12 polarity	
:POLarity	Sets Byte 13 polarity	
:POLarity	Sets Byte 14 polarity	
:POLarity	Sets Byte 15 polarity	
:POLarity	Sets Byte 16 polarity	

2.9 DIGITAL I/O DESIGN GUIDE

The following instructions guide the user through the design of an interface cable between the board's Digital I/O connector and an external device. These steps make a drawing that can be used to fabricate the cable and a table with the configuration settings.

- 1 Make a copy of the appropriate Signal-Pin Table (Table 2-4 or 2-5) and Table 2-6 as your worksheets. Use these copies when directed to record signals, pin numbers etc.
- 2. Make a rough determination of the number of output and input signals from the external device, not exceeding the number of available I/O lines.
- 3. If any of the signal lines are to be monitored, they should be assigned to the first two bytes, CH1-CH15. Line CH16 is not monitored. The monitored lines may also be read by a byte or bit command.
- 4. If input data is to be transferred by a string command, assign the external devices to the data lines starting with first free byte. For BCD/HEX coding, start with the lowest free nibble. i.e. If bytes 1 and 2 are used start with MSN-4 in Byte3. Assign the signals to the bit weights so that the BCD or HEX values are inputted correctly. For byte wise coding, Bit 0 is the LSB. Any unused lines must be tied to logic '0' potential. Note that logic '0' is the opposite level of the true level set by the polarity command. Typical assignments for a small panel meter are:

I/O Signals:	MSN,	MSN-1, MSN-2, MSN-3
Inputs:	Sign/overrange, MSD, MSD-1, .LSD	
Example:	+	1 9 9

Record the signal names and pin numbers in your copy of Table 2-4 or 2-5.

Similar bits such as Sign and Overrange can be combined into one four line group to conserve data lines. Use jumpers to +5 Vdc and to ground to convert a single line signal such as 'Polarity' into a four bit value like hex 'A" or hex 'B'. i.e. This looks like:

101P where P is the polarity signal

Use the format command to change hex 'A' and hex 'B' into '+' and '-' signs when inputting BCD data.

Refer to the sample cable in Section 2.11 for jumper examples. Tie all unused data input lines to the logic '0' potential. If the inputs are low true, leave the unused signals open. The board's internal pullup resistors will pull the signals to +5 V. Record the signal names and pin numbers in your copy of Table 2-4 or 2-5.

If handshaking is enabled, record the external signal (and pin number) that will be connected to the EDR input in Table 2-4 or 2-5. If the Inhibit output is being used, record the signal name and pin number in the Table. In Table 2-6 record the number of input bytes used, the data polarity, Talk handshaking enabled status, EDR polarity and Inhibit polarity.

5. If output data is being transferred by a string command, the output data lines should be assigned next. Start with the next available byte in Table 2-4. If the string data is HEX or BCD characters, start with the lowest available nibble in the byte. If byte 3 is the next free byte, MSN-4 is the lowest available nibble.

i.e. For an output string of "1234", the "0001" code will appear on lines CH20-CH17 and the "0100" code will be on lines CH32-CH29.

Assign the signals to the proper bit weights so that the numbers come out correctly for the external device. For binary data use the binary bit weights shown for the byte. Record the device signal names and pin numbers in Table 2-4 or 2-5. Leave the unused output data lines open. Record the bytes used and their signal polarity in Table 2-6.

If the Output Data Strobe will be used, record the signal and pin number in Table 2-4 or 2-5 and the strobe polarity in Table 2-6.

If Listen Handshaking is to be enabled, connect a signal to Status A in Table 2-4 or 2-5. In Table 2-6 record the Listen Handshake enabled and the polarity of the Status Input when data can be transferred.

6. If any data will be transferred with Byte (Port) commands, their signals should be connected now. Assign the signals to the remaining data lines. All signals on a byte must be in the same direction but they can have individual bit polarities. Record the device signal names and pin numbers in Table 2-4 or 2-5 and the byte's polarity in Table 2-6.

7. If any control signals are being used, they should be connected at this time. Record the signal names and pin numbers in Table 2-4 or 2-5 and the signal polarities in Table 2-6.
8. Use Table 2-6 as the worksheet to generate the configuration commands for the board. Refer to Section 3 for programming instructions.

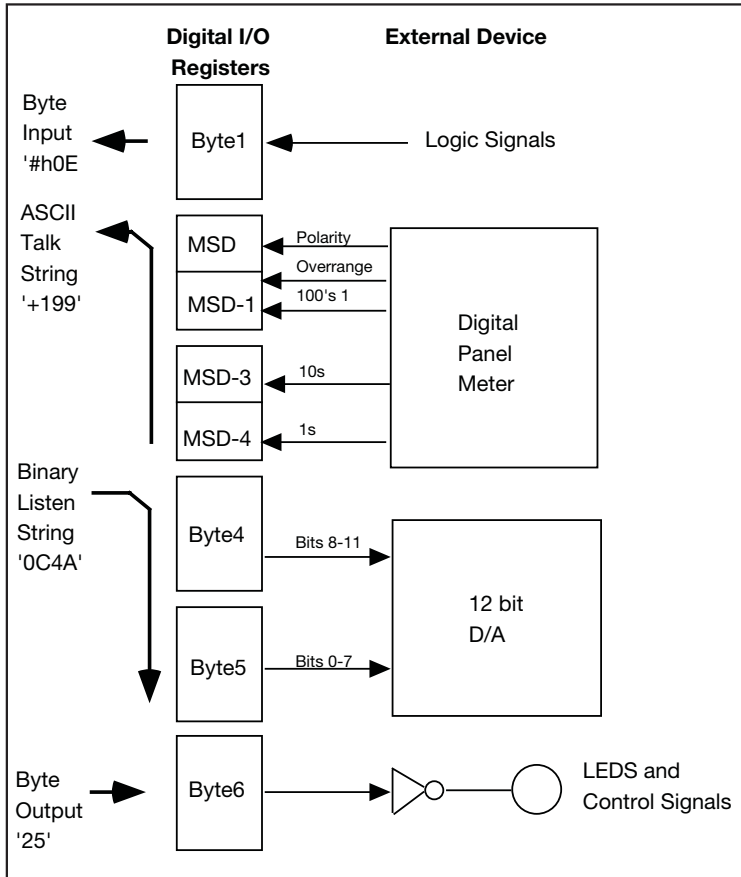


Figure 2-15 Example Digital I/O Connections

2

2.10 DIGITAL I/O EXAMPLE

Figure 2-15 shows how a 8003 or a 8013 may be used to control an external device and input data. In the example below, the external device has a small Digital Panel Meter (DPM), a D/A converter, some control inputs, a four bit counter, two status outputs and three LEDs.

Table 2-7 shows the signal connections for this example. The first byte is used to input the Counter and Status signals so that they could later be monitored by the questionable register if the need arose. The byte is directly queried with the **PORTn?** query. The response format used is the one set by the **FORM: TALK** command.

Bytes 2 and 3 are used to input data from a 2 1/2 Digital Panel Meter (DPM) data as a four character ASCII string with positive data polarity. Logic '0' signal jumpers will go to ground. Logic '1' inputs are left open. The DPM's 'busy' signal is used to pulse the 8013's EDR input. The string could be read either transparently or with the **PORT? (@2,3)** query. Typical in-range responses are -199 to + 199. The Overrange signal is wired to bit 8 of the 100's digit so Overrange responses are -999 or +999. The polarity digit is created by wiring the Plus signal to bit 1. Bits 8, 4 and 2 are fixed with jumpers to be 101. The resulting code is 1011 for plus, 1010 for negative. In the 8013's Talk Conversion Table, character 10 is set to an ASCII '-' and character 11 is set to an ASCII '+'.

Bytes 4 and 5 are used to output 12 bits to a D/A converter. The Strobe line is used to load the data into the D/A's internal latch. Data transfer is done with a string of four HEX characters, two characters per byte. Data transfer can be as a transparent data string or can be done with the **SOURCE:DATA: VALUE** command. In Figure 2-14, Byte 4 is sent as an '0C' and Byte 5 is sent as an '4A'. The resultant output pattern is '0C4A'

In this example, the upper bits of Byte 4 are not used to drive the D/A so they can be used to drive other signals such as LEDs and control lines. The user's program would combine the signal control bits and the D/A bits together to form the output string before it is sent to the 8013. However, you can also use an unused byte to output the LED and control lines.

Table 2-8 shows the completed Configuration Table for this example.

TABLE 2-7 8013 EXAMPLE SIGNAL-PIN ASSIGNMENTS

Signal	Signal Weighting		Pin	User Signals	
	Binary	BCD/HEX		Pin	Signal
CH 4	Byte 1 Bit 7	MSN Bit 8	130	gnd	= 0
CH 3	Byte 1 Bit 6	MSN Bit 4	131	gnd	= 0
CH 2	Byte 1 Bit 5	MSN Bit 2	132	25	Status bit 2
CH 1	Byte 1 Bit 4	MSN Bit 1	133	26	Status bit 1
CH 8	Byte 1 Bit 3	MSN-1 Bit 8	126	27	Counter bit 8
CH 7	Byte 1 Bit 2	MSN-1 Bit 4	127	28	Counter bit 4
CH 6	Byte 1 Bit 1	MSN-1 Bit 2	128	29	Counter bit
CH 5	Byte 1 Bit 0	MSN-1 Bit 1	129	30	Counter bit 1
CH 12	Byte 2 Bit 7	MSN-2 Bit 8	122	+5	= 1
CH 11	Byte 2 Bit 6	MSN-2 Bit 4	123	gnd	= 0
CH 10	Byte 2 Bit 5	MSN-2 Bit 2	124	+5	=
CH 9	Byte 2 Bit 4	MSN-2 Bit 1	125	1	Plus signal = B
CH 16	Byte 2 Bit 3	MSN-3 Bit 8	118	2	Overrange
CH 15	Byte 2 Bit 2	MSN-3 Bit 4	119	gnd	= 0
CH 14	Byte 2 Bit 1	MSN-3 Bit 2	120	gnd	= 0
CH 13	Byte 2 Bit 0	MSN-3 Bit 1	121	3	100's 1
CH 20	Byte 3 Bit 7	MSN-4 Bit 8	114	4	10's 8
CH 19	Byte 3 Bit 6	MSN-4 Bit 4	115	5	10's 4
CH 18	Byte 3 Bit 5	MSN-4 Bit 2	116	6	10's 2
CH 17	Byte 3 Bit 4	MSN-4 Bit 1	117	7	10's 1
CH 24	Byte 3 Bit 3	MSN-5 Bit 8	110	8	1's 8
CH 23	Byte 3 Bit 2	MSN-5 Bit 4	111	9	1's 4
CH 22	Byte 3 Bit 1	MSN-5 Bit 2	112	10	1's 2
CH 21	Byte 3 Bit 0	MSN-5 Bit 1	113	11	1's 1
CH 28	Byte 4-bit 7	MSN-6 Bit 8	106	47	Control out 2
CH 27	Byte 4-bit 6	MSN-6 Bit 4	107	46	Control out 1
CH 26	Byte 4-bit 5	MSN-6 Bit 2	108	45	LED #2
CH 25	Byte 4-bit 4	MSN-6 Bit 1	109	44	LED #1
CH 32	Byte 4-bit 3	MSN-7 Bit 8	102	15	D/A bit 11
CH 31	Byte 4-bit 2	MSN-7 Bit 4	103	16	D/A bit 10
CH 30	Byte 4-bit 1	MSN-7 Bit 2	104	17	D/A bit 9
CH 29	Byte 4-bit 0	MSN-7 Bit 1	105	18	D/A bit 8

 Indicates signals also used as the Questionable Register inputs

TABLE 2-7 8013 EXAMPLE SIGNAL-PIN ASSIGNMENTS

Signal	Signal Weighting		Pin	User Signals	
	Binary	BCD/HEX		Pin	Signal
CH 36	Byte 5 Bit 7	MSN-8 Bit 8	96	19	D/A bit 7
CH 35	Byte 5 Bit 6	MSN-8 Bit 4	97	20	D/A bit 6
CH 34	Byte 5 Bit 5	MSN-8 Bit 2	98	21	D/A bit 5
CH 33	Byte 5 Bit 4	MSN-8 Bit 1	99	22	D/A bit 4
CH 40	Byte 5 Bit 3	MSN-9 Bit 8	92	23	D/A bit 3
CH 39	Byte 5 Bit 2	MSN-9 Bit 4	93	24	D/A bit 2
CH 38	Byte 5 Bit 1	MSN-9 Bit 2	94	25	D/A bit 1
CH 37	Byte 5 Bit 0	MSN-9 Bit 1	95	25	D/A bit 1
Signal	Function		Pin		
EDR#1	EDR #1 Input		137	12	DPM Busy
INH#1	Inhibit Signal Output #1		139		DPM Hold
EDR#2	EDR #2 Input		136		-
INH#2	Inhibit Signal Output #2		138		-
Stat A	Status A Input		135	50	Ext device +5
Stat B	Status B Input		134		- (open)
Trigger	Trigger Output		148	13	DPM Trigger
Remote	Remote State Output		149		-
Reset	Reset/Clear Outputs		150		-
Strobe	Data Strobe Output		147	27	D/A load pulse
-ExtRst	External Reset Input		140		-
Vcc	+5 Vdc Input		100 50		+5 Vdc
Gnd	Signal Ground		101 51 1	1	Ground

TABLE 2-8 EXAMPLE CONFIGURATION SETTINGS

Parameter	Function	New Setting
Talk String		
:INPut	Sets Talk bytes	2,3
:POLarity	Sets Input data polarity	1
:HANDshake	Enables Input Handshaking	ON
:EDR	Sets input polarity of edge	0
:INH	Sets inhibit output polarity	0
:TALK	Selects Input String Format	TABLE
:TRANSLation	Sets input conversion table if needed	0-9-+ ,E and space
:EOM	Sets End-of-message string	10 (LF)
Listen String		
:OUTput	Sets Listen bytes	4,5
:POLarity	Sets Output polarity	0
:HANDshake	Enables Output Handshaking	OFF
:STRobe	Sets Output Strobe polarity	0
:LISTen	Sets Output Format	HEX
:ASTATus	Sets Input Polarity	1
:BSTATus	Sets Input Polarity	0
:REMote	Sets Output Polarity	0
:RESet	Sets Output Polarity	0
:TRIGger	Sets Output Polarity	-
Byte Transfer		
:POLarity	Sets Byte 1 polarity	#h FF
:POLarity	Sets Byte 2 polarity	
:POLarity	Sets Byte 3 polarity	
:POLarity	Sets Byte 4 polarity	
:POLarity	Sets Byte 5 polarity	

This page left intentionally blank

Programming Instructions

3.1 INTRODUCTION

This section describes the operation of the 8003 and 8013 Ethernet to Parallel Interface Boards, Status Reporting Structure, Commands and Programming Guidelines. Because programming the 80xx series boards over a network is different from traditional GPIB programming, a new user should read Appendix A1 and A1 to familiarize himself with the VXI-11 concepts before programming the board. When the client application is linked to an 80xx board, the commands and operation of the Digital Interface is substantially identical to that of the equivalent GPIB board's Digital Interface. The functional description of the 8003 applies to the 8013 series boards unless otherwise stated.

3.2 OPERATION

3.2.1 VXI-11 Operation

The 8003 is a server in the client-server relationship and provides a VXI-11 service. The core channel link to interface *inst0* in the 8003 is used for all commands and responses including 488.2, SCPI and Digital I/O commands. The 8003 has an additional interface personality, *inst1*, that is used to transparently transfer digital I/O data as strings of data bytes. The *inst1* link also accepts the 488.1 like commands such as *device_clear* and *device_trigger*.

The VISA Resource Strings are:

TCPIP:: <i>ip</i> ::inst0::INSTR	'for commands and data transfer
TCPIP:: <i>ip</i> ::inst1::INSTR	'for transparent data transfer

Note: where *ip* is the 8003's ip address

3.2.2 Digital Interface Operation

The 8003's digital interface is user configurable as inputs or outputs in eight bit bytes. Interface configuration and data transfer is done by commands to interface *inst0*. Data transfer is by a combination of direct byte (port) access commands, by bit set/reset commands, by pulse commands or as strings of data characters for multiple byte width transfers. The digital interface configuration, data formats and transfer protocol can be saved in Flash memory and is automatically recalled at power turn-on or when the unit is reset. The 8003 has the additional capability of being able to transparently transfer data as strings of data characters by using a second link to interface *inst1*.

The 8003 is an IEEE-488.2 compatible device and has an expanded Status Reporting Structure. The Status Structure lets the user monitor up to fifteen of the digital I/O lines and two status inputs for changes and generate an Service Request when the enabled changes occur. Service Requests can also be generated when external data is ready, when the output data handshake is not received or when a command error occurs.

The 8003 uses SCPI commands and device specific Short Form commands to control its digital interface. SCPI (Standard Commands for Programmable Instruments) commands are an industry standard and generate a self documenting program listing. Most of the 8003's SCPI commands or queries have a corresponding short form (1 or 2 letter) command for easy programming. Where possible, the short form commands are the same as those used in ICS's 48x3 series GPIB-to-Parallel Interfaces.

The core channel link to interface *inst0* in the 8003 is used for all 488.1, 488.2, SCPI and short form commands, for all queries, all responses and for transferring data as a command parameter. A second link to interface *inst1* is used to bypass the 8003's SCPI parser and transfers data directly to or from the configured output or input bytes.

TABLE 3-1 8003/8013 LINKS AND DATA TRANSFER MODES

Link	Interface Name	Data Transfer and Command Capabilities
Core	inst0	Transfers all 488.1, 488.2, Setup and data as command parameters. The VISA Resource String is: TCP/IP:: <i>ip</i> ::inst0::INSTR where <i>ip</i> is the ip address
Transparent	inst1	Transparent data transfer to configured output bytes and from configured input bytes. The VISA Resource string is: TCP/IP:: <i>ip</i> ::inst1::INSTR

3.2.3 Data Transfer Methods

The 8003 and 8013 have three methods for transferring data to and from the digital interface:

1. Bit commands like ROUTe set/reset/pulse or read an individual bit. Byte direction is automatically set when the command is first executed and should not be set by the configuration commands. Bit/byte polarity is set by the SOURce or SENSE command branches. A PULSE function momentarily sets the output line true.
2. Byte commands like SENSE:DATA:PORTn? and SOURce:DATA:PORTn read or write the specified byte. Byte direction is automatically set when the command is first executed and should not be set by the configuration commands. Bit/byte polarity is set by the SOURce or SENSE command branches.
3. String commands like SENSE:DATA? or SOURce:DATA read or write to one or more bytes. These bytes must be pre-configured as inputs or outputs by the CONFigure and FORMat commands.

3.2.3.1 Inputting Data

The user can directly query individual bits or bytes. The first query sets the byte's data direction and subsequent queries return valid data. String commands read data from bytes configured as inputs. If handshaking is disabled, the unit will simply input the digital data when addressed to talk or when queried, convert it into the selected format, append the EOM character and output the data on network. If talk handshaking is enabled, data transfer will occur if the EDR (External Data Ready) signal is received and the unit has been addressed to talk. The external device must hold the digital inputs stable until the 8003 drops its Inhibit line. If queried by a command and the EDR signal is not received, the 8003 will report an Execution Error. The input data format is set by the FORMAT:TALK command. The input data may be formatted into HEX characters, as user entered ASCII characters, or as a series of decimal values that represent the binary sum of the logic '1' bits in each input byte

The IEEE-488.2 Status Reporting Structure is used to generate *device_intr_srq* messages (SRQs). These messages can be generated when data is available or if input lines change state. If talk handshaking is enabled the EDR bit in the Event Status Register and Operation Condition Register will be set when data is available. The first fifteen digital I/O lines can be monitored for changes

with the Questionable Transition Registers if the lines are assigned as inputs. Detected input signal changes set corresponding bits in the Event register. If the ESR and/or Questionable Register enable bits are set, the register summary bits will appear in the Status Byte Register. If the summary bits are enabled, the Service Request bit will be set and generate a *device_intr_srq* message (SRQs). The *device_intr_srq* messages are sent to the Application over the reverse Interrupt channel. The user can then serial poll or query the Status Byte to determine the cause of the Service Request and read the input data by a command or through the Questionable register. Refer to Application Bulletin AB80-4 on handling RPC Interrupts.

3.2.3.2 Outputting Data

To output data, the user can write to any byte or set a bit in the byte. The byte direction automatically becomes output. Direct data transfer to a specific port or ports is performed with the PORTn commands. Data transfer can be a decimal or hex value. Data polarity can be set on a bit-by-bit basis.

Output bits can be manipulated with the bit or pulse commands. These commands let the user set, reset or pulse bits directly without having to maintain a copy of the byte in the application program. The logical '1' state is determined by the polarity setting command for that bit.

The user can also configure selected bytes as outputs for string data transfers and set their data polarity and listen format. If handshaking is disabled, the unit will accept character strings from the network, convert the characters into digital bits, set the output latches and generate the data strobe pulse. Listen data can be hexadecimal characters (0-9, A-F), BCD characters (0-9 and :<=>?), or as decimal values that represents the binary sum of the bits in each byte to be outputted as logic true levels. If listen handshaking is enabled, data transfer will only occur if the Status A input signal is in its logic true state. Data overrun is reported as an Execution Error.

Multiple control, status and control lines are provided for interfacing with external digital devices. Output control signals include Trigger, Reset, Remote, Clear and Data Strobe lines. Inputs include two general purpose status lines that are inputted via the Operation Condition register in the 488.2 Status Reporting Structure. The user can set the Status Reporting Structure to monitor these lines and generate a SRQ when they change state or periodically poll them by querying the Operation Condition register.

3.2.3.3 Configuring the Digital Interface

At power turn on, the 8003 restores the last configuration and output values saved by the *SAV 0 command as the current configuration. The digital I/O lines are tristated and pulled high until the configuration is restored and then the Stable line is set high. The recalled settings are used as the current configuration until a command is sent to the board with a new value which then becomes part of the current configuration. When power is turned off, the current configuration values are lost.

To configure the Digital Interface, the user should send the board commands with the desired power on port polarities and output values. If string commands are to be used, set the appropriate port directions with the CONFigure commands. Also use the CONFigure commands to set up the handshake and Status lines. When the Digital Interface has been set to its correct power-on values, use the *SAV 0 command to save the current configuration in the Flash memory.

For OEM applications, use the CALibrate commands to enter a custom IDN message. Use the lock command to hide the configuration setting commands and prevent changes to the configuration parameters. The Status Register enable settings are not affected by the lock function.

Recommendation - Create a setup program to automatically setup your boards and provide documentation listing your configuration settings to make it easy for someone to setup future replacement boards.

3.2.4 Raw Socket Operation

Raw Socket is a serial communication method and is similar to the VXI-11 communication with the *inst0* personality. Raw Socket can only execute the commands listed in Tables 3-3, 3-4 and 3-5. There is no *inst1* communication with Raw Socket nor can Raw Socket invoke any action that is not done with the commands in Tables 3-3, 3-4 and 3-5.

If multiple VXI-11 and Raw Socket clients are accessing the interface at the same time, it is possible for the interface to send responses to the wrong client. While multiple Raw Socket connections work, the recommendation is to only use Raw Socket on a back-to-back connection with the controlling computer to avoid the chance of another client trying to communicate with the device. If the Raw Socket connection is lost, the client can immediately make a new connection if a socket is available. Raw Sockets can take up to 40 seconds to close. Too many connections in a short period of time can use up all of the available sockets and result in a 'connector refused' response from the unit.

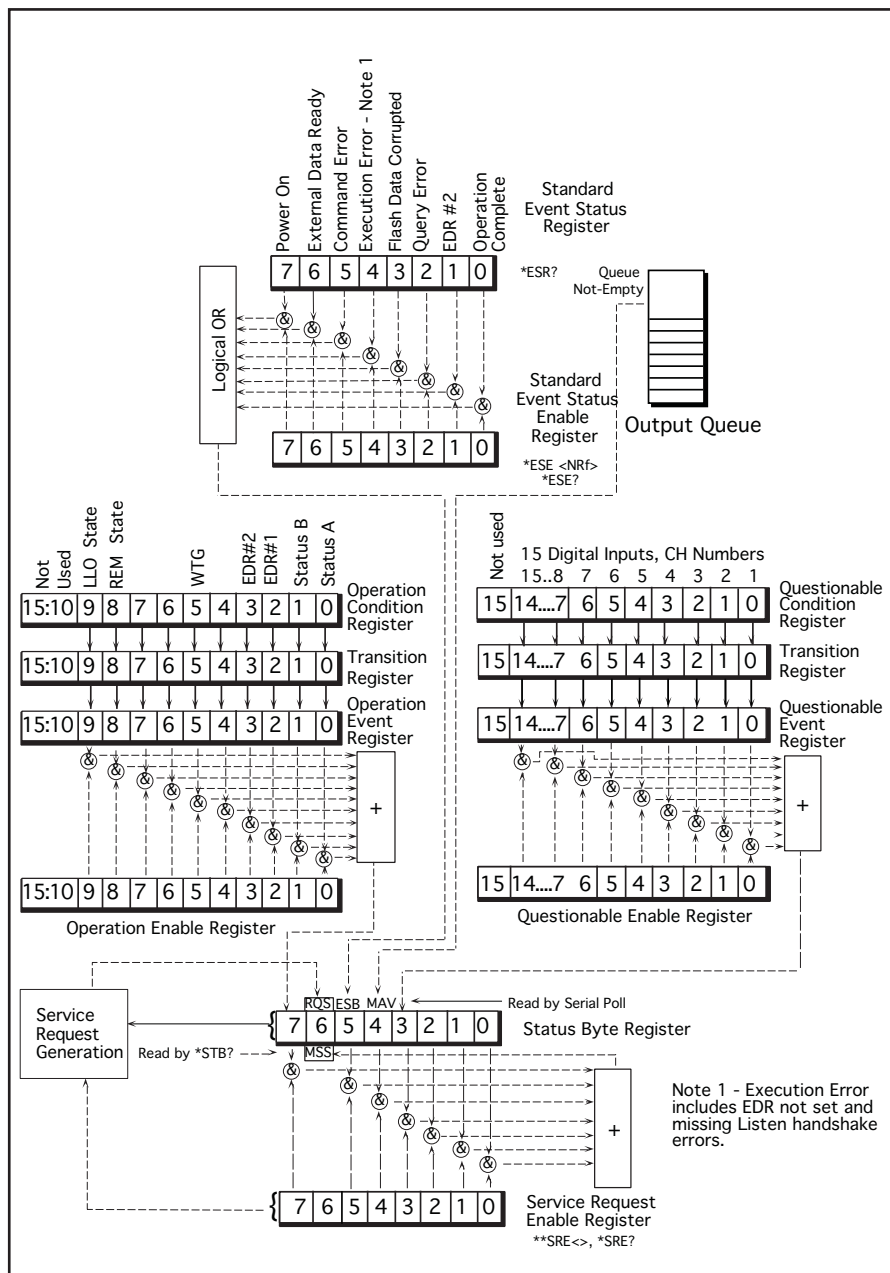


Figure 3-1 Status Reporting Structure

3.3 488.2 STATUS REPORTING STRUCTURE

All units have the expanded IEEE-488.2 status reporting structure shown in Figure 3-1. The expanded status reporting structure conforms to the SCPI 1994.0 Specification and builds on the IEEE-488.2 Standard Status Reporting Structure by adding the Questionable and Operation registers. The Event and Status registers are controlled and queried with the IEEE-488.2 common commands. The Status Byte Register may also be read by serial polling the board. The Questionable and Operation registers are controlled and queried with SCPI commands.

Instead of asserting the GPIB SRQ line, VXI-11.3 Instruments, like the 8003, generate a Service Request message, *device_intr_SRQ*, when the RQS bit in the Status Byte Register becomes true. Service Requests (SRQs) are sent through the Interrupt Channel to alert the client that an event has occurred and/or that the device needs service. SRQ generation is a multilevel function and is determined by the occurrence of an event that has its corresponding enable bit set to '1'. The outputs from the event registers are summarized in separate bits in the Status Byte Register. The Event registers and the Output Queue are cleared when read or by the ***CLS** command.

3.3.1 Event Registers

An event register **captures 0 to 1 transitions** in its associated condition register or in the standard event register. An event bit becomes TRUE (1) when the associated condition bit makes a **logical 0 to 1 transition**. Once an event bit is set it **is held** until the event register is read or cleared with the ***CLS** command.

Each event register contains eight or sixteen bits. When the register is read, its response is a decimal number that is the sum of the binary bit weights of the bits that are logical 1s. e.g.

23 decimal = 0001 0111 or 0000 0000 0001 0111 binary

Each event register bit has a corresponding enable bit. The enabling bits are ANDed with the state of the event bits to create the summary condition in the Status Byte Register. Unwanted conditions can be blocked from generating SRQs by setting their corresponding enabling bit to a '0'. The enabling bits are set by writing the value equal to the sum of all of the desired logic 1 bits to the enabling register. The value is normally decimal but can be expressed in HEX, OCTAL or BINARY by prefixing the number with a #H, #O or #B.

3.3.2 Event Status Register

The Event Status Register reports events that are common to all 488.2 devices. This includes events such as selftest errors, command errors, execution errors, power on and operation complete. The Power-on event occurs at power turn-on and can be used to signal a power off-on occurrence. The External Data Ready flip-flop is included in the Event Status Register. Bit 6 is EDR #1 and Bit 2 is EDR #2. The 488.2 Operation Complete event has no meaning for either unit.

TABLE 3-1 ESR BIT DEFINITIONS

Bit	Weight	Event	Description
7	128	PON	The Power-on event occurs at power turn-on and can be used to signal a power off-on occurrence. External Data Ready Flip-flop #1 set.
6	64	EDR	
5	32	Cmd	Command Error. Bad command, spelling or syntax. Command not executed.
4.	16	Exc	Execution Error. Value out of range. Command not executed
3	8	Flash	Flash configuration data corrupted. Do a CAL:DEFAULT, check and reload the configuration settings and do a CAL:DATE <date> command to clear bit 3. User settings lost and may need reloading. If ICS Serial Number or MAC address is lost, contact ICS Support.
2	4	Query	Query error, data not read or read attempt with no data.
1	2	EDR#2	External Data Ready Flip-flop #2 set.
0	1	OPC	Operation Complete. Operation Complete has no meaning in the 8003 or 8013.

The Event Status Register is read with the ***ESR?** query and cleared with the ***CLS** command. Use the ***ESE** commands to set the Event Status Enable Register as shown in the following example:

***ESE 60**

***ESE 124**

***ESE?**

'enables error bits 2 through 5 for errors

'enables error bits 2 through 5 and the EDR bit

'queries the enabling register setting

3.3.3 Questionable Registers and Digital Inputs

The Questionable Registers lets the user read the first fifteen digital input lines and detect any changes in the digital inputs. Bit alignments are shown in Figure 3-1. The Questionable Transition Register filters the inputs and passes only the enabled state changes to the Questionable Event Register. The Questionable

Event Register bits becomes true (1) when the positive transition bit is enabled and the associated condition register bit makes a 0 to 1 transition or when the negative transition bit is enabled and the associated condition register bit makes a 1 to 0 transition. When both transitions are selected for the same bit, the corresponding Questionable Event Register bit sets whenever the digital input changes state. The Questionable Event Register is cleared when it is read with the **STAT:QUES** query. The response is a decimal sum of the binary values of the bits set to logical 1. e.g.

33 = bits 4 and 0 set to 1, all others set to 0

3.3.3.1 Monitoring Digital Inputs for State Changes

The 8003 can be set to monitor the digital inputs and generate a Service Request when they change state. The following example sets the Questionable Event register to monitor digital inputs CH5 and CH6 by capturing a positive transition on bit 4 and a negative transition on bit 5:

STAT:QUES:PTR 16 'enables bit 4 to set on a positive transition of CH1
STAT:QUES:NTR 32 'enables bit 5 to set on a negative transition of CH2

The Questionable Enable Register enables set Event bits to be included in the summary output to the Status Byte Register. The following example enables bits 0 and 1:

STAT:QUES:ENAB 48 'enables Event bits 4 and 5

The Questionable Event Register has to be cleared after a Service Request is generated to reset the bits. Use either the *CLS command or read the register with a query. e.g.

STAT:QUES? 'reads the Questionable Event reg

3.3.3.2 Reading the Digital Inputs

The Questionable Condition Register reflects the **real time** condition of the first 15 digital inputs. A logical 1 means that the corresponding digital input is high or an open contact. A logical 0 is a low input or a contact closure to ground. To read the Questionable Condition Register use the following SCPI query:

STAT:QUES:COND? 'reads the status inputs

Reading the Questionable Condition Register does not change its contents. The response is a decimal number that is the sum of the binary values of the inputs with levels equal to a logical 1. e.g.

129 = bits 8 and 1 set to logic 1, others = 0

3.3.4 Operation Registers

The 488.2 Operation Registers lets the user read device specific status conditions and detect any changes in the device's status. The Operation Registers are similar to the Questionable Registers described in paragraph 3.4.3.

The Operation Condition Register reports the Status A and Status B inputs, the EDR latches, the WTG (Waiting for Trigger) status and the Local Lockout and Remote GPIB interface states. The Status B input can be used as the Request-to-go-to Local input. When the Status B input is true, the board will go to the Local State if it is not in the Lockout State. The WTG bit is true when the board is armed and is waiting for a trigger. The following commands demonstrate some possibilities of the Operation Registers:

STAT:OPER:PTR 1	'enables bit 0 to set on a positive transition of Status A
STAT:OPER:NTR 2	'enables bit 1 to set on a negative transition of Status B
STAT:OPER:ENAB 3	'enables Event bits 1 and 2
STAT:OPER:COND?	'queries the Operation Condition Reg
STAT:OPER?	'queries the Operation Event Register

3.3.5 Output Queue

The Output Queue is used to send IEEE-488.2 messages back to the bus controller. These messages are responses to 488.2 and SCPI queries sent to the unit by the bus controller. The Output Queue reports a '1' in bit 4 of the Status Byte Register when it contains a message(s) to be read by the bus controller. Reading the contents of the Output Queue clears its summary bit. The Output Queue is read by addressing the board to talk at its GPIB address. If the Output Queue is not read before sending another query, its contents will be lost and an error reported.

Good programming etiquette is to follow each query by reading the result. Testing the Output Queue's summary bit before addressing the device to talk can confuse the board and lead to erroneous results.

3.3.6 Status Byte Register

A service request (SRQ) is generated whenever any of the enabled bits in the Status Byte Register become true and the board is not addressed as a talker. The Status Byte Register may be read by a Serial Poll or with the ***STB?** query. The Status Byte Register is enabled by setting the corresponding bits in the Service Request Enable Register with the ***SRE** command. e.g.

*SRE 160	'Sets the SRE Register to 1010 0000 which enables just the Event Status and Operation register summary bits to generate SRQs.
-----------------	---

3.3.7 Saving the Enable and Transition Register Values

When the PSC flag is set, the Enable and Transition Register values are cleared at power turn-on. The registers can **only** be saved and recalled at power turn-on by disabling the PSC flag. The ***SAV** command does not save these registers. Use the ***PSC 0** command to disable the PSC flag and save the current Enable and Transition register values as shown in the example. e.g.

STAT:OPER:ENAB 1	'enables Status A bit
STAT:OPER:NTR 1	'enables neg transition
*PSC 0; ESE 192; SRE 32	'saves Power-on and EDR bits and current registers values as the new power on settings.

The enable and transition register setting commands must be on the same line or set prior to the ***PSC 0** command to be saved. A later ***PSC 1** command sets the PSC flag which will cause the registers to be cleared at the next power turn-on.

3.3.8 488.2 Differences from 488.1 Devices

The IEEE-488.1 Device Clear command **does not** reset the digital outputs as would be expected of a 488.1 device. To reset the digital outputs, use the ***RST** (Reset) or ***RCL 0** command.

The IEEE-488.2 Standard mandated a list of common commands that are common to all IEEE-488.2 compatible devices. The 8003 responds to these commands and to some optional common commands defined in the IEEE-488.2 Standard. Table 3-3 lists how the 8003 responds to these commands and describes their effect on the 8003 and its status reporting structure.

TABLE 3-3 IEEE-488.2 COMMON COMMANDS

COMMAND	NAME	DESCRIPTION																											
* CLS	Clear Status	Clears all event registers summarized in the status byte, except for "Message Available," which is cleared only if *CLS is the first message in the command line.																											
*ESE <value>	Event Status Enable	<p>Sets "Event Status Enable Register" to <value>. <value> is an integer between 0 and 255, whose binary equivalent corresponds to the state (1 or 0) of bits in the register. If <value> is not between 0 and 255, an Execution Error is generated.</p> <p>EXAMPLE: decimal 16 converts to binary 00010000 which sets bit 4 to a logical 1.</p>																											
*ESE?	Event Status Enable Query	8003 returns the <value> of the "Event Status Enable Register" set by the *ESE command. <value> is an integer whose binary equivalent corresponds to the state (1 or 0) of bits in the register.																											
*ESR?	Event Status Register Query	<p>8003 returns the <value> of the "Event Status Register" and then clears it. <value> is an integer whose binary equivalent corresponds to the state (1 or 0) of bits in the register. The bit weights are:</p> <table><tr><th>Bit</th><th>Weight</th><th>Error</th></tr><tr><td>7</td><td>128</td><td>Power On</td></tr><tr><td>6</td><td>64</td><td>Ext Data Ready F/F</td></tr><tr><td>5</td><td>32</td><td>Command Error</td></tr><tr><td>4</td><td>16</td><td>Execution Error</td></tr><tr><td>3</td><td>8</td><td>Flash Data Corrupted</td></tr><tr><td>2</td><td>4</td><td>Query Error</td></tr><tr><td>1</td><td>2</td><td>EDR F/F#2 Set</td></tr><tr><td>0</td><td>1</td><td>Operation Complete</td></tr></table>	Bit	Weight	Error	7	128	Power On	6	64	Ext Data Ready F/F	5	32	Command Error	4	16	Execution Error	3	8	Flash Data Corrupted	2	4	Query Error	1	2	EDR F/F#2 Set	0	1	Operation Complete
Bit	Weight	Error																											
7	128	Power On																											
6	64	Ext Data Ready F/F																											
5	32	Command Error																											
4	16	Execution Error																											
3	8	Flash Data Corrupted																											
2	4	Query Error																											
1	2	EDR F/F#2 Set																											
0	1	Operation Complete																											

**TABLE 3-3 IEEE-488.2 COMMON COMMANDS
(CONTINUED)**

COMMAND	NAME	DESCRIPTION
*IDN?	Identification Query	8003 returns its identification code as four fields separated by commas. These fields are: manufacturer, model, six-digit serial number and hardware-firmware version and date e.g. ICS Electronics, 8003, S/N 1002001, Rev. 00.00 ver 10.02.03 . The IEEE-488.2 specification states that the word 'model' may not appear in the IDN message.
*OPC	Operation Complete Command	Causes the 8003 to generate the operation complete message in the Standard Event Status Register when all pending selected 8003 operations have been finished.
*OPC?	Operation Complete Query	Places an ASCII character 1 into the 8003's Output Queue when all pending selected 8003 operations have been finished. Pulse commands are not included in the *OPC bit.
*PSC<value>	Power-On Status Clear	Controls the automatic power-on clearing of the SRE and ESE registers. *PSC 0 allows devices to restore the saved SRE and ESE values and to assert SRQ upon power turn-on. *PSC 1 enables the power-on clear and disallows a SRQ at power turn-on. The PSC commands saves the 488.2 SRE and ESE registers and the SCPI transistion and enable register values.
*PSC?	Power-On Status Clear Query	Querys the PSC flag value. A returned value of 0 indicates the registers will retain their saved values, a returned value of 1 indicates the registers will be cleared.
*RCL <value>	Recall	Restores the state of 8003 from a copy stored in its Flash by *SAV command. *RCL 0 recalls saved configuration, updates output levels and re-initializes the UART. Allow the 8003 100 ms and the 8013 150 ms for the *RCL command.

TABLE 3-3 IEEE-488.2 COMMON COMMANDS (CONT'D)

COMMAND	NAME	DESCRIPTION
*RST	Reset	8003 restores its power-up state except that the state of IEEE-488 interface is unchanged, including: instrument address, Status Byte and ESR Register. Disables the trigger function and pulses the Reset output signal. Allow the 8003 100 ms and the 8013 150 ms to complete the *RST command.
*SAV <value>	Save	Saves current 8003 configuration in the Flash. *SAV 0 saves the current setting as the new power on setting. <value>=0
*SRE <value>	Service Request Enable	Sets the "Service Request Enable Register" to <value>. The value of bit six is ignored because it is not used by the Service Request Enable Register. <value> is an integer between 0 and 255, whose binary equivalent corresponds to the state (1 or 0) of bits in the register. If <value> is not between 0 and 255, an Execution Error is generated.
*SRE?	Service Request Enable Query	8003 returns the <value> of the "Service Request Enable Register" (with bit six set to zero). <value> is an integer whose binary equivalent corresponds to the state (1 or 0) of bits in the register.
*STB?	Read Status Byte	8003 returns the <value> of the "Status Byte" with bit six as the "Master Summary" bit. <value> is an integer whose binary equivalent corresponds to the state (1 or 0) of bits in the register.
*TRG	Device Trigger	Pulses the Trigger Output line.
*TST?	Self-Test Query	Queries the results of the last self test. A zero response indicates no failures. Other responses are not returned as the unit will be running in a blink LED loop and will be unable to respond to the query.
*WAI	Wait-to-continue	Prevents the 8003 from executing any further commands or queries until the No-Operation-Pending flag is TRUE.

3.5 SCPI CONFORMANCE INFORMATION

The 8003 accepts SCPI commands and command extensions to configure its digital interface, to set the data formats and to transfer data. The SCPI commands conform to SCPI Standard 1994.0 and provide an industry standard, self-documenting form of code that makes it easy for the programmer to maintain the application program.

Table 3-4 shows the SCPI command tree. The command tree uses portions of the SCPI SYSTEM, STATUS, CONFIGURE, FORMAT, SENSE, SOURCE, INITIATE and CALIBRATE subsystems. The boards follow SCPI's hierarchal 'tree like' structure which starts with a root keyword and branches out to the final action keyword. Each command can be used as a query except where noted. The SCPI commands are **not** case sensitive. The portion of the command shown in capitals denotes the abbreviated form of the keyword. Either the abbreviated or whole keyword may be used when entering a complete command. Bracketed keywords are optional and may be omitted. There must be a space between the command and the parameter or channel list. Commands ending in lower case 'n' apply to the individual I/O Ports and 'n' designates the port number, 1 to 5.

e.g.	CONFigure:INPUT 1	is the same as
	CONF:INP 1	or
	conf:inp 1	

Table 3-5 lists the SCPI keywords and describes their functions in detail. Keywords other than those listed in the table or locked keywords will have no effect on the unit's operation and a command error will be reported. Refer to Appendix A-1 for additional information about SCPI commands.

Note: A SCPI command that ends with a question mark '?' is a query. All queries should be followed by reading their response to avoid data loss.

Concatenated commands on a command line are executed sequentially. In the case of a pulse command, execution means setting up a table for later generating the correct width pulses. Pulse generation starts when the command line has been parsed. Pulse generation is not included in the *OPC or *WAI commands or query.

3.6 SHORT FORM COMMANDS

The 8003 also accepts short form commands which invoke the same action as do the corresponding SCPI commands. The short form commands are one to three characters long and are not case sensitive. The short form commands have the advantage of reduce the typing load on the programmer when operating the interface from a terminal or from a terminal emulation program. In a time critical program they reduce GPIB bus traffic and the parser's execution time.

Table 3-4 shows the short form commands alongside the SCPI commands. Their parameter form is the same as that of the SCPI commands. A space is required between the short form command and its parameter. The SCPI command descriptions in Table 3-5 also apply to the appropriate short form commands.

Short form commands ending in lower case 'n' apply to the individual I/O Ports and 'n' designates the port number, 1 to 5. The following are some short form command examples:

e.g. **TP 1** 'is the same as
CONFigure:INPut:POLarity 1

N (@1:3) 'is the same as
CONFigure:INPut (@1:3)

BO5 32 'is the same as
SOURce:DATA:VALue:PORT5 32 or
DATA:PORT5 #h20

BI3? 'is the same as
SENSe:DATA:PORT3?

TABLE 3-4 SCPI COMMAND TREE

Keyword	Parameter Form	Notes & Short Form Commands
SYSTem		System Address
:ERRor?	(0, "No error")	
:VERSion?	(1994.0)	
CONFigure		Configure Strings
[:DIGital]		
:INPut	<channel list> [(@1:5)]	N
:POLarity	0 1 [1]	TP
:HANDshake	OFF ON [ON]	TH
:OUTput	<channel list> [0]	LN
:POLarity	0 1 [1]	LP
:HANDshake	OFF ON [OFF]	LH
:EDR	0 1 [0]	E
:INHibit	0 1 [1]	I
:REMOte	0 1 [0]	R
:RESet	0 1 [0]	X
:STRobe	0 1 [0]	S
:TRIGger	0 1 [0]	TR
:ASTatus	0 1 [1]	A
:BSTatus	0 1 [0]	B
FORMat		Format Strings
[:DATA]		
:TALK	ASCii HEX HEXL TABLE [HEX]	FT
:TRANSlation	<16 char string> [0123456789;,<=>?]	V
:LISTen	ASCii HEXL HEX 4833 BINary [HEX]	FL
ROUTE		Bit Commands
:CLOSe	byte, bit	CLOSE
:OPeN	byte, bit	OPEN
:RESEt	byte	BRESET
:PULSe	byte, bit	PL
:WIDTh	10-30000 [50]	PW
:CHannel	1-40 (1-128 for 8013)	PC

TABLE 3-4 SCPI COMMAND TREE (CONT'D)

Keyword	Parameter Form	Notes & Short Form Commands
SENSe		
[:DIGital]		Input
:DATA		
[:VALue]?		PI?
:PORT?	number or <channel list>	BI?
:PORTn?		BI n?
:POLarity	0- 255	IP n
:RESet:EDR		ER
:RESet:EDR2		ER2
:BIT?	byte, bit	READ?
:BYTe?	1-5 (1-16 for 8013)	BREAD?
[SOURce]		Output
[:DIGital]		
:DATA		
[:VALue]	value format dependent	PO
:PORTn	0-255	BO n
:POLarity	0-255	OP n
:STRobe		SP
STATus		
:OPERation		Status Inputs, WTG
[:EVENT]?	bit 0,1 and 5 active (0)	
:CONDition?	bit 0,1 and 5 active (0)	
:ENABle	bit 0,1 and 5 active (0)	
:ENABLe?		
:PTRansition	0-#h7FFF [All 1s]	
:PTRansition?		
:NTRansition	0-#h7FFF [0]	
:NTRansition?		
:QUESTionable		Digital Inputs
[:EVENT]?	bits 0-14 active (0)	
:CONDition?	bits 0-14 active (0)	
:ENABle	bits 0-14 active (0)	
:ENABLe?		
:PTRansition	0-#h7FFF [All 1s]	

TABLE 3-4 SCPI COMMAND TREE (CONT'D)

Keyword	Parameter Form	Notes & Short Form Commands
:PTRansition? :NTRansition :NTRansition? :PRESet INITate [:IMMeditate] :CONTinuous ABORT CALibrate :IDN :DATe :DEFault :LOCK	0-#h7FFF [0] 1(On) 0(Off) [0] string mm/dd/yy 1(On) 0(Off) [0]	 Trigger TI TC TA Calibrate

Notes:

1. Parameter enclosed by [] - denotes factory default
2. Parameter enclosed by () - denotes power on default
3. SCPI name ends with ? - denotes query only
4. Unless otherwise noted SCPI command is also a query
5. Keyword enclosed by [] - denotes optional use
6. Only a configuration command that has one of its parameters enclosed by [] can change its parameter setting and have this setting stored in the 8003's Flash (with the *SAV command).
7. The format for a SCPI list is (@1,2, n) or (@ 1:n). There must be a space between the @ and the first number and parenthesis are required. A list of numbers is separated by commas or uses a colon to denote a range of numbers.
8. Numeric entries conform to IEEE-488.2 section 7.7.2.4 for decimal numeric parameters.
9. ASCII formatted data is a series of decimal values (0-255) for each byte separated by commas. e.g. 64, 132, 8
10. The CAL:DATe commands stores the CAL:IDN and CAL:DATe parameters in the 8003's Flash.
11. The CAL:DEFault command resets the Flash memory to it factory settings. Caution - All user settings will be overridden by this command.
12. Most parameters can be output in various numeric formats (radix). The parameters with decimal 0-255 value ranges may also be output as HEX using #h00-#hFF or Binary using #b00000000-#b11111111. Conversely, the parameters shown with HEX (#h) values can also be output in Decimal form.

TABLE 3-5 SCPI COMMANDS AND QUERIES

Keyword	Default Value	Description
SYSTem	-	Starts System command branch.
:ERRor?	0, "No error"	Requests next entry in 8003's error/event queue. Error messages are: 0, "no error" -100, "Command error" -200, "Execution error" -400, "Query error"
:VERSion?	1994.0	Unit returns the <value> of the applicable SCPI version number.
CONFigure		Starts string configuration branch
DIGital		Optional digital data identifier
INPut	(@ 1:5)	Defines bytes on the digital interface that are used to create the talk string. Value is a channel list with the format in note 7. Factory default is (@ 1:5) for the 8003.
POLarity	1	Sets logical true level for input data. Requires that inputs be assigned first. Values is 0 or 1.
HANDshake	ON	Enables or disables input data handshaking. Disable handshaking to read static signals . Values are OFF ON.
OUTput	0	Defines bytes on the digital interface that are used to output data from the listen string. Bytes can be in any order. Value is a channel list with the format in note 7. Factory default is none (All bytes set as inputs)
POLarity	1	Sets logical true level for output data. Requires that outputs be assigned first. Values is 0 or 1.

**TABLE 3-5 SCPI COMMANDS AND QUERIES
(CONTINUED)**

Keyword	Default Value	Description
HANDshake	OFF	Enables or disables output data handshaking. Enable handshaking to output data to devices that only accept data at certain times or that need to handshake in data. Values are OFF ON.
EDR	0	Sets the active level of both EDR input signals. A value of 1 selects positive signal transition. A value of 0 selects negative signal edge. Values are 0 1.
INH	1	Sets the active level for the Inhibit output signal (both Inhibits in 8003). Values are 0 1.
REMOte	0	Sets the active level for the Remote output signal. Values are 0 1.
RESet	0	Sets the active level of the Reset output pulse. Values are 0 1.
STRobe		Sets the active level of the Data Strobe output pulse. Values are 0 1.
TRIGger	0	Sets the active level of the Trigger output pulse. Values are 0 1.
AStATus	1	Sets the logical true level for the Status A input signal. Values are 0 1.
BStATus	1	Sets the logical true level for the Status A input signal. Values are 0 1.
FORMat		Starts string format branch.
DATA		Optional digital data identifier
TALK	HEX	Sets talk string and data query response format. ASCII expresses a byte's input bit pattern as a decimal value equal to the binary sum of the input bits. Multiple bytes are separated by commas. HEX

**TABLE 3-5 SCPI COMMANDS AND QUERIES
(CONTINUED)**

Keyword	Default Value	Description
TRANSlation	noted	<p>converts each four bit nibble into the ASCII characters 0-9 and A-F. HEXL converts bytes into two hex characters separated by commas. TABLE allows the user to define his own character set. All talk strings end with a linefeed. Values are ASCii HEX HEXL TABLE.</p> <p>i.e. ASCii example = 128,5,255 HEX example = 8005FF HEX example = 80, 05, FF</p> <p>Defines the HEX to ASCII conversion table. May be the only command on the line. Value is a 16 character string without separators. i.e.0123456789+-.E ?</p>
LISTen	HEX	<p>Sets listen string and data output format. ASCii converts a decimal value into an eight bit binary bit pattern by bit weights. Multiple byte values are separated by commas. HEX converts the incoming ASCII characters 0-9 and A-F into the equivalent four bit HEX values. 4833 converts incoming ASCII characters in the 30 - 3F ASCII subset (0-9 and :;<=>?) into the equivalent four bit HEX values similar to ICS's Model 4833 Parallel Interface. HEXL allows the user to insert commas between hex byte values. BINary is a single data character per output byte without separators. BINary messages are multiples of the output bytes and end with EOI asserted on the last byte. The 8003 generates an Output Strobe each time all output bytes have been updated. Values are ASCii HEX HEXL 4833 BINary</p> <p>i.e. ASCii example = 161,35,69 HEX example = A12345 HEXL example = A1,23,45 4833 example = + 1 2 3 4 5 BINary example = bbb</p>

**TABLE 3-5 SCPI COMMANDS AND QUERIES
(CONTINUED)**

Keyword	Default Value	Description
ROUTE	-	Starts bit command branch
:CLOSe byte,bit	-	Sets a bit to logic 1 or ON as defined by the prior SOURce:DATA:PORTn:POLArity command. Specify the bit as byte, bit. Byte is defined as 1 to 16, bit is 0 to 7. i.e. ROUT:CLOS 1,0.
:OPEN byte,bit	-	Sets a bit to logic 0 or OFF state which is opposite the polarity defined by the prior SOURce:DATA:PORTn:POLArity command. Specify the bit as byte, bit. Byte is defined as 1 to 16, bit is 0 to 7. i.e. ROUT:OPEN 2,7.
:RESET	-	Resets a byte to it default value. Byte is defined as 1 to n. i.e. ROUT:RESET 1.
:PULSe byte,bit	-	Pulses a bit to logic 1 or ON as defined by the prior SOURce:DATA:PORTn:POLArity command. Pulse width set by the ROUTe:WIDTh command. See para 3.5. Specify the bit as byte, bit. Byte is defined as 1 to 16, bit is 0 to 7. i.e. ROUT:PULS 1,0.
:CHannel number	-	Pulses a bit to logic 1 or ON as defined by the prior SOURce:DATA:PORTn:POLArity command. Pulse width set by the ROUTe:WIDTh command. See para 3.5. Specify the bit as a channel number or list. i.e. ROUT:PULS:CH (@ 5,7)
:WIDTh value	50	Sets the pulse width for all ouput lines. Value is 10 to 30000 ms. The default is 50 ms
SENSe		Starts direct digital byte input branch.
:DIGital	-	Optional digital data identifier
:DATA	-	Digital data identifier

**TABLE 3-5 SCPI COMMANDS AND QUERIES
(CONTINUED)**

Keyword	Default Value	Description
[:VALue]?	-	Reads input ports selected by Configure branch. Format set by FORMat:TALK setting.
:PORT?	-	Sets bytes to inputs if not already inputs and returns value of signals on the bytes specified in the attached channel list. Format set by FORMat:TALK setting. i.e. SENS:DATA:PORT? (@ 1:3)
:PORTn?	-	Five queries that set the byte as an input if not already an input and returns the value of the signals on the associated byte. Format set by FORMat:TALK setting. Value of n is 1 to 5 i.e. SENS:DATA:PORT3?
:POLarity	[255]	Sets input byte polarity. Value is 0 to 255 in decimal or HEX
:RESet:EDR		Resets EDR #1 flip-flop without reading the input data.
:RESet:EDR2		Resets EDR #2 flip-flop without reading the input data.
:BIT? byte,bit	-	Reads bit status. Logic levels set by the prior SENSE:DATA:PORTn:POLarity command. . Bit is defined by byte, bit where byte is 1 to 16 and bit is 0 to 7. i.e. SENS:BIT? 1,0.
BYTe? n		Reads byte 1 to 16. Value is 0 to 255. i.e. BYTe? 2.
[SOURce]		Starts direct digital byte output branch Optional branch identifier
[:DIGital]	-	Optional digital data identifier

**TABLE 3-5 SCPI COMMANDS AND QUERIES
(CONTINUED)**

Keyword	Default Value	Description
:DATA	-	Digital data identifier
[:VALue]	-	Writes data string to output bytes set by the Configure branch. Format set by FORM: LISTen setting.
:PORTn	-	Five output commands that set the specified byte to output if not already an output and then outputs the value to selected byte. Value of n is 1 to 5. Value of parameter is 0 to 255 in decimal or HEX.
:POLarity	[255]	Sets polarity for PORTn or VALue commands. Value is 0 to 255 in decimal or HEX.
:STRobe		Pulses the Data Strobe without having received string data. Use to transfer SOURCE data to the external device.
STATus	-	Starts Status Reporting Structure
:OPERational	-	Identifies Operational registers.
:QUESTionable	-	Identifies Questionable registers.
[:EVENT?]		Returns contents of the event register associated with the command.
:CONDition?		Returns contents of the condition register associated with the command.
:ENABle	0	Sets the enable mask which allows the true conditions in the associated event register to be reported in the summary bit.
:PTRansition	255	Sets positive transition enable register. Value = 0 to 255 in decimal or HEX.
:NTRansition	0	Sets the negative Transition register. Values = - 0 to 255 in decimal or HEX.

**TABLE 3-5 SCPI COMMANDS AND QUERIES
(CONTINUED)**

Keyword	Default Value	Description
:PREset	OFF	Sets the selected Enable Register, PTR and NTR registers to their default values (0, 255 and 0 respectively) so the 8003 detects a positive changes
INITiate		Starts Trigger branch
[:IMMediate]		Enables a single trigger operation
:CONTinuous		Enables ongoing external triggers. Values = 0 1 or OFF ON.
ABORt		Disables trigger function
CALibrate		Starts calibrate branch
:IDN <string>		Sets user IDN message. String is up to 72 characters and consists of four fields (manufacturer,model code,serial number and firmware revision) separated by commas. e.g. ICS Electronics, 8003, S/N 707123, Rev 01.01 (07-10-07) .
:DATE <date>		Saves IDN message and date. The save operation lights all the LEDs. Date is in mm/dd/yy format.
:DATE?	0	Queries the calibration date. The response is 00/00/00 when the unit is not calibrated.
DEFault		Sets Flash memory to factory settings. Does not reset the VXI-11 parameters.
:LOCK		Disables configuration commands when On. Values = 0 1 or OFF ON. Table 1-4 lists the locked commands.

3.7 PROGRAMMING GUIDELINES

There are two steps to programming a VXI-11.3 instrument like the 8003 and being able to send it commands to control its digital interface. The first step is to settle on a LAN communication method. One method is to make calls to a VXI-11 compatible VISA or SICL library which sends VXI-11 commands over RPC to the 8003. This is the recommended method for Windows PCs. A second method is to make RPC calls directly to the 8003. This is the recommended method for Apple's OS X and all UNIX and LINUX like operating systems. Another method is to use the JAVA project library on SourceForge and use Java commands. Raw Socket communication can also be used to control the 8003. The following sections describe some LAN programming concepts and the VISA, SICL and RPC programming methods.

The second step, involves using the correct commands to control the 8003's digital interface. Section 3.8 provides several examples of how to control the 8003. ICS's VXI-11 Keyboard Program can be used to try out the different commands with the 8003 before you write your program. See section 3.9 for directions on using the VXI-11 Keyboard Program.

NOTE

Refer to ICS's AB80 series Application Notes and the Appendices for additional information about programming ICS's VXI-11 devices.

3.7.1 General Setup

The VXI-11 protocol provides a way to send data packets and commands to an instrument and to receive data back from instruments over your company network or LAN. As with GPIB Programming, there are a couple things the user must do before using the 8003.

Windows users should:

1. Install a VXI-11 compliant VISA Library from Agilent, Kikasui or National Instruments in the computer. National Instruments VISA version 5.2 has numerous issues and should be avoided.
2. Define the 8003 as a VISA TCP/IP Resource.
3. Write and test the Application Program in C, Visual Basic, LabVIEW or in any language that can make VISA calls

Windows users can also install a commercial RPC package and use RPC calls to control the 8003. Microsoft's RPC is not ONC compliant and cannot be used to control the 8003.

Apple, Linux/UNIX users should:

1. Install the RPC client-side support (rpcgen) for your operating system.
2. Use the rpcgen utility to install the VXI-11 RPC. (See AB80-3)
3. Write and test the program.

3.7.2 LAN Programming and Timeouts

Programs written for LAN instruments need to be organized in the following manner:

1. Open sockets and links to the 8003 and to the other instruments.
2. Body of the test program with instrument reads, writes etc.
3. An exit routine that closes all links and sockets.

Leave the instrument links and channels open until the program is finished to avoid unnecessary program delays and exhausting the devices' resources. Error testing should be built into the program to verify that the called function worked as planned. Test your commands with ICS's VXI-11 keyboard program (See Section 3.9 for VXI11_kybd directions). ICS also provides a Error Log Utility to read back soft errors from the 8003 during program debugging. See Section 3.10 for more information about the ErrorLog Utility.

LAN systems have multiple timeouts. There is the VXI-11 IO_timeout which includes the wait-for-lock-release time and the delays in the device. The wait-for-lock-release time is specified in the client's command packet and controls how long your command will wait if the device is being used by another client. The 8003's digital I/O actions are immediate except when they are being controlled by an external handshake signal. The VISA communication or RPC network timeouts are long timeouts designed to catch network failures.

COMM_timeout in ICS VXI-11 devices refers to the time the 8003's service will go without getting a message from the client before declaring the link dead. KeepAlive is a background function of the 8003's TCP/IP Stack to check the socket connection and is invisible to the application. Both of these functions will terminate a broken link or channel, close the socket, release any locks and release the resources for use by another connection. If the dead sockets are not recovered, the 8003 can run out of resources and become inaccessible.

COMM_timeout should be set to a reasonably low period to keep the system healthy. We suggest 2 minutes when you are first debugging a program and tend to breakout of the program without properly closing the sockets. Later, with a finished program, extend the time to 10 minutes or to a couple of hours

to avoid prematurely closing the socket while you are not communicating with the 8003. Hard wired systems are pretty dependable and you can safely extend the 8003's COMM_timeout to several hours. Do not set COMM_TIMEOUT to 0, which disables the timeout, or to long time periods unless you have a way to physically reset the 8003 when it runs out of resources.

The 8003's Keep_Alive function should be enabled. Keep_Alive will put a short message on the network, once every 2 hours if there has been no traffic from the client in this time. Only use Keep_Alive if your client application supports it.

ICS recommends that you install a background function in your test program to prevent unwanted socket closure during work breaks or unplanned test stop-pages. This function can be set to perform some RPC VXI-11 activity through the Core channel when nothing has been done for a period of time less than the 8003's COMM_timeout setting which can then be kept short. The background functions should not alter the state of the devices or of the interface. A non-altering action is the opening and closing a second link to the 8003.

3.7.3 VISA Libraries

VISA libraries provide an standardized application interface for user programs and outputs that communicate with standard hardware ports such as PC COM ports, PCI bus, and USB ports. VXI-11 compliant VISA libraries provide VXI-11 calls over the TCP/IP network to communicate with VXI-11 instruments as shown in Figure 3-2. Agilent, Kikusui and National Instruments (NI) provide VXI-11 capable VISA and SICL libraries. While all three VISA libraries run in WIN32 operating systems, VISA libraries variations are available for some UNIX, LINUX and other operating systems.

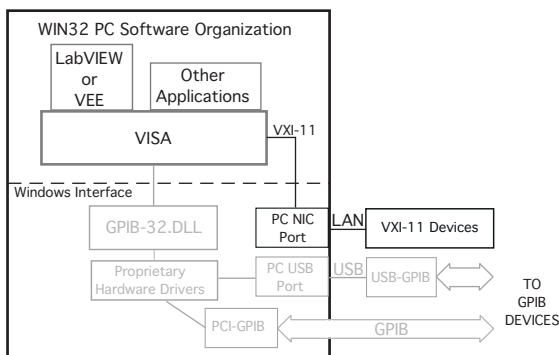


Figure 3-2 VISA to VXI-11 Communication Path

3.7.4 Using Agilent's VISA

Use version 15.0 or later of Agilent's VISA. It includes VXI-11.2 and VXI-11.3 functions and has been rewritten to work better with VXI-11.3 instruments. You do have to be sure to keep the link to the 8003 open since Agilent's VISA will close the channel when the link count drops to zero. It also inserts non-VXI-11 commands in with valid commands while doing instrument discovery which causes the 8003's ERR LED to blink. Agilent is aware of these problems and may fix them in future releases.

Configure the 8003 with Auto-disconnect on when using Agilent's VISA.

Perform the following steps to make a connection to the 8003:

1. Open the Agilent Connection Expert
2. Establish that a LAN interface is created. If none exists, create one.
3. Highlight the LAN TCPIPO interface and change its properties.
Select VXI-11 and reduce the default timeout to 10 seconds.
4. Click the Add Instrument button on the menu bar. The auto discovery algorithm should find the 8003 and display a list of found devices.
5. Select the 8003 from the resulting instrument list.
Check Allow IDN query. Uncheck Host Names. Click OK
6. The Connection Expert should verify the device and add it to the Instrument IO tree.

The user should exit the Agilent Connection Expert after the device is found. Run ICS's VXI-Error Log Utility if you experience unstable operation, program hangups or see the 8003's ERR LED blinking often. Use the error log to debug and modify your VISA program.

3.7.5 Using Agilent's SICL Library

Agilent's SICL Library includes a complete set of VXI-11.2 and VXI-11.3 functions and works well with C language and Visual Basic. It is very stable and has been around a long time. Agilent's SICL User's Guide lists all of SICL's functions and provides easy to follow instructions for creating a LAN Session and for controlling VXI-11 devices. The SICL help file provides detailed function explanations. ICS' Application Note, AB80-2, describes how to write an interactive Visual Basic program (SICL_kybd) using SICL functions. The example SICL program can be used as a starting point for your program. The Application Note, executable and source files may also be downloaded from ICS's website.

The hardest part about using Agilent's SICL library is setting up the open function to link to the 8003. The SICL_kybd program has a comboBox where the user can enter the 8003's IP address. The Create Link button calls the cmdLink function that closes any open interface links and then opens an interface link to the 8003's IP address.

The 8003 interface link format is:

```
intfc = iopen("lan;vxi-11[192.168.0.254]:inst0")
```

Note that the command ends with 'inst0' which specifies the interface link to an instrument. The IP address shown in the above example is the 8003's default IP address and is a placeholder for the 8003's current IP address. The SICL_kybd program inserts the user supplied IP address from the comboBox, if the user had entered an address, or it uses the 8003 default IP address if no IP address was entered.

Establishing an alias like '8003' and then referring to the alias in the iopen command eliminates the need to enter the ip address in your program. The alias is defined in National Instrument's MAX or Agilent's IO Connection Expert. The command becomes:

```
intfc = iopen("8003")
```

3.7.6 Using Kikusui VISA

Kikusui provides a VISA library that is compliant with IVI VISA specification 5.0. It supports USB, RS-232C, LAN, and GPIB interfaces. It has been tested and found to be compatible with ICS Electronics' GPIB and LAN interfaces. You can download the Kikusui Visa (ki-visa) from Kikusui's website at <http://www.kikusui.co.jp/en/download/>.

Use ICS's VISAKybd program with the Kikusui VISA to interactively control instruments. VISAKybd does not have a search and find feature so you will have to manually enter the instrument's GPIB address or LAN IP address.

3.7.7 Using National Instruments' VISA and MAX

National Instruments' VISA comes with the NI Measurement and Automation Explorer (MAX). It only has VXI-11.3 capability. When manually running the NI VISA Interactive Control Panel, run MAX first to define the TCP/IP Resource. Use LabVIEW version 8.5 or later, to minimize problems.

The following steps will let you use MAX to link to the 8003.

1. Run MAX.
2. Right click on Devices and Interfaces in the left hand window and select Create New.
2. Select VXI TCP/IP Resource from the pop up window. Press Next.
3. Select VXI-11 LAN Instrument. Press Next.
4. Select Auto-discovery. Press Next.
5. Select which instruments to add. Press Finish to save the instrument.

You can now run the VISA Interactive Control Panel.

1. Select the INSTR TCP/IP Resource you just entered.
2. Click the Open VISA Session button.
3. Select the BASIC I/O tab.
4. Select the Write tab. *IDN? is prefilled in the text box but you can change it to any command. Press Execute to send the command to the GPIB device.
5. Select the Read tab. Press Execute to read the device's response. Only execute the read function if you are expecting a reply from the device. Otherwise you will get a timeout error.
6. The remaining tabs let you send a Device Trigger or Device Clear to the device and Serial Poll the device.

3.7.8 Raw Socket Programming

Because TCPIP Socket communication does not have a find server capability, set the 8003 to a static IP address when using Raw Sockets. Under Windows, Mac and Linux, a user can write Socket code with nearly every programming language. The languages usually provide extensions to provide this socket capability. There are many C# examples on the Internet that can be used to make C /C++ applications. Application Note AB80-20 provides VB.Net example code for socket communication with the 8003.

Under Mac and Linux it is possible to do redirection which means that the stdin/stdout devices are redirected to an external device via Ethernet. This simply means that when the client program is called, the OS is told that instead of using the keyboard and monitor, I/O to and from the console are sent via Ethernet to/from the remote device.

3.7.8.1 Using VISA with Raw Sockets

VISA libraries provide an standardized application interface for user programs and outputs that communicate with standard hardware ports such as PCI bus, serial COM ports, USB ports and Ethernet ports. VISA easily communicates with TCP/IP devices like the 8003 when you open a session as a socket resource. The following example shows how to open a VISA Socket session for the 8003.

i.e.

```
ip="192.168.0.254"  
port="23"  
CmdStr = "TCPIP::" & ip & "::" + port + "::SOCKET"  
status = viOpen(defrm, CmdStr, 0, 0, dev) 'open instrument  
resource  
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler
```

Where ip is the 8003's IP address and port is 23 or the port number set on the 8003's Configuration Page. Set the following VISA attributes when using a SOCKET resource.

```
status = viSetAttribute(dev, VI_ATTR_TERMCHAR_EN, VI_  
TRUE)  
                                'recognize terminator  
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler  
status = viSetAttribute(dev, VI_ATTR_IO_PROT, VI_  
PROT_4882_STRS)                'enable 488.2 vi  
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler
```

The 8003 outputs a '8003 Connected' message back to the client when a connection is made to it. Do the following to read the connect message.

```
inbuf = instance.GetBytes(Space$(100)) '100 spaces  
InCount = 100  
bufSize = 100  
instring = ""  
status = viRead(udlink, inbuf, bufSize, InCount)  
                                'read the connected string  
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler  
txtResults.Text = inbuf
```

Writing commands to and reading responses back from the 8003 are the same as you would do for any other TCP/IP resource.

3.7.8.2 National Instruments' VISA Control Panel and MAX

National Instruments' VISA comes with the NI Measurement and Automation Explorer (MAX). You can define and connect to the 8003 with MAX but must use the NI_VISA Control Panel to communicate with the 8003. NI's VISA Control Panel apparently does not set VISA to recognize the terminating character so each 8003 response is only shown after the read communication times out. Move on to LabVIEW or to some other application where you can control the VISA session.

3.7.8.3 LabVIEW Notes

LabVIEW is an application that calls the VISA library to communicate with the devices it is controlling. Enable the terminating character recognition when opening the VISA session. A vi example is as follows:



Also enable the IO Protocol to support IEEE-488.2 commands because the 8003 supports the 488.2 Common Commands.

3.8 PROGRAMMING EXAMPLES

This section shows the user how to configure the 8003's Digital Interface, how to transfer data, how to use the IEEE-488.2 Status Reporting Structure to generate SRQs, to set the IDN message and how to save and lock the configuration using an existing link. The example statements in this section are 8003 command strings. The SCPI commands used in these examples may be full length commands, the shorter abbreviated commands or the short form commands listed in Table 3-4. As a reminder, all commands and queries are sent with the core *inst0* link. Transparent data transfer uses the *inst1* link.

Paragraph 3.8.1 shows how to transfer data. Paragraphs 3.8.2 through 3.8.11 describe some of the configuration commands and show how they are applied to the example application developed in Section 2.10. The example Block Diagram is repeated in Figure 3-3. The example is not meant to limit the user to the data formats shown, but rather to illustrate some of the unit's capabilities. Before you start, review Section 2.10 to learn how the example circuits are connected to the board.

The remaining paragraphs describe other applications and show how to monitor the digital inputs, generate digital waveforms, output binary data, personalize the unit and how to save the configuration.

3.8.1 Data Transfer Methods

Data can be transferred between the network and the digital I/O lines as command parameters or by transparent data strings.

The SENSE, SOURCE and ROUTE command branches give the user two ways of transferring data to or from the digital interface. The PORTn commands pass data to or from a **specific** port (or byte). The Bit Manipulation commands set/reset or read specific bit. These are the easiest commands to use as they **do not require configuring** the data ports. Example commands for bytes 1 and 6 in Figure 3-3 are:

SOUR:DATA:PORT6 128	'sets bit 7 in byte 6
SENS:DATA:PORT1?	'reads byte 1
ROUT:CLOSE 6,2	'sets bit 3 in byte 6
SENS:BIT? 1,0	'reads bit 1 in byte 1

Skip to paragraph 3.8.6 for instructions on using the PORTn commands or to 3.8.8 for the Bit commands.

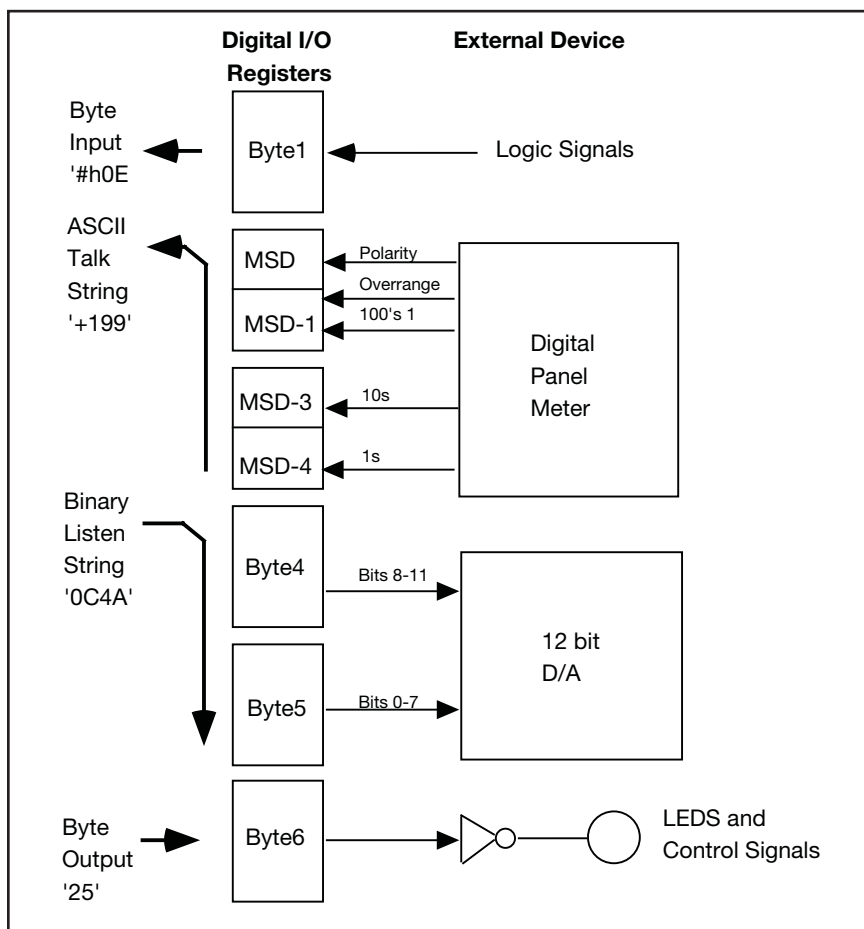


Figure 3-3 Example Application

The PORT and VALue commands can pass strings of one or multiple bytes to **bytes configured as outputs** or from **bytes configured as inputs**. These commands use bytes configured by the CONFIgure and FORMat settings. Some examples are:

SENSe:DATA?	'reads configured input bytes
SOURCe:DATA 12FD	'writes to configured output bytes

The 8003 can also transfer data as transparent data strings without the command syntax by addressing the board at *inst1*. Transfer is from the Network to **bytes configured as outputs** when the board is sent data. Transfer is from **bytes configured as inputs** to the Network when the board is addressed as a Talker. The data transfer is controlled by the handshake setting. e.g. the above write becomes:

12FD <nl>	'transparent data string
------------------------	--------------------------

The CONFIgure branch commands let the user define and configure the input and output bytes for the PORT and VALue commands or for transparent data transfer.

The FORMat command lets the user select how the data is represented. Byte data values can be decimal numbers (0-255) or hex numbers (#h0-#hFF). Data string characters can be decimal values for each byte separated by commas, HEX numbers separated by commas, HEX characters for each four bits or user defined characters for each four bits. See the FORMat command in Table 3-5. Some examples are:

128,03,174	'decimal for three bytes
80,03,AE	'HEXL for same three bytes
8003AE	'HEX for same three bytes
8003+E	'user TABLE format where '+' is substituted for the value #hA

It is a good idea to include all of the configuration parameters when setting up a board with a test program in case an unexpected value was stored in the Flash memory by an earlier application.

3.8.2 Configuring for a Talk String (Digital Inputs)

The SCPI CONFIGURE and FORMAT subsystems are used to configure the digital interface for an input string before inputting data. In the example of Figure 3-3, bytes 2 and 3 are used to input 16 bits of data as BCD/HEX characters from a digital panel meter (DPM). The DPM has a 2 1/2 digit output with a polarity signal and a conversion done signals. The data signals are

positive true and the conversion signal is a high true BUSY pulse. The DPM accepts a low true HOLD signal to inhibit future conversions. The configuration commands are:

CONFigure:INPut (@ 2,3)	'selects bytes 2 and 3
CONFigure:INPut:POL 1	'high true input data
CONFigure:INP:HAND ON	'enables input handshake
CONFigure:EDR 0	'for low going input when BUSY ends
CONFigure:INH 0	'for low going INHIBIT output to hold the DPM

Note that the parameters in the above commands are the values recorded in Table 2-8 when the example was designed.

The DPM's polarity output signal is used to generate plus and minus signs on the talk data. The technique is to jumper the 2, 4 and 8 bits of the input nibble to a value between 10 and 14. Use the external signal to select one of two characters in the Talk Translation Table. In this case, the characters are set to - and +. In the example, the digital inputs are wired so the input code is 101P with P being the DPM's Plus signal. When Plus is true, the input code is 1011 or a HEX B. When plus is false, the input code is 1010 or a HEX A. The custom format table replaces the A and B characters with - and +. The characters ".,E" and a space will be used to finish the table. The format commands then are:

FORMat:TALK TABLE	'selects a custom conversion
FORMat:TALK:TRANS 0123456789-+.,E space	'specifies the custom table. Note that 'space' is just a space character.

3.8.3 Configuring the Listen String (Digital Outputs)

The SCPI CONFIGURE and FORMAT subsystems are used to configure the digital interface for an output string before outputting data to an external device. The FORMat:LISTen command provides the user with four formats for outputting data to the 8003. ASCii lets the user send the byte output value as decimal numbers from 0 to 255 separated by commas. HEXL is similar but it uses two HEX characters from 00 to FF separated by commas. HEX is the same as HEXL but without the commas between bytes. The 4833 format is like HEX but it uses the Model 4833 rule of only accepting ASCII characters with hex values of 30 thorough 3F. These are the numbers 0-9 and ;;<=> and

?. The following example shows the four formats being used to output 24 bits of data (0000 0001 0001 0111 1111 1110):

Format	Sequence
ASCII	1,23, 254
HEXL	01,17,FE
HEX	0117FE
4833	0117?>

In Figure 3-3, bytes 4 and 5 are used to output 12 bits of data to a D-to-A converter. The data output is two bytes of low true bits with a low true pulse to load the data into the D-to-A converter's latch. HEX characters are used as they are easy to program and debug. No handshaking is required. The example configure and format commands are:

CONFigure:OUTput (@ 3,4)	'selects bytes 3 and 4
CONFigure:OUTput:POL 0	'low true input data
CONFigure:OUT:HAND OFF	'no output handshaking
CONFigure:STR 0	'sets low going data strobe
FORMAT:LISTen HEX	'selects HEX coded data

The output command or data is:

SOURCE:DATA A135 <nl>	'command and string
or	
A135 <nl>	'transparent data string

3.8.4 Outputting Multiple Data Sets

For most applications, the user will send a single set of output data in a command that is terminated with a linefeed or with END asserted on the last character. However, in some applications, the user might want to send multiple sets to the output ports. (See also paragraph 3.8.13) The boards will accept multiple sets of output values in the same command line and generate data strobes for each set as long as the sets are separated by commas. An output strobe pulse will be generated for each comma between the sets of data. List formats like ASCII or HEXL that use commas to separate bytes, require a double comma between sets to generate an output strobe. If the comma is missing, the board will not generate a data strobe. The board continues outputting data to the output ports listed in the last CONFigure:OUTput command until the command terminator is reached. The message length should not exceed the GPIB buffer size. Some examples are:

Format

ASCII

HEX

Sequence

1,23, 254,,129, 255,1 <nl>

1234,ABCD,1AC2 <nl>

3.8.5 Outputting Data with Listen Handshake Enabled

When Listen Handshaking is enabled, the board tests the Status A input and only outputs data if the Status input is in its true state. Else an Execution Error will be reported and data disboarded. The recommended procedure is for the user to test the Status A input before outputting data if Listen handshaking is enabled. If multiple sets of data are being outputted to a device with Listen Handshaking enabled, the board will wait 100 msec for the external device to complete its handshaking before starting the next data set. If the external device does not complete its handshake within the 100 msec period, the board will abort the command and report an Execution Error.

3.8.6 Configuring Polarity and Reading Individual Bytes

Individual bytes are configured and read with the SCPI SENSE subsystem. The :PORTn? command automatically sets the byte for inputting data and tristates the input lines when the port is read. The :POLarity command configures the input polarity on a bit-by-bit basis. Polarity must be programmed before data is read. In the example cable, the input bits are all high true. The example sense commands are:

```
SENSE:DATA:PORT1:POLarity #hFF 'sets all bits high true
SENSE:DATA:PORT1? 'reads byte 1
```

3.8.7 Configuring and Outputting to Individual Bytes

Output binary bytes are configured and outputted with the SCPI SOURCE subsystem. The SOURCE:DATA:PORTn command sets the byte as an output byte and the output lines to a high or low state when the port is written to. The :POLarity command configures the output polarity on a bit-by-bit basis. Polarity must be programmed before data is outputted. Example source commands are:

```
SOURCE:DATA:PORT6:POLarity #hF0 'sets byte 6 to low true
                                polarity, 4 bits hi true
                                and 4 bits low true.
SOURCE:DATA:PORT6 0 'selects byte 6 for out-
                                put and sets output bits
                                to 0 state.
```

If the optional **SOURCE** word is not used, the output commands become:

::DATA:PORT6:POLarity #hF0 'sets byte 6 to low true polarity, 4 bits hi true and 4 bits low true.
:DATA:PORT6 0 'selects byte 6 for output and sets output bits to 0 state.

3.8.8 Bit Commands

The bit commands manipulate bits directly and save the user from having to maintain a local copy of the output byte in the user's program. The **Bit ROUTe:CLOSE** command sets the specified bit to the logic 1 or ON state as defined by the prior **SOURCE:DATA:PORTn:POLarity** command. The **ROUTe:OPEN** command sets the specified bit to the logic 0 or OFF state. Bits are specified by a *byte*, *bit* variable. Bytes are numbered 1 to n where n is determined by the number of bytes on the board. Bits are numbered 0 to 7 where 0 is the LSB. Example commands are:

ROUTe:CLOSe 1,0 'sets first bit in byte 1
ROUTe:OPeN 5,2 'resets third bit in byte 5

Bits can be read with the **SENSe:BIT?** command. Bits are specified by the *byte*, *bit* variable. An example query is:

SENSe:BIT? 1,0 'query's first bit in byte 1

3.8.9 Pulse Commands

Bits can be pulsed with the **ROUTe:PULSe** command. The bits can be specified by the *byte*, *bit* variable or as signal channel numbers.

ROUTe:PULSe 1,0 'pulses first bit in byte 1
or
ROUTe:PULS:CH 5 'pulses first bit in byte 1

One or more lines can be pulsed at the same time by concatenating multiple pulse commands into one command or by using the channel list parameter. See the SCPI command rules in the Appendix for concatenation. Some examples are:

ROUT:PULS 1,0 ; PULS 2,3 ; PULS 4,7
or
PL 1,0 ; PL 2,3 ; PL 4,7

or
ROUTe:PULS:CH (@ 5,16, 28)

or
PC (@ 5,16, 28)

Set the output polarity with the **SOURCE:DATA:PORTn:POLarity (OPn)** command and follow it with a **SOURCE:DATA:PORTn 0** or **BOn 0** command to set the bytes with the bits to be pulsed to their off value. Subsequent pulse commands will generate logic true pulses. Use the **ROUTe:PULSe:WIDTh** command to set the pulse width in milliseconds. Use even 10 ms increments. i.e.

ROUT:PULS:WIDTH 70

3.8.10 Configuring the Control Signals

The control signals are configured with the **CONFIGURE** subsystem. In the example, Clear is wired into the D/A to reset it to zero, Reset is wired to the DPM to reset the meter at power turn-on time. Trigger could be used by the DPM or it could be set to free run. The Status A input is connected to 5 Vdc on the external board to monitor the external power supply. Status B and the Remote output signal are not used in the example. The example configure commands are:

CONFigure:CLEAr 0	'selects low true output
CONFigure:REMOte 0	'selects low true output
CONFigure:RESet 0	'selects low true output
CONFigure:TRIGger 0	'selects low true output
CONFigure:ASTATus 1	'selects high true input
CONFigure:BSTATus 0	'selects low true input

3.8.11 Reading the Digital Input String

Now that the talk string has been defined and digital interface has been configured, the board can input data. Be sure that the external device is pulsing the External Data Ready line or that talk handshaking is disabled. To read the data with a command use the **PI?** query or the **SENSe:DATA:VALue?** query.

SENSe:DATA:VALue?

or

PI?

If you do not receive data, query the Event Status Register (ESR) to get the status of the EDR flip-flop. (It must be on to read data if handshaking is enabled.). If ESR bit 6 is set, the data is ready to be read. If the bit is not set, the EDR input line was not pulsed.

```
*ESR?                                'Query ESR reg
IF (VAL(Rdg$) AND 32)=32 THEN.,      'Input data if EDR F/F set
```

3.8.12 Triggering the External Device

Triggering provides a way to control when an external device performs some action. Triggers can be single shot or continuous. Use the SCPI INITATE subsystem to enable the trigger output. e.g.

```
INITate:CONTinuous ON                'Enables continuous triggers
*TRG                                  'Generates the trigger pulse
```

The board responds to a *TRG command by pulsing its trigger output line. Any other action depends upon the external device. Provide a delay for the triggered action to occur and then read the input data. In the example in Figure 3-3, the Trigger output is wired to the DPM's external trigger input.

3.8.13 Using SRQs to Input Data

The status inputs, digital lines or the EDR flip-flop can generate a *device_intr_srq* message (SRQ) when selected input signals change state. The signal changes are collected in the individual event registers and summarized in the Status Byte Register. Refer to the Status Reporting Structure described in Figure 3-1.

VXI-11.3 devices like the 8003 use a reverse interrupt channel to send a Service Request message (*device_intr_srq*) to the Application. This message contains a key string to identify the device that is requesting service. When the Service Request message detected, the user queries the 8003's Status Byte and then the appropriate event register to determine the cause of the *device_intr_srq* message (SRQ). Refer to Application Bulletin AB80-4 for more information on setting up the reverse channel.

The following example generates an Service request (SRQ) whenever the EDR flip-flop becomes set (external data is ready) or the Status A signal (bit 0) goes low (In the example of 2.10, the external device may have lost power).

Setup:

*ESE 64	'EDR flip-flop set
STAT:OPER:NTR 1	'bit 0 set for negative transition
STAT:OPER:ENAB 1	'event bit 0 enabled
*SRE 160	'bits 5 and 7 are enabled (128 + 32)

After the SRQ occurs:

```
Serial Poll the 8003
If Status Byte bit 5 is on then
  Read the ESR Register
  If ESR bit 6 then
    Read the input data
  Else
    Report error!!!
End If
Else If Status Byte bit 7 is on then
  Print "External Unit off line" + (TIME)
  Beep
Else
  Report error!!!
End If
```

3.8.14 Monitoring the Digital Input Signals

The board can monitor the digital inputs and generate a *device_intr_srq* messages (SRQs) when selected input signals change state. The state changes are collected in the Questionable Event Register and summarized in the Status Byte Register. Refer to AB80-4 for information on setting up the Reverse Interrupt Channel to handle the *device_intr_srq* messages.

When the *device_intr_srq* message (SRQ) is generated, the user's application should query the Status Byte and then the Questionable Event Register to determine the cause of the SRQ. The Questionable Condition Register can be read to get the current digital input signal levels.

The following example generates an 8003 SRQ whenever Digital Input 1 (bit 0) goes high, Digital Input 2 (bit 1) goes low or Digital Input 3 (bit 2) toggles. (Note that the Digital Inputs are numbered 1-8 while the register bits are numbered 0-7)

Setup:

STAT:QUES:ENAB:PTR 5	'bits 0 and 2 high going
STAT:QUES:ENAB:NTR 6	'bits 1 and 2 low going
STAT:QUES:ENAB 7	'bits 0, 1 and 2 are enabled
*SRE 40	'bits 3 and 5 are enabled

After the SRQ occurs:

- Serial Poll the 8003
- If bit 3 of the Status Byte is on then
 - Read the Questionable Event Register
 - Test bits 0, 1 and 2 and take the appropriate action
- End If

3.8.15 Creating Digital Waveforms

The 8003's digital lines can be used to generate waveforms by successively outputting data to the same byte. Each bit in the output byte can represent a data line. An example might be using bits 0 and 1 to mimic a two line waveform from an incremental encoder as shown in Figure 3-4. If a data strobe is required, it can be created by coding a third bit or it can be automatically generated by the board after it has updated the output port(s). Refer to Application Bulletin AB48-20 at <http://www.icselect.com> for more information about waveform generation.

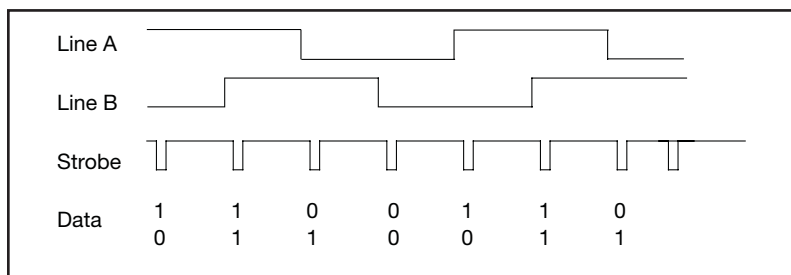


Figure 3-4 Simulated Encoder Waveforms

In Figure 3-4, the two waveforms are created by outputting the data values shown along the bottom of the figure and using the Output Data Strobe as the clock. More complex waveforms or more signals can be easily created by programming more bits in the output byte. The data rate depends upon the output command used and the speed of the controller and network. Table 1-2 lists the board's transfer times vs command syntax. For slow uniform waveforms, output one step per command. Delays can be inserted in the program to produce the desired waveform rate. If the waveforms can have bursts such as when doing

file transfers, then send the multiple steps in one command or output string. The board will need some time to input the data, parse the command and to output the first step. Subsequent steps will be outputted very quickly. The initial delay varies with the command used and is shortest for transparent data.

When outputting multiple data values, place a comma between the data values so the board will pulse the Output Data Strobe after it loads each set of data values in the configured output byte(s). The user can configure the polarity of the Output Data Strobe to create the desired clock edge. The data message length should not exceed the input buffer size. The following example shows the above waveform outputted as transparent data.

Setup:

CONF:OUT (@ 1) 'sets a byte as the output byte
FORM:LIST HEX

Sending data to the *inst1* link:

2,3,1,0,2,3,1 LF 'multiple data sets with commas to
generate data strobes for each byte

3.8.16 Outputting Binary Data (inst1 only)

The BINary format outputs 8-bit bytes to the configured output bytes and generates a Output Data Strobe each time all of the output bytes (ports) have been updated. The BINary data format only works as transparent data sent to *inst1*. The output message size must be a multiple of the number of configured output bytes and be terminated with an EOI. Output handshaking is ignored. Data rates can be > 50 Kbs. The following is the same data shown in paragraph 3.8.13 with the BINary format.

Setup:

CONF:OUT (@ 1) 'sets port 1 as an output port
FORM:LIST BIN 'sets BINary output format

Send data to inst1 with END asserted on the last byte:

2310231 '8003 outputs 7 bytes on port 1
with a strobe for each byte

3.8.17 Personalizing the IDN Message

The IDN message is changed with the CALIBRATE:IDN command. Change the IDN message to personalize the board's response, to identify the end product as being from your company and to record product history or revision dates. The IDN message is a lockable parameter. If locked, it needs to be unlocked before being changed. The format for the IEEE-488.2 IDN message is four fields (company, model#, serial number and revision) separated by commas and a maximum of 72 characters long. The word "model" may not be used in an IEEE-488.2 IDN message. An example IDN message change sequence is:

CAL:LOCK OFF	'unlocks all parameters
 CAL:IDN Acme Widgets Co, 101, s/n 007, Rev 10 0/08/99	
CAL:DATE 01-15-99	'saves new IDN message
	'Note-use the current date
CAL:LOCK ON	'relocks all parameters
*SAV 0	'saves lock status

3.8.18 Locking the Setup Parameters

All of the digital interface configuration parameters can be locked to prevent accidental change by the end-user. Locking the 8003's setup parameters is not to be confused with locking a communication link to prevent another client from taking control of the 8003. See paragraph 3.2.2.

These lockable parameters are noted by a # symbol in Table 1-3. Locked parameters cannot be queried or changed while locked. Any command that addresses a locked parameter is not executed, the Command Error bit in the Event Status Register is asserted and the ERR LED is lit. The lock function is saved by the *SAV 0 command. An example is:

CAL:LOCK ON	'blocks unauthorized changes
*SAV 0	'saves lock condition
 CAL:LOCK OFF	
	'unlocks the setup parameters

While lock is enabled, the end-user can only change and save any non-locked parameter.

3.8.19 Saving the Configuration

The 488.2 *SAV 0 command saves the current configuration in Flash Memory. This includes the data polarity and the current output signal levels. The saved configuration is recalled and the digital lines are restored to their saved state at power turn-on time or by the *RCL 0 command.

WARNING - Because the Flash Memory has a finite number of write cycles, the *SAV command should not be used inside a program loop. Configure the digital lines to their desired states before saving the configuration. e.g.

***SAV 0** 'saves current values and configuration

***RCL 0** 'recalls the saved configuration

3.8.20 Restoring the Factory Settings

If the user is not be certain of what is saved in the unit's flash memory, use the CAL:DEFAult command to restore the unit to the factory default settings. CAL:DEFAult only restores the 8003's Digital Interface settings and does not alter the network settings. e.g.

CAL:DEFAult 'restores factory settings

***SAV 0** 'saves factory configuration as
the new power on configuration

3.9 VXI-11 KEYBOARD PROGRAM

The VXI-11 Keyboard Program (`VXI11_kybd`) is a utility program that lets a user interactively control VXI-11 instruments directly from the computer's keyboard. The VXI-11 Keyboard Program is the recommended way to test the 8003 or 8013 after its installation or to try out commands before incorporating them into a program. The VXI-11 Keyboard Program (`VXI11_kybd`) runs on any WIN32 PC and is provided on the included Support CD.

3.9.1 VXI-11 Keyboard Installation

Select the 'Install VXI-11 Support' option to run the `ICS_VXI-11_Install` program. This program installs the VXI-11 Keyboard on your computer along with ICS's `VXI-11.DLL`, Microsoft's Visual Basic 6 Runtime Package and the VXI-11 support documentation.

3.9.2 VXI-11 Keyboard Operation

To run the VXI-11 Keyboard Program, double click the desktop `VXI11_kybd` icon or run it from the Window's Start menu by pointing to Programs and then to `ICS_Electronics`. Select `VXI11_kybd` from the submenu.

When the VXI-11 Keyboard Program launches, only the Find Server and Create Link buttons are enabled. The initial steps are to discover and link to the server (8003) and then to the desired instrument (*inst0* or *inst1*)

Press the Find Server button (located in the VXI-11 Server frame) to scan for servers. If you know the IP address, it can be manually entered in the VXI-11 Server window. The number of servers found is displayed in the Device Response window at the lower left. In the VXI-11 Server frame, use the pulldown arrow to display the found servers. Select the 8003's IP address and press the Select and Create Link button. If the 8003 is not found, use the directions in Section 2.5 to review and correct your network connections to the 8003.

The message in the Device Response window will tell you the link to the 8003 has been established and two instruments have been discovered. Use the pull down arrow on the right side of the Instrument Resource Address window to expose the instruments in the 8003. Select an instrument and press the Select Inst and Link button to link to the instrument. A message is displayed in the Device response window when the link has been created and the remaining `VXI11_kybd` buttons and controls are enabled.

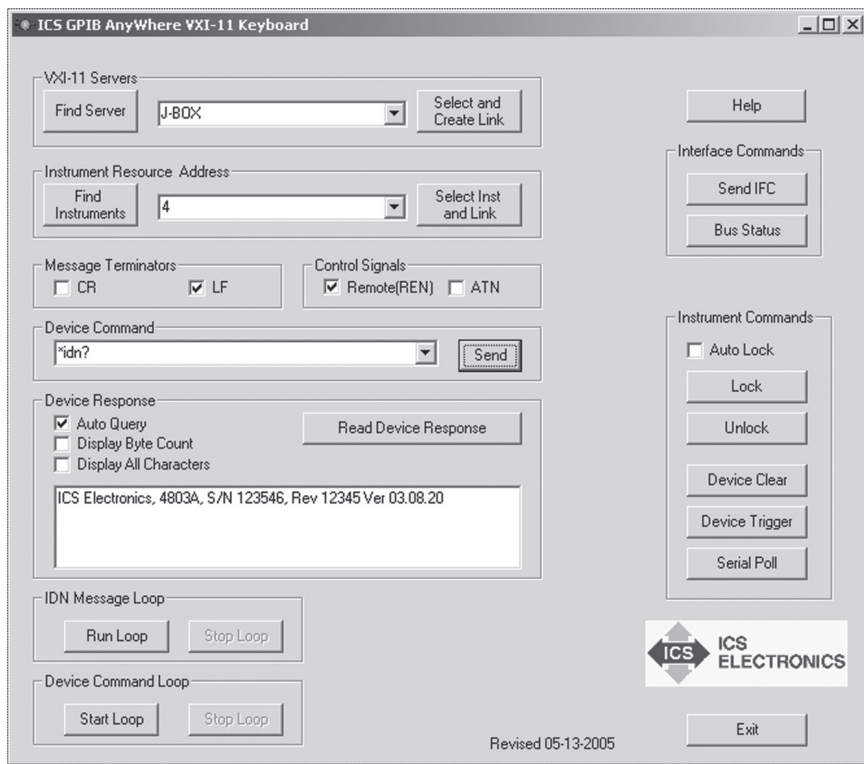


Figure 3-5 VXI-11 Keyboard Program Panel

At this point, you can communicate with the linked instrument just like any other GPIB device. The VXI11_kybd default setup appends a linefeed terminator to all outgoing messages, looks for an EOI or linefeed terminator and automatically reads the response to a query (any string that includes a question mark). The *ESR?, *IDN?, and *STB? queries only work with IEEE-488.2 compatible GPIB devices. Figure 3-5 shows the VXI11_kybd panel that has just read the *IDN response.

To output a message, enter the message in the Device Command window and press Send. If the message was a query (contains a '?'), the VXI11_kybd program automatically reads and displays the device response. To read a response manually, press the Read device response button. If you uncheck the Auto Query button or if your query does not contain a question mark, you have to do a manual read after each query. If you do not read a response from a device and then send it another command or if you attempt to read when the device has nothing to output, you will generate a device query error in an

IEEE-488.2 device. A read that does not get a response will produce a timeout error on the VXI11_kybd.

The Interface Command buttons on the right let you send commands to a GPIB Controller and do not apply to a VXI-11 instrument like the 8003 or 8013.

The Instrument Command buttons on the right let you Lock and Unlock the instrument. Locking an instrument prevents other clients from changing its status or giving it new commands while you are using it to perform an operation. Always Unlock the instrument when done with it. When the Auto Lock check box is checked, the VXI11_kybd program automatically locks the instrument when sending it a command and unlocks it when the command has been completed or when it has received a response to a query. A Red 'Locked' message is visible when the instrument is locked.

The Device Trigger and Device Clear buttons send the corresponding GPIB commands to the instrument. The Serial Poll button reads the instruments Status Register and displays the results in the Device Response window.

Use the Help button in the upper right hand corner to access the VXI11_kybd Help File.

3.10 VXI-11 ERRORLOG UTILITY

ICS'S VXI-11 ErrorLog Utility periodically queries an ICS VXI-11 device at its configuration port to see if it has any logged errors. If there are errors, they are read and listed by the ErrorLog Utility. ICS'S VXI-11 ErrorLog Utility is designed to work with ICS Electronic's VXI-11 products with 80xx series model numbers. The ErrorLog Utility is not intended, nor will it work with VXI-11 products from other companies.

3.10.1 Error Types

An error is defined as a non-standard event. An error may be either a hard error or a soft error. A hard error is such that it prevents further operation of the device, resulting in a hang condition after which the device is no longer functional. Hard errors are shown by a blinking LED pattern if the processor is able to operate the LEDs. A soft error is an error condition that is of momentary condition and will not prevent the device from normal operation.

The occurrence of a soft error will cause the ICS VXI-11 device to momentarily flicker the red ERR LED (typically 1/10th of a second). Multiple soft errors may extend the time the ERR LED is turned on. In addition, an error entry is made in the device internal error log. The error log is accessible through the ErrorLog utility. Launching the ErrorLog utility will clear the current contents of the device error log. Future error log entries will then be displayed by the ErrorLog utility.

The ErrorLog utility reports soft error conditions and provides some minimal information as to what the error condition means. It is not intended to provide indepth information as to the conditions causing the error. However, the fact that an error occurred and the nature of the error can quite often provide important information and allows the user to isolate the cause of the error and preventing future error conditions.

3.10.2 Running the ErrorLog Utility

To launch the ErrorLog utility, open an MSDOS window and navigate to the ICS Electronics VXI-11 Utilities folder. The ErrorLog utility requires a single command line parameter, which is the IP of the ICS VXI-11 device to be monitored. If the IP is not provided, the ErrorLog utility will default to 192.168.0.254, which is the default shipping address of ICS's VXI-11 equipment. The MSDOS window will display an error message if it fails to connect to the VXI-11 server.

The occurrence of a soft error will cause the ErrorLog utility to generate a single-line error report. It consists of a time/date that the error was detected, the numeric code of the error and a short English translation of the error code.

Normal usage of the ErrorLog utility can provide two key pieces of information. The first is that an error did happen. Normal operation would not result in errors and the occurrence of an error can indicate an abnormal condition that should be investigated. Therefore it may be advantageous to have the ErrorLog utility operating over a period of time if unexpected errors occasionally happen and/or a new client utility is being developed.

The second way in which the ErrorLog may be of value is for client program debugging. Usually it is possible to single-step through a program, allowing the user to determine exactly which operation results in the error condition. Then the user can investigate the proper usage of the operation to prevent errors in the new program.

There are two basic types of soft errors. The first type is a RPC protocol type error which are usually not seen by the typical application developer. They are normally caused by communication protocol errors and should be reported to the developer of the communication package being used. Normally this would be a VISA library, RPC developer, or some other type of communication package.

The second type of error is a procedural type of error. An example of a procedural error is attempting to perform an I/O operation to/from a non-existent device or trying to read data that is not there. These errors are typically seen by the application developer and should be corrected.

TABLE 3-6 ERRORLOG ERROR CODES

Error Code	Error Description
1	VXI-11 Syntax Error
3	GPIB Device Not Accessible
4	Invalid VXI-11 Link ID
5	VXI-11 Parameter Error
6	Invalid VXI-11 Channel, Channel Not Established
8	Invalid VXI-11 Operation
9	Insufficient Resources (normally related to Link IDs or Locks)
11	Device Locked By Another Link ID
12	Device Not Locked
15	I/O Timeout Error
17	I/O Error
21	Invalid GPIB Address Specified
23	Operation Aborted (indicator, not a true error)
29	Channel Already Established
60	Channel Not Active
110	Device Already Locked
111	Timeout Attempting To Gain A Device Lock
999	Unspecified/unknown error
1000	VXI-11 Protocol Error
2000	RPC Protocol Error
2001	Unsupported RPC Function
2002	Insufficient RPC Message Length

NOTE: Usage of the Agilent I/O Library will result in some error log entries. These are due to the incorrect usage of the VXI-11 protocol in the current revision of the library and also by opening and closing links to determine the *inst* personalities in a device. Agilent is aware of their protocol errors and may correct these errors in later releases of the Agilent I/O Suite. These errors normally only show up during the opening of a SICL/VISA device and can be safely ignored.

3.11 OEM DOCUMENTATION AND CONFIGURATION

3.11.1 Firmware Settings

OEM users can set the board's IDN message to identify the company and product. The Digital I/O configuration can be set to the power turn-on values and saved. The settings can then be locked so the end-user cannot change them.

3.11.2 WebServer Pages

The OEM can customize the board's WebServer configuration pages to identify the product and incorporate the company logo by following the guidelines in Application Bulletin AB80-5.

3.11.3 End-User Documentation

OEM users of the this interface board should provide the end-user with the instructions and utility programs necessary to operate the complete system. This is not done by passing on the 8003 manual to the end-user since it does not relate to the end product. In most cases the end-user needs directions for:

1. Setting the product's Network Settings.
2. Resetting the Network Settings when he forgets them.
2. Using commands to control the overall device. (Includes sending outputs and reading inputs if applicable). The OEM needs to define the commands in terms of what they do to the overall product and show the end-user how to use them.
3. Using the trigger functions if applicable.
4. Using the 488.2 Status Reporting Structure. The OEM needs to define what the digital inputs mean if they are part of the product, how to enable SRQs and how to read the registers.

The SCPI Standard requires that the SCPI command tree and SCPI conformance information be passed on to the end-user. This means only the active or applicable commands. Edit out all unused commands. All locked commands become invisible to the end-user and should be omitted from the end-user's SCPI command tree and list.

3.11.4 Utility Programs and Drivers

The following utility programs and drivers should be given to the end-user:

1. ICS's VXI-11 Keyboard Program
2. ICS's VXI-11 Error Log Utility
3. Any programs generated by the OEM to control the end product such as LabVIEW VIs, etc.

3.11.5 Copyright Release

OEM users of the 8003 or 8013 Ethernet to Parallel Interface Board are hereby given permission to copy any portion of this manual, referenced ICS material and utility or example programs for the purpose of documenting systems, maintaining or enhancing sales of systems that incorporate ICS's interface boards. Reproduction of this manual for other purposes without the expressed written consent of ICS Electronics is forbidden.

Theory of Operation

4.1 INTRODUCTION

This section describes the theory of operation of the 8003 and 8013 Parallel Interface Boards. While the description refers to the 8003 it applies to all 80xx boards unless otherwise stated.

4.2 FUNCTIONAL DESCRIPTION

The 8003 is a microprocessor based device that performs the VXI-11 service functions to control its Digital interface from an IEEE-802.11 network. The 8003 is made up of seven major elements, most of which are interconnected to the microprocessor by a common data, address and control signal bus.

Incoming Ethernet messages are received by the LAN Interface chip. Each received message interrupts the microprocessor so it can fetch the message from the LAN Interface chip. If the decoded message is an Digital Interface command, it is converted into a command for the Digital IO section. Write commands cause the 8003 to pass the data characters on to the designated byte. Read commands cause the 8003 to input data characters from the designated bytes. The responses are sent as Ethernet packets back to the client program.

The other VXI-11.3 commands like Device Clear and Device Trigger cause the 8003 to respond as a GPIB device would respond to the equivalent IEEE-488.1 commands. Device Clear resets the Digital IO lines to their power-on state. Device Trigger pulses the Trigger output line. The ReadSTB command causes the 8003 to respond as a GPIB device would to a Serial Poll on the designated device.

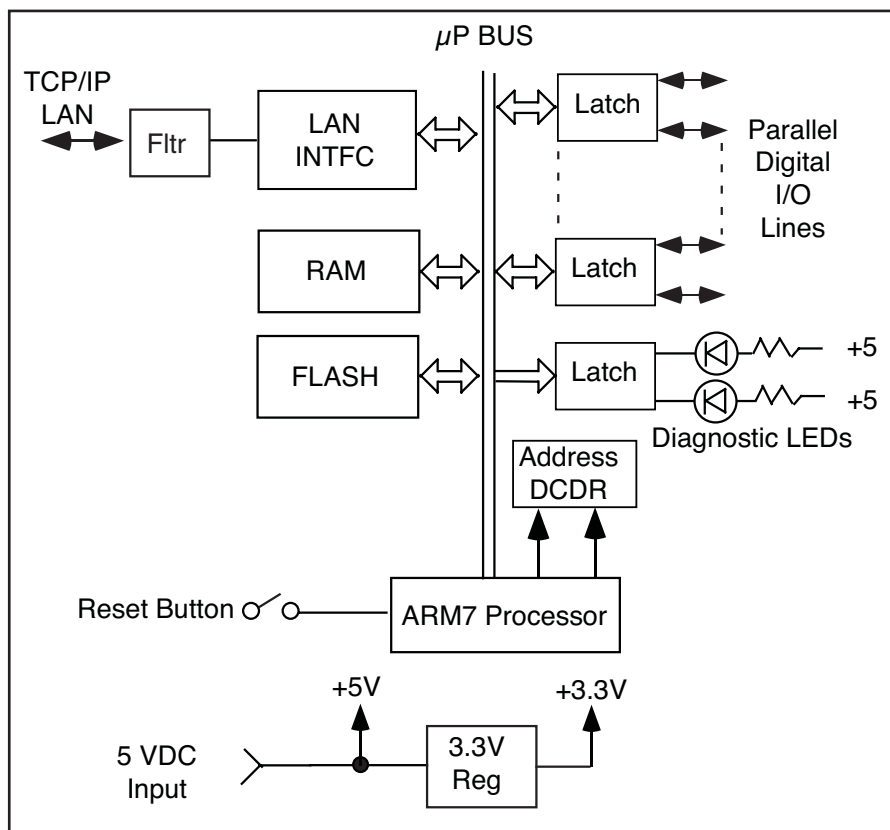


Figure 4-1 8013 Block Diagram

4.3 BLOCK DIAGRAM DESCRIPTION

Figure 4-1 shows a block diagram of the 8003's internal logic. The 8003 is a microprocessor based device that accepts commands from the TCP/IP network to control its digital outputs and to read digital inputs. The 8003 is made up of nine major elements, most of which are interconnected to the microprocessor by a common data, address and control signal bus.

Incoming network data and commands are received by the LAN Interface chip. Incoming network messages are received by the LAN Interface chip. The LAN Interface chip interrupts the microprocessor so it can fetch the message from the LAN Interface chip. The command characters are parsed and used to change the 8003's operational settings or invoke some action or a response. The typical action for the Model 8003 is to transfer data to or from the Digital interface.

The 8003's Digital Interface chips are bidirectional 8-bit registered latches. Each digital I/O line has a 33 Kohm pullup resistor to +5 Vdc. As inputs, the digital I/O lines are high impedance lines with pullup resistors that accommodate contact closures, TTL logic inputs or CMOS logic inputs. As outputs the digital I/O lines are heavy duty ABT type drivers that can source 24 mA or sink 48 mA. The Status inputs and the EDR input are HCT type inputs.

At power turn-on, the processor clears the LEDs, does a brief check of the logic elements and then checks the FLASH to find the correct program. If the processor had found a hardware error, the loading process would stop and the LEDs would blink the appropriate error code. New units default to using the program stored in the program0 space. Units that have been updated in the field may use the program1 space. If the code in program1 space is corrupted, the 8003 reverts to the factory installed code in program0.

After the correct program code is transferred to RAM, the LAN and GPIB Interfaces are initialized and the 8003 is ready to receive messages from the network. For static IP addresses, the LAN initialization takes about 3 seconds. For dynamic IP addresses, it may take up to 30 seconds to obtain an IP address from the DHCP server. The operating system handles all of the LAN communication issues, decodes the messages and decides if they require action on the Digital Interface.

The 8003 receives regulated 5 Vdc to run its Digital Interface logic and to power the 3.3 V regulator. The 3.3 V regulator regulates the 5 Vdc down to 3.3 volts to power the processor and the major logic chips. (RAM, FLASH and LAN Interface). Satisfactory DC input range is +4.8 to +5.2 Vdc.

This page intentionally left blank

Troubleshooting and Repair

5.1 INTRODUCTION

This section describes the maintenance testing, troubleshooting, and repair procedures for ICS's Model 8003 and 8013 Ethernet to Digital Interfaces. While the directions refer to the 8003, they apply equally to the 8013 unless otherwise stated.

5.2 MAINTENANCE

The 8003 does not require periodic calibration and has no internal adjustments.

5.3 TROUBLESHOOTING

Troubleshooting is broken down into selftest errors and operating errors.

5.3.1 Self Test Errors

Selftest errors occur at power turn-on time. The boards indicate self- test errors by blinking one or more of their LEDs at a 2 Hz rate. Refer to paragraph 5.4 for more information about the Self-Test Errors. The Self-Test error codes and their most likely causes are listed in Table 5-3

5.3.2 Operating Failures

Operating failures are those failures that occur while using a unit that has passed its power turn-on self test. Use the fault isolation information in Table 5-1 to narrow the problem down to a specific area. The majority of installation hookup faults can be fixed by following the table and making the necessary corrections to the installation wiring or to the application program.

Hardware failures after the unit has been properly installed and running can be isolated by substituting a known good unit in to the system. If the problem goes away, the fault lies with the removed board. Note the pin or command that fails and contact ICS for repair information. See paragraph 5.5 for repair instructions. If the fault persists, check the wiring especially any flat ribbon cables for faulty or open connectors and the devices connected to the interface board for possible faults.

WARNING

If the fault isolation procedure requires internal measurements, always remove power when disassembling or assembling the unit. Use extreme caution during troubleshooting, adjustments, or repair to prevent shorting components and causing further damage to the unit.

TABLE 5-1 TROUBLESHOOTING GUIDE

Symptom	Possible Fault	Action or Check
Unit will not turn on PWR LED off	Power supply not plugged in or on	Check for AC power at the 5 volt power supply.
	Power on board	Check power at testpoints on the board.
	3.3 volts missing	Check 3.3 volt testpoint.
	Defective crystal	Check output of Y1 across resistor R6
LEDs stuck on	Internal fault	Check TP3 (3.3V) and TP2 (5 V) for proper voltage. Return unit for repair
Unit shows blinking LEDs	Self test fault	Refer to Self Test errors in Table 5-3
Unit does not respond to client PC	LAN LED off	Wrong cable. Use a standard Ethernet cable to connect to a hub or switch. Use a cross-over cable to connect to a PC.
		No network Check hub, switch or PC for proper operation
	RDY LED Blinking	All 8003 sockets or links used. Wait for the COMM timeout or power cycle the 8003 if you are the only client connected to the 8003.
	LAN LED on ACT LED never on	No network messages received by the 8003. Check network router and gateways for proper settings Verify 8003 IP settings correct for the network location
	LAN LED on ACT LED blinks	Rescan for VXI-11 services. Select correct 8003 or IP address

TABLE 5-1 TROUBLESHOOTING GUIDE (CONT.)

Symptom	Possible Fault	Action or Check
Byte Data Transfer fails	Wrong command or command syntax error	Recheck program. Start with short form commands like BIn? or BOn space data when n = 1 to 16.
	Wrong output value	Use value 0 to 255 decimal.
	Format setting	PO command uses format set by the FORMat command Check the FORMat:LISTen setting.
Wrong input data value	Format setting	PI?, BI? and BIn? queries use the FORMat:TALK setting
	Translation Table	Check FORM:TALK:TRANS command setting.
	Data lines wired wrong	Check lines. Put 8003 in a query loop and successively ground each input.
Data output did not respond to a polarity command.	Output command not sent	Data polarity switches with next output command. Write a new value or same value to the output port(s).
	Configured Ports: Polarity command out of sequence	Configure ports as inputs or outputs before assigning the polarity.
No talk data	EDR Flip-flop not set	Query ESR or Operational Condition Registers for EDR status. Verify EDR input is being pulsed.
	Talk handshake on when not needed for static data.	Set talk handshake off.

TABLE 5-1 TROUBLESHOOTING GUIDE (CONT.)

Symptom	Possible Fault	Action or Check
Transparent string data transfer fails.	Wrong interface name used	Use <i>inst1</i> for transparent data transfer.
	Wrong format	Check selected data format
	Extra commas	Check for extra commas or terminators between data bytes in HEX and 4833 formats. Remove extra commas.
	EDR not set	EDR missing with Talk handshaking enabled. Query ESR or Operational Condition Registers for EDR status.
Second transparent string failed	Missing END	Check data command for END flag.
Data readback causes 8003 crash	Data lines jumpered together	Check for a resistor between the data lines. Replace jumpers with 1-2 kohm resistors.
Initial reading errors	Reading from Output bytes	Check byte power-on direction. Use CAL:DEF to set bytes to inputs. Resave new configuration.
		Bytes used as outputs in the program. Verify bytes not sent data.
		Bytes used for bidirectional data transfer. Do a dummy read to change bytes to inputs.
Unit hangs up when LabVIEW Starts	LabVIEW search for <i>inst</i> personalities creates/destroys many links.	Unit network resources exhausted. Enable Auto Disconnect to recover the 8003's resources when the link count goes to zero.

TABLE 5-1 TROUBLESHOOTING GUIDE (CONT.)

Symptom	Possible Fault	Action or Check
Unit hangs up after working for while	Application had not sent the 8003 a command for several minutes	COM_Timeout period expired. so 8003 disconnected the socket. Lengthen the COM_Timeout period. Add a background keep alive function to the application to periodically send the 8003 a command.

TABLE 5-2 VXI-11 ERRORS

Error	Meaning
0	No error
1	Syntax error
3	Device not accessible
4	Invalid link identifier
5	parameter error
6	Channel not established
8	Operation not supported
9	Out of resources
11	Device locked by another link
12	No lock held by this link
15	I/O timeout
17	I/O error
21	Invalid address
23	Abort
29	Channel already established

5.4 SELF TEST ERROR CODES

At power turn on, the 8003 conducts a selftest of its major components. The test takes about 5 seconds. During the selftest the PWR LED is on and the RDY LED is off. The network LAN and ACT LEDs may be on during selftest. A successful test ends when the RDY LED on. Test failures are indicated by the blinking LED patterns shown in Table 5-3. If a self test failure occurs, turn the unit off for 10 seconds and turn power back on. If the failure persists, refer to paragraph 5.7 for repair instructions. Note that some of the failures could occur while the 8003 is running.

TABLE 5-3 8003 SELF TEST ERROR CODES

Board LEDs								Fault
RDY	LAN	ACT	RDY	TALK	LSTN	SQR	ERR	
⊕	-	-	-	-	-	-	-	fatal error (CPU, FLASH, RAM..etc.)
⊕	-	-	-	-	-	-	-	fatal error (power supply)
⊕	-	-	-	-	-	-	B	LAN IC, Network Socket Failure or DHCP
⊕	-	-	-	-	-	B	-	GPIB IC or GPIB Transceivers
⊕	-	-	-	-	-	B	B	Configuration Error or Flash Failure
⊕	-	-	-	-	B	-	-	OS Issued Exit
⊕	-	-	-	-	B	-	B	RAM IC or Memory overflow error
⊕	-	-	-	-	B	B	-	OS Error
⊕	-	-	-	-	B	B	B	Flash Error

Symbols: ⊕ = solid on, B = blinking

5.5 REVERTING TO FACTORY SETTINGS

5.5.1 Factory Network Settings

The board can be reset to the default **network** settings listed in Table 1-3 at any time by holding the LAN Reset Button in for 5 seconds while applying power to the board. The TALK, LSTN and SRQ LEDs all blink momentarily when the board is changed back to its default settings. The Digital I/O configuration values are not affected by the LAN Reset operation.

5.5.2 Factory Digital I/O Configuration

The board's Digital I/O configuration can be reset to the factory settings with the CAL:DEFAULT command or with an internal jumper as directed below. CAL:DEFAULT does not change the network settings.

1. Turn the unit off.
2. Touch your hands to the shiny metal part of the host chassis or to some other instrument that is grounded to the AC power lines and to earth.
3. Locate the W1 (DEF) jumper pins as shown in Figures 2-9 or 2-10. Use a jumper or clip lead to short the two W1 (DEF) pins.
4. Apply 5 Vdc power to the board. Keep the DEF pins shorted while the board boots up. This will take approximately 10 seconds. The TALK, LSTN and SRQ LEDs will blink for 0.5 seconds as the default configuration is saved in the flash memory. Repeat step 4 if you do not see the TALK, LSTN and SRQ LEDs blink.
5. Power the unit off. Remove the W1 (DEF) jumper.

5.6 UPDATING FIRMWARE

The 80x3 boards are designed so that their firmware can be updated in the field. This is a simple procedure that the user can perform without returning the unit to the factory.

1. Connect the 8003 or 8013 to your PC as described in section 2.5.1. Apply 5 Vdc power to the board.
2. Download the latest Firmware Update for the 8003 or 8013 from ICS Electronics website, <http://www.icselect.com>.
3. Unzip the file into new directory and run the Update Utility.
4. Click on the Find Server button to scan for VXI-11.3 servers. The IP addresses of the found servers are displayed in the adjacent window. Use the pulldown arrow to display the found servers. Select the board's IP address and press the Select and Create Link button to link to your board.

When you link to the board, the program will check its revision status to see if it is the correct model number and if it needs updating.

5. If the board needs updating, click the Update Flash button to start the update process. Do not interrupt this process. The three yellow GPIB LEDs (TALK, LSTN and SRQ) will blink while the new program is being stored in the board's flash memory.
6. The Update Utility will reboot the board so the changes just made can take affect. Wait until the RDY LED comes on and then relink to the board as described in step 4 above.

CAUTION

Do not exit the Update Utility until told to do so. The Update Utility may need to reboot several times to complete the update process.

7. When done, power the unit off and back on. Verify that it passes its self test and that you can communicate with it using the VXI-11 keyboard program or with a browser.

5.7 SANITIZING PROCEDURE

The 8003 and 8013 have the following memory elements:

- RAM, 1 M x 16-bit memory
- Flash, 2 M x 8-bit memory
- Processor memory
- Webserver memory

The RAM and processor memory contents are lost when power is turned off. The Flash memory has one sector that contains user saved variables and a second sector that contains the webserver files. The other sectors are used for the unit's firmware or are spare sectors for future program use. A user does not have access to the program or to the spare sectors except when using the update program.

The Flash memory variable sector can be restored to factory settings by performing the following steps:

1. Follow the steps in paragraph 5.5.1 to restore the network settings
2. Use ICS's VXI11_kybd utility to link to the board. (See para 3.9)
3. Send the board the following commands with all capital letters:
CAL:DATE UNCAL
CAL:DEFAULT
CAL:DATE mm-dd-yy 'use today's date

The Flash memory webserver files can be read with a browser requesting pages from the webserver. The sector can be written to with the ICS's VXI-11_html_ utility or by updating the 8013's firmware and selecting YES when asked if the Update Program should load the standard HTML pages. This will reload the webserver sector with the ICS standard webserver directory and files, blocking access to any other data that might be in the webserver sector. See paragraph 5.6 for directions on updating the 8013's firmware.

NOTE: The Update Utility saves the user IDN message so it is necessary to restore the Flash memory to its factory setting by following the three steps listed above.

The RAM memory contents are normally lost when power is turned off but sometimes, a residual small charge will linger on the capacitors for several hour, holding some data in the RAM. If the Update Utility was run, most of

the RAM was overridden several times and the unit can just be turned off. To be 100% sure that all data and code remnants are fully removed from the RAM, perform the following steps to fully discharge the power capacitors:

1. Turn the unit off and disconnect all cables from the unit.
2. Locate the 3.3 V testpoint (TP4) and use a jumper to connect it to the GND testpoint (TP3). Leave the test points connected for 5 minutes or longer to drain any residual charge out of the CMOS RAM chip.
3. Remove the jumper.

The unit has now been reset to its factory shipped condition.

5.8 REPAIR PROCEDURE

Repair of the 8003 or 8013 is done by the user or by returning the unit to the factory or to your local distributor. Units in warranty should **always** be returned to the factory or else repaired only after receiving permission to do so from an ICS customer service representative.

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

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

Appendix

APPENDIX	PAGE
A1 IEEE-488 Bus Description	A-2
A1.1 IEEE-488.1 Bus	A-2
A1.2 IEEE-488.2 Standard	A-5
A1.3 SCPI Commands	A-8
A2 VXI-11 Concept	A-11
A3 VXI-11 RPC Gen Information	A-15
A4 ICS RPC Configuration Commands	A-19

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

The IEEE Std 488 Bus is a convenient means of connecting instruments and computers together to form a test system or to transfer data between two computers. The IEEE Std 488.1 covers the electrical and mechanical bus specifications and the state diagrams for each bus function. The IEEE Std 488.2 expanded on the original specification and established data formats, common commands for each 488.2 device and controller protocols. The SCPI standard developed a tree like series of standard commands for programmable instruments so that similar instruments by different manufacturers can be controlled by the same program. This appendix covers aspects of these standards that apply to a VXI-11.3 Instrument. Refer to the VXI-11.2 Standard for a complete definition of a VXI-11.3 Instrument.

A1.1 IEEE-488.1 BUS

The IEEE Std 488 Bus, or GPIB as it is commonly referred to, provides a means of transferring data and commands between devices. The physical portion of the bus is not relevant to a VXI-11.3 Instrument but the message concepts are important.

A.1.1.1 Relationship to IEEE-488.1

VXI-11.3 Instruments are servers and respond to VXI-11 Remote Procedure Calls (RPCs) from the client application (program). The *device_write* and the *device_read* RPCs transfer Device-dependent messages which are used for device control and data transfer. Examples are IEEE-488.2 Common Commands, SCPI commands and other device dependent messages.

VXI-11.3 Instruments may act as talkers or listeners

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

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

VXI-11.3 Instruments are addressed by their IP addresses. They then are connected to by opening a socket connection to them. Finally a link is made to the Instrument Interface inside the VXI-11.3 Instrument. Each VXI-11.3 Instrument contains at least one Instrument Interface referred to as *inst0*. VXI-11.3 Instruments may have multiple Instrument Interfaces for sub-functions similar

to the dual primary or secondary address capability in GPIB devices. These additional Instrument Interfaces are referred to as *inst1* to *instN*. The links are created and destroyed by the *create_link* and *destroy_link* RPCs.

VXI-11.3 Instruments have a remote/local capability that is controlled by the *device_remote* and *device_local* RPCs. This is equivalent to a GPIB device operation with the REN line and the LLO and GTL commands.

VXI-11.3 Instruments have the ability to generate Service Requests similar to SRQs in a GPIB device. The VXI-11.3 Instrument does this by acting as a client and sending the *device_intr_srq* RPC to a service running on the same computer with the client application. The service then handles the interrupt and often generates a flag for the client indicating what device wanted service. The user's client application can then Serial Poll the VXI-11.3 Instrument with the *device_readstb* RPC to learn the cause of the Service request.

VXI-11.3 Instruments respond to a Selected Device Clear message when they receive the *device_clear* RPC.

VXI-11.3 Instruments respond to a Device trigger message when they receive the *device_trigger* RPC. The instrument's response is conditioned by the SCPI INST and ABORT commands if they are supported by the Instrument.

VXI-11.3 Instruments have no equivalent functions for the following GPIB interface messages: SPE, SPD, PPC and PPU.

A.1.1.2 New VXI-11.3 Functions

VXI-11.3 Instruments have three additional functions not related to the GPIB bus or to the IEEE-488.1 Standard. They are:

The *device_abort* RPC causes the instrument to abort any operation associated with the link that sent the RPC.

The *device_lock* and *device_unlock* RPCs causes the instrument to block access to the instrument from other links while locked

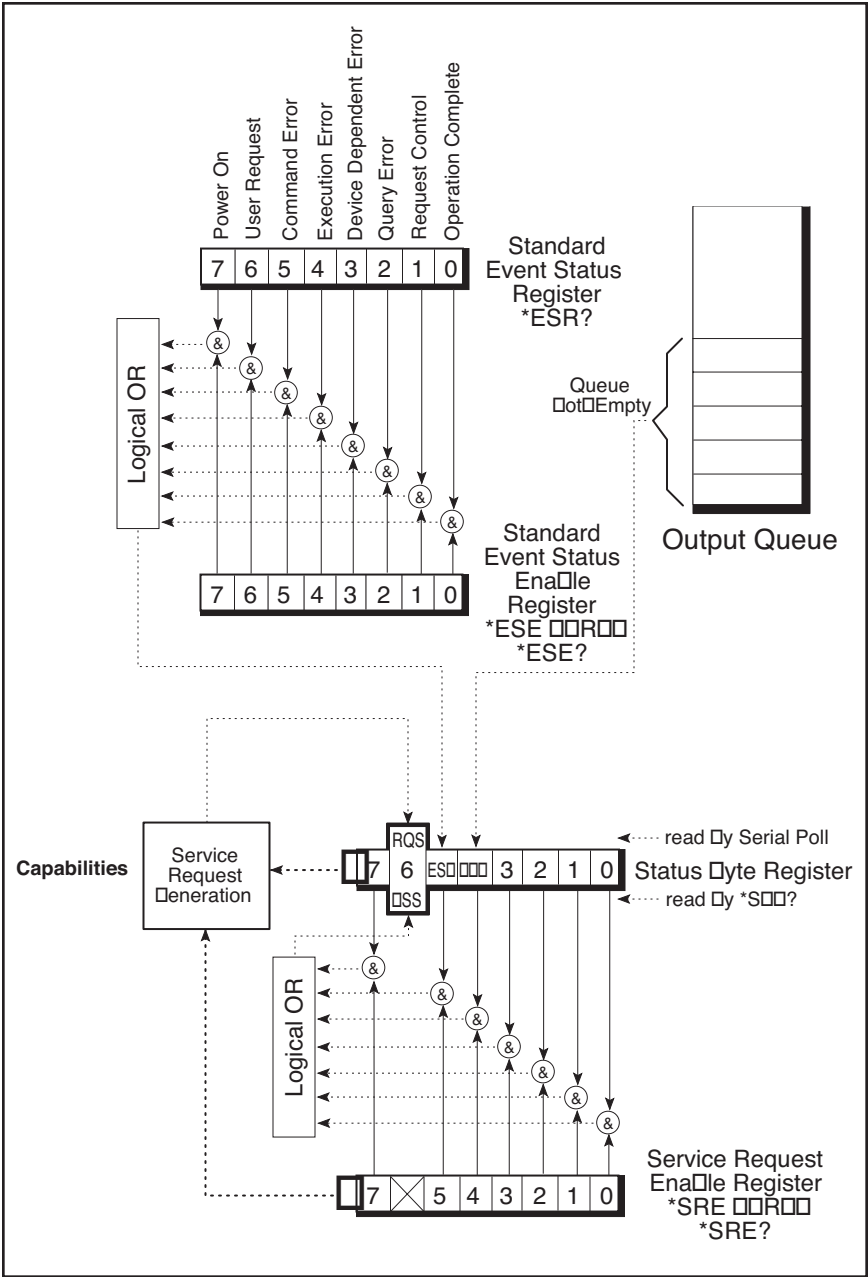


Figure A-1 488.2 Required Status Reporting Capabilities

A1.2 IEEE-488.2 STANDARD

A1.2.1 IEEE-488.2 Message Formats

The IEEE-488.2 Standard was established in 1987 to standardize message protocols, status reporting and define a set of common commands for use on the IEEE-488 bus. A VXI-11.3 TCP/IP-IEEE-488.2 Instrument Interface and its associated instrument follows all the requirements in IEEE-488.2 except where the difference between the TCP/IP-IEEE-488.2 Instrument Interface and IEEE-488.1 requires clarification.

IEEE-488.2 devices are supposed to receive messages in a more flexible manner than they send. A message sent from GPIB controller to GPIB device is called: PROGRAM MESSAGE. A message sent from device to controller is called: RESPONSE MESSAGE. As part of the protocol standardization the following rules were generated:

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

A1.2.2 IEEE-488.2 Reporting Structure

With IEEE-488.2, status reporting was enhanced from the simple serial poll response byte in IEEE-488.1 to the multiple register concept shown in Figure A-1. The IEEE-488.2 Standard standardized the bit assignments in the Status Byte Register, added eight more bits of information in the Event Status Register and introduced the concept of summary bits reporting to the Status Byte Register. The Status and Event registers have enabling registers that can control the generation of their summary reporting bits and ultimately SRQ generation. Each 488.2 device must implement a Status Byte Register, a Standard Event Status Register and an Output Message Queue as a minimum status reporting structure. A device may include any number of additional condition registers, event registers and enabling registers providing they follow the model shown in Figure A-1.

A1

TABLE A-2 IEEE-488.2 COMMON COMMANDS

Required common commands are:

- *CLS** Clear Status Command
- *ESE** Standard Event Status Enable Command
- *ESE?** Standard Event Status Enable Query
- *ESR?** Standard Event Status Register Query
- *IDN?** Identification Query
- *OPC** Operation Complete Command
- *OPC?** Operation Complete Query
- *RST** Reset Command
- *SRE** Service Request Enable Command
- *SRE?** Service Request Enable Query
- *STB?** Status Byte Query
- *TST?** Self-Test Query
- *WAI** Wait-to-Continue Command

Devices that support parallel polls must support the following three commands:

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

Devices that support Device Trigger must support the following commands:

- *TRG** Trigger Command

Controllers must support the following command:

- *PCB** Pass Control Back Command

Devices that save and restore settings support the following commands:

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

Devices that save and restore enable register settings support the following commands:

- *PSC** Saves enable register values and enables/disables recall
- *PSC?** PSC value query

A1.2.3 IEEE-488.2 Common Commands

The IEEE-488.2 Standard also mandated a list of required and optional Common Commands that all 488.2 devices could support. All of the Common Commands start with an asterisk. Commands that end with a question mark are queries. Query responses can be an ASCII number or an ASCII string. Other numerical formats are legal as long as the device supports the required ASCII format. Table A-2 lists the IEEE-488.2 Common Commands.

A1.2.4 IEEE-488.2 Differences From IEEE-488.1

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

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

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

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

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

A1.3 SCPI COMMANDS

A1.3.1 Introduction

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

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

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

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

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

A1.3.2 Command Structure and Examples

SCPI commands are based on a hierarchical structure that eliminates the need for most multi-word mnemonics. Each key word in the command steps the device parser out along the decision branch - similar to a squirrel hopping from the tree trunk out on the branches to the leaves. Subsequent keywords are considered to be at the same branch level until a new complete command is sent to the device. SCPI commands may be abbreviated as shown by the capital letters in Figure A-2 or the whole key word may be used when entering a command. Figure A-2 shows some single SCPI commands for setting up and querying a serial interface.

SYSTem:COMMunicate:SERial:BAUD 9600 <nl>	'Sets the baud rate
SYST:COMM:SER:BAUD? <nl>	'Queries the current baud setting
SYST:COMM:SER:BITS 8 <nl>	'Sets character format to 8 data bits

Figure A-2 SCPI Command Examples

Multiple SCPI commands may be concatenated together as a compound command using semi colons as command separators. The first command is always referenced to the root node. Subsequent commands are referenced to the same tree level as the previous command. Starting the subsequent command with a colon puts it back at the root node. IEEE-488.2 common commands and queries can be freely mixed with SCPI messages in the same program message without affecting the above rules. Figure A-3 shows some compound command examples.

SYST:COMM:SER:BAUD 9600; BAUD? <nl>
SYST:COMM:SER:BAUD 9600; :SYST:COMM:SER:BITS 8 <nl>
SYST:COMM:SER:BAUD 9600; BAUD?; *ESR?; BIT 6; BIT?; PACE XON; PACE?; *ESR?<nl>

Figure A-3 Compound Command Examples

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

The response includes five items because the command contains 5 queries. The first item is 9600 which is the baud rate, the second item is ESR=0 which means no errors (so far). The third item is 8 (bit/word) which is the current setting. The BIT 6 command was not accepted because only 7 or 8 are valid for this command. The fourth item XON means that XON is active. The last item is 32 (ESR register bit 5) which means execution error - caused by the BIT 6 command.

A1.3.3 Variables and Channel Lists

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

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

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

Figure A-4 Channel List Examples

A1.3.4 Error Reporting

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

A1.3.5 Additional Information

For more information about SCPI refer to the SCPI Standard or to the SCPI section in any SCPI compatible instrument manual.

A2 VXI-11 CONCEPT

This Appendix describes the VXI-11 Standard and its applicability to ICS's 80xx series Interfaces

A2.1 VXI-11 Introduction

The VXI-11 Standard was created as a way to control instruments over a TCP/IP network. VXI-11 is the overall VXI-11 document and describes the network protocol. There are three sub-standards. VXI-11.1 is for a VXI chassis. The VXI-11.2 Standard is for a VXI-11 GPIB Interface like ICS's Model 80xx Ethernet to GPIB Controller. VXI-11.3 describes the control of LAN instruments and applies to the 80xx series Interfaces.

The 80xx series Interfaces respond to all VXI-11.3 commands to control their operation. These commands include such familiar GPIB tasks as writing commands to or reading responses from an instrument, Selected Device Clear, Device Trigger (GET), set Remote or Local state (GTL), and Read Status Byte (Serial Poll). The 80xx follows all the requirements in IEEE-488.2 Standard except where the difference between the TCP/IP-IEEE-488.2 Instrument Interface and IEEE-488.1 requires clarification. VXI-11 messages are similar to IEEE-488.2 messages in that can have a user embedded terminating character (linefeed) added to the end of the message and an End flag that can be asserted on the last character of the message. The 80xx cards include an IEEE-488.2 Status Reporting Structure and SCPI Command Parser that are similar to those in ICS's GPIB Interfaces.

The 80xx can be programmed with familiar graphical applications like LabView or VEE. Both applications call a VISA layer that makes VXI-11 calls over the TCP/IP network. C and Visual Basic programs can be written with RPC, SICL or VISA calls to control the 80xx. Users of Linux or any other flavor of Unix like SunOS, IBM-AIX, HP-UX, or Apple's OS X, can communicate with the 80xx through either through RPC over TCP/IP or with VISA calls. Java can be used to write applications that run on nearly all computer platforms with the Java library from jGPIBenet project on SourceForge. Refer to ICS's Application Notes for detailed information on programming VXI-11 devices.

RPC (Remote Procedure Call) provides an invisible communication medium allowing the developer to concentrate on his program. The VXI-11 Specification, available at <http://www.vxibus.org>, includes the necessary RPCgen header files to generate the RPC calls. RPC calls can be used with virtually any operating system that has TCP/IP communication capability.

A2.2 Sockets, Channels and Links

The 80xx is a VXI-11 server and uses a socket connection for bi-directional communication with a client which is the application program running in a computer. The 80xx has 15 sockets for connection with up to 15 clients at a time and a 16th socket for UDP communication. UDP protocol may be used initially when the client scans for a VXI-11 server. The TCP/IP socket communication is used when the client links to the 80xx. The 80xx only supports VXI-11 commands via TCP and not via UDP.

To use the 80xx, the client must first locate and open a socket connection to it and then link to an instrument interface in the 80xx. The 80xx cards have multiple instrument personalities and *inst0* is the first instrument personality. Linking to *inst0* gives the user access to the parser and control logic. These are the same steps that the VXI-11 keyboard utility goes through before you can send commands and queries to the 80xx.

Locating the 80xx can be done by scanning for a VXI-11 service or by addressing the 80xx at its IP address. Linking to the 80xx is done by opening a TCP/IP socket connection to the 80xx. This socket stays open until it is relinquished by the client or until the 80xx discovers the connection is broken. Two broken link discovery methods are COMM_Timeout and KeepAlive. The COMM_Timeout setting allows the 80xx to recover channels that have not had any client activity for a set period of time. The KeepAlive setting allows the 80xx to test the client socket connection on a periodic basis. When the 80xx closes a socket, the socket and all of its resources (links and locks) become available to another client. If the 80xx runs out of resources (sockets, links or locks), it blinks its RDY LED until the shortage has been cured, typically by a socket or channel timeout.

A2

The initial socket connection to the 80xx establishes the Core channel which can handle multiple device links and locks. The client can create an Abort channel to clear Core channel command hangups. The Abort channel uses an additional TCP/IP socket. A Reverse Interrupt channel can be created from the 80xx (as an RPC client) to notify the application that a Service Request (SRQ) has occurred. Interrupt channels share a separate TCP/IP socket that does not count as one of the 80xx's 15 Core or Abort sockets.

The user normally links to the *inst0* instrument interface or personality. *inst0* is used for all configuration and data transfer commands. Once the link is made to *inst0*, the client can communicate with, control and query the 80xx just as he would do with a standard GPIB test program. The client can lock the link so that no other client can communicate with that instrument until he is finished

with it. Locking is only recommended if the device connection is such that it could be operated by another user. Some devices like ICS's 80x3 Parallel Interfaces have an additional interface, *inst1*, that is used for transparent data transfers similar to the upper GPIB address in the 4813's dual address mode.

Sockets should be closed gracefully to prevent the 80xx from running out of resources. Graceful socket closure requires several socket layer messages between the client and server sockets. Most socket close commands just ask the operating system to close the socket which can take 10s of seconds. The best way to close a socket is to do an immediate graceful socket closure instead of just letting the operating system close the socket at some later time. Note that if a connection is broken while one of its links is locked, the link stays locked until either the broken connection is detected (by *COMM_Timeout* or *KeepAlive*) or the 80xx is power cycled. Reconnecting from the same client does not allow unlocking since the new connection is through a different socket. The new socket has its own channels, links and locks and cannot manipulate resources owned by another socket.

The 80xx's initial *COMM_Timeout* is factory set to a short 2 minute period to let the 80xx quickly recover unused resources when you are first debugging a program and tend to breakout of the program without properly closing the sockets. Later on, when the program is debugged, extend the *COMM_Timeout* to 1 or more hours to avoid prematurely closing the socket while you are not communicating with the 80xx. Hard wired systems are pretty dependable and you can extend the 80xx's *COMM_Timeout* to several hours or even days. **DO NOT** set *COMM_Timeout* to 0, which disables the timeout, unless you have a way to physically reset the 80xx if it runs out of resources.

Auto Disconnect, when enabled, causes the 80xx service to abort the socket connection when the link count goes to zero and to release all resources. Auto Disconnect lets the 80xx service mimic Agilent instruments and should be enabled when operating with Agilent's IO Connection Expert. Auto Disconnect is not recommended for normal operation since it can cause communication loss in programs that are not expecting this behavior.

A2.3 Service Requests (SRQs)

The 80xx series Interfaces can generate Service Requests in a fashion similar to the SRQ generation in a GPIB device. Instead of asserting the GPIB SRQ line, VXI-11.3 Instruments, like the 80xx, generate a Service Request message, *device_intr_SRQ*, when the RQS bit in the Status Byte Register becomes true. Service Requests (SRQs) are sent through an Interrupt Channel to alert the 'client' that an event has occurred and/or that the device needs service. One

method is for the user to set up a separate task in the program that can receive the message with the id key string (handle) and set a flag. The task will be a one-way RPC service that only has to receive a message and should not reply to the 80xx Interface. You need to provide the 80xx with the IP address and initial port number of the PC. You will also need to install or activate the RPC service in your computer. The creation of the RPC service will provide you the port number since the RPC handler will establish the TCP listening socket. The IP address can be obtained through a socket call, but it does require you to know which NIC to use (remember that a PC can have multiple Ethernet NIC ports) which may require a configuration setup for the application. Refer to ICS's Application Note AB80-4 for more detailed information on RPC SRQ programming and interrupt handling.

A2.4 Transferring Data

The 80xx Interfaces normally transfer small amounts of data so there are no data transfer problems. However, when reading, the user should not attempt to limit the amount of data in a read operation from an 80xx Interface unless the Interface specifically allows it. Otherwise, the unread data will be discarded. e.g. reading 10 bytes in a 22 byte data message will result in the loss of the last 12 bytes.

ICS's 80xx series Interfaces have a *maxRecvSize* of 1024 bytes that limits the amount of the data that can be transferred in device_write RPC operation. *maxRecvSize* is the last parameter returned when the link is created to the 80xx by the create_link function.

The user can transfer larger amounts of data or long commands to the 80xx by dividing the data size by the *maxRecvSize* and then sending *maxRecvSize* blocks of data until all of the data has been transferred. The device_write function has a flags parameter which is used to determine whether an END indicator (EOI) shall be set at the completion of the write operation. The END indicator (EOI) is only asserted on the last packet. The 80xx does not terminate a write operation until it receives a packet with the end condition set.

Reading large amounts of data from a GPIB device works the same way. The 80xx does not terminate a read operation until the end condition is met. There is no readdressing of the GPIB device between packets when multiple packets are used to transfer large amounts of data. The client can request a read of more than the *maxRecvSize* number of bytes but the 80xx will only send a packet with 1024 bytes and with no reason bits set if there is more data to be sent. The client continues reading packets until it receives a packet with one or more reason bits set. See the VXI-11 Specification, Rule B.6.23.

A3 VXI-11 RPC PROTOCOL

The following information about the RPC Protocol is reprinted from the VXI-11 Specification. Consult the VXI-11 Specification for more information about using the VXI-11 RPC Protocol.

C Network Instrument RPCL

An ONC RPC protocol is described using RPCL. This section contains the complete listing of the protocols for the core, abort , and interrupt channels.

RULE C.1:

A network instrument host SHALL implement the following RPCL constructs.

C.1 Core and Abort Channel Protocol

```
/* Types */
typedef long Device_Link;
enum Device_AddrFamily {          /* used by interrupts */
    DEVICE_TCP,
    DEVICE_UDP
};

typedef long Device_Flags;        /* Error types */
typedef long Device_ErrorCode;
struct Device_Error {
    Device_ErrorCode error;
};

struct Create_LinkParms {
    long clientId;                /* implementation specific value */
    bool lockDevice;              /* attempt to lock the device */
    unsigned long lock_timeout;   /* time to wait on a lock */
    string device<>;             /* name of device */
};

struct Create_LinkResp {
    Device_ErrorCode error;
    Device_Link lid;
    unsigned short abortPort;     /* for the abort RPC */
    unsigned long maxRecvSize;    /* specifies max data size in bytes
                                   device will accept on a write */
};
```

A3

```

struct Device_WriteParms {
Device_Link lid;           /* link id from create_link */
unsigned long io_timeout;  /* time to wait for I/O */
unsigned long lock_timeout; /* time to wait for lock */
Device_Flags flags;
opaque data<>;           /* the data length and the data
                           itself */
};

```

```

struct Device_WriteResp {
Device_ErrorCode error;
unsigned long size;        /* Number of bytes written */
};

```

```

struct Device_ReadParms {
Device_Link lid;           /* link id from create_link */
unsigned long requestSize; /* Bytes requested */
unsigned long io_timeout;  /* time to wait for I/O */
unsigned long lock_timeout; /* time to wait for lock */
Device_Flags flags;
char termChar;            /* valid if flags & termchrset */
};

```

```

struct Device_ReadResp {
Device_ErrorCode error;
long reason;              /* Reason(s) read completed */
opaque data<>;           /* data.len and data.val */
};

```

```

struct Device_ReadStbResp {
Device_ErrorCode error;   /* error code */
unsigned char stb;        /* the returned status byte */
};

```

```

struct Device_GenericParms {
Device_Link lid;          /* Device_Link id from connect call */
Device_Flags flags;       /* flags with options */
unsigned long lock_timeout; /* time to wait for lock */
unsigned long io_timeout;  /* time to wait for I/O */
};

```

A3

```

struct Device_RemoteFunc {
    unsigned long hostAddr;           /* Host servicing Interrupt */
    unsigned short hostPort;          /* valid port # on client */
    unsigned long progNum;             /* DEVICE_INTR */
    unsigned long progVers;            /* DEVICE_INTR_VERSION */
    Device_AddrFamily progFamily;      /* DEVICE_UDP | DEVICE_
};                                     TCP */

struct Device_EnableSrqsParams {
    Device_Link lid;
    bool enable;                       /* Enable or disable interrupts */
    opaque handle<40>;                 /* Host specific data */
};

struct Device_LockParams {
    Device_Link lid;                   /* link id from create_link */
    Device_Flags flags;                /* Contains the waitlock flag */
    unsigned long lock_timeout;         /* time to wait to acquire lock */
};

struct Device_DocmdParams {
    Device_Link lid;                   /* link id from create_link */
    Device_Flags flags;                /* flags specifying various options */
    unsigned long io_timeout;           /* time to wait for I/O to complete */
    unsigned long lock_timeout;         /* time to wait on a lock */
    long cmd;                           /* which command to execute */
    bool network_order;                /* client's byte order */
    long datasize;                      /* size of individual data elements */
    opaque data_in<>;                  /* docmd data parameters */
};

struct Device_DocmdResp {
    Device_ErrorCode error;             /* returned status */
    opaque data_out<>;                  /* returned data parameter */
};

program DEVICE_ASYNC{
    version DEVICE_ASYNC_VERSION {
        Device_Error device_abort (Device_Link) = 1;
    } = 1;

    } = 0x0607B0;

```

```

program DEVICE_CORE {
version DEVICE_CORE_VERSION {
Create_LinkResp create_link (Create_LinkParms) = 10;
Device_WriteResp device_write (Device_WriteParms) = 11;
Device_ReadResp device_read (Device_ReadParms) = 12;
Device_ReadStbResp device_readstb (Device_GenericParms) = 13;
Device_Error device_trigger (Device_GenericParms) = 14;
Device_Error device_clear (Device_GenericParms) = 15;
Device_Error device_remote (Device_GenericParms) = 16;
Device_Error device_local (Device_GenericParms) = 17;
Device_Error device_lock (Device_LockParms) = 18;
Device_Error device_unlock (Device_Link) = 19;
Device_Error device_enable_srq (Device_EnableSrqParms) = 20;
Device_DocmdResp device_docmd (Device_DocmdParms) = 22;
Device_Error destroy_link (Device_Link) = 23;
Device_Error create_intr_chan (Device_RemoteFunc) = 25;
Device_Error destroy_intr_chan (void) = 26;
} = 1;

} = 0x0607AF;

```

C.2 Interrupt Protocol

```

/* Types */
struct Device_SrqParms {
opaque handle<>;
};

program DEVICE_INTR {
version DEVICE_INTR_VERSION {
void device_intr_srq (Device_SrqParms) = 30;
}=1;

}= 0x0607B1;

```

A3

A4 ICS CONFIGURATION RPC PROTOCOL

The following document describes ICS's Configuration RPC Protocol. This information is supplied to enable a RPC programmer to configure ICS devices that have an Ethernet interface with RPC commands.

A4.1 INTRODUCTION

This document defines the configuration interface to the ICS Ethernet devices (hereafter referred to as the Edevice). The purpose of this document is to allow the communication between the controlling computer and the Edevice, for the purposes of modifying the operational characteristics of the Edevice. Edevices are ICS products whose Model number is in the 8xxx range. Not all commands are supported by all Edevices.

A4.2 SCOPE

This specification addresses the Edevice communication for the purposes of operational configuration.

This specification is to be considered an addendum to the VXI-11 specification for communication to the VXI-11 compliant ICS Edevice Interfaces. The Edevice follows the VXI-11.2 and/or VXI-11.3 specifications.

It is assumed the reader is conversant with ONC/RPC and XDR specifications as published by Sun Microsystems. All client/Edevice communication is performed through ONC/RPC and thus requires knowledge of both (ONC/RPC and XDR) specifications. In addition, it is assumed the reader is conversant with the VXI-11 and VXI-11.2 specifications as published by the VXIbus Consortium. In some cases, the reader will require an understanding of the GPIB (IEEE-488) specification as published in the IEEE Standards.

A4.3 SPECIFICATION OBJECTIVES

This specification has the following objectives:

1. To enable the creation of tools to perform Edevice configuration.
2. To enable applications to perform temporary modifications to the Edevice configuration.
3. To define the ONC/RPC protocol used by the Edevice configuration.

A4.4 REFERENCES

- [1] IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation.
- [2] IEEE Std 488.2-1992, IEEE Standard Codes, Formats, Protocols, and Common Commands For Use With IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation.
- [3] XDR: External Data Representation Standard, Request for Comments 1014, Sun Microsystems, DDN Network Information Center, SRI International, June, 1987.
- [4] RPC: Remote Procedure Call Protocol Specification, Request for Comments, 1057, Sun Microsystems, DDN Network Information Center, SRI International, June, 1988.

B EDEVICE CONFIGURATION PROTOCOL

The Edevice Configuration protocol uses the ONC remote procedure call (RPC) model. This model allows an application executing on one computer to conceptually call a function on another computer.

The client identifies the remote procedure by means of a program ID, program version, and procedure number. This information is encoded into an RPC communication packet with the procedure argument values. The message is then sent to the RPC service running on the server device, where the target procedure is then executed. The server is required to respond to all procedure calls with an RPC reply message containing any/all procedure return values.

Table B.1 lists the RPC messages used by the Edevice configuration protocol. Messages that apply to the 8065 are marked by an 'x' in the 8065 column.

B.1 PROTOCOL BEHAVIOR

The client shall issue an RPC command to the Edevice directing the action to be taken. The Edevice shall attempt to execute the action and will then reply with an RPC reply.

All configuration messages pertaining to values or modes shall contain an Action boolean indicating whether the action is to be a read of the current setting or a modification of the current setting. Edevice default values are listed in Section 1 of this manual.

TABLE B CONFIGURATION RPC MESSAGES

Message	ID	Description Req'd	8013	Reboot
interface_name	1	VXI-11 logical name	X	No
rpc_port_number	2	RPC TCP port		
core_port_number	3	VXI-11 core TCP port		
abort_port_number	4	VXI-11 abort TCP port		
config_port_number	5	configuration TCP port		
comm_timeout	6	TCP timeout	X	No
hostname	7	Edevice TCP hostname		
static_ip_mode	8	static/dynamic IP	X	Yes
ip_number	9	network IP number	X	Yes
netmask	10	network netmask	X	Yes
gateway	11	network gateway	X	Yes
keepalive	12	keepalive time	X	No
gpib_address	13	Edevice GPIB bus address		
system_controller	14	system controller		
ren_mode	15	REN active at boot		
eos_8_bit_mode	16	EOS 8 bit comparison		
auto_eos_mode	17	automatic EOS on EOI		
eos_active	18	EOS active		
eos_char	19	EOS character		
reload_config	20	force reload of default config	X	No
reload_factory	21	reload factory config settings	X	No
commit_config	22	commit (write) current config	X	No
reboot	23	cause a reboot of the Eth488	X	No
fw_revision	24	firmware revision		
idn_string	25	read IDN type string		
error_logger	26	read current error log contents		

A4

All configuration messages pertaining to actions shall respond with an RPC reply and then (if the status is No Error) execute the action.

All Edevice configuration commands will reply with statuses corresponding to Error Codes as defined by the VXI-11 (section B.5.2).

All Edevice configuration command data will use XDR encoding. Numerical values will be of 4-byte unsigned integer format. String and binary fields will be of opaque array format. Variable length string values will be NULL terminated and will contain a leading length numerical value defining the total length (inclusive of the NULL).

All Edevice configuration commands and replies will result in RPC messages which are multiples of 4-byte lengths. Padding will occur following the last data field and may consist of any byte value.

When the Action boolean signals a read of a mode/value setting, the RPC command must contain a dummy mode/value. While the mode/value in the RPC command is not used, it must exist. If the mode/value is not contained within the RPC command, an error status will result.

The successful modification of a configuration setting will result in the change taking effect immediately, except where noted. Thus, it is strongly advisable to not make configuration changes if VXI-11 device links are currently active. Doing so can cause unpredictable results and Edevice misbehavior. However, such dynamic modifications may be desirable and are possible at the discretion of the user. Messages that require rebooting will not take affect until the Edevice is rebooted.

B.2 Edevice PROGRAM ID AND VERSION

The Edevice configuration procedures shall use an RPC program ID of 1515151515 and an RPC version number of 1.

C.1 interface_name

The `interface_name` procedure is used to read/modify the current VXI-11 logical interface name.

```
struct Int_Name_Parms {
    unsigned    int    action;
    unsigned    int    length;
    opaque      name<>;
};
struct Int_Name_Resp {
    unsigned    int    error;
    unsigned int    length;
    opaque      name<>;
};
```

`Int_Name_Resp interface_name (Int_Name_Parms) = 1;`

The `action` value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

`action = 0` = read of current value

`action = 1` = modify current value

If the `action` value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The name string must be a NULL terminated string with a 32-byte maximum length (exclusive of the NULL). An error of 5 is returned and the Interface Name is unchanged if the name field exceeds 32-bytes.

The returned `Int_Name_Resp` structure will always contain the current Interface Name, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.2 `rpc_port_number`

The `rpc_port_number` procedure is used to read/modify the TCP port used by the RPC server.

```
struct Rpc_Port_Parms {  
    unsigned    int    action;  
    unsigned    int    port;  
};  
struct Rpc_Port_Resp {  
    unsigned    int    error;  
    unsigned int    port;  
};
```

`Rpc_Port_Resp rpc_port_number (Rpc_Port_Parms) = 2;`

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The port value must be within the range of 0x0001 and 0xFFFF. An error of 5 is returned and the RPC Port value is unchanged if the port value is outside of this range.

The returned `Rpc_Port_Resp` structure will always contain the current RPC Port value, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

A4

C.3 core_port_number

The core_port_number procedure is used to read/modify the TCP port used by the VXI-11 core channel.

```
struct Core_Port_Parms {
    unsigned int    action;
    unsigned int    port;
};
struct Core_Port_Resp {
    unsigned int    error;
    unsigned int    port;
};
```

Core_Port_Resp core_port_number (Core_Port_Parms) = 3;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The port value must be within the range of 0x0001 and 0xFFFF. An error of 5 is returned and the VXI-11 Core Port value is unchanged if the port value is outside of this range.

The returned Core_Port_Resp structure will always contain the current VXI-11 Core Port value, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.4 abort_port_number

The abort_port_number procedure is used to read/modify the TCP port used by the VXI-11 abort channel.

```
struct Abort_Port_Parms {
    unsigned    int    action;
    unsigned    int    port;
};
struct Abort_Port_Resp {
    unsigned    int    error;
    unsigned int    port;
};
```

Abort_Port_Resp abort_port_number (Abort_Port_Parms) = 4;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The port value must be within the range of 0x0001 and 0xFFFF. An error of 5 is returned and the VXI-11 Abort Port value is unchanged if the port value is outside of this range.

The returned Abort_Port_Resp structure will always contain the current VXI-11 Abort Port value, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

A4

C.5 config_port_number

The config_port_number procedure is used to read/modify the TCP port used by the Edevice configuration channel.

```
struct Config_Port_Parms {
    unsigned    int    action;
    unsigned    int    port;
};
struct Config_Port_Resp {
    unsigned    int    error;
    unsigned int    port;
};
```

Config_Port_Resp config_port_number (Config_Port_Parms) = 5;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The port value must be within the range of 0x0001 and 0xFFFF. An error of 5 is returned and the VXI-11 Abort Port value is unchanged if the port value is outside of this range.

The returned Config_Port_Resp structure will always contain the current VXI-11 Abort Port value, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.6 comm_timeout

The comm_timeout procedure is used to read/modify the TCP timeout value. An inactive TCP channel will be left open this length of time before being closed. A value of zero means no timeout checking.

```
struct Comm_Timeout_Parms {
    unsigned    int    action;
    unsigned    int    timeout;
};
struct Comm_Timeout_Resp {
    unsigned    int    error;
    unsigned int    timeout;
};
```

Comm_Timeout_Resp comm_timeout (Comm_Timeout_Parms) = 6;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The timeout value is not range checked, thus it is possible to define an impossible timeout period. A timeout value of zero prevents timeout checking. If a channel remains inactive for the specified timeout period, then the channel is closed in the belief that the TCP connection is broken.

The comm_timeout procedure applies only to the VXi-11 core and Edevice configuration channels. The timeout period is defined as the number of seconds until a timeout is detected. The returned Comm_Timeout_Resp structure will always contain the current communication timeout value, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.7 hostname

The hostname procedure is used to read/modify the hostname used by the Edevice. The hostname is only applicable if a dynamic DNS service is available.

```
struct Hostname_Parms {
    unsigned    int    action;
    unsigned    int    length;
    opaque      name<>;
};
struct Hostname_Resp {
    unsigned    int    error;
    unsigned int    length;
    opaque      name<>;
};
```

Hostname_Resp hostname (Hostname_Parms) = 7;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The name string must be a NULL terminated string with a 32-byte maximum length (exclusive of the NULL). An error of 5 is returned and the Interface Name is unchanged if the name field exceeds 32-bytes.

The returned Hostname_Resp structure will always contain the current hostname value, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.8 static_ip_mode

The static_ip_mode procedure is used to read/modify the static IP mode. If static_ip_mode is set TRUE, then the Edevice will use a static IP and will need a netmask and gateway IP.

```
struct Static_IP_Parms {  
    unsigned    int    action;  
    unsigned    int    mode;  
};  
struct Static_IP_Resp {  
    unsigned    int    error;  
    unsigned    int    mode;  
};
```

Static_IP_Resp static_ip_mode (Static_IP_Parms) = 8;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The mode must be either 0 (dynamic) or 1 (static). An error of 5 is returned and the static IP mode is unchanged if the mode field is any other value.

The returned Static_IP_Resp structure will always contain the current static IP mode, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

A4

C.9 ip_number

The `ip_number` procedure is used to read/modify the static IP number. If `static_ip_mode` is set `TRUE`, then the Edevice will use a static IP (see the `static_ip_mode` function) and will need a netmask and gateway IP.

```
struct IP_Number_Parms {
    unsigned    int    action;
    unsigned    char   ip[4];
};
struct IP_Number_Resp {
    unsigned    int    error;
    unsigned    char   ip[4];
};
```

`IP_Number_Resp ip_number (IP_Number_Parms) = 9;`

The `action` value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

`action = 0` = read of current value

`action = 1` = modify current value

If the `action` value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The `ip` must be exactly 4-bytes in length. An error of 5 is returned and the current IP is unchanged if the IP is determined to be invalid.

The returned `IP_Number_Resp` structure will always contain the current IP, irrespective of the error value.

* Note that the IP will only be used if Static IP is selected.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.10 netmask

The netmask procedure is used to read/modify the netmask. If `static_ip_mode` is set TRUE, then the Edevice will use a static IP (see the `static_ip_mode` function) and will need a netmask and gateway IP.

```
struct Netmask_Parms {
    unsigned    int    action;
    unsigned    char    netmask[4];
};
struct Netmask_Resp {
    unsigned    int    error;
    unsigned    char    netmask[4];
};
```

`Netmask_Resp netmask (Netmask_Parms) = 10;`

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The netmask must be exactly 4-bytes in length. An error of 5 is returned and the current netmask is unchanged if the netmask is determined to be invalid.

The returned `Netmask_Resp` structure will always contain the current netmask, irrespective of the error value.

* Note that the IP will only be used if Static IP is selected.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

A4

C.11 gateway

The gateway procedure is used to read/modify the gateway IP. If `static_ip_mode` is set TRUE, then the Edevice will use a static IP (see the `static_ip_mode` function) and will need a netmask and gateway IP.

```
struct Gateway_Parms {
    unsigned    int    action;
    unsigned    char    gateway[4];
};
struct Gateway_Resp {
    unsigned    int    error;
    unsigned    char    gateway[4];
};
```

Gateway_Resp gateway (Gateway_Parms) = 11;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The gateway must be exactly 4-bytes in length. An error of 5 is returned and the current gateway IP is unchanged if gateway is determined to be invalid. The returned Gateway_Resp structure will always contain the current gateway, irrespective of the error value.

* Note that the IP will only be used if Static IP is selected.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.12 keepalive

The keepalive procedure is used to read/modify the keepalive value. If set to zero, then keepalives will not be used. If used, then this is the time (in seconds) of inactivity prior to a keepalive being sent.

```
struct Keepalive_Parms {
    unsigned    int    action;
    unsigned    int    time;
};
struct Keepalive_Resp {
    unsigned    int    error;
    unsigned    int    time;
};
```

Keepalive_Resp keepalive (Keepalive_Parms) = 12;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The time value is not range checked, thus it is possible to define an impossible timeout period. A time value of zero prevents keepalive from being used. If a channel remains inactive for the specified time period, then a keepalive is sent (assuming time is non-zero). The returned Keepalive_Resp structure will always contain the current keepalive value, irrespective of the error value.

* Note that the Keepalive time may be fixed and not variable. If non-zero time is specified, Keepalive will be active regardless of time.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

A4

C.13 gpib_address

The gpib_address procedure is used to read/modify the Edevice GPIB bus address.

```
struct Gpib_Addr_Parms {
    unsigned    int    action;
    unsigned    int    address;
};
struct Gpib_Addr_Resp {
    unsigned    int    error;
    unsigned    int    address;
};
```

```
Gpib_Addr_Resp gpib_address (Gpib_Addr_Parms) = 13;
```

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The address must be within the range of 0 and 30. An error of 5 is returned and the current GPIB address is unchanged if address is determined to be invalid.

The returned Gpib_Addr_Resp structure will always contain the current Edevice GPIB bus address, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.14 system_controller

The `system_controller` procedure is used to read/modify the system controller mode. If the system controller mode is set `TRUE`, then the Edevice will initialize at boot time as the GPIB bus controller.

```
struct Sys_Control_Parms {
    unsigned    int    action;
    unsigned    int    controller;
};
struct Sys_Control_Resp {
    unsigned    int    error;
    unsigned    int    controller;
};
```

`Sys_Control_Resp system_controller (Sys_Control_Parms) = 14;`

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The controller mode must be either 0 or 1. An error of 5 is returned and the current system controller mode is unchanged if controller is determined to be invalid.

The returned `Sys_Control_Resp` structure will always contain the current system controller mode, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.15 ren_mode

The ren_mode procedure is used to read/modify the REN mode. If the REN mode is TRUE, then REN will be asserted at boot time.

```
struct Ren_Parms {
    unsigned    int    action;
    unsigned    int    ren;
};
struct Ren_Resp {
    unsigned    int    error;
    unsigned    int    ren;
};
```

Ren_Resp ren_mode (Ren_Parms) = 15;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The ren mode must be either 0 or 1. An error of 5 is returned and the current REN mode is unchanged if ren is determined to be invalid.

The returned Ren_Resp structure will always contain the current REN mode, irrespective of the error value.

* Note that this requires System Controller mode.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.16 eos_8bit_mode

The eos_8bit_mode procedure is used to read/modify the 8-bit EOS compare mode. If the 8-bit compare mode is TRUE, then EOS compare will be 8-bits. If 8-bit compare mode is FALSE, then EOS compare will be 7-bits.

```
struct Eos_8bit_Parms {
    unsigned    int    action;
    unsigned    int    eos8bit;
};
struct Eos_8bit_Resp {
    unsigned    int    error;
    unsigned    int    eos8bit;
};
```

Eos_8bit_Resp eos_8bit_mode (Eos_8bit_Parms) = 16;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The eos8bit mode must be either 0 or 1. An error of 5 is returned and the current 8-bit EOS compare mode is unchanged if eos8bit is determined to be invalid.

The returned Eos_8bit_Resp structure will always contain the current 8-bit EOS compare mode, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.17 auto_eos_mode

A4

The `auto_eos_mode` procedure is used to read/modify the automatic EOS on EOI mode. If the `autoEos` mode is `TRUE`, then an EOS character will be sent with EOI.

```
struct Auto_Eos_Parms {
    unsigned    int    action;
    unsigned    int    autoEos;
};
struct Auto_Eos_Resp {
    unsigned    int    error;
    unsigned    int    autoEos;
};
```

`Auto_Eos_Resp auto_eos_mode (Auto_Eos_Parms) = 17;`

The `action` value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

`action = 0` = read of current value

`action = 1` = modify current value

If the `action` value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The `autoEos` mode must be either 0 or 1. An error of 5 is returned and the current automatic EOS mode is unchanged if `autoEos` is determined to be invalid.

The returned `Auto_Eos_Resp` structure will always contain the current automatic EOS mode, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.18 eos_active_mode

The eos_active_mode procedure is used to read/modify the EOS active mode. If the EOS mode is TRUE, then an EOS character will terminate reads.

```
struct Eos_Active_Parms {
    unsigned    int    action;
    unsigned    int    eosActive;
};
struct Eos_Active_Resp {
    unsigned    int    error;
    unsigned    int    eosActive;
};
```

EosActive_Resp eos_active_mode (Eos_Active_Parms) = 18;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The eosActive mode must be either 0 or 1. An error of 5 is returned and the current automatic EOS mode is unchanged if eosActive is determined to be invalid.

The returned Eos_Active_Resp structure will always contain the current EOS mode, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.19 eos_char

The eos_char procedure is used to read/modify the EOS character.

```
struct Eos_Char_Parms {
    unsigned    int    action;
    unsigned    int    eos;
};
struct Eos_Char_Resp {
    unsigned    int    error;
    unsigned    int    eos;
};
```

Eos_Char_Resp eos_char (Eos_Char_Parms) = 19;

The action value determines whether the client wishes to execute a read of the current setting, or a modification of the current value.

action = 0 = read of current value

action = 1 = modify current value

If the action value is other than 0 or 1, then an error value of 5 is returned.

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The eos character must be in the range of 0x00 through 0xFF. An error of 5 is returned and the current EOS char is unchanged if eos is determined to be invalid.

The returned Eos_Char_Resp structure will always contain the current automatic EOS character, irrespective of the error value.

error	Meaning
0	No error
1	Syntax error
5	Parameter error

C.20 reload_config

The reload_config procedure is used to cause a reload of the configuration settings. Any modified configuration settings will be restored to default settings.

```
struct Reload_Config_Resp {  
    unsigned int error;  
};
```

```
Reload_Config_Resp reload_config (void) = 20;
```

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The returned Reload_Config_Resp.error value determines whether the default configuration was reloaded.

error	Meaning
0	No error
1	Syntax error

C.21 reload_factory

The reload_factory procedure is used to cause the Edevice to reset the default configuration back to factory loaded defaults. Any/all modifications to the default configuration are lost as a result. Note that dynamic in-memory configuration settings are not modified until a reload_config or reboot is executed.

```
struct Reload_Factory_Resp {  
    unsigned int error;  
};
```

Reload_Factory_Resp reload_factory (void) = 21;

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The returned Reload_Factory_Resp.error value determines whether the Edevice has reset the default configurations to the factory default settings.

error	Meaning
0	No error
1	Syntax error

C.22 commit_config

The `commit_config` procedure is used to cause the current configuration settings to be saved. Any modified configuration settings now become default settings and will be reloaded as the default settings with either `reload_config` or a reboot.

```
struct Commit_Config_Resp {  
    unsigned int error;  
};
```

```
Commit_Config_Resp commit_config (void) = 22;
```

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The returned `Commit_Config_Resp.error` value determines whether the current configuration was saved as the default configuration.

error	Meaning
0	No error
1	Syntax error

C.23 reboot

The reboot procedure is used to cause the Edevice to reboot. This causes all device links to be cleared, all connections closed, all resources released, and the default configuration to be loaded and used during initialization.

```
struct Reboot_Resp {  
    unsigned    int    error;  
};
```

```
Reboot_Resp reboot (void) = 23;
```

If the RPC message is of insufficient length to satisfy the required length, an error value of 1 is returned.

The returned `Reboot_Resp.error` value determines whether the Edevice has initiated a reboot process. Note that the timing of the reboot process may block the RPC reply.

* Note that certain configuration settings are only set at boot time. Thus when setting configuration settings, it is recommended that the Reboot command always terminates the configuration setting.

error	Meaning
0	No error
1	Syntax error

C.24 idnReply

The idnReply procedure is used to obtain a response similar to the GPIB *IDN? string. It contains the FW revision, the ICS product model number, and other miscellaneous information.

```
struct idn_Parms {  
    };  
struct idn_Resp {  
    unsigned    int    error;  
    unsigned    int    length;  
    unsigned    char    reply[128];  
    };
```

```
idn_Resp idnReply (idn_Parms) = 25;
```

The length determines the length of the data buffer. The maximum length is 128 bytes.

Error	Meaning
0	No error

C.25 errorLogger

The errorLogger procedure is used to obtain the current contents of the error log.

```
struct error_log_Parms {  
    };  
struct error_log_Resp {  
    unsigned    int    error;  
    unsigned    int    count;  
    unsigned    int    errors[100];  
    };
```

```
idn_Resp errorLogger (error_log_Parms) = 26;
```

The error log will contain 100 entries. The count will signify how many are valid. The remaining values will be of indeterminate values.

Note this function returns all entries and flushes the error log. Do not run this function more than 5 times per second to avoid impacting the 8065's performance and overloading the network.

Refer to the ErrorLogger utility for the error value definitions.

Error	Meaning
0	No error

This page intentionally left blank

A4

Index

Symbols

488.2

Common Commands Table of3-12

Operational Register3-10

Saving Enable Registers3-11

8003

Crossover connection2-3

Default Settings1-14

Description1-1

Hub connection2-5

Included Accessories1-20

Jumper Settings2-12

Model Specification1-3

Mounting Directions2-14

Outline Drawing1-16

Physical Specifications1-17

Rear Panel Mounting Dimen-
sions2-15

Signal Pin Assignment Ta-
ble2-21, 2-22

8013

Crossover Connection2-3, 2-9

Default Settings1-14

Description1-1

Example Configuration Settings2-35

Example Signal Assign-
ments2-33, 2-34

Hub Connection2-5, 2-9

Included Accessories1-20

Indicators1-15

Jumper Settings2-13

Mating Connectors2-16

Model Specification1-3

Mounting Instructions2-14

Optional Accessories1-20

Outline Drawing1-16, 1-18

Physical Specifications1-17

Rear Panel Mounting Dimen-
sions2-16

Signal-Pin Assignment Table2-23
Specifications1-3

A

Accessories1-20

Agilent SICL Library3-30

Agilent VISA3-29, 3-30

Auto Disconnect2-4

B

Bit Commands

Definition3-23

Programming example3-41

Block Diagram4-2

Block Diagram description4-2

Bulkhead Adapter Cutout2-14

Byte order2-18

C

- Channels1-4, A-12
- Commands
 - SCPI, exampleA-9
- Configurable Functions1-14
- Configuration
 - Settings2-4, 2-6
 - Utility2-8
- Configuration Chart
 - Example2-35
- Configuration Settings2-4, 2-6
- Conformance information
 - 488.23-12
 - SCPI3-15
- Connectors1-17
- Copyright Release3-56
- Crossover Cable Connection2-3, 2-9

D

- Data Strobe2-19
- Data Transfer Methods3-3, 3-35
- Default Settings1-14
- Description1-1
- device_intr_sq3-43, 3-44
- Digital Inputs
 - Generating SRQs from3-44
 - Monitoring changes3-8
 - Monitoring Signals3-9
 - Reading Input Signals3-9, 3-42
- Digital Interface
 - Configuring3-5
 - Data Lines1-9
 - Data Strobe1-9, 2-19
 - Description2-17
 - EDR1-9
 - Inputting Data3-3
 - LED Driver Outputs1-11
 - Monitoring1-10
 - Outputting Data3-4
 - Remote Output1-10, 2-20
 - Reset Output2-20
 - Specifications1-9
 - Stable Output1-11, 2-20

- Status Inputs1-10
- Timing Diagrams1-11
- Trigger Output1-10, 2-19
- Digital Interface Operation3-2
- Digital I/O
 - Data Strobe2-19
 - Design Guide2-29
 - Digital Lines2-17
 - Example2-32
 - Input Handshaking2-19
 - Output handshaking2-19
 - Overview2-17
 - Pin Assignments2-23
 - Remote Output2-20
 - Reset Output2-20
 - Stable Output2-20
 - Status Inputs2-19
 - Trigger Output2-19
- Digital Signals
 - Configuration Chart
 - Example2-35
 - Data Transfer Methods3-35
 - Design Example2-32
- Digital Waveforms3-45

E

- EDR1-9
 - Timing2-19
- EMI/RFI Warningii
- Error Codes
 - ESR error bits3-12
 - Self test5-7
 - VXI-11 Commands3-54
- ErrorLog Utility3-52
 - Error codes3-54
 - Running the utility3-52
- Ethernet
 - Port Usage1-5
- Ethernet Interface1-5
- Event Registers3-7, 3-8
- Example Configuration Settings2-35
- Example I/O Connections2-31
- Example Pin Assignments2-33
- External Reset Input1-11

F

Factory

- Default settings, recovering5-9
- Factory Default Settings1-14
- Firmware Settings3-55
- Firmware Updating5-9
- Functional Description4-1

H

- Handling SRQsA-13
- HP-UXA-11
- HTML Pages1-7
- Hub Connection2-5, 2-9

I

- IBM-AIXA-11
- IEEE 488
 - Message formats (IEEE 488.2)A-7
- IEEE-488
 - Bus DescriptionA-2
- IEEE-488.2A-5
 - Common CommandsA-6, A-7
 - Status Reporting StructureA-5
- IEEE 488.2 Common Commands
 - *CLS3-12
 - *ESE3-12
 - *ESE?3-12
 - *ESR?3-12
 - *IDN?3-13
 - *OPC3-13
 - *OPC?3-13
 - *PSC3-13
 - *PSC?3-13
 - *RCL3-13
 - *RST3-14
 - *SAV3-14
 - *SRE3-14
 - *SRE?3-14
 - *STB3-14
 - Table of3-12
 - *TRG3-14
 - *TST?3-14

*WAI3-14

- IEEE 488.2 STANDARDA-5
 - Common CommandsA-7
 - Differences from 488.1A-7
 - Message FormatsA-5
 - Reporting StructureA-5
- IEEE 488 Bus DescriptionA-2
 - IEEE 488.1A-2
- IEEE 488 Interface
 - 488.2 required status reporting capabilitiesA-4
 - SCPI command structure and examplesA-8
 - SCPI error reportingA-10
- Indicators1-15
 - inst03-1, 3-2, 3-35, A-12
 - inst13-2, 3-35, 3-46, A-13
- Installation2-1
- Installation guide2-2
- Interface Name1-4
- I/O Cable
 - Example2-32

J

- JavaA-11
- JAVA Programming3-27
- Jumpers2-12

K

- KeepAlive3-29
- Kikusui VISA3-29, 3-31

L

- LabVIEW
 - Raw Socket Notes3-34
- LAN Programming Guidelines3-27, 3-28
- LAN Timeouts3-27, 3-28
- Lead Free1-17, 1-19
- LED Driver Outputs1-11
- LEDs1-15
- Links1-4, A-12

LinuxA-11
Locking Setup Parameters3-47
Locks1-4, A-12
LXI differences1-2

M

Maintenance5-1
Mating Connector1-19
Mating Connectors2-16
MAX3-31, 3-34
maxRecvSizeA-14
Measurement & Automation Explorer3-32
Monitored Digital Inputs1-10
Mounting Instructions2-14

N

National Instruments VISA3-29, 3-31, 3-34
Network Settings
Resetting Defaults2-11
Nibble order2-18

O

OEM
Copyright wavier3-56
Documentation3-55
Operation
SCPI conformance information3-15
Theory of4-1
OS XA-11
Outline Dimensions1-16, 1-18
Output handshaking2-19
Output Queue3-10
Outputting Binary Data3-46

P

Physical Specifications1-17
Pin Assignment Table2-23
Port Commands
Program example3-37
Port usage1-5
Programming
Control Signals3-41, 3-42
Creating Waveforms3-45
Enabling SRQs3-43
Guidelines3-35
Individual Bytes3-40
Listen Handshake3-40
Listen String (Outputs)3-38
Locking/Unlocking Parametersparameters3-47
Monitoring Digital Inputs3-44
Multiple Data Sets3-39
Reading data with SRQs3-43
Reading Digital inputs3-42
Saving the setup3-48
SRQsA-13
Talk String (Inputs)3-37
Trigger Pulses3-43
User's IDN Message3-47
Programming Guidelines3-27
Pulse Commands
Definition3-23
Program example3-41

Q

Questionable Event Register3-8
Questionable Register
Monitoring Inputs3-44
Quick Installation Guide2-2

R

- Raw Socket
 - Operation3-5
 - Programming3-32
 - Specifications1-6
- Rear Panel Cutouts
 - for 80132-16
 - for Bulkhead adapter2-14
- Recovering Default Settings5-9
- Relay Contacts
 - Signal-pin assignments2-21
- Relay Driver
 - Signal-pin assignments2-21
- Remote Output1-10, 2-20
- Remote Procedure CallA-11
- Repair Procedure5-12
- Reset Output2-20
- Resetting Default Settings2-11
- Resetting Digital IO Configuration5-8
- Resetting Network Default Settings2-11, 5-8
- Reverse interrupt channel3-43
- Reverse Interrupt ChannelA-13
- Reverting to Factory Settings5-8
- RMA number5-12
- RoHS1-17, 1-19
- RPCA-11
 - VXI-11 RPCL messages3-49
- RPC Protocol1-4, A-15, A-19

S

- Sanitizing Procedure5-10
- SCPI
 - ABORT3-26
 - Channel list
 - ExamplesA-10
 - Command Reference Table3-20
 - Commands
 - ExampleA-9
 - Command structure and exampleA-8
 - Command Tree3-17
 - Compound commands example-
sA-9, A-10

- Conformance information3-15
- Error reportingA-10
- INITiate3-26
- STATus3-25
- SYSTem3-20
- SCPI CommandsA-8
- Self test
 - Errors5-1
- Self Test Error Codes5-7
- Service RequestsA-13
- Shipment verification2-1
- Short Form Commands3-16
- SICL Library3-30
- Simulated Encoder Waveforms3-45
- Sockets1-4, A-12, A-13
- SourceForge3-27
- Specifications1-3
- SRQsA-13
- Stable Output1-11, 2-20
- Status Byte Register3-11
- Status Inputs1-10
- SunOSA-11

T

- Theory of Operation4-1
- Timing Chart1-13
- Timing Diagrams
 - Data Outputs at power turn-on1-12
 - Digital Interface1-11
 - Outputs1-12
- Trademarksii
- Transferring DataA-14
- Transparent data transfer
 - Example3-37
- Triggering the External Device3-43
- Trigger Output1-10, 2-19
- Troubleshooting5-1
 - Guide5-3, 5-4, 5-5, 5-6
 - Operating Failures5-1
 - Self test errors5-1
- Troubleshooting Guide5-3

U

Updating Firmware5-9

Using VISA with Raw Sockets3-33

V

VISA

Agilent3-30

Libraries3-29

National Instruments3-31, 3-34

Resource String3-1

with Raw Sockets3-33

VISA Resource Strings3-2

VXI-11

ConceptA-11

Conformance1-4

Error chart5-6

Interface Name1-4

IntroductionA-11

Operation3-1

PC Communication Diagram3-29

RPC ProtocolA-15

Service RequestsA-13

Sockets, Channels and LinksA-12

Transferring DataA-14

VXI-11 Configuration Utility2-8

VXI-11 Conformance1-4

VXI-11 Keyboard Program3-49

W

Warrantyii

WebServer

described1-2

Pages3-55

Specifications1-7

Wiring Kits2-14