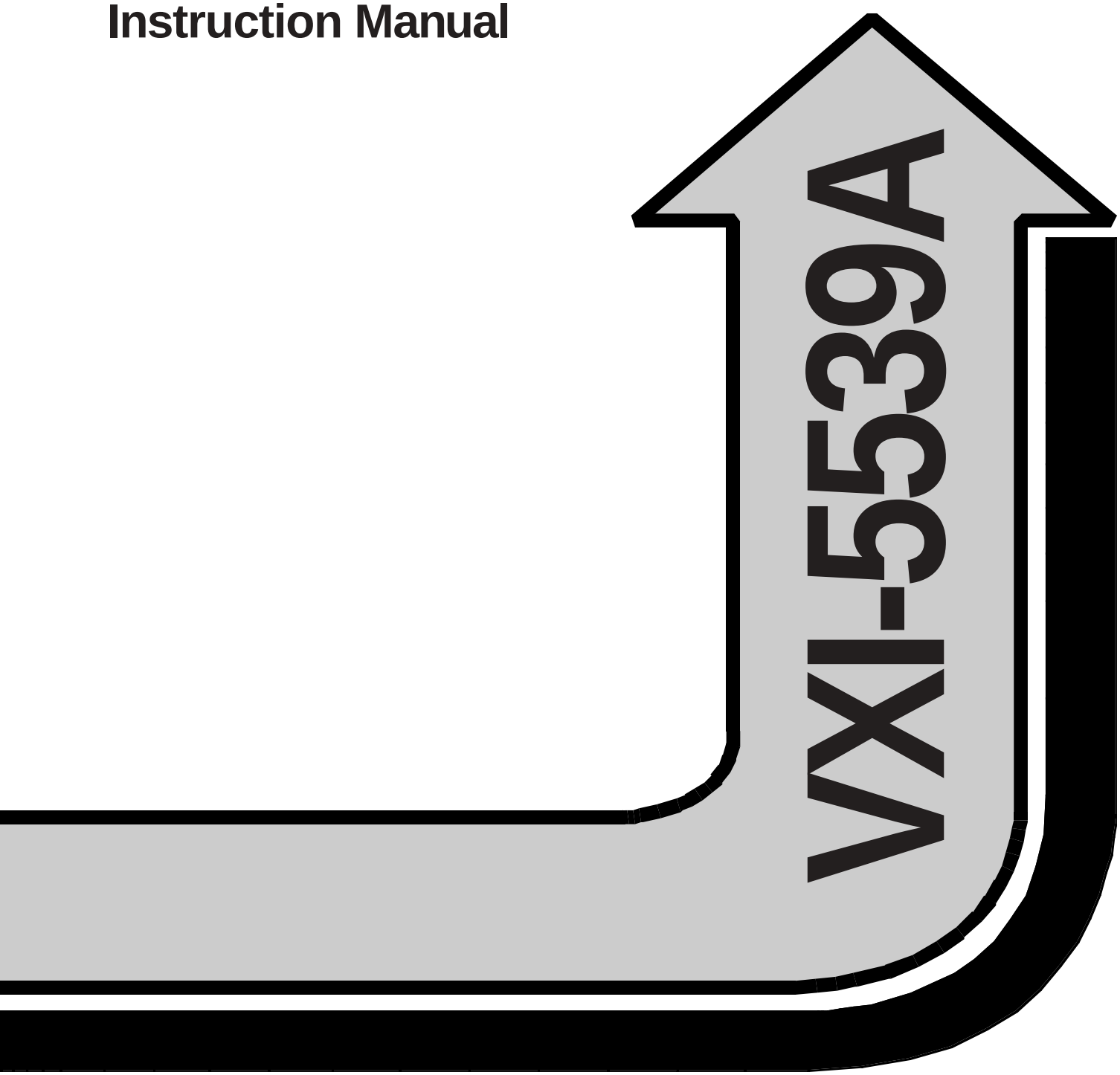
ICS
ELECTRONICS

# MODEL VXI-5539A
# VXIbus Quad and Octal Serial Module
# Instruction Manual



# MODEL VXI-5539A
# VXIbus Quad and Octal Serial Module
# Instruction Manual

ICS ELECTRONICS
*division of Systems West Inc.*
7034 Commerce Circle, Pleasanton, CA 94588
Phone 925.416.1000, Fax 925.416.0105
Website http://www.icselect.com

Publication Number 120158
November 2004 Edition Rev 0.2

## 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. It has 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:

HP is a trademark of Hewlett-Packard Corporation, Palo Alto, CA
ICS is a trademark of ICS Electronics div Systems West, Inc., Milpitas, CA
MXIbus is a trademark of National Instruments, Austin, TX

## COPYRIGHTS

This product contains software with the following copyrights:

Flash Loader Utility - Copyright 1994, 95, 96 Cyber Quest, Inc., All Rights Reserved

## REVISION NOTES

# Contents

## TABLES



## FIGURES

# 1

# Specifications

## 1.1 DESCRIPTION

### 1.1.1 VXI-5539A Module

The Model VXI-5539A is a single slot, C-size VXI module with four or eight independent serial channels for transmitting and receiving serial messages or data. Any or all of the serial channels can be active at the same time. Data transfer between the VXI-5539A and the VXI controller is via VXI Fast Data Channels to minimize VXIbus transfer time and assure continuous data flow when operating at high baud rates. Each channel has two modes of operation, Asynchronous and SDLC.

The Asynchronous mode handles asynchronous data characters like those sent from a PC COMM port or from a CRT terminal. Asynchronous characters have a start bit, data bits, an optional parity bit and a stop bit. Messages are composed of one or more characters and do not have a character limit. Baud rates can be individually programmed for each channel in standard baud rates from 50 baud up to 460.8 Kbaud. The module generates nonstandard baud rates by dividing down from 460,800 Hz and selecting the closest divider ratio for the commanded rate.

The SDLC mode transmits data bytes in packets with a CRC checksum. SDLC characters have 8 data bits and no start or stop bits. The VXI-5539A's SDLC packet can hold from 2 to 30 data bytes. The packet size is set by the Frame:Size command. The SDLC packet format is:

Start byte - Data Bytes - CRC Bytes - End byte

SDLC data rates can be up to 1 Mbs. The transmit 1X clock is normally transmitted with SDLC data. The data from each received packet is placed in the FDC data buffer and is immediately followed by the 16-bit status byte from the channel's UART. The status byte contains information about the packet and identifies the packet as being a good or bad packet.

The VXI-5539A module is a Message Based I4 class VXI instrument with interrupter capability and supports all of the required VXIbus Word Serial Commands under the VXIbus Specification VXIbus-1, Rev. 1.4. Control and setup commands are Word Serial Messages and Word Serial Commands. Serial data is normally sent and received via Fast Data Channel buffers in A32 address

space. Test messages can be sent and received as Word Serial Messages. The VXI-5539A should only be used with a Slot 0 Controller that supports the A32 address space.

The data input and output FDC channels operate as *A/B buffer pair*s and in *Stream Transfer* mode for enhanced throughput. Each channel uses two FDC buffers for transmit and two for receive. This makes a total of thirty-two FDC channels used to transfer the serial data over the VXIbus. The VXI-5539A module supports Fast Data Channel (FDC) data transfer per VXIbus Specification VXIbus-10, Rev. 2.10 using the Standard FDC Word Serial Commands and an expanded command set that handles up to 32 FDC channels. In addition the module supports SCPI commands (1994.0 ) and IEEE Std 488.2-1987 commands. Refer to ICS's Application Note, AB55-5, for more information about the VXI Fast Data Channel operation and commands.

### 1.1.2 VXIbus Description

#### 1.1.2.1 VXIbus Objectives

The goal of the VXIbus Consortium is to create an open industry standard for modular instruments by defining interoperability between vendors, mechanical and environmental requirements, EMC compatibility, system initialization and software communication protocols. The physical portion of the VXIbus specification was adapted from the existing VME bus specification (IEEE-STD 1014). VXI is an acronym for "**V**ME bus E**x**tensions for **I**nstrumentation."

The VXIbus specification details the technical requirements of VXIbus compatible components such as mainframes, backplanes, power supplies and modules. The specification also provides for interconnecting and operating different manufacturers' products within the same chassis. The latest revision of the specification is version 1.4. The success of the specification is evidenced by over 300 manufactures who make over 800 different VXIbus products and hundreds of users .

The IEEE Standards Committee, in its IEEE-STD 1155, has adopted the VXIbus consortium's specifications. The U. S. Air Force has also accepted the specifications as the basis for its Modular Automatic Test Equipment (MATE) Instrument on a Card (IAC) standard.

The VXIplug&play Alliance has defined several additional standards that simplifies the integration of multi-vendor VXI systems. The VXIplug&play Alliance has created a standard system framework concept that allows test programs in any language and operating system to be able to control VXI chassis and instrument modules through Standard Instrument Drivers. The VXIplug&play Framework identifies the operating system (OS) and applications development environment (ADE) used to generate the test software. The VXIplug&play Alliance just adopted an updated specification for a set of standard VISA Transition Library drivers that the Slot 0 Controllers and Embedded Computer vendors should adhere to assure VXIplug&play compatibility. The VISA (Virtual Instrument Software Architecture) drivers are used between the end user's test application program or general purpose test programs purchased from software vendors and the physical VXI modules or GPIB instruments. VISA Transition Library (VTL) and full VISA drivers are described in the VXIplug&play VPP-4.2 Specifications.

### 1.1.2.2 Advantages of VXIbus Based Systems

The VXIbus provides the user with the following advantages:

Higher density packaging
Increased system throughput
More precise timing and device synchronization
Standard protocols for instrument communication and control
Ability to utilize existing VME modules
Lower costs due to shared resources.

### 1.1.2.3 VXIbus System Configurations

VXIbus systems utilize a chassis with a backplane and a common power supply. Modules plug into the chassis from the front and communicate to each other over the backplane. The left most slot in each chassis is labeled slot 0 and it is reserved for the system or chassis controller. Modules in the other slots are servants to the system controller but can also be controllers and have their own servants. The Slot 0 controller must be capable of performing the resource manager function which initializes the other modules and assigns logical addresses for dynamically addressed devices, interrupt lines, and trigger lines. The Slot 0 controller may be an embedded computer or it may simply be a translator module driven by an external computer.

Each VXI device is addressed by its logical address. The address may be static and preset by the user or dynamic and set by the resource manager function during system initialization. The VXIbus specification allows for 256 logical addresses. Address 0 is reserved for the Slot 0 controller and address 255 is reserved for dynamic addressable devices that will have their address defined by the resource manager. A VXI system can contain a maximum of 254 logical devices.

A module may be a single logical device or contain multiple logical devices. Physically the module can also be one slot wide or occupy multiple slots. Full rack wide VXI chassis have 13 slots on a 1.2 inch centers and can hold up to 13 one slot wide modules. Chassis extenders allow multiple chassis to be interconnected together producing systems of up to 254 logical devices.

VXI based systems can also incorporate non-VXIbus devices. The most common variations are the inclusion of GPIB instruments in the system or the use of VME cards in the VXI chassis. Slot 0 controllers commonly have a GPIB interface for controlling the GPIB instruments so that their operation can be controlled from the same program as are the VXI modules. VME cards can function in a VXI chassis because the VXIbus Specification maintained compatibility with the VME bus by retaining the VME signals definitions for P1 and the center row of P2. Provisions were also made to address the registers in the VME modules just as they are currently addressed in a VME bus system.

### 1.1.2.4 Data Transfer Methods

VXI modules can be register or message based . Register based modules are typically controlled by direct reads or writes to registers in the module. Message based modules communicate with word serial messages that are strings of ASCII or binary bytes. Word transfer uses the VXI word serial protocol that examines bits in the modules's response register to maintain an orderly data transfer.

Because of the word serial protocol, register based modules are typically faster than message based modules but they lack the intelligence of message based modules. A new Fast Data Channel specification, VXI-1, provides for direct transfer of data from a module's memory to the Slot 0 Controllers at rates up to 32 Mbytes per second. This allows smart message based modules to have the same high data transfer rates as do register based modules.

#### 1.1.2.5 Additional information about the VXIbus

For additional information, contact the VXI Consortium for a copy of the VXI specification and the VXI Fast Data Channel specification. Contact the VXIplug&play Alliance for a copy of the VISA specification or ICS Electronics for Application Bulletins that describe various VXI applications.

## TABLE 1-1 VXIBUS GLOSSARY

**Commander:** A VXIbus device that has VME bus master capability and may have VXIbus servants under it in the system hierarchy. A Commander may act as a Servant to another Commander. A Commander must be message based.

**Servant:** A VXIbus device (with or without VME bus master capability) that is under control of a Commander in the VXIbus system hierarchy. A Servant may also be a Commander to other Servants. A Servant may be either message or register based.

**Interrupt Handler:** The module in the VXIbus system that generates the hardware interrupt acknowledge for a particular VME interrupt level. In VXIbus, the software interrupt handler may or may not be on the same module as the hardware interrupt handler.

**Logical Address:** A unique 8 bit number (0-255) which identifies each VXIbus device in a system. It defines the device's A16 register addresses

**Resource Manager:** A message based commander located at logical address 0 which provides configuration management services, including address map configuration, Commander/Servant mapping, self test, and diagnostic management.

**VXI Message Based Instrument:** An intelligent instrument that implements the defined VXIbus registers and, at a minimum, word serial protocol.

**VXI Word Serial Messages:** The simplest required communication protocol supported by Message Based devices in a VXIbus system. It utilizes the A16 communications registers to transfer data or commands as a series of characters on the VXIbus backplane. The end bit is asserted on the last character of the message. Uses Word Serial Commands: Byte Available and Byte Request.

**VXI Word Serial Commands:** Single word, 16-bit commands sent from the commander to its servants. Some Word Serial Commands have a response word. The VXI specification defines a number of standard Word Serial commands that are reserved for use by the Slot 0 Controller or the Resource Manager. The specification also allows instrument designers to define their own Word Serial Commands.

**VXI Commands:** These are commands passed from a Commander to a Servant within the VXIbus environment. There are three broad categories of commands: VXIbus Instrument Protocols, IEEE 488.2 Common Commands, SCPI commands, and Device specific commands. A command may or may not be stimulated by an external event. For example an IEEE-488 Group Execute Trigger will generate a trigger command to all addressed devices. However, a Begin Normal Operations command is generated by the VXIbus resource manager and has no external source.

**VXI Events:** VXIbus Events are passed from a Servant to a Commander. They may be generated by the Servant either in response to a command (e.g., an invalid command error), or due to an external condition (e.g., data ready or status change).

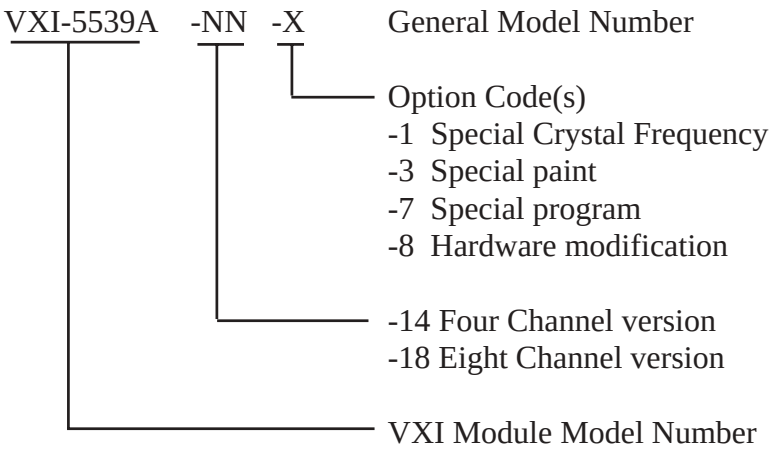
**VXI Fast Data Channel:** A method for exchanging data between a commander and a servant module that utilizes a minimum of handshaking to transfer data so that the data transfer rate approaches the theoretical VME bus transfer rate. Allows a commander access to portions of the device's memory or a registers in the A32 address space. Data transfer is unidirectional for each channel and can be D16 or D32 bit words. Multiple channels may be opened for each device using A/B channel pairs for continuous data transfer.

**488-VXIbus Interface Device:** An IEEE-488 to VXIbus Interface Device is a message based device which provides communication between the IEEE-488 bus and VXIbus instruments. Typically this function is included in the Slot 0 card for external control of the VXI chassis.

## 1.2 DETAIL SPECIFICATIONS

The following specifications apply in general to all VXI-5539A modules.

### 1.2.1 Model Number Designation




### 1.2.2 VXIbus Capabilities

The VXI-5539A has the following VXIbus capabilities:

| | |
|---|---|
| Addressing | Static configured addresses 1-254 or Dynamic configuration |
| Manufacturer ID | 4073 (ICS Electronics) |
| Device Class | Message based I4 class VXI instrument |
| Address Space | A16 (D16) / A32 (D32)<br>64 bytes in A16 space<br>1 or 4 Mbytes in A32 space |
| Model Code | 12 bits, 539 decimal for Model 5539A |
| Interrupter | D16, Programmable Interrupter |
| Event Generator | Programmable, Interrupts only |
| FDC Event Generator | Programmable, Interrupts only |
| Response Generator | Not Supported |
| VMEbus Master | Not Supported |
| Commander | Not Supported |
| Signal Register | Not Supported |
| Handshake | Normal handshake only |
| Fast Data Channel | A32 (D32) Random Access,<br>16 Channels: I/O Pair & Stream Transfer mode,<br>Standard FDC Word Serial Commands and ICS Expanded FDC Word Serial Commands per FDC Addendum |
| FDC Transfer Rate | approximately 4 Mbytes per second |
| Primary Classification | I4 |

### 1.2.3 VXIbus Instrument Protocols

The VXI-5539A is a message based instrument that supports the VXIbus instrument protocols for operation as an I4 class VXI instrument and Fast Data Channels, using the VXIbus word serial protocol and 488.2 common commands. As such it supports the following instrument protocol commands as described below and ICS's expanded FDC command set in the FDC Addendum.

**TABLE 1-2 SUPPORTED INSTRUMENT PROTOCOLS**

| Command | Description |
|---|---|
| Byte Available | Sends Word Serial Messages to the VXI-5539A. |
| Byte Request | Reads command responses or data from the VXI-5539A. |
| Clear | The clear command causes the VXI-5539A to clear its internal buffers, reset its command parser to its initial state. Does not affect the Fast Data Channel buffers or other buffers. |
| Trigger | The Trigger command causes the unit to execute commands in the program buffer specified by the Arm Trigger command. |
| Set and Clear Lock | Not supported by the VXI-5539A, no front panel controls. |
| Read STB | Response is a byte equivalent to the serial poll and *STB? response in IEEE Standard 488.2. |
| Abort Normal operation | Standard response. |
| Begin Normal operation | Standard response. |
| End Normal operation | Standard response. |
| Read Protocol | Standard response. |
| Read Protocol Error | Standard response. |
| Assign Interrupter Line | Standard response. |
| Read Interrupter Line | Standard response. |
| Read Interrupters | Standard response. |
| Asynchronous mode control | Standard response. |
| Control Event | Standard response. |
| Arm Trigger | An ICS defined command (0x02bbb ettt) used to specify the TTL Trigger line (ttt) that will be used to run buffer (bbbb) when the VXI-5539A is sent a TTL Trigger or a Word Serial Trigger command. The buffer is one of the VXI-5539A's program storage buffers. Bits 4-7 are the buffer number. Bit 3 is the enable bit. Bits 0-2 specify the TTL Trigger line. No response |
| Channel Address High | Standard response. |
| Channel Address Low | Standard response. |
| Channel Close | Standard response. |
| Channel Initialize | Standard response. |
| Channel Size High | Standard response. |
| Channel Size Low | Standard response. |
| Enable Passed Buffer | Standard response. |
| Passed Buffer | Standard response. |
| FDC Event | Standard response. |
| FDC Supported | Standard response. |
| Go to Idle | Standard response. |
| Transfer to Commander | Standard response. |
| **Transfer to Servant** | **Standard response.** |

### 1.2.4 VXIbus Fast Data Channels

The VXI-5539A supports up to thirty-two VXI Fast Data Channels for transferring data from the VXI-5539A's serial channels to the Slot 0 Controller. The channels are used as A/B pairs in stream mode for continuous data transfer. Even numbered channels are the A channel. The channel assignments and their maximum size is shown in Table 1-3. The user can set the receiver buffer size to a value less than the maximum shown in the Table. Buffer starting addresses are preassigned and do not change if the buffer size is changed.

The VXI-5539A supports the standard VXI Fast Data Channel Commands and ICS's expanded Fast Data Channel command set for setting up the FDC Channels and controlling their operation. Both commands sets are described in the Fast Data Channel Addendum. ICS's expanded command set is preferred because the standard VXI FDC Commands only support eight FDC channels and have limited usefulness with the VXI-5539A.

**TABLE 1-3 FDC CHANNELS**

| FDC Channel | Function | Buffer Sizes | |
|---|---|---|---|
| | | Four Channels | Eight Channels |
| 0 | Ch 1 Receive | 128 Kbytes | 64 Kbytes |
| 1 | Ch 1 Receive | 128 Kbytes | 64 Kbytes |
| 2 | Ch 1 Transmit | 128 Kbytes | 64 Kbytes |
| 3 | Ch 1 Transmit | 128 Kbytes | 64 Kbytes |
| 4 | Ch 2 Receive | 128 Kbytes | 64 Kbytes |
| 5 | Ch 2 Receive | 128 Kbytes | 64 Kbytes |
| 6 | Ch 2 Transmit | 128 Kbytes | 64 Kbytes |
| 7 | Ch 2 Transmit | 128 Kbytes | 64 Kbytes |
| 8 | Ch 3 Receive | 128 Kbytes | 64 Kbytes |
| 9 | Ch 3 Receive | 128 Kbytes | 64 Kbytes |
| 10 | Ch 3 Transmit | 128 Kbytes | 64 Kbytes |
| 11 | Ch 3 Transmit | 128 Kbytes | 64 Kbytes |
| 12 | Ch 4 Receive | 128 Kbytes | 64 Kbytes |
| 13 | Ch 4 Receive | 128 Kbytes | 64 Kbytes |
| 14 | Ch 4 Transmit | 128 Kbytes | 64 Kbytes |
| 15 | Ch 4 Transmit | 128 Kbytes | 64 Kbytes |
| 16 | Ch 5 Receive | | 64 Kbytes |
| 17 | Ch 5 Receive | | 64 Kbytes |
| 18 | Ch 5 Transmit | | 64 Kbytes |
| 19 | Ch 5 Transmit | | 64 Kbytes |
| 20 | Ch 6 Receive | | 64 Kbytes |
| 21 | Ch 6 Receive | | 64 Kbytes |
| 22 | Ch 6 Transmit | | 64 Kbytes |
| 23 | Ch 6 Transmit | | 64 Kbytes |
| 24 | Ch 7 Receive | | 64 Kbytes |
| 25 | Ch 7 Receive | | 64 Kbytes |
| 26 | Ch 7 Transmit | | 64 Kbytes |
| 27 | Ch 7 Transmit | | 64 Kbytes |
| 28 | Ch 8 Receive | | 64 Kbytes |
| 29 | Ch 8 Receive | | 64 Kbytes |
| 30 | Ch 8 Transmit | | 64 Kbytes |
| 31 | Ch 8 Transmit | | 64 Kbytes |

### 1.2.5 IEEE 488.2 Common Commands

The VXI-5539A supports the following VXIbus IEEE 488.2 Common Commands to perform functions such as service request enable, event status register query and status byte query, etc. IEEE 488.2 Common Commands begin with an asterisk (*), and may include one or more parameters. Detailed descriptions of the commands can be found in Section 3 and in the IEEE Std 488.2-1987 Standard.

*CLS, *ESE, *ESE?, *ESR?, *IDN?, *OPC, *OPC?, *PSC, *PSC?, *RCL, *RST, *SAV, *SRE, *SRE?, *STB, *TRG, *TST?, *WAI

#### 1.2.5.1 STB Response

The VXI-5539A responds to the Read STB command by sending a byte equivalent to the serial poll and *STB? response in IEEE Standard 488.2. The VXI-5539A's Read STB response byte has the following bit assignments:

| Bit | STB Functions | ESR Functions |
|---|---|---|
| 7 | OPERation Status | PON, Power On |
| 6 | MSS, Master Service Summary | URQ, User Request, not used |
| 5 | ESB, Event Status Summary bit | CME, Command Error |
| 4 | MAV, Message Available bit | EXE, Execution Error |
| 3 | QUEStionable Status | DDE, Device Dependent Error, not used |
| 2 | Not used | QYE, Query Error |
| 1 | Not used | RQC, Request Control, not used |
| 0 | Not used | OPC, Operation Complete, not used |

The on condition of the corresponding bit in the RQS mask byte (set by *SRE command) enables generation of a Request Service Interrupt at the next occurrence of the unmasked bit. Bits with 0's in the RQS mask byte will not generate an interrupt, but they will be reported in the STB response byte.

### 1.2.6 SCPI Commands

The VXI-5539A supports SCPI commands that allow the user to configure the module, set the data buffer sizes and control the Questionable and Conditional registers in the VXI-5539A's Status Reporting Structure. The SCPI commands conform to the SCPI 1995.0 Standard and are listed in Table 3-5. Table 1-4 lists the VXI-5539A's configurable parameters and the factory settings.

#### 1.2.6.1 Operational Register

The VXI-5539A provides an Operational Register that reports the status of the Waiting for Trigger bit. The register structure is a four register set with a Condition register, Transition register, Event register and an Enable register. Refer to section 3.5 for a description of the registers' operation. Bit assignments are:

Bit 5 Waiting for Trigger

#### 1.2.6.2 Questionable Register

The SCPI Questionable Register is not used in the VXI-5539A module. The register structure is inplace and the commands are supported by the VXI-5539A's parser so their use will not report a command error. The Questionable Register commands have no functions in the VXI-5539A module.

**TABLE 1-4 VXI-5539A DEFAULT PARAMETER SETTINGS**

| Keyword | Description | ASYNC Mode | SDLC Mode |
|---|---|---|---|
| PROTOCOL | Serial message/character type | ASYNC | - |
| FRAMe:SIZe | Sets number of data bytes in a packet | n/a | 30 |
| BAUD | Serial bit rate | 9600 | 1000000 |
| BITS | Data bits per character | 8 | n/a |
| SBITS | Stop bits | 1 | n/a |
| PARITY | Parity | NONE | n/a |
| EOMChar | End of Message Character | LF | n/a |
| ECHO | External Message Loopback | OFF | OFF |
| LOOPBACK | Internal Message Loopback | OFF | OFF |
| MODE | Serial Interface Signal Type | 232 | 422FD |
| HANDSHAKE | Enables RS-232 handshake signals | OFF | n/a |
| RXCLK | RX Clock Source | INT | INT |
| TXCLKOUT | TX Clock Transceivers | IN | OUT |
| TERMINATION | RS-485 Termination Network | OFF | n/a |
| DMA | Enables DMA for receive data | OFF | n/a |
| EOMChar | Sets EOM character,enables messages | 10 | n/a |
| MESSages | Sets Rx FDC buffer size in messages | 1 | n/a |
| BUFFer:SIZE | Sets Rx FDC Buffer size in bytes | 65,535 | 65,535 |
| FREQUENCY | Clock rate into the UARTs | 7.3728 | 7.3728 MHz |
| PERIOD | Pulse period | 0.01 sec | 0.01 sec |
| WIDTH | Pulse width | 0.0025 sec | 0.0025 sec |

### 1.2.7 Serial Interfaces

The VXI-5539A provides four or eight serial interfaces for Asynchronous and SDLC communication with external devices. Each interface supports RS-232, RS-422 or RS-485 serial links.

#### 1.2.7.1 Modes

Asynchronous A character oriented protocol where each character has a start bit, data bits, an optional parity bit and one or more stopbit(s). There is no limit to the number of characters in a message. A null fill character is automatically inserted in the FDC buffer after 3 character times if the mesage contains an odd number of characters. Use RS-232, RS-422 or RS-485 signals. Maximum module data rate for all channels is 460,000 baud.

| | |
|---|---|
| Data bits | 5, 6, 7 or 8 bits/character |
| Parity | Odd, even or none |
| Stop bits | 1 or 2 |

SDLC A packet oriented protocol where each packet contains 2 to 30 eight-bit data bytes. The packets have a start byte, data bytes, a 16-bit CRC checksum (CRC-16) and a stop identifier. There are no start bits or stop bits between the characters. Maximum data rate for all channels is 1 Mbs. The SDLC packet format is:

Start byte-data bytes-CRC-End byte

#### 1.2.7.2 RS-232 Signals

Single ended signals referenced to signal ground. DTE signal configuration.

| | |
|---|---|
| Signals | BA, BB, CA, CB, CD, and CF<br>Signal ground is AB. Shield is connector shell<br>Handshaking enabled/disabled by separate command<br>CA is always on.<br>CD on when enabled and FDC receive buffers enabled or on when handshaking is disabled.<br>CB required for transmission when handshaking enabled.<br>CF required for receiving when handshaking enabled. |
| Transmit Levels | +6 ± 1 Vdc = logic '0' or On<br>-6 ± 1 Vdc = logic '1' or Off |
| Receive Levels | ± 1.5 Vdc minimum<br>± 15 Vdc Maximum |

| | |
|---|---|
| Mode | Full duplex |

### 1.2.7.3 RS-422 Signals

| | |
|---|---|
| Signals | TX, RX, TXCLK, RXCLK balanced line signal pairs<br>Signal ground is SG. Shield is chassis ground |
| Transmit Levels | +4 ± 1 Vdc differential for binary '0' or on<br>-4 ± 1 Vdc differential for binary '1' or off |
| Receive Levels | ± 0.2 Vdc minimum<br>± 10 Vdc maximum differential or between any signal and SG. |
| Modes | Full and half duplex with optional RX and TX clock signals<br>RXCLK is receive only when enabled.<br>TXCLK can be receive or transmit. |
| Termination | Termination network can be switched across the TX lines to bias the TX lines in the mark state when the transmitter is tristated. |

### 1.2.7.4 RS-485 Signals

| | |
|---|---|
| Signals | TX and RX on a single balanced signal pair<br>Signal ground is SG. Shield is chassis ground |
| Transmit Levels | +4 ± 1 Vdc differential for binary '0' or on<br>-4 ± 1 Vdc differential for binary '1' or off |
| Receive Levels | ± 0.2 Vdc minimum<br>± 10 Vdc maximum differential or between any signal and SG. |
| Mode | Half duplex only |
| Termination | Switchable on to TX lines |

### 1.2.7.5 Baud Rates

Baud rates for asynchronous characters can be generated from the internal 7.3728 MHz oscillator or from and eternal RXCLK input. Standard internally generated rates start at 50 Hz and include: 110, 300, 600, 1200, 2.4 K, 4.8 K, 7.2 K, 9.6 K, 19.2 K, 38.4 K, 57.6 K, 76.8 K, 115.2 K, 153.6 K, 230.4 K and 460.8 Kbaud. Baud rate accuracy is ± 0.02 %. Unit selects closest integer value for nonstandard rates by dividing down from 460,800 Hz.

Baud rates for SDLC generated from RXCLK or from internal 7.3728 MHz oscillator.

#### 1.2.7.6 Serial Buffers

See FDC buffer specifications in Table 1-2. DATA and DATAH messages limited to 1024 bytes.

#### 1.2.7.7 Termination Resistor Network

A 150 ohm load with 2.2 Kohm pullup and pulldown resistors to +5 Vdc and ground.

#### 1.2.7.8 Diagnostic Capabilities

Internal loopback for VXIbus test messages with or without transmission of the test messages. DATA is a Word Serial Message for transmitting test strings. DATA? reads back any data in the receive FIFO. When loopback is enabled, DATA? reads back the DATA message. DATA? only works in Asynchronous mode.

### 1.2.8 Timing Pulse Output

Eight channel units have a RS-422 Timing Pulse Output which can be user programmed.

| | |
|---|---|
| Period | 0.0001 to 3.0000 seconds |

Pulse Width 0.0001 to 3.0000 seconds

### 1.2.9 Front Panel Indicators

#### 1.2.9.1 Alphanumeric Display

The VXI-5539A has a four character alpha numeric display on the front panel which displays module status, logical device addresses and diagnostic messages. Normal mode display messages are listed below. The complete message set is listed in Section 3.

| Display | Meaning |
|---|---|
| SysF | SYSFAIL asserted, Self test in progress |
| Init | Unit Initialized and in configure state, Self test passed |
| BNO | Begin Normal Operation received |
| Rdy | Ready State |
| SERn | Serial Channel n addressed |
| FAIL | Failed Self Test |
| ERnn | Error Code Number (nn) |

### 1.2.10 Physical

| | |
|---|---|
| Size | C size VXI module |
| Dimensions | 352.43 mm long x 233.35 mm high x 30.18 mm wide |
| Weight | 1.79 kg (3.94 lbs) including RF shields |
| Power | 5 Vdc at 3 A max<br>±12 Vdc at 0.01A |
| Temperature | 0 °C to 55 °C operating<br>-20 °C to 70 °C storage |
| Connectors | Four Channel Units:<br>Four 9-pin DE-9P Connectors with lock studs<br><br>Eight Channel Units:<br>Two 37-pin, DC 37P Connectors with lock studs |

## 1.3 ACCESSORIES

### 1.3.1 Supplied Accessories

The following accessories are supplied with each VXI-5539A module.

| Qty. | Part Number | Description |
|---|---|---|
| 1 | 120158 | VXI-5539A Instruction Manual. |
| 2 | 902050 | DC-37S Mating Connector (8 Channel unit only) |
| 2 | 902105 | DC Hood (8 Channel unit only) |

### 1.3.2 Optional Accessories

The following accessories are available for use with the VXI-5539A module.

| Part Number | Description |
|---|---|
| 120158 | Spare VXI-5539A Instruction Manual. |

# 2

# Installation

## 2.1 UNPACKING AND INSPECTION

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

## 2.2 SHIPMENT VERIFICATION

Take a moment to verify that you have everything you need. If you ordered a standard VXI-5539A module, we should have sent you:

| | | |
|---|---|---|
| (1) | VXI-5539A-14 or -18 | VXI Module |
| (1) | 120158 | VXI-5539A Instruction Manual |

If anything is missing or defective, please contact ICS Electronics immediately.

## 2.2 INSTALLATION

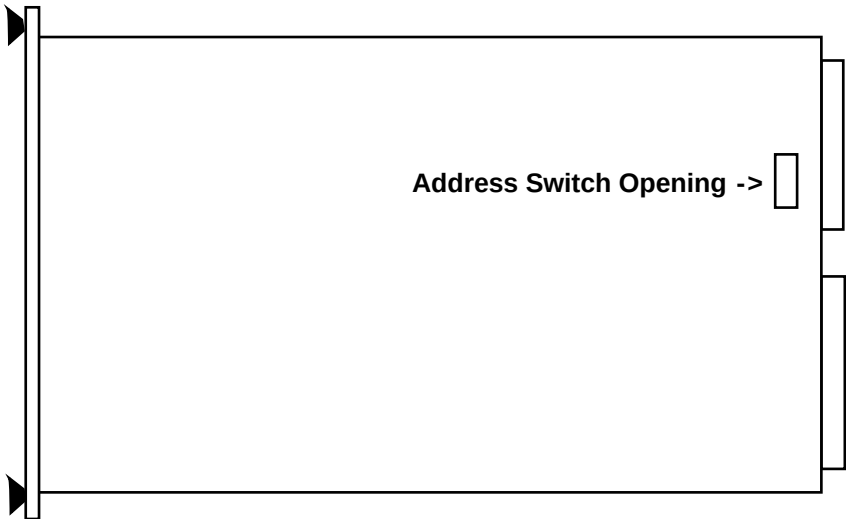
The VXI-5539A Quad Serial Module is ready for RS-232 and RS-422/RS-485 operation when shipped.

To install the module in a C size VXI chassis, select an empty slot and remove the slot cover plate. Turn off chassis power. Set the module's logical address as described in paragraph 2.3 before installing the module in a VXI chassis. Slide the module into the chassis with the LEDs up (or to the left in the case of a horizontal chassis) until the connectors start to engage the backplane connectors. Press the module firmly into the backplane connectors until the front panel stops at the chassis rails.

## 2.3 INSTALLATION

### 2.3.1 Address Switch Setting

The VXI-5539A has an internal address switch for setting its logical address. The switch is accessible from the side of the module when the unit is outside the VXI chassis. Set the VXI-5539A's address switch to any unused value between 1 to 254 for static addressing or to 255 for dynamic address assignment. Figure 2-1 shows the switch layout and rocker bit weights. The address switch is set to logical address 16 (HEX 10) by the factory.




Logical address switch set to 10 Hex

**Figure 2-1 Address Switch Location**

**Figure 2-2 Address Switch Layout**

### 2.3.2 Checking the VXI-5539A's Logical Device Address

To check the address setting of the module while the unit is installed in a chassis, press the DISPLAY MODE button with a small thin nonconducting object (a tooth pick will do) until **Addr** is displayed, then release the button. The display will indicate the selected address in the form **A=xx**, where xx is the VXIbus address in hexadecimal notation.

## 2.4 SERIAL INTERFACE

### 2.4.1 Four Channel Units

The VXI-5539A-14 Quad Serial Module has four 9-pin DE-9P connectors with lock studs that independently provide either RS-232, RS-422 or RS-485 signals. Table 2-1 shows the serial signal assignments and the signal direction relative to the VXI-5539A.

**TABLE 2-1 SERIAL INTERFACE PIN ASSIGNMENTS FOR FOUR CHANNEL UNITS**

| Channel Number | Pin Number | Signals and Direction | | | | | |
|---|---|---|---|---|---|---|---|
| | | RS-232 | | RS-422 FD/HD | | RS-485 HD | |
| 1 to 4 | 1 | DCD | ← | Rx(B)+ | ← | - | |
| | 2 | Rx | ← | Rx(A)-- | ← | - | |
| | 3 | Tx | → | Tx(A)- | → | Tx/Rx- | ↔ |
| | 4 | DTR | → | Tx(B)+ | → | Tx/Rx+ | ↔ |
| | 5 | DGND | | DGND | | DGND | |
| | 6 | (DSR) | | RxCLKO+ | ← | RxCLKO+ | ← |
| | 7 | (RTS) | | RxCLKO- | ← | RxCLKO- | ← |
| | 8 | CTS | ← | TxCLKI+ | ↔ | TxCLKI+ | ↔ |
| | 9 | - | | TxCLKI- | ↔ | TxCLKI- | ↔ |
| | Shell | Chassis | | Chassis | | Chassis | |

Note: Signal direction arrows ← = in to VXI-5539A, → = out from VXI-5539A

### 2.4.2 RS-232 Connections

When RS-232 signals are selected, the signal pinouts are compatible with those on the 9-pin COM port of an IBM type computer. The VXI-5539A's RS-232 signals can be expanded out to a 25 pin connector with a 9 pin to 25 pin adapter. The RS-232 column in Table 2-1 above shows the RS-232 signal pin assignments and their direction relative to the VXI-5539A. Figure 2-3 shows the wiring diagram for a typical 9 pin to 25 pin adapter. For minimum RS-232 connections, use the serial signals on pins 2, 3 and 5. Jumper the unused RS-232 control lines back to themselves as follows: DTR to CTS and DCD. Implement either the hardware or software flow control if the serial messages could overflow either devices' receive buffer.

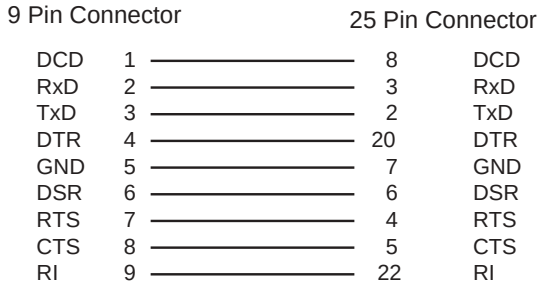



**Figure 2-3 DTE 9 Pin to 25 Pin Adapter**

### 2.4.2 RS-422 Connections

When RS-422 signals are selected, the pinouts appear as shown in the RS-422 FD/HD column in Table 2-1. For asynchronous RS-422 data transfer, the TXCLK and RXCLK signals are not used and may be left open. Use the VXI-5539A's internal oscillator to generate baud rates as listed in paragraph 1.2.7.5. For SDLC operation, TXCLK may be an input or output signal and RXCLK can be an input or it can be not used.

In half-duplex mode, the transmitter is tristated between messages. The VXI-5539A's internal termination network can be used to bias the TX data lines in the 'mark' state when the transmitter is not active.

### 2.4.3 RS-485 Connections

When RS-485 signals are selected, data is transmitted and received on the Tx/Rx signal pair in the RS-485 HD column in Table 2-1. The RXCLK and TXCLK are not used and maybe ignored. The VXI-5539A's internal termination network can be used to bias the TX/RX data lines in the 'mark' state when the transmitter is not active.

### 2.4.4 Eight Channel Units

The VXI-5539A-18 Octal Serial Module has two 37-pin DE-9P connectors that each provide four serial channels. Channels 1, 2, 5, 6 and the Pulse Output are on J5. Channels 3, 4 7 and 8 are on J6. The signal definitions and functions are the same as for the four channel unit. See paragraphs 2.4.1, 2.4.2 and 2.4.3. Table 2-2 shows the serial signal assignments and the signal direction relative to the VXI-5539A.

Eight channel units have an additional RS-422 Timing Pulse output on J5.

## TABLE 2-2 SERIAL INTERFACE PIN ASSIGNMENTS FOR EIGHT CHANNEL UNITS

| Channel Number | Pin Number | Signals and Direction | | |
|---|---|---|---|---|
| | | RS-232 | RS-422FD/HD | RS-485 HD |
| 1 or 3 | 1 | DCD ← | Rx(B)+ ← | - |
| | 2 | Rx ← | Rx(A)-- ← | - |
| | 3 | Tx → | Tx(A)- → | Tx/Rx- ↔ |
| | 4 | DTR → | Tx(B)+ → | Tx/Rx+ ↔ |
| | 5 | DGND | DGND | |
| | 20 | (DSR) | RxCLKO+ ↔ | - |
| | 21 | RTS → | RxCLKO- ↔ | - |
| | 22 | CTS ← | TxCLKI+ ↔ | - |
| | 23 | | TxCLKI- ↔ | - |
| 5 or 7 | 24 | DCD ← | Rx(B)+ ← | - |
| | 25 | Rx ← | Rx(A)- ← | - |
| | 26 | Tx → | Tx(A)- → | Tx/Rx- ↔ |
| | 27 | DTR → | Tx(B)+ → | Tx/Rx+ ↔ |
| | 6 | (DSR) | RxCLKO+ ↔ | - |
| | 7 | RTS → | RxCLKO- ↔ | - |
| | 8 | CTS ← | TxCLKI+ ↔ | - |
| | 9 | | TxCLKI- ↔ | - |
| 6 or 8 | 29 | DCD ← | Rx(B)+ ← | - |
| | 30 | Rx ← | Rx(A)- ← | - |
| | 31 | Tx → | Tx(A)- → | Tx/Rx- ↔ |
| | 32 | DTR → | Tx(B)+ → | Tx/Rx+ ↔ |
| | 33 | DGND | DGND | |
| | 11 | (DSR) | RxCLKO+ ↔ | - |
| | 12 | RTS → | RxCLKO- ↔ | - |
| | 13 | CTS ← | TxCLKI+ ↔ | - |
| | 14 | | TxCLKI- ↔ | - |
| 2 or 4 | 15 | DCD ← | Rx(B)+ ← | - |
| | 16 | Rx ← | Rx(A)- ← | - |
| | 17 | Tx → | Tx(A)- → | Tx/Rx- ↔ |
| | 18 | DTR → | Tx(B)+ → | Tx/Rx+ ↔ |
| | 19 | DGND | DGND | |
| | 34 | (DSR) | RxCLKO+ ↔ | - |
| | 35 | RTS → | RxCLKO- ↔ | - |
| | 36 | CTS ← | TxCLKI+ ↔ | - |
| | 37 | | TxCLKI- ↔ | - |
| | J5-10 | | Pulse(B)+ → | |
| | J5-28 | | Pulse(A)- → | |
| | Shell | Chassis | Chassis | Chassis |

# 3

# Operation and Programming

## 3.1 INTRODUCTION

This section describes how to use and program the VXI-5539A. This section includes directions for using the Front Panel Display, on how to configure the VXI-5539A interfaces and on using the VXI-5539A interfaces to control GPIB devices.

## 3.2 FRONT PANEL DISPLAYS AND CONTROLS

The VXI-5539A has a four character alphanumeric display on the top of its front panel and a Data LED for each GPIB port. The Data LED blinks when data is being transferred on the GPIB interface.

### 3.2.1 Alphanumeric Display

At power turn-on or when reset, the alphanumeric display operates in its normal mode and displays the VXI-5539A's status and addressed interface. The normal power-on display sequence is:

| Display | Meaning |
|---|---|
| **SysF** | On during selftest while SYSFAIL is asserted. If on after 4.9 seconds, the module could be defective or SYSFAIL is still being held on. |
| **ICS** | Momentary on after selftest is passed. |
| **5539** | Momentary on after selftest is passed. |
| **FAIL** | On only if selftest failed or the module was commanded into the failed state. |
| **Init** | On after selftest passed while module is waiting for a Begin-Normal-Operation command. Module is in configure sub-state. |
| **Rdy** | Unit ready for Word Serial commands. |

If the unit fails selftest, **FAIL** will be displayed for 5 seconds and then the display will show the appropriate failure error code (**ERxx**). The error codes are listed in Section 5.

When the unit saves data to Flash memory. **SAVE** is displayed for one second followed by **SvOK** if the save operation was successful. **ER03** is displayed if the save operation failed.

### 3.2.2 Display Button

The Display Button is a recessed bush button located below the display as shown in Figure 3-1. The Display Button selects the VXI-5539A display and diagnostic modes described in Table 3-1. Tables 3-2 through 3-5 list the messages for the various display modes.

To change the display mode, use a toothpick, Q-stick or some other thin nonconducting rod to gently hold the button closed. The display will first show the current mode and after two seconds advance through the next mode. Release the button when the display shoes the desired mode. Momentarily pressing the Display Mode Button will cause the display to show the current mode without changing the mode.

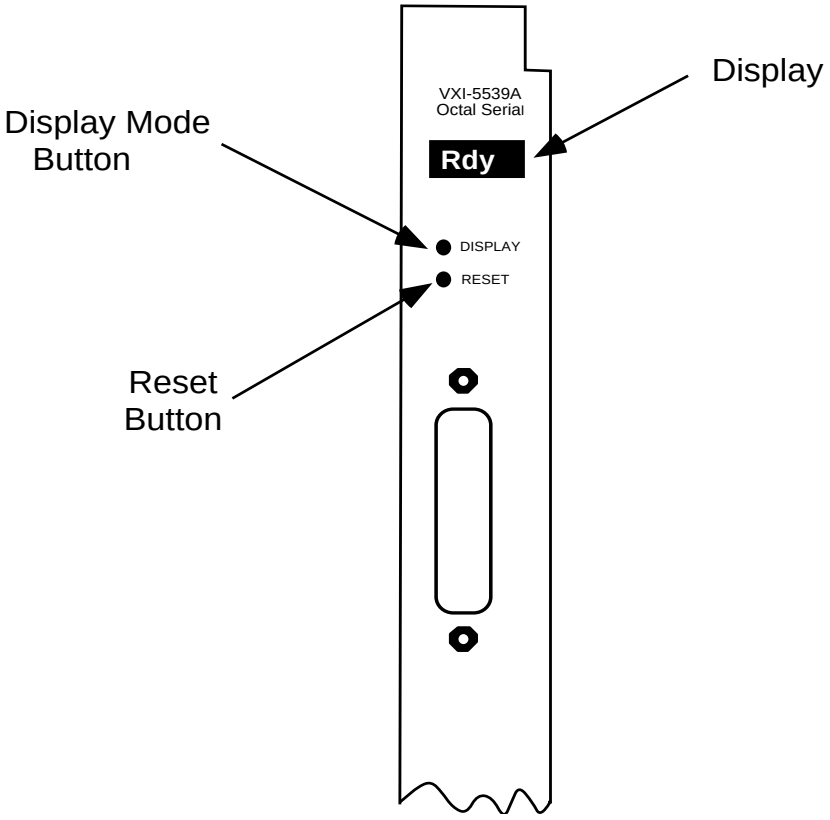



**Figure 3-1 Display Mode Button Location**

### 3.2.3 Reset Button

The Reset Button is a recessed bush button located below the Display Button as shown in Figure 3-1. Pressing the Reset Button resets the VXI-5539A processor and internal logic. The Resource Manager must be run or the Begin-Normal-Operation command must be issued to resume Normal operation after a reset.

### 3.2.3 Display Modes

The VXI-5539A display has alternate display modes to give the user more information about the module and its status. Table 3-1 describes the module's display modes. Tables 3-2 through 3-5 list the displayed parameters.

**TABLE 3-1 VXI-5539A DISPLAY MODES**

<table>
<tr><th>Display Mode</th><th>Description</th></tr>
<tr><td>Norm</td><td>The Norm or normal mode shows the power on messages listed in Table 3-2.</td></tr>
<tr><td>Extd</td><td>The Extd or extended mode shows the messages listed in Table 3-3. These messages are useful for system testing and debugging.</td></tr>
<tr><td>Addr</td><td>The Addr or address mode is a temporary mode that displays the units current VXIbus address setting in decimal form for 5 seconds. The unit then returns to is prior mode.</td></tr>
<tr><td>Resp</td><td>The Resp mode is a temporary mode that displays some of the VXI Response register information in hexadecimal form for 5 seconds. The unit then returns to is prior mode. The Stat display is :
<table>
<tr><td>Digit</td><td colspan="4">MSD</td><td colspan="4">MSD-1</td><td colspan="2">MSD-2</td><td>LSD</td></tr>
<tr><td>Bit</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6-4</td><td>3-0</td></tr>
<tr><td>Contents</td><td>0</td><td>Rsvd</td><td>DOR</td><td>DIR</td><td>Err#</td><td>Read Rdy</td><td>Wrt Rdy</td><td>FHS Act#</td><td>Lock -ed#</td><td>Device Dep</td><td>Device Dep</td></tr>
</table></td></tr>
<tr><td>SfSt</td><td>The SfSt mode is a temporary mode that displays the current Self-Test status on the least significant digit for 5 seconds. The unit then returns to its prior mode. The SfSt display is:<br>Least Significant Digit<br>Bit 1 Channel #1<br>Bit 2 Channel #2</td></tr>
</table>

**TABLE 3-2 NORMAL DISPLAY MODE MESSAGES**

| Display | Meaning |
|---|---|
| 5539A | Power on ID message |
| SAVE | Data written to the Flash memory |
| FAIL | Self Test failed |
| ICS | Power on ID message |
| Init | Initialized, waiting for begin normal operation command |
| Rdy | Normal ready message, begin normal operation received |
| SysF | SystFail Asserted |
| Ch*n* | Serial Channel *n* selected (*n* = 1 to 8) |

## TABLE 3-3 EXTENDED MODE MESSAGES

| Display | Meaning |
|---|---|
| BNO | Begin Normal command received. |
| ER*nn* | Error message - see Table 3-5 |
| Err*n* | Unsupported command error |
| IDCr | Identify commander |
| SfGP | Soft reset |

## TABLE 3-4 TEST MODE MESSAGES

| Display | Meaning |
|---|---|
| FAIL | Self test failed |
| Pass | Selected slave test passed |

## TABLE 3-5 ERROR MESSAGES

| Display | Meaning |
|---|---|
| ER01 | VXI Flash memory error |
| ER02 | VXI RAM error |
| ER03 | VXI Flash memory error |
| ER04 | VXI interface register error |
| ER?? | Unknown error |

## 3.3 GENERAL OPERATING INSTRUCTIONS

This section provides the user with the necessary information to operate the VXI-5539A. It is required reading for new VXI-5539A users.

### 3.3.1 Pre-Operation Setup

Before using the VXI-5539A, set the VXI-5539A's address switch and install the module in a VXI chassis as directed in Section 2.4.

### 3.3.2 Power Turn-on

Turn the VXI chassis power on. The VXI-5539A's display should cycle through its identification sequence while the unit does its power-on selftest. When the display shows **Init**, the module has passed its self test and is waiting for the Begin-Normal Command. After the Resource Manager has run the display shows **Rdy,** which indicates that the module is good and has received the Begin-Normal-Operation command. For the meaning of other messages, refer to paragraph 3.2 and to Tables 3-2 and 3-3.

The VXI-5539A can also be Soft Reset by asserting the Reset bit (bit 0) in the VXIbus Control Register for a minimum period of 100 microseconds. This causes a software reset of the module. All settings revert to their power turn-on values. **The VXI-5539A remains inactive after a Soft Reset until activated by a Begin Normal command.** The Begin Normal Operation command is normally sent by the Resource Manager but can be sent by the user after a reset. If interrupts are used, the Assign Interrupt Line command will have to be sent also.

### 3.3.3 Interface Programming Concepts

Communication with the VXI-5539A over the VXIbus is done with either Word Serial Commands and Word Serial Messages or via the Fast Data Channel buffers.

**Word Serial Commands** are single word, 16-bit coded commands from the Slot 0 Controller to the VXIbus device. Word Serial Commands are **NOT** sent with ASCII characters. Refer to your VXI Controller's instruction manual for directions on how to generate Word Serial Commands. If you are using VISA or SCIL, refer to the appropriate software manual for the proper function call.

The applicable Word Serial Commands for the VXI-5539A are described in paragraph 3.4 and are listed in Table 3-6. ICS's expanded Word Serial Commands for FDC Channels are described in the FDC Addendum. Note that some Word Serial Commands are queries and return a response. The response must be read before sending the module a new command.

**Word Serial Messages** are multi-word messages that pass a series of 8-bit characters between the Slot 0 Controller and a VXI module. These messages may be commands, queries, responses from the module or short strings of serial data up to 1024 characters in length. Using the DATA and DATAH commands to send and receive serial data with Word Serial Messages is not the same as placing data in or reading data from the Fast Data Channel buffers.

The **Fast Data Channel** buffers are the normal way serial data is transferred between the VXI Commander and the VXI-5539A. The Fast Data Channel buffers are used as pairs to pass serial data in one direction. The controller places data in one buffer while the VXI-5539A empties the other channel's buffer. The Controller can switch buffers when ready. The VXI-5539A places received data in a FDC buffer until it reaches the full point. The buffers are then switched and the 'full' buffer is passed to the Controller. The Controller learns of the buffer switch either by a VXI interrupt or by polling the buffer header.

Four fast data channels are required for each VXI-5539A serial channel, two for transmit and two for receive. Bits in the Fast Data Channel header define the number of bytes in the FDC buffer and buffer ownership. The FDC buffer space is allocated by the Resource Manager function in the Slot 0 Controller. The Fast Data Channel buffers are initialized by the user's program. Refer to Section 3.4 for information about the FDC buffers and their organization in the VXI-5539A. Refer to ICS Application Bulletin AB55-5 for detailed information on how to setup and use the FDC buffers.

The VXI-5539A generates **VXI Interrupts**. The VXI-5539A is a single VXI interrupter. The VXI interrupts can be FDC events or normal VXI events. The normal VXIbus interrupts are enabled by setting the appropriate bits in the 488.2 Status Enable Register and Event Enable Register in the VXI-5539A's Status Reporting Structure. The FDC channels can be set to interrupt when the Receive buffer has a message, has received a preset byte count or is full. FDC interrupts can also occur when there is a full buffer that needs to be emptied or an empty buffer that needs filling. The VXIbus interrupt line is assigned when the Resource Manager is run. When an interrupt occurs, examine the upper 8 bits of the interrupt response word to determine the interrupt type. If the interrupt is a normal VXI event interrupt, use the Word Serial Read STB command to read the interface(s) Status Byte Register to determine the cause of the interrupt. If the interrupt is a FDC interrupt, the FDC buffer number is specified in the upper byte of the interrupt response word.

The VXI-5539A has a **VXI Status Register** (at an offset of four bytes above its base A16 address) which can be read by the VXI Slot 0 Controller. Bits 6-13 of the VXI Status Register are device dependent bits that the device designer can use to simplify device programming. In the VXI-5539A, these bits are used to pass status back to the user. Figure 3-2 shows the register bit assignments and their channel usage.

| **BIT#** | **15** | **14** | **13-6** | **5** | **4** | **3** | **2** | **1-0** |
|---|---|---|---|---|---|---|---|---|
| **VXI Spec** | **A24/A32 Active** | **MODID** | **Device Dependent** | | | **Rdy** | **Passed** | **Device Dependent** |
| GPIB | 0 | X | 0 | X | X | 1 | 1 | 0 |

**Figure 3-2 VXI Status Register Bit Assignments**

### 3.3.4 Program Recommendations

The following recommendations are offered to improve your program's performance.

1. Do not continuously poll the VXI-5539A with the VXI Read STB command. This command requires internal processor time and continuous polling degrades the interface's performance.

2. FDC Buffer headers are in memory and may be polled if desired. Polling the FDC headers does affect the modules performance so it should **not** be done in a tight loop.

### 3.3.5 Controlling the VXI-5539A with Multi-Tasking Operating Systems

Outputting data to the VXI-5539A using Word Serial Messages from a task in a Multi-Tasking environment requires careful programming to avoid confusing the module or accidentally changing interface channels. These concerns are not relevant when passing data in FDC channel buffers.

If multiple tasks are used to control the VXI-5539A, the user needs to preface each Word Serial command with the Channel selection command to be sure of sending the command to the correct channel. Do not rely upon the channel setting being left in your channel number. A Channel selection command is not necessary when using FDC buffers to send and receive data as the FDC channels are dedicated to a specific serial channel..

## 3.4 Word Serial Commands

The VXI-5539A supports the standard Word Serial Commands defined in the VXI Specification, the VXI FDC commands and ICS's expanded FDC commands. Word Serial Commands are single word, 16-bit binary commands and are coded differently from the data words used in Word Serial Messages. The VXI-5539A always responds to VXI Word Serial Commands regardless of the mode of the interface. The Word Serial Commands are used to configure the VXI-5539A's VXI interface, alter its operation or control its FDC buffers.

The user should consult his VXI Slot 0 Controller manual for specific directions on sending Word Serial Commands to a VXIbus module. The following example is taken from an ICS Model VXI-5543 Slot 0 Controller.

e.g. Command Format = *int* vxiWScmd ( *int* Logical Address, *int* WScmd )

An example is: error = vxiWScmd( 16, 0xedff )

Using mnemonics instead of absolute values, the above becomes:
error = vxiWScmd( Logical Address, Trigger)

**TABLE 3-6 VXI-5539A WORD SERIAL COMMANDS**

| Command | Code | Description |
|---|---|---|
| Abort Normal operation | C8FF | Cease normal operation and return to default configuration. |
| Begin Normal operation | FCFF | Notifies a device to begin normal operation. |
| Byte Available | BCdd | Sends a data byte to a servant. The code for the last byte with the END bit asserted is BDdd. |
| Byte Request | DEFF | Reads a data byte from the servant device. |
| Clear | FFFF | Clears a device's VXIbus interface, buffers and reinitializes the |
| End Normal operation | C9FF | End normal operation an orderly fashion. The device becomesinactive. |
| Trigger | EDFF | Trigger a previously armed operation. |
| Read STB | CFFF | Reads devices status word. Equivalent to 488.2 *STB? query. |
| Read Protocol | DFFF | Finds what protocols a servant device supports. |
| Read Protocol Error | CDFF | Tells a servant to report its current error state. |
| Assign Interrupter Line | | AAxx Assigns a IRQ line to an Interrupter in the device. |
| Read Interrupter Line | 8Dxx | Determines which IRQ line a particular Interrupter in a servant is connected to. |
| Read Interrupters | CAFF | Determines number of Interrupters in a device. |
| Asynchronous mode cntl | A8xx | Directs the way a device responds to events. |
| Control Response | 8Fxx | Enables response signals or response interrupts. |
| Control Event | AFxx | Enables generation of events in a device. |

## 3.5 Fast Data Channel Usage

### 3.5.1 VXI-5539A FDC Buffer Organization

FDC channel buffers are divided into a header area and a data buffer area. In Figure 3-3, the header area is shown above the dotted line and the data buffer area is below the dotted line. The header area contains two 32-bit words. The first word contains the ReadReady and WriteReady bits that define buffer ownership. The second word is the Data Buffer Size Word and contains the number of bytes used in the data area. Message organization in the data buffer area is in ascending byte order for a serial message. The lowest numbered byte the first character of the message.

Each 16-bit word in the FDC buffer contains two serial characters. The first or most significant character is in byte 0. The second or next character is in byte 1. Additional characters appear in the same order in subsequent 16-bit words. Serial messages can have an even or odd number of characters. If an odd number of characters is received, the VXI-5539A waits for 3 character times and then inserts a null character in byte 1 to complete the last 16-bit word of the message. The message byte count is updated and stored in the Data Buffer Size Word.

The user can put multiple messages in the transmit buffer. If a message has an odd number of characters (bytes), the first character of the subsequent message is packed in the word with the last character of the previous message. If the last message ends with only one character in the last word in byte 0, then byte 1 can be set to a null value of zero. The byte count is then set to the total number of characters in the buffer and does not include the null byte if there is one.

Figures 3.3 and 3-4 show the layout of the message spaces in the FDC channel buffer by VXIbus word size. The word is as it would be read by a 32 or by a 16-bit access over the VXI bus.

| Word Count | Byte 3 (MSB) | Byte 2 | Byte 1 | Byte 0 (LSB) |
|---|---|---|---|---|
| 0 | Revision | Reserved Bits | Reserved TRIG | Rsvd ABT RDY WDY END |
| 1 | DB Size Bits 31-24 | DB Bits 23-16 | DB Bits 15-8 | DB Bits 7-0 |
| 2 | Character #4 | Character #3 | Character #2 | Character #1 |
| 3 | Character #8 | Character #7 | Character #6 | Character #5 |
| 4 | Character #12 | Character #11 | Character #10 | Character #9 |
| 5 | Character #16 | Character #15 | Character #14 | Character #13 |
| 6 | Character #20 | Character #19 | Character #18 | Character #17 |
| . | . | . | . | . |
| . | . | . | . | . |
| n | Unused bits | Character #n | Character #n-1 | Character #n-2 |

Note: Bit definitions are listed in paragraph 3.5.2. Data area starts below the dashed line.
Byte numbers are in Intel byte order as seen by the Slot 0 Controller
Data sample contains n characters.

**Figure 3-3 FDC Buffer Layout for 32 Bit Words**

| Word Count | Byte 1 (MSB) | Byte 0 (LSB) |
|---|---|---|
| 0 | Rsvd Rsvd Rsvd Rsvd Rsvd Rsvd Rsvd TRIG | Rsvd Rsvd Rsvd Rsvd ABT RDY WDY END |
| 1 | Minor Revision Major Revision | Rsvd Rsvd Rsvd Rsvd Rsvd Rsvd Rsvd Rsvd |
| 2 | Data Buffer Size Bits 31-24 | Data Buffer Size Bits 23-16 |
| 3 | Data Buffer Size Bits 15-8 | Data Buffer Size Bits 7-0 |
| 4 | Character #2 | Character #1 |
| 5 | Character #4 | Character #3 |
| 6 | Character #6 | Character #5 |
| 7 | Character #8 | Character #7 |
| - | | |
| - | | |
| n | Last character or unused bits | Next to last or last character |

Note: Bit definitions are listed in paragraph 3.5.2. Data area starts below the dashed line.
Byte numbers are in Intel byte order as seen by the Slot 0 Controller.

**Figure 3-4 FDC Buffer Layout for 16-Bit Words**

### 3.5.2 FDC Buffer Definitions

**FDC BUFFER:** Total buffer area

**HEADER AREA** : First eight bytes of the FDC channel buffer. Contains the channel Revision and Data Buffer Size words as defined by the FDC Specification.

**RSVD** : These bits are reserved and should be set to 0.

**MAJOR REVISION**: Must be 2 (3 bit code)

**MINOR REVISION**: Must be 1 (2 bit code)

**TRIG**: The TRIG bit is utilized only within Message Transfer Protocol (MTP) to send the MTP Trigger command. It has no meaning outside of MTP and should be set to 0.

**END**: The END bit indicates whether *this* buffer of data is the *last* buffer of data in a data block. If the END bit is set to 1, this is the last buffer of data. If the END bit is set to 0 , this is not the last buffer of data.

**WDY**: The WDY flag is utilized when data is transferred from the Commander to the Servant. If the WDY bit is set to 1, the Commander owns the FDC area. It can place a buffer of data into the FDC area and then set WDY to 0 to pass the buffer of data to the Servant. If the WDY bit is set to 0, the VXI Servant owns the FDC area. It may read the buffer of data and then set WDY high to pass the FDC area back to the Commander. When the channel is in the idle state, WDY is 0.

**RDY**: The RDY flag is utilized when data is transferred from the Servant to the Commander. If the RDY bit is set to 0, the VXI Servant owns the FDC area. It can place a buffer of data into the FDC area and set the RDY bit to 1 to pass the buffer of data to the Commander. If the RDY bit is set to 1, the Commander owns the FDC area. The Commander can read the buffer of data and then set the RDY bit to 0 to pass the FDC area back to the Servant. When the channel is in the idle state, RDY is 0.

**ABT**: The ABT bit indicates that an abort transfer is being requested for this block of data.

**DATA BUFFER SIZE**: A 32-bit word in the FDC header that contains the number of bytes contained in the data buffer portion of the FDC buffer. Byte count starts at byte 8 and goes through byte n.

**DATA BUFFER**: Memory area for the module's data. Buffer organization defined by the module designer. Includes bytes 8 through n.

**DATA BUFFER ORGANIZATION**: In the VXI-5539A the whole data buffer section is used for data. Transmit data is right justified with the first (most significant ) character or byte in byte 0. Received data is placed in the buffer in the same manner.

Note: Refer to the Fast Data Channel Specification VXIbus-10 for detailed information on the usage of the bits in the Channel Header

### 3.5.3 FDC Buffer Initialization

The FDC buffers need to be initialized before they can be used to transfer data. The initialization sequence and commands are described in the FDC Addendum. The buffers should be initialized as streaming pairs where they automatically switch as they become full (receive) or empty (transmit). The VXI-5539A only supports Pair and Stream mode. The FDC mode selection is made by setting flag bits in the *Transfer to Commander* or *Transfer to Servant* initialization command.

Initializing the FDC buffers also initializes the channel's UART and enables it to receive data. If handshaking lines are enabled, they are set to the appropriate true levels at this time.

### 3.5.4 FDC Buffer Operation

In the VXI-5539A module, the Fast Data Channel (FDC) buffers are used as A/B buffer pairs in stream mode to transfer data over the VXIbus backplane at a high data rate. For a transmit pair, the Controller first loads the first (lower numbered) buffer and passes it to the VXI-5539A The buffer can contain as little as one message. The VXI-5539A transmits the data from the first buffer while the Controller places any additional data in the second buffer. The VXI-5539A returns the empty buffers to the Controller. This operation continues, with the Controller loading data into alternate buffers and passing them to the VXI-5539A as long as there is data to send. The Controller should not reuse the same buffer twice in a row, even when both buffers are empty.

Data is received in the opposite fashion. The VXI-5539A loads the received data into alternate buffers and passes them to the Controller. A receive buffer is 'full' when it has been filled to its assigned size as set by the SYST:COMM:SER:BUFF:SIZe command. When a receive buffer is 'full', the module switches to the other buffer and continues receiving data. The VXI-5539A can generate a VXIbus interrupt to notify the controller that it has a buffer ready to pass. This operation continues with the VXI-5539A filling alternate receive buffers as it receives data.

The user should consider the message length and VXI access word size when setting the buffer size. For multiple messages, the buffer size should be a multiple of the received message size to avoid splitting messages. Else the result could be very confusing data in the buffer. The user has to read out complete words when reading the buffer, even if the full count ended in the middle of a VXI bus word.

Refer to the FDC Addendum for information on how to pass buffers and how to test for buffer ownership
.

## 3.6 488.2 COMPLIANCE

The VXI-5539A is an IEEE-488.2 compliant VXIbus instrument. As an IEEE-488.2 compliant VXIbus instrument, the VXI-5539A responds to the IEEE-488.2 commands and queries listed in Table 3-7.

### 3.6.1 488.2 Status Reporting Structure

The VXI-5539A includes the IEEE-488.2 Status Reporting structure shown in Figure 3-5. The expanded status reporting structure conforms to the SCPI 1994.0 Specification and builds on the IEEE 488.2 Standard with the addition of the Questionable and Operational registers. The Event and Status registers are controlled and queried with the IEEE-488.2 common commands. The added Questionable and Operational registers are controlled and queried with SCPI commands.

As shown in Figure 3-5, IEEE 488.2 service request generation is a multilevel function and is determined by the occurrence of an event that has its corresponding enable bit set to '1'. A service request uses a VXI interrupt to signal the bus controller that an event has occurred and/or that the VXI-5539A needs service. There are four major sources of service requests, each of which is summarized in a bit in the Status Byte Register. Three of the sources are event registers with their own enabling bits and the fourth is the Output Queue. The Event registers and the Output Queue are cleared when read or by the *CLS command.

The *STB? query places the Status Byte Register response message in the data buffer. If there is data or other query responses in the buffer, they must be read first. The Read STB VXI command reads the status directly from the VXIbus interface, bypassing the receive buffer, therefore this command is preferable for determining the status of the buffers.

> **WARNING** - Continuously polling the VXI-5539A with the Read STB command ties up the internal processor and degrades module performance. The recommended method to test for data in the receive buffer is to enable the VXIbus interrupt and then execute a Read STB command when the interrupt is received.

The VXI-5539A logical devices respond to the Read STB command and *STB? query with a response byte equivalent to the serial poll response in IEEE Standard 488.2. The VXI-5539A's STB response byte is shown in Figure 3-5.

### 3.6.2 Event Registers

An event register **captures 0 to 1 transitions** in its associated condition register or in the standard event conditions. An event bit becomes TRUE (1) when the associated condition bit makes 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

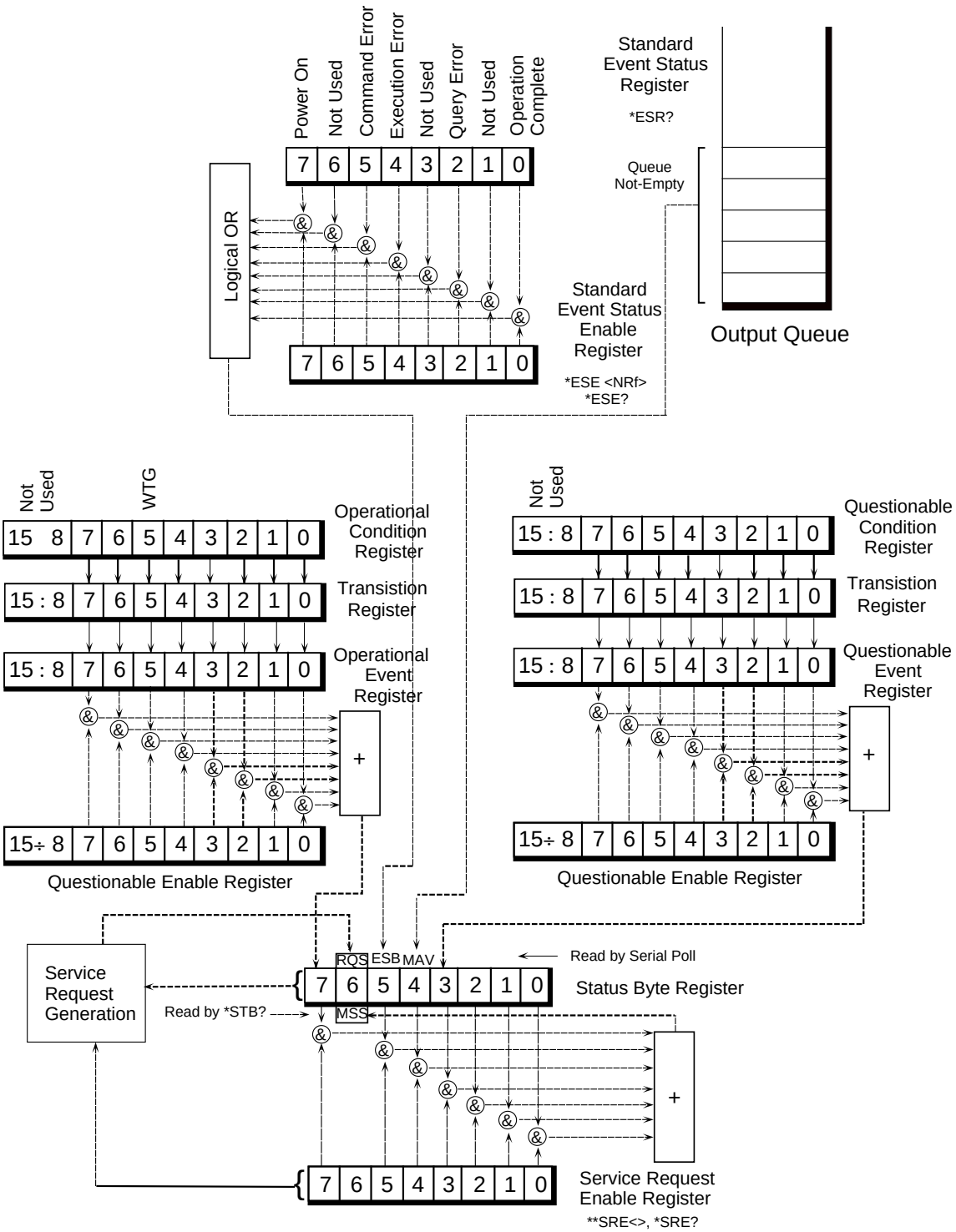



**Figure 3-5 VXI-5539A 's Status Reporting Structure**

set by writing the equivalent value to the desired 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.

At power turn-on or when the VXI-5539A is reset, all of the enabling registers are set to their saved values if *PSC (Power-on-clear) is not enabled. The *ESE and *SRE commands are used to change the enabling register values. New values are saved with the 488.2 *SAV 0 command. Table 3-8 lists the recommended settings for the Event Status and Status Byte enable registers.

### 3.6.3 Standard Event Status Register

The Standard Event Status Register reports events that are common to all 488.2 devices. This includes events such as self test 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 Event Status Register is read and cleared with the ***ESR?** query. Use the ***ESE** commands to set the Event Status Enable Register as shown in the following example:

```
*ESE 60        'enables error bits 2 through 5
*ESE?          'quires the enabling register setting
```

### 3.6.4 Questionable Registers

The Questionable Registers has no use in the VXI-5539A module. If used, the Questionable Condition Register reports the status of its input signals. **A logic 1 means that the signal is asserted.** 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 Enable Register enables set Event bits to be summarized in the Status Byte Register. The following example enables bit 0:

```
STAT:QUES:ENAB 1      'enables Event bit 0
*SRE 8                'enables Questionable Summary bit in the STB register.
```

The Questionable Condition Register reflects the **real time** condition of its inputs. A logical 1 means that the corresponding digital input is high. To read the Questionable Condition Register use the following SCPI query:

```
STAT:QUES:COND?       'reads the digital inputs
```

The response is a decimal number that is the sum of the weights of the register bits that are a logical 1 (i.e. 1010 = decimal 10). Reading the Questionable Condition Register does not change its contents.

### 3.6.5 Operational Condition Registers

The 488.2 Operational Registers lets the user read device specific status conditions and detect any changes in the device's status. The Operational Registers are similar to the Questionable Registers described in paragraph 3.6.1.5.

In the VXI-5539A, the Operational Condition Register only reports the WTG (Waiting for Trigger) status. The WTG bit is true when the VXI-5539A has been armed and is waiting for a trigger. The following command will query the Operational Condition Register:

**STAT:OPER:COND?** 'queries the Operational Condition Register

### 3.6.6 Output Queue

The Output Queue is used by the VXI-5539A to send IEEE 488.2 messages back to the bus controller. These messages are device responses and VXI-5539A responses to 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 a Word Serial message. The Output Queue can be tested for presence of data by testing the DOR bit in the response register or by testing the MAV bit in the STB register. If the Output Queue is not read before sending another query, its contents will be lost and an error reported.

### 3.6.7 Status Byte Register

The VXI-5539A generates a service request whenever any of the enabled bits in the Status Byte Register become true. The Status Byte Register may be read by 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 40** 'Sets the SRE Register to 0010 1000 which enables just the Event Status and Questionable summary bits to generate SRQs.

## TABLE 3-7 IEEE 488.2 COMMAND LIST

| Command | 488.2 Function |
|---|---|
| ***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 - Sets "Event Status Enable Register" to <value>, an integer between 0 and 255. <value> is an integer 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.<br><br>EXAMPLE: decimal 16 converts to binary 00010000 which sets bit 4 to a logical 1. |
| ***ESE?** | EVENT STATUS ENABLE QUERY - 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 - 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. |
| ***IDN?** | IDENTIFICATION QUERY - 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. A typical IDN response is: "**ICS Electronics, VXI-5539A, S/N 00101, Rev. 00.00 Version 00.04.12**". |
| ***OPC** | OPERATION COMPLETE COMMAND - Causes the Unit to generate the operation complete message in the Standard Event Status Register when all pending selected Unit operations have been finished. |
| ***OPC?** | OPERATION COMPLETE QUERY - Places an ASCII character 1 into the Unit's Output Queue when all pending selected Unit operations have been finished. |
| ***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 save the SRE and ESE values in the $E^2$PROM. Not implemented in Version 00.00. |
| ***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. Not implemented in Version 00.00. |
| ***RCL <value>** | RECALL - Restores the state of Unit from a copy stored in its E2PROM by *SAV command. *RCL 0 restores the power on setting. <value> = 0. Not implemented in Version 00.00. |
| ***RST** | RESET - Unit 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. |
| ***SAV <value>** | SAVE - Saves current Unit configuration in the $E^2$PROM. *SAV 0 saves the current setting as the new power on setting. <value>=0. Not implemented in Version 00.00. |
| ***SRE <value>** | SERVICE REQUEST ENABLE - Sets the "Service Request Enable Register" to <value>, an integer between 0 and 255. The value of bit six is ignored because it is not used by the Service Request Enable Register. <value> is an integer 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. |

**TABLE 3-7 IEEE 488.2 COMMAND LIST (CONT'D)**

| Command | 488.2 Function |
|---|---|
| ***SRE?** | SERVICE REQUEST ENABLE QUERY - Unit 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 - Unit 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** | TRIGGER - Unit initiates an action determined by the manufacturer. In the VXI-5539A, the action taken is to run the GPIB commands stored in a Program Buffer. The ArmTrigger command be sent first to select which buffer will be run when triggered. |
| ***TST?** | SELF-TEST QUERY - Causes the Unit to run internal self-test. Test takes about 1 second and clears all internal buffers. Unit will not respond to any GPIB inputs while performing its internal self-test. A zero response indicates no failures. Other responses are listed in Table 5-1. |
| ***WAI** | WAIT-TO-CONTINUE - Prevents the Unit from executing any further commands or queries until the No-Operation-Pending flag is TRUE. |

### 3.6.8 Saving the Enable Register V alues

The ESE and SRE Register values can only be saved and recalled at power turn-on by disabling the PSC flag. **The *SAV command does not save the ESE and SRE register values.** Use the *PSC 0 command to disable the PSC flag and save the ESE and SRE register values. The following example saves the ESE and SRE values and enables a SRQ at power turn-on.

```
*PSC 0; ESE 192; SRE 32     'checks Power-on and Not Calibrated bits
```

Note that the ESE and SRE 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 ESE and SRE registers to be cleared at the next power turn-on and prevent a power on SRQ.

**Table 3-8 RECOMMENDED ESE and SRE BIT V ALUES**

| Register | Recommended Value | Comments |
|---|---|---|
| ESE | 60 (3C HEX) | All error reporting bits enabled |
| SRE | 40 (28 HEX) | Event Status and Questionable summary bits enabled |

## 3.7 SCPI CONFORMANCE INFORMATION

The VXI-5539A module 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 1995.0 and provide an industry standard, self-documenting form of code that makes it easy for the programmer to maintain the application program.

Table 3-9 shows the VXI-5539A SCPI command tree. The VXI-5539A uses portions of the SCPI SYSTEM, STATUS, INSTRUMENT, SOURCE, and CALIBRATE subsystems. The commands 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 in the VXI-5539A 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.

e.g., **SYSTEM:COMM:SERIAL:BAUD 9600**

is the same as

**SYST:COMM:SER:BAUD 9600**
or **syst:comm:ser:baud 9600**

Table 3-9 repeats the SCPI commands used to set up the serial configuration. The first page shows the commands used for the Asynchronous Mode. The second page shows the commands used for the SDLC mode. Unused commands in each mode have been grayed out to avoid confusion.

The SYSTEM and STATUS commands are repeated for each serial channel. The user must use the INSTRUMENT command to select the channel before setting or querying a channel parameter.

**e.g. INST:NSEL 3**
**SYST:COMM:SER:BAUD 9600** 'sets channel 3's baud rate

Table 3-10 is the Command Reference Table and it lists the SCPI keywords and describes their functions in detail. Keywords other than those listed in the table will have no effect on the module's operation and a command error will be reported. Refer to the FDC Addendum for information about the Fast Data Channel 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 and a command execution error.**

## 3.8 SHORT FORM COMMANDS

The VXI-5539A also accepts short form commands which invoke the same action as do the corresponding SCPI commands. The short form commands are one to five 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 VXI bus traffic and the VXI-5539A's parser execution time.

Table 3-9 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 3-10 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.** **C 3**
is the same as
**INST:NSEL 3**

**BAUD 9600**
is the same as
**SYST:COMM:SER:BAUD 9600**

**C 3**
**BAUD?**
is the same as
**INST:NSEL 3**
**SYST:COMM:SER:BAUD?**

## TABLE 3-9 VXI-5539A SCPI COMMAND TREE - ASYNCHRONOUS MODE

| Keyword | Parameter | Short Form | Notes |
|---|---|---|---|
| **SYSTem** | | | |
| :COMMunicate | | | |
| :SERial | | | |
| [:RECeive] | | | |
| :PROTocol | [ASYNC] \| SDLC | PROT | |
| :FRAMe | | | |
| :SIZe | 2-30 [30] | SIZE | |
| :BAUD | 50-460800 [9600] | BAUD | |
| :BITS | 5-8 [8] | BITS | |
| :SBITs | [1] \| 2 | SBITS | |
| :PARity | | | |
| [:TYPE] | EVEN \| ODD \| [NONE] | PAR | |
| :ECHO | ON \| [OFF] | ECHO | |
| :LOOPback | [OFF] \| INT \| MON | LOOP | |
| :MODE | [232] \| 422FD \| 422HD \| 485 | MODE | |
| :HANDshake | ON \| [OFF] | HAND | |
| :RXCLK | [INT] \| EXT | RXCLK | |
| :TXCLKOUT | [IN] \| OUT | TXCLK | |
| :TERMination | ON \| [OFF] | TERM | |
| :BUFfer | | | |
| :SIZe | 8 - [65535] | | |
| :DATA | D16 \| [D32] | | |
| :CLOCK | | | |
| :FREQuency | value | | |
| :DATA | string | DATA | Send data |
| :DATA? | string | DATA? | Read data |
| :DATAT | string | DATAT | Send terminated data |
| :UPdate | | UP | Update UART |
| :ERRor? | (0, “No error”) | | Reads error |
| :VERSion? | (1994.0) | | Reads version |
| **INSTrument** | | | |
| :NSELect | 1 - 8 [1] | C | Channel select |

## TABLE 3-9 VXI-5539A SCPI COMMAND TREE - SDLC MODE

| Keyword | Parameter | Short Form | Notes |
|---|---|---|---|
| **SYSTem** | | | |
| :COMMunicate | | | |
| :SERial | | | |
| [:RECeive] | | | |
| :PROTocol | [ASYNC] \| SDLC | PROT | |
| :FRAMe | | | |
| :SIZe | 2-30 [30] | SIZE | |
| :BAUD | 50 - 1 Mbs [1 Mbs] | BAUD | |
| :BITS | 5-8 [8] | BITS | |
| :SBITs | [1] \| 2 | SBITS | |
| :PARity | | | |
| [:TYPE] | EVEN \| ODD \| [NONE] | PAR | |
| :ECHO | ON \| [OFF] | ECHO | |
| :LOOPback | [OFF] \| INT \| MON | LOOP | |
| :MODE | [232] \| 422FD \| 422HD \| 485 | MODE | |
| :HANDshake | ON \| [OFF] | HAND | |
| :RXCLK | [INT] \| EXT | RXCLK | |
| :TXCLKOUT | [IN] \| OUT | TXCLK | |
| :TERMination | ON \| [OFF] | TERM | |
| :BUFfer | | | |
| :DATA | D16 \| [D32] | | |
| :SIZe | 8 - [65535] | | |
| :CLOCK | | | |
| :FREQuency | value | | |
| :DATA | string | DATA | Send data |
| :DATA? | string | DATA? | Read data |
| :DATAT | string | DATAT | Send terminated data |
| :UPdate | | UP | Update UART |
| :ERRor? | (0, “No error”) | | Reads error |
| :VERSion? | (1994.0) | | Reads version |
| **INSTrument** | | | |
| :NSELect | 1 - 8 [1] | C | Channel select |

## TABLE 3-9 VXI-5539A SCPI COMMAND TREE - ALLMODES

| Keyword | Parameter | Short Form | Notes |
|---|---|---|---|
| **STATus** | | | |
| :OPERational | | | WTG status in bit 5 |
| [:EVENt]? | | | |
| :CONDitional? | | | |
| :ENABle | 0-7FFF [0] | | |
| :PTRansistion | 0-7FFF [All 1s] | | |
| :NTRansistion | 0-7FFF [0] | | |
| :OPERational | | | |
| [:EVENt]? | | | |
| :CONDitional? | | | |
| :ENABle | 0-7FFF [0] | | |
| :PTRansistion | 0-7FFF [All 1s] | | |
| :NTRansistion | 0-7FFF [0] | | |
| :PRESet | | | |
| **SOURCE** | | | **Pulse Output** |
| :PULSe | | | |
| :PERiod | .0001-3. [.01] sec | | |
| :WIDTh | .0001-3. [.0025] sec | | |
| **CALibrate** | | | **Setup** |
| :DATe | mm/dd/yyyy | | |
| :DEFault | | | |
| :IDN | string | | |
| :LOCK | ON \| [OFF] | | |
| :MANufacturer | 0-4095 [4073] | | VXI Mfgr Number |
| :MODel | 0-4095 [539] | | VXI Model Number |

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 Flash memory (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 Flash memory.
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.

## TABLE 3-10 SCPI COMMAND REFERENCE

| Command | Default | Function |
|---|---|---|
| **SYSTem** | | **Starts System Subsystem** |
| :COMMunicate | | Starts COMMunicate branch |
| :SERial | | Serial keyword |
| [:RECeive] | | Optional Receive branch keyword |
| :PROTocol | ASYNC | Selects serial protocol. ASYNC uses a Start and Stop bit for each character. SDLC is a packet protocol with flag bytes, data bytes and CRC bytes. There are no stop or start bits. Values are ASYNC and SDLC |
| :FRAMe | | SDLC Frame keyword |
| :SIZe | 30 | SDLC command that sets the number of data bytes in the SDLC packet. Value must be an even number of bytes from 2 to 30. |
| :BAUD | 9600 (ASYNC)<br>1 Mbs (SDLC) | Sets serial bit rate to standard rates or to closest value for nonstandard rates. ASYNC value is 50 to 460800 b/s. SDLC value is 50 to 1000000 b/s. Some rates may require a special oscillator frequency. |
| :BITS | 8 | ASYNC command sets number of bits per character. Value is 5,6,7 or 8. |
| :SBITs | 1 | ASYNC command sets number of stop bits. Value is 1 or 2. |
| :PARity | NONE | ASYNC command enables parity generation and checking. Values are ODD, EVEN or NONE. |
| [:TYPE] | | Optional Parity keyword. |
| :ECHO | OFF | Enables echo transmission of any received character when internal loopback is off. Used to test connections to a serial device. Echo on blocks VXI to serial transmission. Values are OFF and ON. |
| :LOOPback | OFF | Enables internal UART loopback for testing internal logic and UART. INT selects internal loopback only. MON selects internal loopback and transmits the test message. Values are OFF , INT and MON. |
| :MODE | 232 (ASYNC)<br>422FD (SDLC) | Selects serial interface signal type. 232 enables RS-232 drivers and receivers. 422FD enables RS-422 drivers and receivers for four-wire data systems and keeps the transmitter enabled at all times. 422HD tristates the RS-422 driver when not transmitting data. 485 enables the RS-485 transceiver for two wire systems and tristates the RS-485 driver when not transmitting. Async values are 232, 422FD, 422HD and 485. SDLC values are 422FD. |

## TABLE 3-10 SCPI COMMAND REFERENCE

| Command | Default | Function |
|---|---|---|
| HANDshake | OFF | ASYNC command enables/disables RS-232 handshake lines. When handshaking is enabled, DTR is set true when the channel's FDC receive buffers are enabled or when the DATA? command is executed. When handshaking is disabled, DTR is set true.<br>Also when enabled, CTS must be true before a channel will transmit and DCD must be true before a channel can receive data. |
| :RXCLK | INT | Selects internal 7.3728 MHz clock for ASYNC baud rates or external RX clock. The EXT selection is valid only when RS-422 or RS-485 signals are selected. RXCLK is 16x for Asynchronous mode, 1x for Synchronous mode. Values are INT and EXT. |
| :TXCLKOUT | IN | Enables TX clock transmitter Valid only when RS-422 or RS-485 signals are selected. TXCLK is generated from the RXCLK signal. TXCLK is 16x for Asynchronous mode, 1x for Synchronous mode. Values are IN and OUT. |
| :TERMination | OFF | ASYNC command connects RS-422 RX lines and RS-485 TX/RX lines to a termination network. Values are OFF and ON. |
| :DMA | OFF | ASYNC command enables DMA transfer of received data to the FDC receive buffers. Use DMA only for high-speed, continuous data transfer. DMA on disables EOM character checking and Message counting. Value is ON and OFF. |
| :BUFfer | - | Identifies FDC buffer settings. |
| :DATA | D32 | Sets FDC buffer access word size. Values are D16 and D32. |
| :SIZe | 65535 | Sets the number of bytes the FDC receive buffer can hold before the VXI-5539A switches buffers (switch point). Will override MESSage setting. Values are 8 to 65535 bytes and should include the 8 FDC header bytes. |
| :CLOCK | | Identifies Clock settings. |
| :FREQuency | 7372800 | Sets oscillator frequency in Hz for baud rate calculations. Default is the VXI-5539A's internal clock frequency. The clock setting must be changed if an external RXCLK source is selected or if the internal oscillator is changed. Values are 100 000 to 20 000 000. (spaces shown for clarity) |
| :DATA | | Diagnostic command that sends a test serial data message without adding any termination characters to the message. Uses the VXI Word Serial Message to send or receive serial data. Maximum data size is 1024 characters. The DATA command will time out if it cannot place data in the UART's FIFO, set the Execution Error bit in the ESR Register and display Err on the front panel. Value is a string of 8-bit characters.<br>Note-Null characters are not allowed in the DATA command. |

## TABLE 3-10 SCPI COMMAND REFERENCE

| Command | Default | Function |
|---|---|---|
| :DATA? | | DATA? is an ASYNC mode query that enables the selected channel's receiver. A query when no data is present returns a 'no data available' response. The serial channel should be re-initialized before using it with the FDC buffers. DATA? does not check for EOM character. |
| :DATAT | | ASYNC diagnostic command that sends a test serial data message and automatically appends a CR LF termination sequence to the message. Uses the VXI Word Serial Message to send or receive serial data.. Maximum data size is 1024 characters. The DATAT command will time out if it cannot place data in the UART's FIFO, set the Execution Error bit in the ESR. and display Err on the front panel. Value is a string of 8-bit characters.<br><br>DATAT? query is identical to DATA? |
| :ERRor? | | Queries SCPI error value. The standard response for no error is 0, "No error" |
| :VERsion? | | Queries SCPI version. Response is 1995.0 |
| **INSTrument** | | **Starts Instrument Subsystem** |
| :NSELect | 1 | Selects channel number. Values are 1 to 8 for Octal units and 1-4 for Quad units. |
| **STATus** | | **Starts Status Reporting Subsystem** |
| :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 is 0 to 255. |
| :NTRansition | 0 | Sets the negative Transition register. Value is 0 to 255. |
| :PREset | | Sets the selected Enable Register, PTR and NTR registers to their default values (0, 255 and 0 respectively) so the VXI module detects positive changes. |

## TABLE 3-10 SCPI COMMAND REFERENCE

| Command | Default | Function |
|---|---|---|
| **SOURce** | | **Starts Pulse Subsystem** |
| :PULSe | | Identifies Pulse Output. |
| :PERiod | 0.01 | Sets Pulse period in seconds. Value is 0.0001 (100 microseconds) to 3.000 seconds. |
| :WIDTh | 0.0025 | Sets Pulse width in seconds. Value is 0.0001 (100 microseconds) to 3.000 seconds. |
| **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, 5539A, S/N 012345, Rev 00.00 Ver 00.04.17.** |
| :DATe <date> | | Saves IDN message and date. Date is in mm/dd/yyyy format. |
| :DATe? | | Queries the calibration date. The response is 00/00/0000 for factory default settings. |
| :DEFault | | Sets Flash to factory settings. |
| :LOCK | 0 | Disables configuration commands when On. Disabled commands are not recognized as valid SCPI commands. Values = 0\|1 or OFF\|ON. Table 1-4 indicates which commands can be locked. |
| :MANufacturer | 4073 | Sets VXI Manufacturer Number in the VXI ID Register. The Manufacturer Number is obtained from the SCPI Consortium. Default value is ICS Electronics' number. Value is 0 to 4095. |
| :MODel | 539 | Sets module Model Number in the VXI Device Type Register. Default is 539 for the Model 5539A. Value is 0 to 4095. |

## 3.9 Programming Guidelines

### 3.9.1 Overview

The VXI-5539A has four or eight serial channels that can independently transmit and/or receive data. Each channel has its own baud rate generator, transmit and receive circuits. To send a channel commands, the user must **first select the channel with the INST subsystem command**. This will allow the SYSTEM configuration commands to go to the correct channel. Next set the channel protocol to select Asynchronous or SDLC operation. Next set the baud rate, mode, signal type, character format and other parameters as required to configure the channel. When done, use the *SAV 0 command to save the current configuration for all of the channels as the module's power-on default setting. Note that changing any serial parameter resets the channel's UART to the idle state.

e.g. The following commands to set channel 3 to 19,200 baud rate and saves the change:

```
*CLS
INST:NSEL 3
SYST:COMM:SER:BAUD 19200
*SAV 0
```

### 3.9.2 VXI Programming Requirements

It is important that the programmer understand the difference between a Word Serial Command and a Word Serial Message when working with the VXI-5539A. Refer to paragraphs 3.3.3. and 3.4 for the definitions and command examples. Consult your Slot 0 Controller's instruction manual or contact the manufacturer's application support people for the directions on how to send VXI Word Serial Commands.

Check your VXI Slot 0 Controller for A32 Address Space Capability. This capability is required to address the VXI-5539A's FDC buffers. Most GPIB-VXI modules do **not** have this capability

### 3.9.3 Testing an Asynchronous Channel

Connect a serial device or terminal to the channel to be tested. A PC running Hyperterminal can also be used as the terminal. Use an interactive control program to send the setup commands to the VXI-5539A. Setup the channel for the desired baud rate, mode etc.

For the initial test, set :LOOPBACK to INT. Use the DATAT? command to enable the channel's receiver and to send and receive a simple data message.

e.g. The following commands read back an internal loopback message from channel 1:

```
*CLS
INST:NSEL 1
SYST:COMM:SER:LOOP INT
SYST:COMM:SER:DATAT?                'read 'no data available' response
SYST:COMM:SER:DATAT test message
SYST:COMM:SER:DATAT?                'read 'test message' response
```

Set :LOOPBACK to OFF and send a test message to the serial device or terminal. Read back a simple message from the terminal of serial device.

e.g. The following commands send and receive test messages from a serial device or terminal

```
*CLS
INST:NSEL 1
SYST:COMM:SER:LOOP OFF
SYST:COMM:SER:DATAT?                'read 'no data available' response
SYST:COMM:SER:DATAT test message    'sends a message to the device
      .                             'device sends a response to the VXI-5539A
SYST:COMM:SER:DATAT?                 'read device response message
```

### 3.9.4 Transmitting Data

Fast Data Channel buffers are the normal way serial data is transferred between the VXI Commander and the VXI-5539A. The Fast Data Channel buffers are used as pairs to pass serial data in one direction. The VXI controller places data in one buffer while the VXI-5539A owns the other buffer. Bits in the Fast Data Channel header define the number of bytes in the FDC buffer and who 'owns' the buffer.

Four fast data channels are required for each VXI-5539A serial channel. The Fast Data Channel buffers are initialized by the Resource Manager function in the Slot 0 Controller. Refer to Section 3.5 for information about the FDC buffers and their organization in the VXI-5539A. Refer to Fast Data Channel Addendum for detailed information on how to setup and use the FDC buffers.

Initialize the Transmit Fast Data Channel buffers as Transfer to Servant and in Pair and Stream Mode before transmitting data. The following expanded FDC Word Serial Commands will initialize channel 2's FDC transmit buffers in pair and stream mode. Note that the commands initialize both transmit FDC channels and need only be sent to the first (even numbered) channel.

```
Expand Channel Initialize (0x9F92)
Expand Transfer to Servant (0x7F62)      */sets transfer direction/*
```

The sample program in the Fast Data Channel Addendum includes the function XFDC_OpenWriteBuffers which shows how to use these commands. Refer to Figure A1 in the Fast Data Channel Addendum for detailed instructions on initiating the FDC buffers.

To transmit data, the user writes data into the first transmit FDC buffer and sets the header bit to RDY. Transmission of the first message starts as soon as a buffer with data is passed to the VXI-5539A. The user can 'fill' the second FDC transmit buffer while the VXI-5539A is transmitting data from the first

FDC buffer. When the transmitter empties the first FDC buffer, its header WDY bit is set and the user can again write data into it. The FDC buffers must be used in their A/B sequence to avoid getting out of sync with the VXI-5539A transmitter.

If the serial channel was tested with the DATA and/or the DATAT commands, the UART should be reinitialized before using it with the FDC bufers.

### 3.9.5 Receiving Data

Serial data is received after the receive Fast Data Channels are initialized in the Transfer to Commander direction and in the Pair and Stream Mode. Serial data is received, packed into 16-bit words and placed into the first FDC receive buffer until the number of characters in the buffer reaches the preset 'full' value. The preset 'full' value can be a character (byte) count or when the buffer is physically full. The receiver automatically switches to the second FDC receive buffer and continues receiving serial data.

The SYST:COMM:SER:BUFF:SIZE command sets the point (in bytes) at which the receive buffer becomes 'full' and the receiver switches to the other buffer. This command only sets the buffer's usable area, it does not change the number of bytes allocated to the buffer. When the data in the receive buffer reaches the 'full' point, the VXI-5539A can generate a VXIbus interrupt to notify the user that the buffer is 'full'. The user should set this point so that the receive buffers are swapped when a fixed number of messages have been received. The byte count in the Size Command is all of the bytes in the FDC buffer including the FDC header.

e.g. The following commands will set the VXI-5539A's channel 2's FDC receive buffers to receive 48 characters.

```
INST:NSEL 2
SYST:COMM:SER:REC:BUF:SIZE 56
```

An alternate reception method is to set the buffers full point to a large number, run the test, receive the data and then use the swap buffer command to take possession of the buffer and read out the data.

### 3.9.6 Receiving Messages with an EOM Character

The VXI-5539A can receive data as messages rather than as bytes. This method is easier to implement than byte counting when the messages asynchronous and of varying lengths. The EOM character recognition increments the Message counter. The FDC buffers are swapped when the desired number of messages have been received.

To use EOM character detection, leave the buffer size set to the default value of 65,535 bytes. Set the EOM character to the desired termination character. For multiple messages in the FDC buffer, set the MESSage parameter to the desired number.

e.g. The following commands set the VXI-5539A to terminate messages with a carriage return (13) character and to accumulate 10 messages in the buffers.

```
INST:SEL 2
SYST:COMM:SER:EOM 13
SYST:COMM:SER:MESS 10
```

### 3.9.7 SDLC Mode Operation

In the SDLC mode, the VXI-5539A sends and receives data as packets of bytes. To use the SDLC mode, the baud rate and frame size need to be set. The frame size is the number of bytes transmitted. Frame size is an even number between 2 and 30. The transmit bufers should be loaded with exact multiple of the frame size.

. **INST:NSEL 3** 'selects channel 3
**SYST:COMM:SER:FRAME:SIZE 30** 'sets channel 3 to send 30 bytes

When receiving SDLC data, the VXI-5539A will save a frame of received data in the FDC buffer. Next the VXI-5539A appends a 16-bit status word from the Serial Interface controller to the data in the FDC buffer. The bit pattern for the Status Word is shown in Figure 3-6. This process repeats until the buffer reaches the switch point. The buffer size for setting the switch point is :

8 header bytes + ( n frames x (frame size + 2))

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2ndBE | | 1stBE | | Rx Residue | | | ShortF CV Type | Exited Hunt | Idle Rcvd | Break/ Abort | Rx Bound | CRCE /FE | Abort /PE | RX Over | RX Avail |

**Figure 3-6 Serial Controller Status W ord**

The Serial Interface Controller Status Word is a 16-bit word as shown in Figure 3-6. It has a normal value of 0x00E1 for good messages. Bit 5 becomes a 0 if there is a problem with the packet.

### 3.9.8 Clock Selection

A VXI-5539A serial channel can use several sources for its transmit and receive clocks. The default setting is to use the internal 7.3728 MHz oscillator as the clock source for the channel's baud rate generator. In the RS-232 mode, this is the only source that you can use. In the RS-422 and RS-485 modes, an external RX clock and/or an external TX clock can be used in place of the internal clock. Also in the RS-422 and RS-485 modes, the TX clock transceiver can be set to provide a clock output for an external device.

e.g. The following commands will set the VXI-5539A's channel 3 for external clock inputs.

**INST:NSEL 3**
**SYST:COMM:SER:MODE 422FD**
**SYST:COMM:SER:RXCLK EXT**
**SYST:COMM:SER:TXCLKOUT IN**

### 3.9.9 Timing Pulse Output

Eight channel VXI-5539As have an RS-422 Pulse output that can be programed from 10 kHz down to once every three seconds. The pulse width is also programmable. Program pulse width first when reducing the period. Program period first when increasing the pulse width.

e.g. The following commands will set the Pulse Output to generate a 1 ms wide pulse at a 25 Hz rate.

```
SOURCE:PULSE:PERIOD 0.04
SOURCE:PULSE:WIDTH 0.001
```

# 4

# Theory of Operation

## 4.1 GENERAL

The VXI-5539A is a VXIbus module with four or eight programmable serial interfaces and clock pulse output for interfacing with devices that use asynchronous or synchronous serial data. The serial interfaces are independently programmable and can be set to operate with RS-232, RS-422 or RS-485 signals. The VXI-5539A has an internal oscillator for generating all common baud rates and can work with an external clock for synchronous data. The VXI-5539A uses VXI Fast Data Channels to achieve continuous operation on all channels without tieing up the VXI backplane.

The VXI-5539A module is available in two versions. The four channel version has four 9-pin DE connectors on the front panel. The eight channel version has two 37-pin connectors on the front panel. The eight channel version includes a programmable pulse output that can be used to trigger events or for other test functions. Their FDC transmit algorithm has been modified to match the end user's software.

The VXI-5539A is a Message based VXI instrument in the A16 address space and also contains Fast Data Channel (FDC) buffers in the A32 address space. The user may send commands and data to the VXI-5539A as Word Serial Messages. Serial is normally transmitted by placing the data in the FDC buffers. Refer to the Fast Data Channel Addendum for a description of how the Fast Data Channel buffers work.

## 4.2 VXI-5539A BLOCK DIAGRAM DESCRIPTION

A simplified Block Diagram of the VXI-5539A is shown in Figure 4-1. The VXI-5539A is assembled from two printed circuit board assemblies, a Model VXI-5526-14 VXI Interface Card and a Serial Interface card. The Model VXI-5526 -14 is a 386EX processor based interface with 4 Mbyte of DRAM memory that contains the FDC buffers. The Serial Interface Card contains the serial channel drivers, UARTs and FIFOs. A 16-bit 386EX expansion bus is used to communicate with Serial Interface Card.

Communication with the VXIbus is via a 16 or 32 bit wide data bus. Word serial messages are

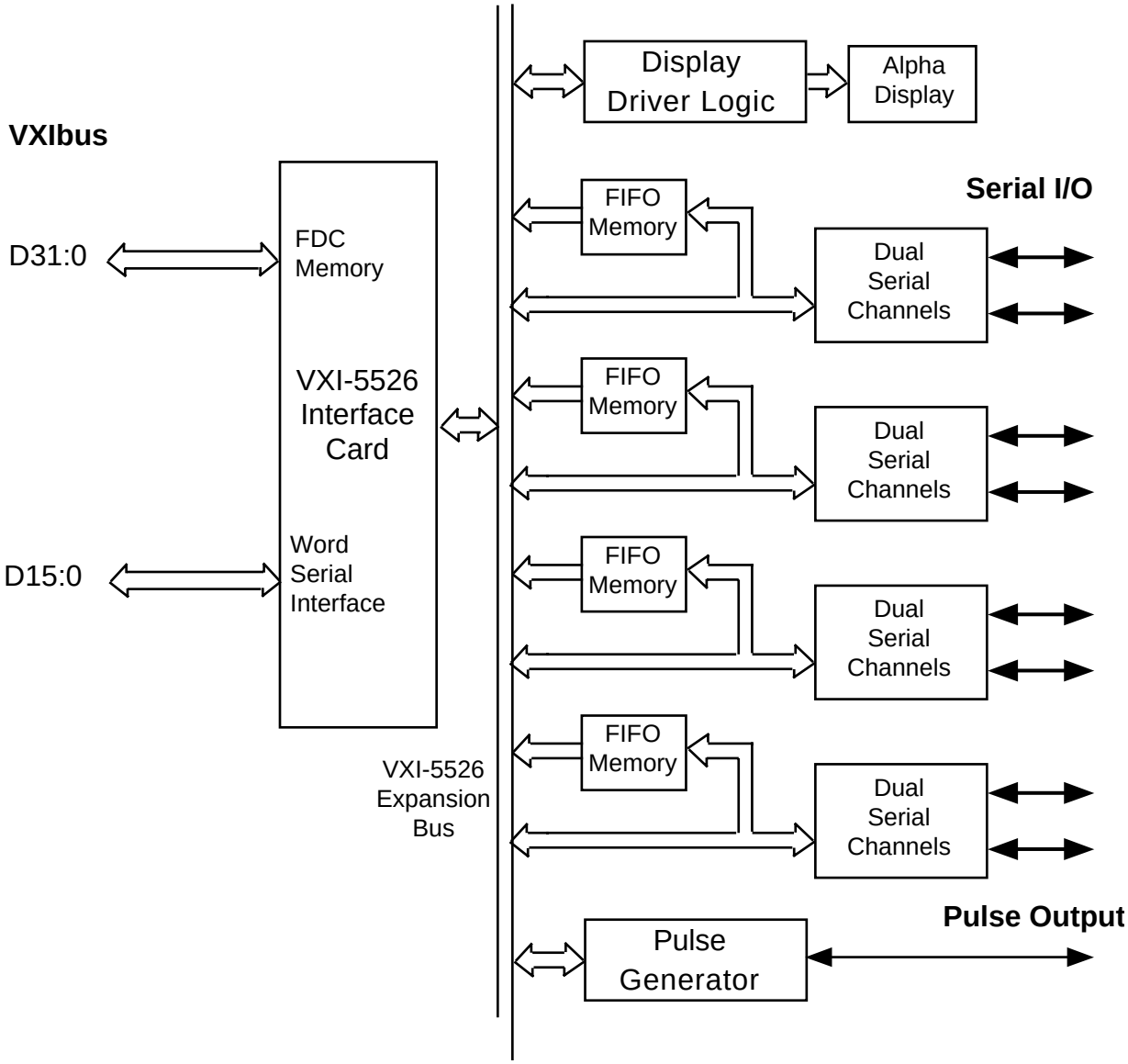



**FIgure 4-1 VXI-5539A Block Diagram**

handshaked in on a word-by-word basis, parsed and executed. The Word serial messages are used to configure and test the serial channels, to initialize or close the FDC channels and to control the pulse output. Transmit data is placed in the FDC buffers located in the VXI-5526's DRAM memory. FDC buffer accesses place the 386EX processor in a hold condition while the commander directly accesses buffers the VXI-5526's memory. Access and buffer ownership is controlled by management bits in the FDC buffer headers. When the FDC buffer is passed to the module, the 386 transfers data from the FDC buffer into the TX FIFO in the UART. It keeps the TX FIFO full until the message is completely transmitted.

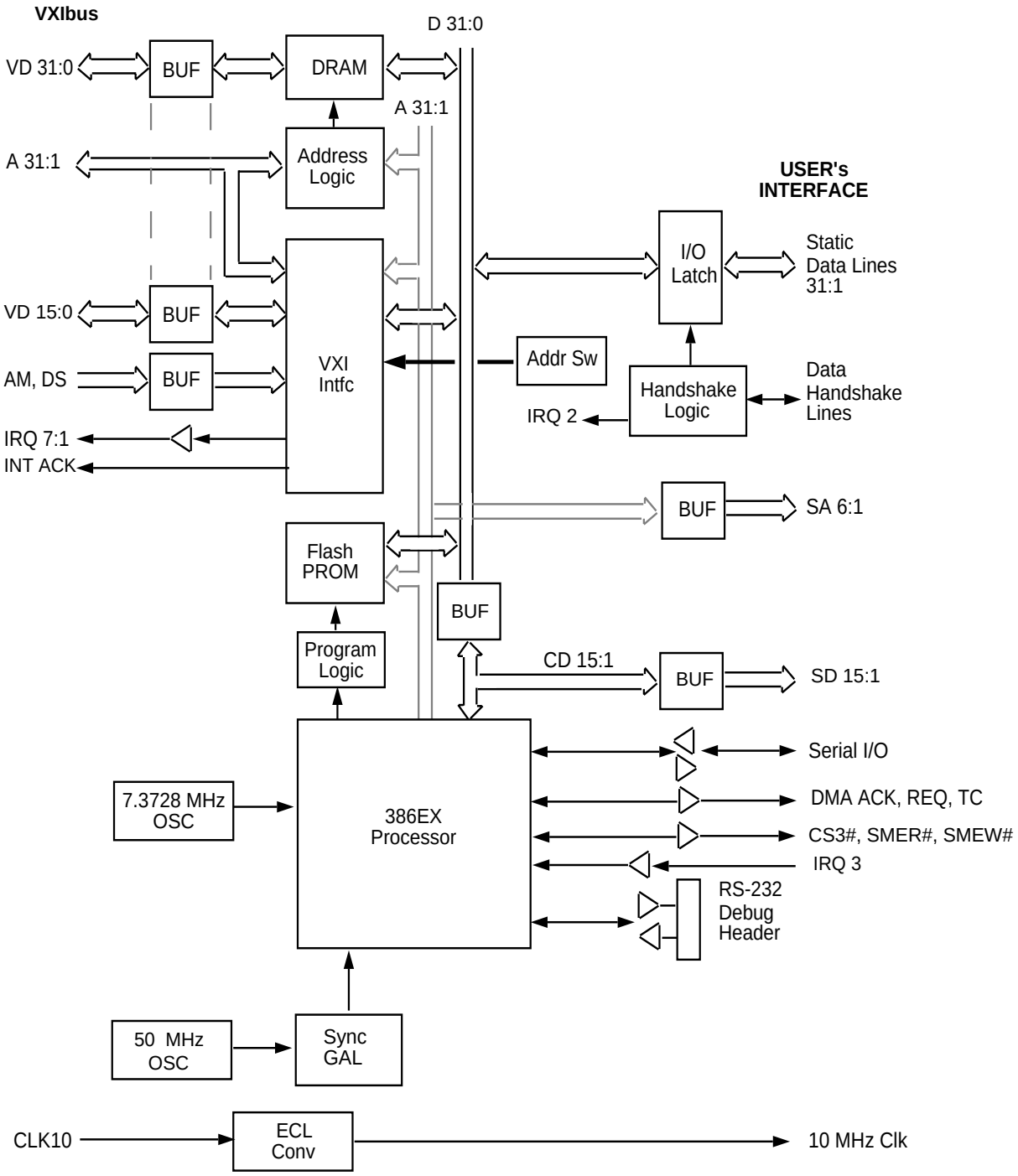



**Figure 4-2 VXI-5526 Block Diagram**

Received data is placed in the channel's RX FIFO and then moved into the receive FDC buffer. When the FDC buffer reaches the user set 'full' point, the receive buffers are switched and a VXI Interrupt is generated to alert the Controller that a buffer is 'full'.

## 4.3 VXI-5526 DESCRIPTION

A simplified Block Diagram of the VXI-5526-14 VXI Interface Card is shown in Figure 4-2. The VXI-5526-14 is basically a flat memory mapped PC. It contains a 386EX processor with 4 Mbyte of DRAM memory and Flash memory for program storage. It also has a local RS-232 interface for running the flash loader and debugger. The stored program is changed by downloading it over the serial link.

The VXI-5526-14 has a VXI Interface chip that contains the required VXI registers and communicates with the VXI bus. The on-card logical address switch is read and latched into the VXI Interface at power turn-on. VXI communication is done as a 16-bit word (D16) in A16 address space.

The VXI-5526-14 also contains two 16-bit registers that provide static digital I/O signals for the Serial Interface Card. The static signals can be programmed as inputs or outputs. In the case of the VXI-5539A, one register outputs higher order address bits to select the serial channel, display of pulse generator logic. The second register is used to read the serial channel's interrupt status. The VXI-5526-14 communicates with most of the logic on the Serial Interface Card via a 16-bit expansion bus. The expansion bus lets the program read or write to the Serial Interface at I/O addresses 300 HEX to 37E HEX.

## 4.4 SERIAL INTERFACE BOARD

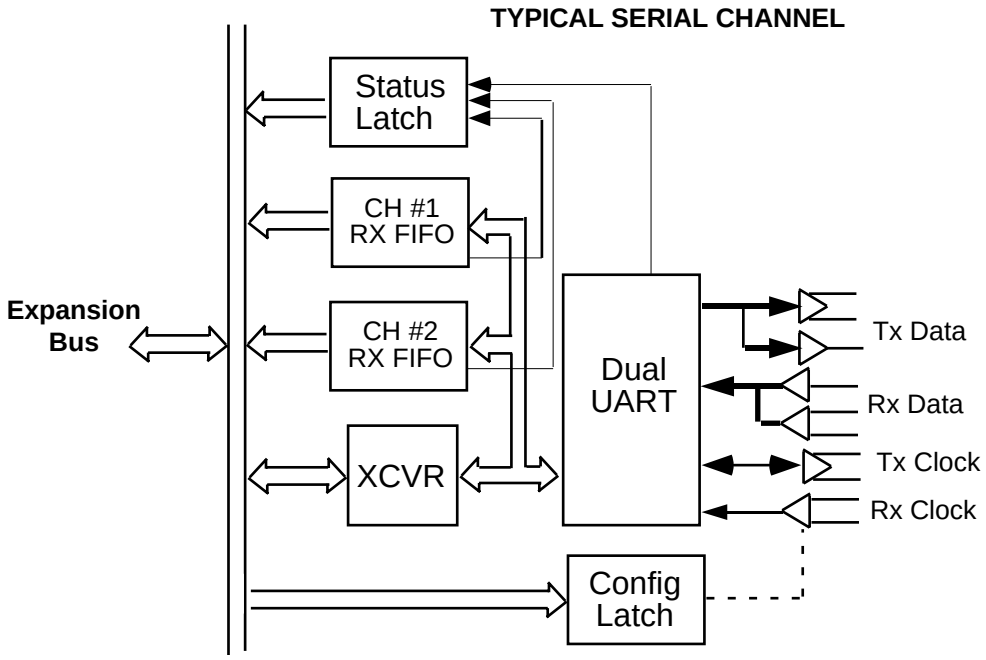
The Serial Interface Board contains the internal reference oscillator, the pulse generator, decode logic and dual serial channels. The Quad Serial Card has two dual channels; the Octal Serial Card has four dual channels. Figure 4-3 shows a typical Dual Serial Channel.

The standard value for the reference oscillator is 7.3728 MHz. This frequency is a 16x clock for 460,800 baud and can be divided down into all of the standard rates from 230,400 to 50 baud. The VXI-5539A's parser accepts any baud rate up to 460,800 and selects the closed divider ratio to produce the commanded baud rate. At high baud rates, the divider steps are very coarse and the resulting frequency might not be very close to the commanded value. At lower baud rates, below 10,000, the divider steps are much finer and the resulting frequency is very close to the commanded value. For an exact nonstandard baud rate, replace the VXI-5539A's reference oscillator with one that can be divided down to the desired baud rat e and change the oscillator frequency setting.

The pulse generator is an 82C54 Timer that is programmed to divide the reference oscillator down to 62,500 Hz. The 62,500 clock is then used as the clock to generate the pulse period and width clocks. The period clock sets the output latch, the width clock resets the latch at the end of the pulse width time. The output of the latch goes to a RS-422 driver and then to Connector J5.

When the user selects a channel, the processor writes to the output register on the VXI-5526-14 to set the correct value on address lines AL11 to AL8. Decoder logic on the Serial Interface Card decodes the binary value to generate the strobe pulses for each channel pair or for the display or for the pulse generator.

Figure 4-3 shows a typical VXI-5539A Dual Serial Channel. Each dual channel has a Status Latch,




Note - Second Configuration Latch and serial transceivers not shown

**Figure 4-3 Block Diagram - VXI-5539A Dual Serial Channel**

two RX FIFOs, a transceiver that connects the VXI-5526 expansion bus to the local UART bus, a PLD, a dual UART, two Configuration Latches and the necessary RS-232, RS-422 and RS-485 drivers and receivers. When the user selects a transmission mode, the processor writes tot he Configuration Latch to enable the desired drivers and receivers. The dual UART has two complete transmit and receive sections with independent baud rate generators. The UART transmit and receive sections have an internal 32 byte FIFOs. Each VXI-5539A serial channel has an external 512 word FIFO that supplements the UART's receive FIFO. The processor writes the baud rate, character format, and similar commands to the selected half of the UART to configure its operation.

When transmitting, the processor fetches transmit data from the FDC buffer and loads it into the TX FIFO in the selected channel's UART. Loading is done every 450 microseconds to maintain a continuous transmit data flow until the complete message has been transmitted. Depending upon the selected mode, the TX and RX clock can be selected from the internal reference oscillator or from an external source.

As RX data is received, it is assembled into 16-bit words and stored in the RX FIFO. The processor reads the Status Latch every 500 microseconds to see if there is any received data. If there is data in the FIFOs, it is transferred into the channel's RX FDC buffer. Because the length of a received message may not end on a 16 bit boundary, the processor checks the UART for a last character when the FDC channel is closed or shut down.

# 5

# Maintenance

## 5.1 INTRODUCTION

This section describes the maintenance, troubleshooting and repair procedures for the VXI-5539A Serial Interface card. The VXI-5539A does not require periodic calibration and has no internal adjustments.

## 5.2 TROUBLESHOOTING PROCEDURES

### 5.2.1 Self Test Failures

The VXI-5539A has a built in self test routine which is designed to detect basic circuit faults. The self test routine checks the Flash, DRAM and to some extent, the interface chips. Any self test failure is indicated by a blank front panel display or by a FAIL indication on the front panel display.

The failures are not field repairable and the unit must be returned to ICS for repair. Refer to paragraph 5.4 for return instructions.

Units under warranty should be returned to the factory for repair. Any attempts to repair a unit with out ICS's specific approval will void your warranty. If a failure is experienced, contact ICS before proceeding with any repairs.

## 5.3 TROUBLESHOOTING GUIDE

This section describes some of the common problems a user may encounter and the corrective action.

**TABLE 5-1 TROUBLESHOOTING GUIDE**

| Problem | Check | Probable Cause or Corrective Action |
|---|---|---|
| Unit does not respond to Slot 0 Controller or Resource Manager | LEDs all off | Internal processor malfunction Return for repair |
| | FAIL shown | Selftest failure. Reset the module and retry. If the failure persists, return for repair. |
| | Logical address setting | Check address switch setting. See Section 2 for directions on setting the address switch. |
| No TX Data | Interface not enabled | Verify interface enabled |
| | Wrong cable connection | Check cable wiring. See Section 2 for signal pin assignments. |
| | Clock not selected | Verify clock selection command and rate command is within module limits. Check continuous clock output for proper clock bit rate. |
| | Missing CB input | RS-232 mode with handshaking on requires CB input be true (high). Check input signal. |
| | Interface driver faulty | Test by jumpering TX outputs to RX inputs on same channel. Send and receive a test message on another channel to verify module operation. Repeat on the suspect channel. |
| No data in the receive buffer | Receive FDC channel not setup | Reinitialize the correct receive FDC buffer pair. Refer to the FDC Addendum for instructions. |
| | FDC channels not off while changing serial parameters | Check program. Close FDC channels before changing receiver setup. |
| | Missing CF input | RS-232 mode with handshale on reguires CF true to receive data. Check CF signal level for high ture input. |

**TABLE 5-1 TROUBLESHOOTING GUIDE CONTINUED**

| Problem | Check | Probable Cause or Corrective Action |
|---|---|---|
| Incorrect data in receiver buffer | Buffer full point setting | Messages may be garbled if buffer full point splits a message. Make buffer full point a multiple of a whole message or large enough to capture all of the expected receive messages. |
| Transmitter or receiver parameters were not responded to | FDC channel not closed during setup commands | Check program. Appropriate FDC channels must be closed when changing the transmit or receive setup parameters. |
| No Pulse Generator output | Bad pulse width and period settings | Check program for logical conflict between the current and new settings. Do not set a pulse width greater than the period or a period shorter that the pulse width. |
| | | Reset the module to reset the Pulse Generator |
| | -14 Model | Pulse Generator not installed in four channel modules. |
| | Bad driver | Check output with cable disconnected |
| DATA? command doesn't receive data | Channel receiver not enabled | Execute a DATA? query before attempting to receive data. |

## 5.4 RETURNING FOR FACTORY SERVICE

When returning a VXI-5539A 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 applications engineer will want to discuss the problem at this time to verify that the unit needs to be returned, or assist in correcting the problem. We have discovered that one-third of the difficulties customers call about can be resolved over the phone, rather than having to return a unit.

2. Write a description of the problem and attach it to the material being returned. Describe the installation, systems 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 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.

# 6

# Parts List and Location

## 6.1 INTRODUCTION

This section contains all information necessary to locate, identify, and order parts. The listing of parts and information for an exact replacement are shown in the attached parts list. All component locations are identified by reference designators in the parts list and in the attached circuit board assembly drawings. Figure 6-1 shows the component locations on the circuit board.

## 6.2 REPLACEMENT PARTS

### 6.2.1 Standard Parts

All parts can be purchased directly from ICS Electronics, at current market price, plus a handling charge. However, since most parts are standard electronic components, it is suggested that they be secured locally for prompt replacement. The parts list gives all pertinent information including the recommended manufacturers and manufacturers' part numbers. Where not noted, equivalent parts meeting the original part specifications may be substituted. A list of manufacturers by abbreviations is given in Table 6-1.

### 6.2.2 Special Parts

Parts marked with an ICS part number should be procured directly from ICS Electronics. These parts are manufactured or selected to satisfy specific requirements. Substitution of other parts might not yield equivalent performance and will void any warranty.

### 6.2.3 Parts Ordering Information

All orders should be directed to ICS Electronics, at the address shown on the front page, or to ICS Electronics in care of your local representative. When ordering, be sure to include the following information:

a. Part description and reference designator
b. part number, manufacturer, and stock number
c. Instrument model, serial number, and program revision level

## TABLE 6-1 LIST OF MANUFACTURERS BY ABBREVIATION

| Code | Company Name | Location |
|---|---|---|
| ABC | Allen Bradley | Milwaukee, WI |
| ACT | Advanced Components Tech | Redwood City, CA |
| ACW | Alpha - CW | Elizabeth, NJ |
| AD | Analog Devices | Norwood, MA |
| ADI | ADI Electronics, Inc. | Bohemia, NJ |
| ADV | Advanced Interconnections | Warwick, RI |
| AEP | Alco Electronics Products, Inc | Lawrence, MA |
| AIW | American Insulated Wire | Pawtucket, RI |
| AMA | Amatom | New Haven, CT |
| AMD | Advanced Micro Devices | Sunnyvale, CA |
| AMP | AMP, Inc. | Harrisburgh, PA |
| AMR | American Relays, Inc. | Gardena, CA |
| AMZ | American Zettler, Inc. | Irvine, CA |
| AND | Alpha Numeric Displays | Burlingame, CA |
| APH | Amphenol Corp. | Oakbrook, IL |
| APL | Amplicon | Brighton,England |
| APT | Aptronics Corporation | Mentor, OH |
| APX | Ampex | Sunnyvale, CA |
| ARC | Arco | Commack, NY |
| ARI | Aries Industries, Inc. | Gardena, CA |
| ARO | Aromat Corp./Matsushita | San Jose, CA |
| ARR | Arrow Hart | Hartford, CT |
| ARX | Amperex Electronics | Slatersville, RI |
| ASC | American Switches Corp. | Wakefield, MA |
| ASS | Assmann Corporation | Chandler, AR |
| AST | Asyst Software Technology, Inc. | Rochester, NY |
| ATM | Atmel Corporation | San Jose, CA |
| AUG | Augat, Inc. | Attleboro, MA |
| AVD | Aavid Eng., Inc. | Laconia, NH |
| AVT | Advantech Co. LRD | Taipei, Taiwan |
| AVX | AVX Corp. | Myrtle Beach, SC |
| BDI | Bud Industries, Inc. | Peoria, AZ |
| BEC | Beckman | Fullerton, CA |
| BEL | Belden Corp. | Richmond, VA |
| BER | Bergquist | Minneapolis, MN |
| BEV | Bevmar Industries, Inc. | Carson, CA |
| BHE | BH Electronics, Inc. | Burnsville, MN |
| BLF | Bel Fuse, Inc. | Jersey City, NJ |
| BOR | Bourns | Riverside, CA |
| BRG | Berg Electronics | New Cumberland, PA |
| BRN | Burr-Brown | Tuscon, AZ |
| BTD | BT&D Technologies | San Jose, CA |
| BUC | Buchanan | Union, NJ |
| BUD | Budwig | Los Angeles, CA |
| BUR | Burndy | Norwalk, CT |
| BUS | Bussmann | Earth City, MO |
| BVR | Bivar | Santa Ana, CA |
| C&K | C&K Components | Newton, MA |
| CAB | Cable Connections | Cambell, CA |
| CAC | Circuit Assembly Corp. | Irvine, CA |
| CAD | Caddock | Riverside, CA |
| CAM | Ecam Technology | Scottsdale, AZ |
| CAN | Cannon | Santa Ana, CA |
| CAT | Catalyst Semiconductor | Santa Clara, CA |
| CAX | Calex Mfg. Co., Inc. | Pleasant Hills, CA |
| CDE | Cornell Dublier | Newark, NJ |
| CEL | Cellotape | Sunnyvale, CA |
| CHE | Cherry Electronics Products | Highland Park, CA |
| CHO | Chomerics, Inc. | Woburn, MA |
| CHR | Cuttler Hammer | Milwaukee, WI |
| CJO | Cinch Mfg. Company | Elk Grove Village, IL |
| CMS | Component Mfg. Service | Bridgewater, MA |
| CNT | Centralab | Los Angeles, CA |
| COL | Columbia Electronic Cables | Los Angeles, CA |
| CON | Condor Electronics | Sunnyvale, CA |
| COR | Corcom, Inc. | Chicago, IL |
| CRD | Concord Electronics Corp. | New York, NY |
| CRN | Corning Glass Works | Corning, NY |
| CRZ | Crazy Glue | Chicago, IL |
| CTK | Crystek Corp. | Fort Meyers, FL |
| CTR | Centre Engineering | State College, PA |
| CTS | CTS Corp. | Elkhart, IN |
| CYP | Cypress Semiconductor | San Jose, CA |
| DAL | Dale Electronics, Inc. | Columbus, NB |
| DEL | Delta Product Corporation | Fremont, CA |
| DEU | Deutsch Fastener Corp. | El Segundo, CA |
| DIA | Dialco | Brooklyn, NY |
| DII | Diodes Incorporated | Chatsworth, CA |
| DLL | Dallas Semiconductor | Dallas, TX |
| DSP | DSP Development Incorp. | Cambridge, MA |
| EBI | Ensign-Bickford Ind. | Simsburg, CT |
| ECL | Ecliplek Corporation | Fountain Valley, CA |
| EDC | Edac, Inc. | Ontario, Canada |
| EEE | Eaton Electrical/Electronic | New Haven, CT |
| ELL | Elec-trol | Saugus, CA |
| ELM | Elma Electronic, Inc. | Fremont, CA |
| ELN | Elna America, Inc. | Carson, CA |
| EMC | Electronic Molding Corp. | Woonsocket, RI |
| ENT | Entrelec | Spring Valley, NY |
| EPI | Epitek, Kanta | Ontario, Canada |
| ERG | Endicott Research Group | Endicott, NY |
| ERI | Erie Technology Products | Erie, PA |
| ESC | ESC Electronics Corp. | Palisades Park, NJ |
| ESI | EMC Shielding Incorprated | Fall River, MA |
| EUR | Euro-Dip | Dattwil, Switzerland |
| EVT | Electrovert, Inc. | Elmsford, NY |
| EWC | EWC, Inc. | Kenilworth, NJ |
| EXE | Exel Microelectronics | San Jose, CA |
| EXL | Excel | Chicago, IL |
| EXR | Exar Corporation | Sunnyvale, CA |
| FAS | Fastex | Des Plains, IL |
| FCL | Freedom Crystal Lab | Freedom, CA |
| FIS | Fairchild Inst. & Controls Div. | Mountain View, CA |
| FLK | Fluke | Santa Clara, CA |
| FOX | Fox Electronics | Cape Coral, FL |
| FSC | Fairchild Semiconductor Div. | Mountain View, CA |
| FUJ | Fujitsu Microelectronics | San Jose, CA |
| GAL | Gillis and Lane | Redwood City, CA |
| GAM | G & A Mounting Bracket | Issaquah, WA |
| GCE | GC Electronics | Rockford, IL |
| GE | GE Semiconductor | Syracuse, NY |
| GHI | Grayhill | La Grange, IL |
| GHW | Grant Hardware Company | City of Industry, CA |
| GI | General Instruments | Hicksville, NY |
| GMS | Globe Manufacturing Sales, Inc. | Mountainside, NJ |
| GOR | Gordos Corp. | Bloomfield, NJ |
| GRC | Gries Reproduction Co. | New Rochelle, NY |
| GSI | General Semi. | Tempe, AZ |
| GTI | Goldstar Tecnology, Inc. | Sunnyvale, CA |
| HAM | Hamlin Relays | Lake Mills, WI |
| HAR | Harris Semiconductor | San Jose, CA |
| HEY | Heyboer Transformers | Grand Haven, MI |
| HHS | Herman Smith, Inc. | Brooklyn, NY |

## TABLE 6-1 LIST OF MANUFACTURERS BY ABBREVIATION (CONT.)

| Code | Company Name | Location |
|---|---|---|
| HIT | Hitachi | Toyko, Japan |
| HOL | Holmberg Electronics Corp. | Inman, SC |
| HP | Hewlett-Packard | Palo Alto, CA |
| HRS | Hirose Electric, Inc. | Chatsworth, CA |
| HUR | Hurricane Electronics | Hurricane, UT |
| HYU | Hyundai Electronics America | San Jose, CA |
| ICO | Ico-Rally Corp. | Palo Alto, CA |
| ICS | ICS Electronics | Milpitas, CA |
| IDE | IDEC Corporation | Sunnyvale, CA |
| IDI | Industrial Devices, Inc. | Edgewater, NJ |
| IDT | Intergrated Device Technology | Santa Clara, CA |
| IEE | Industrial Electronic Engineer | Van Nuys, CA |
| IER | IERC | Burbank, CA |
| IMC | Inmac | Santa Clara, CA |
| INE | Ines GMGH | Koln, Germany |
| INL | Intel | Sunnyvale, CA |
| INT | Intervor | Melville, NY |
| IRC | TRW/IRC Resistors | Philadelphia, PA |
| ISC | Instrument Specialties | San Jose, CA |
| ISL | Intersil | Mountain View, CA |
| ITT | ITT General Controls | Glendale, CA |
| JAR | Jarome Wire | Mountain View, CA |
| JFR | Jeffers Electronics | Nogales, AZ |
| JWM | J.W. Miller | Rancho Dominguez, CA |
| KAP | Kappa Networks Inc. | Rahway, NJ |
| KCC | Keltron Connector Company | Bayshore, NY |
| KEM | Kemet | Greenville, SC |
| KES | Kester | Chicago, IL |
| KEY | Keystone Electronics | New York, NY |
| KOA | KOA Speer Electronics | Santa Ana, CA |
| KOS | Koszeigi | South Bend, IN |
| KYO | Kyocera International, Inc. | San Diego, CA |
| LAN | Lansing Instrument Corp. | Ithaca, NY |
| LAT | Lattice Semiconductor | Portland, OR |
| LCM | L-Com Incorporation | N. Andover, MA |
| LDI | Logic Dynamic, Inc. | Gardena, CA |
| LEA | Leader Tech | Tampa, FL |
| LED | Ledtronics, Inc. | Torrance, CA |
| LFS | Little Fuse, Inc. | Des Plaines, IL |
| LOC | Loctite Corporation | Hollister, CA |
| LOP | Zollin J. Lopaugh | San Francisco, CA |
| LYN | Lyntron, Inc. | Burbank, CA |
| LYT | Lytel Incorporated | Somerville, NJ |
| LYX | Lynx Enterprises | Watsonville, CA |
| MAN | Manhattan Electric Cable Corp. | Station Plaza-Rye, NY |
| MAR | Marcon | Northbrook, IL |
| MAX | Maxim Integrated Prod., Inc. | Sunnyvale, CA |
| MCB | Mepcocentalab | Fort Dodge, IA |
| MCC | Microchip Technology, Inc. | Chandler, AZ |
| MCI | MCI Transformer Corp. | Babylon, NY |
| MCK | McKenzie Technology | Fremont, CA |
| MCS | Microsemi Corporation | Scottsdale, AZ |
| MDA | Media Products | San Jose, CA |
| MDY | Midway Mfg. Co. | Franklin Park, IL |
| MEC | Magnum Electric Corp. | Erie, MI |
| MED | Meadows Mfg. | Sunnyvale, Ca |
| MIC | Microntran Company, Inc. | Valley Stream, NY |
| MIN | Minicomputer Accessories | Sunnyvale, CA |
| MIT | Mitsubishi Semiconductors | Sunnyvale, CA |
| MLR | Midland Ross Corporation | North Mankato, MN |
| MLX | Molex Products Co. | Downers Grove, IL |
| MLY | Mallory Dist. Prod. Co. | Indianapolis, IN |
| MMI | Monolithic Memories | Sunnyvale, CA |
| MMM | 3M | Minneapolis, MN |
| MON | Monsanto (Genl. Instr.) | Palo Alto, CA |
| MOS | Thompson Components | Carrollton, TX |
| MOT | Motorola Semicond. Prod. | Phoenix, AZ |
| MPC | Montior Products Corp. | Oceanside, CA |
| MPI | Multi Products International | Cedar Grove, NJ |
| MPR | Micropower Systems | Santa Clara, CA |
| MSC | Master Specialties Co. | Costa Mesa, CA |
| MSL | Mosel | Sunnyvale, CA |
| MTN | M-Tron Corporation | Yankton, SD |
| MUR | Murata Corporation | Marietta, GE |
| NAT | National Semi | Santa Clara, CA |
| NCS | Nor-Cal Seal Co. | San Leandro, CA |
| NDK | NDK America Inc. | Cupertino, CA |
| NEC | NEC Microcomputers, Inc. | Wellesley, MA |
| NEL | NEL Frequency Control, Inc. | Burlington, WI |
| NEM | Nova Electronics Mfg. Co. | Nutley, NJ |
| NIC | Nichicon | Chicago, IL |
| NKK | NKK Switches | Scottsdale, AZ |
| NMB | NMB Technologies | Chatsworth, CA |
| NSC | National Semicond. Corp. | Santa Clara, CA |
| OFT | Optical Fiber Tech. | Billerica, MA |
| OKI | OKI Semiconductor | Sunnyvale, CA |
| OMG | Omega Printing | Palo Alto, CA |
| OPT | Optrex Corp. | Torrance, CA |
| PAN | Panduit | Tinley Park, IL |
| PBD | Potter & Brumfield Company | Princeton, IN |
| PCD | Preci Dip | Oyster Bay, NY |
| PCL | Precision Crystal Lab | Santa Monica, CA |
| PEC | Pacific Electricord Co. | Gardena, CA |
| PIC | Piher International Corp. | Mt. Prospect, IL |
| PLX | PLX Technology, Inc. | Mountain View, CA |
| PMI | Precision Monolithics, Inc. | Santa Clara, CA |
| PNS | Panasonic | Seacaucus, NY |
| POS | Positronic Industries, Inc. | Springfield, MO |
| POT | Potter Company | Wesson, MS |
| POW | Power Sonic | Long Beach, CA |
| PRE | Precicontact, Inc. | Langhorne, PA |
| PRM | Prem Magnetics Incorporated | McHenry, IL |
| PTB | Phoenix Terminal Blocks, Inc. | Harrisburg, PA |
| PTS | Promptus Electronic Hdwr. | Lomita, CA |
| PUL | Pulse Engineering, Inc. | San Diego, CA |
| RAF | RAF Electronics Hardware | Seymour, CT |
| RAY | Raytheon | Mountain View, CA |
| RCA | RCA Corporation | Iselin, NJ |
| RCD | RCD Components, Inc. | Manchester, NH |
| RCH | Richlock | Chicago, IL |
| RED | Redwood Stationers | Menlo Park, CA |
| REM | Remee Products Corporation | Florida, NY |
| REN | Renco Electronics, Inc. | Deer Park, NY |
| RGA | RG Allen | Van Nuys, CA |
| RGS | Rogers Corporation | Tempe, AZ |
| RIC | Richco | Chicago, IL |
| RNU | Robinson Nugent | New Albany, IN |
| ROM | Rohm | Irvine, CA |
| RTE | RTE Aerovox, Inc. | New Bedford, MA |
| RUS | Russell Industries, Inc. | Oceanside, NY |

## TABLE 6-1 LIST OF MANUFACTURERS BY ABBREVIATION (CONT.)

| Code | Company Name | Location |
|---|---|---|
| RXD | RXD, Incorporated | Norfolk, NE |
| SAE | Stanford Applied Engineering | Long Beach, CA |
| SAM | Samsung Semi Conductor, Inc. | San Jose, CA |
| SAN | Sangamo Western, Inc. | Pickens, SC |
| SAR | Saronix | Palo Alto, CA |
| SCH | Schadow, Inc. | Eden Prairie, MN |
| SCN | Scanbe | El Monte, CA |
| SEC | Secma, Inc. | Irvine, CA |
| SEI | Seiko Instruments | Torrance, CA |
| SEK | Seiko Circuits | Sunnyvale, CA |
| SEQ | Seeq Technology, Inc. | San Jose, CA |
| SGC | Silicon General | Westminster, CA |
| SGL | Signal Transformer | Inwood, NY |
| SGS | SGS-Ates Semiconductor | Phoenix, AZ |
| SHA | Saha | New York, NY |
| SHK | Shakeproof | Edgin, IL |
| SHO | Shoin | Japan |
| SHP | Sharp | Manwah, NJ |
| SHR | Schurter, Inc. | Petaluma, CA |
| SIE | Siemens Components, Inc. | Iselin, NJ |
| SIG | Signetics Corp. | Sunnyvale, CA |
| SIL | Siliconix, Inc. | Santa Clara, CA |
| SLN | Sullins Electronics Corp. | San Marcos, CA |
| SMC | Standard Microsystems Corp. | Hauppauge, NY |
| SMK | SMK Corporation | Placentia, CA |
| SMN | Seimens | Iselin, NJ |
| SMO | S-Mos Systems, Inc. | San Jose, CA |
| SMT | Samtec | New Albany, IN |
| SOC | Socket Express | Princeton, NJ |
| SOL | Solid Electric Inc. | San Jose, CA |
| SOU | Souriau, Inc. | Van Nuys, CA |
| SPC | Spectra Strip | Garden Grove, CA |
| SPR | Sprague Electronics | Chicago, IL |
| STR | Star Micronics, Inc. | Piscataway, NJ |
| SWC | Switchcraft, Inc. | Chicago, IL |
| SYN | Synertec | Santa Clara, CA |
| SYO | Sanyo Electric, Inc. | Compton, CA |
| TAD | Tadiran | Tel-Aviv, Isreal |
| TAM | Tamura Corporation of America | Carson, CA |
| TAW | TAW | Burbank, CA |
| TBA | T&B Ansley | Los Angeles, CA |
| TCN | Tra-Con | Medford, MA |
| TEE | Telemecanique, Inc. | Westminster, MD |
| TEK | Tektronics, Inc. | Beaverton, OR |
| TEL | Teledyne Relays | Hawthorne, CA |
| TEM | Temple Industries, Inc. | Tecate, CA |
| TEX | Textool | Irving, TX |
| THC | Thermalloy Co. | Dallas, TX |
| TI | Texas Instruments | Dallas, TX |
| TKR | Thacker Container Co. | None available |
| TMI | Tri-Mag, Incorporated | Visalia, CA |
| TMX | Thermax Wire | Flushing, NY |
| TOM | Thomson Passive Comp. Corp. | Woodland Hills, CA |
| TOS | Toshiba America, Inc. | Tustin, CA |
| TRM | Trompeter | Chatsworth, CA |
| TRW | TRW Capacitors | Ogallala, NE |
| TYT | Tyton Corporation | Milwaukee, WI |
| UCH | United Chemcon | Rosemont, IL |
| USE | Useco | Van Nuys, CA |
| USI | Universal Semiconductor, Inc. | San Jose, CA |
| VAT | Varta | Elmford, NY |
| VEC | Vector Electronics Corp., Inc. | Sylmar, CA |
| VEM | Vermaline | Warwick, RI |
| VER | Bicc-Vero Electronics Corp. | Hamden, CT |
| VIK | Viking | Chatsworth, CA |
| VIR | Virginia Plastics | Roan Oke, VA |
| WDC | Western Digital Corp. | Newport Beach, CA |
| WEI | Weidmuller | Richmond, VA |
| WEK | Weckesser | Chicago, IL |
| WFD | Wakefield Engineering, Inc. | Wakefield, MA |
| WIN | Winchester | Oakville, CT |
| WLD | Waldom | Chicago, IL |
| WMA | Wima (Distr. by TAW, Inc.) | Burbank, CA |
| WPG | Western Packaging | Santa Clara, CA |
| WPS | Wescon Production Sockets | South Bend, IN |
| XEC | Xecom, Incorporated | Milpitas, CA |
| XIC | Xicor, Incorprated | Milpitas, CA |
| ZIL | Zilog | Cupertino, CA |

## 6.3 PARTS LISTS

Table 6-2 lists the recommended spare parts for 1 to 5 units. Tables 6-3 and 6-4 show the VXI-5539A module assembly breakdown. The subsequent tables break the circuit board assembly down to the component part level.

The parts list table contains both the ICS Electronics stock numbers and the manufacturers' true part numbers where applicable. True manufacturers' part numbers are not shown for industry standard parts nor for items unique to the VXI-5539A Serial Interface Card.

The Recommended Spare Parts list shows assemblies only because the VXI-5539A is constructed with surface mounted components that are very delicate and require a highly skilled electronic assembly depot to isolate and replace the defective part. The recommended procedure is to either return the whole module for repair or to replace the printed circuit assemblies in the field and then return the defective assembly to ICS Electronics for repair.

### TABLE 6-2 VXI-5539A RECOMMENDED SPARE PARTS

| LN# | PART# | QTY | DESCRIPTION | REF-DES | MFGR |
|---|---|---|---|---|---|
| 1 | 114602-03 | 1 | Assy VXI-5526-14 Board | A1 | ICS |
| 2 | 115034-01 | 1 | Assy VXI 5539A-14 4 Channel Intfc Card | A2 | ICS |
| 3 | 115034-02 | 1 | Assy VXI 5539A-18 8 Channel Intfc Card | A2 | ICS |

### TABLE 6-3 VXI-5539A-14 FOUR CHANNEL ASSEMBLY STRUCTURE

| LN# | PART# | QTY | DESCRIPTION | REF-DES | MFGR | MFGR PART # |
|---|---|---|---|---|---|---|
| 1 | 115030-01 | 1 | Ship Config VXI-5539A-14 | | ICS | |
| 2 | 115032-01 | 1 | Assy Module VXI-5539A-14 | | ICS | |
| 3 | 114602-03 | 1 | Assy VXI-5526-14 Board | A1 | ICS | |
| 4 | 114570 | 1 | Assy Display PCB | A3 | ICS | |
| 5 | 115034-01 | 1 | Assy VXI-5539A 4 Ch Board | A2 | ICS | |

### TABLE 6-4 VXI-5539A-18 EIGHT CHANNEL ASSEMBLY STRUCTURE

| LN# | PART# | QTY | DESCRIPTION | REF-DES | MFGR | MFGR PART # |
|---|---|---|---|---|---|---|
| 1 | 115030-02 | 1 | Ship Config VXI-5539A-18 | | ICS | |
| 2 | 115032-02 | 1 | Assy Module VXI-5539A-18 | | ICS | |
| 3 | 114602-03 | 1 | Assy VXI-5526-14 Board | A1 | ICS | |
| 4 | 114570 | 1 | Assy Display PCB | A3 | ICS | |
| 5 | 115034-02 | 1 | Assy VXI-5539A 8 Ch Board | A2 | ICS | |

## TABLE 6-5 VXI-5526-14 PCB ASSEMBLY PARTS LIST (114614-03)

| LN# | PART# | QTY | DESCRIPTION | REF-DES | MFGR | MFGR PART# |
|---|---|---|---|---|---|---|
| 1 | 114616 | 1 EA | PCB C&T VXI LOGIC BD | A1 | | |
| 2 | 590000 | 51 EA | CAP DIP .01UF 50V | C1-51 | | |
| 3 | 701021 | 3 EA | LED SUBMIN GREEN RT-A | D1;D2;D5 | | |
| 4 | 701020 | 2 EA | LED SUBMIN RED RT-ANGLE | D3;D4 | | |
| 5 | 902026 | 1 EA | CONN DIN 96 PIN DIP M R/A | P1 | | |
| 6 | 902264 | 2 EA | CONN 50 PIN FEM R/A DD-50S | J1;J2 | | |
| 7 | 410207 | 6 EA | SCREW PH PAN 2-56 X 7/16 | (J1;P1;2) | | |
| 8 | 431200 | 6 EA | L/W STD SPLT 2-56 | (J1;P1;2) | | |
| 9 | 420200 | 6 EA | NUT STD HEX 2-56 | (J1;P1;2) | | |
| 10 | 902075 | 1 EA | POST 3 PIN SIP ASSY | J3 | | |
| 11 | 902087 | 1 EA | CONN 2 PIN SHORTING | (J3) | | |
| 12 | - | - EA | DASH | | | |
| 13 | 602102 | 1 EA | RES 1/4W 1K 5% | R1 | | |
| 14 | 602332 | 1 EA | RES 1/4W 3.3K 5% | R2 | | |
| 15 | 696043 | 1 EA | SIP 680 OHM 5 COMM RES | RP1 | | |
| 16 | 696045 | 4 EA | SIP 33 OHM 4 ISOLATED RES | RP2;3;8;9 | | |
| 17 | 696042 | 4 EA | SIP 33 OHM 6 ISOLATED RES | RP4;5;10;11 | | |
| 18 | 696034 | 2 EA | SIP 3.3K 9 COMM RES | RP6;7 | | |
| 19 | 96004 | 1 EA | SIP 10K 9 COMM RES | RP12 | | |
| 20 | 907042 | 1 EA | SWITCH MIN PUSHBUTTON | S1 | | |
| 21 | 735175 | 3 EA | IC QUAD DIFF RCVR | U1;2;9 | | |
| 22 | 735176 | 6 EA | IC DIFF BUS XCVR RS-485 | U3-6;11;12 | | |
| 23 | 735174 | 5 EA | IC QUAD DIFF XMTR | U7;8;10;13;14 | | |
| 24 | 797200 | 4 EA | IC FIFO 256X9 300 MIL | U15;16;27;28 | | |
| 25 | 749015 | 1 EA | IC PA7024 PEEL ARRAY | U17 | | |
| 26 | 748574 | 8 EA | IC OCTAL TRIG 3-STATE F/FS | U18;21;24;25;—<br>U34;41;42;51 | | |
| 27 | 748126 | 1 EA | IC QUAD 3-STATE BUF HI ENA | U19 | | |
| 28 | 748245 | 4 EA | IC OCTAL BUS XCVR | U20;33;47;48 | | |
| 29 | 747595 | 4 EA | IC 8-BIT SIPO SR LTCHD 3-ST | U22;29;35;38 | | |
| 30 | 748688 | 4 EA | IC 8-BIT COMPARATOR | U23;31;36;37 | | |
| 31 | 748125 | 2 EA | IC QUAD 3-STATE BUF LO ENA | U26;39 | | |
| 32 | 748165 | 2 EA | IC PARALLEL 8 BIT SHIFT REG | U30;40 | | |
| 33 | 748074 | 2 EA | IC DUAL F/F | U32;43 | | |
| 34 | 798254 | 3 EA | IC PROGRAMMABLE TIMER | U44-46 | | |
| 35 | 772210 | 2 EA | IC PAL 22V10 LOW PWR 35NS | U49;52 | | |
| 36 | 748138 | 1 EA | IC 3-TO-8 DECODER/DEMUX | U50 | | |
| 37 | - | - EA | DASH | | | |
| 38 | 903009 | 3 EA | SOCKET 24 PIN SKINNY DIP | (U17;49;52) | | |
| 39 | 112936 | 3 EA | LABEL PAL COPYRIGHT NOTE | (U17;49;52) | | |
| 40 | 706100 | 1 EA | OSCILLATOR 10.000MHZ | Y1 | | |

## TABLE 6-6 VXI-5539A-14 PCB ASSEMBLY PARTS LIST (115034-01)

| LN# | PART# | QTY | DESCRIPTION | REF-DES | MFGR | MFGR PART # |
|---|---|---|---|---|---|---|
| 1 | 115036 | 1 EA | PC BOARD | A1 | | |
| 2 | 522476 | 1 EA | CAP TANT 47MF 16V | C1 | | |
| 3 | 548510 | 1 EA | CAP CER .01UF 50V | C2 | | |
| 4 | 555103 | 50 EA | CAP CER .01UF 50V | C3-53 | | |
| 5 | 524105 | 3 EA | CAP TANT 1MF 35V | C57;58;63 | | |
| 6 | 715248 | 2 EA | DIODE ZENER 18V | CR1;2 | | |
| 7 | 902212 | 1 EA | HEADER 3X3 R/A | JP1 | | |
| 8 | 902206 | 1 EA | CONN DIN 96 PIN DIP MALE | P1 | | |
| 9 | 410207 | 2 EA | SCREW PH PAN 2-56 X 7/16 | (P1) | | |
| 10 | 431200 | 2 EA | L/W STD SPLT 2-56 | (P1) | | |
| 11 | 420200 | 2 EA | NUT STD HEX 2-56 | (P1) | | |
| 12 | 672103 | 10 EA | RES 1/4W 10K 5% | R1;6'12;13;15;27 30;33;34;46 | | |
| 13 | 672472 | 4 EA | RES 1/4W 4.7K 5% | R3;9;44;45 | | |
| 14 | 672222 | 8 EA | RES 1/4W 2.2K 5% | RP4;4;10;11 | | |
| 15 | 672151 | 4 EA | RES 1/4W 150K 5% | R5;7;26;28 | | |
| 16 | 696017 | 2 EA | SIP 33 OHM 5 COMM RES | RP1,3 | | |
| 17 | 907042 | 1 EA | SWITCH MINI PUSHBUTTON | S1;2 | | |
| 18 | 769024 | 4 EA | IC PA7024 PEEL ARRAY | U7;23 | | |
| 19 | 767200 | 1 EA | IC FIFO | U2 | | |
| 20 | 735176 | 17 EA | IC DIFF RCVR | U3;70 | | |
| 21 | 758074 | 1 EA | IC DUAL F/F | U4 | | |
| 22 | 769254 | 1 EA | IC PROG TIMER | U5 | | |
| 23 | 763403 | 4 EA | IC DUAL DPST SW | U6;34;51;67 | | |
| 24 | 763185 | 4 EA | IC 232 DVR/RCVR | U7;25;52;68 | | |
| 25 | 768273 | 4 EA | IC OCTAL F/F | U10;19;55;63 | | |
| 26 | 769645 | 4 EA | IC 16-BIT BUS XCVR | U11;20;56;64 | | |
| 27 | 769215 | 4 EA | IC FIFO 512 WORDS X 18 | U12;21;57;65 | | |
| 28 | 769630 | 2 EA | IC UNIV CTLR | U13;58 | | |
| 29 | 769010-410 | 2 EA | ICPAL 22V10 | U23;39 | | |
| 30 | 760126 | 1 EA | IC QUAD 3 STATE BUF | U32 | | |
| 31 | 768138 | 1 EA | IC 3-TO-8 DECODER/DEMUX | U50 | | |
| 32 | 903037 | 6 EA | SOCKET 24 PIN PLCC | (U1;14;22;23 ;39;66 ) | | |
| 33 | 706737 | 1 EA | OSCILLATOR 7.3728 MHZ | Y1 | | |

# ADDENDUM

# VXIbus FDC ADDENDUM

## A.0 INTRODUCTION

This addendum provides background information about the VXI-10 Fast Data Channel, features of the FDC that are specific to ICS's VXI modules that use the Fast Data Channel transfer protocol and ICS's Expanded FDC Command Set. A sample initialization program is also included. While this addendum provides recommended channel initialization and data passing procedures, it is not a replacement for the FDC Specification. Users of the Fast Data Channel transfer protocol are urged to read the VXI-10 Fast Data Channel Specification. A copy of the VXI-10 Specification may be downloaded from ICS's Web Site at www.icselect.com.

## A.1 DESCRIPTION

Depending upon the module type and usage, commands and/or data may be transferred between the Slot 0 Controller and the module via Fast Data Channel buffers in the module's A32 memory space. In most of ICS VXIbus modules, the FDC channels operate as *A/B buffer pair*s and in *Stream Transfer* mode for enhanced throughput. In this mode, two FDC channels (Buffers A and B) are used for a data transfer direction and four channels for a bidirectional data port. i.e. a serial port or a GPIB Interface. The number of FDC channels in a module is determined by multiplying the number of interface channels or ports in the module by four. Other uses of the FDC protocol, such as passing static data, may require different numbers of FDC channels or different modes so it is best to check the exact FDC channel count in the specifications section of your module.

Currently, ICS VXIbus modules support Fast Data Channel (FDC) data transfer per VXIbus Specification VXI-10, Rev. 2.10 using the Standard FDC Word Serial Commands and ICS's expanded command set that handles up to 32 FDC channels. Refer to the VXIbus Specification VXI-10, Rev. 2.10 for detailed information on the standard commands and other Fast Data Channel configurations.

## A.2 BACKGROUND INFORMATION

### A.2.1 Fast Data Channel Advantages

VXIbus message based modules have slow data transfer rates because of the complex protocol used to transfer word serial messages. Data transfer with word serial messages is restricted to one byte of information for every 2 or 4 four bus transactions. The larger amount of data needed in newer modules has pushed need for an easier, faster protocol to transfer data.

VXI Shared Memory concept was the VXIbus Consortium's first attempt to provide high speed data transfer. The Shared Memory protocols were too complex and it was not a practical solution. In 1995, the VXIbus Consortium defined the new VXI Fast Data Channel specification. The Fast Data Channel concept has the speed advantages of shared memory without the software overhead.

Data transfer with the Fast Data Channel approaches the maximum transfer rate for the VXIbus. With Word Serial messages, reading a data character requires the controller to make a minimum of four VXIbus accesses - read the response register, send byte request, read the response register again and then read the data byte. To read a register, the Controller must make a read to see that the data is present and then read the data. With the Fast Data Channel, the Controller reads the word count and then makes one read per word. Each word can contain four data bytes which again quadruples the data transfer rate.

VXI Fast Data Channels can be used to transfer data and/or commands between the Commander and the Servant module. The number, size and organization of the channels and their use are up to the module designer. Channels (or buffers) can be used singly or in pairs. Control bits in the FDC channel header (see Tables A1 and A2 define who 'owns' the buffer and if the buffer is full or empty. The control bits prevent data contention since only the channel owner can read or write data in the buffer.

### A.2.2 Fast Data Channel Usage In ICS's VXI Modules

In ICS VXI modules, the VXI-10 Fast Data Channel protocol is used as a way to improve the data transfer rate across the VXIbus. Because most of ICS's modules have data ports, the FDC channels are organized to maintain a continuous flow of high-speed data. A pair of FDC channels is used for each data transfer direction. The channel pairs usually operate as A/B type buffer pairs in the FDC Stream Transfer mode to shuttle data between the ICS module and the Slot 0 Controller. One device fills the first buffer while the other module is emptying the second buffer. The buffers are exchanged and the process repeats until all of the data has been transferred.

Because two buffers are required for a data transfer direction, it takes four FDC channels for a bidirectional data port such as a serial port or a GPIB bus interface. Multiport modules require four FDC channels per port times the number of ports in the module. In multiport modules, the number of FDC channels quickly exceeds the eight channels provided for in the VXI-10 standard command set. Fortunately, the VXI-1 Specification allows designers to create user defined commands sets to meet the needs of more complex modules. ICS's Expanded FDC Command Set handles up to 32 FDC channels which is adequate for modules with up to 8 bidirectional data ports. ICS's Expanded FDC Command Set is described in Section A.6. These commands mirror the standard FDC command set.

The number and usage of the FDC channels in a module is established when the module is designed. The number of channels in a module can be determined by checking the module's specifications or by querying the module with the *FDC Supported* command.

It is important to note that FDC channel ownership is indicated by the *RDY* and *WDY* bits in the FDC header. If both bits are cleared, the channel belongs to the servant and should not be written to or read by the commander. Reading the buffer will not cause an error, but the data in the buffer may not be valid. If the *RDY* bit is set, the commander owns the buffer and it contains data to be read. If the *WDY* bit is set the commander also owns the buffer and may write data into it.

FDC channels are passed to the commander in two ways. When a buffer is ready to be passed, the *RDY* or *WDY* bit is set in the header. If FDC Events are enabled for the channel, an interrupt is generated to the commander containing the FDC channel number being passed. If the FDC Events are not enabled, the commander must poll the FDC header and check the bits to detect the passed buffer.

If the *Channel Initialize* command response indicates that the *Pass Command* is supported, the *Enable Passed Buffer* command must be sent to the servant and FDC channels are passed to the servant with the *Passed Buffer* command. If the *Pass Command* is not supported, the channels are passed by clearing the *WRY* or *WDY* bit and the servant must poll the bits to detect the channel pass. Most ICS modules require the *Pass Buffer* command to minimize the internal processor overhead.

The FDC channels must be initialized before they are used to establish data direction for the Slot 0 Controller. The recommended initialization process is shown in Figure A1. Figure A1 resets the channels at the beginning of the initialization in case they had been previously used.

FDC channels are initialized by sending the *Channel Initialize* command, and either the *Transfer to Commander*, or *Transfer to Servan*t commands. The Transfer commands set the data transfer direction and pair and stream modes for the channel and must be set to the modes supported by the channel. Receive channels should be set with the *Transfer to Commander* command and transmit channels with the *Transfer to Servant* command. These commands should only be sent to the even numbered channel of a channel pair. This also applies to the *Go To Idle* and *Channel Close* commands.

### A.2.3 Transmit Channels

Transmit channels are initialized with the FDC *Transfer to Servant* command to establish the data direction from commander to servant for the channel or channel pair. The Pair and Stream mode bits should be set in this command if these modes are required by the module. Once a transmit channel is initialized with the *Channel Initialize* and *Transfer to Servant* commands the buffer will be owned by the commander. The commander should check the FDC header for ownership before writing to the buffer. Then the first data message may be written to the buffer and passed to the servant with the *Passed Buffer* command. When the buffer is received by the servant, it will transmit the data.

When writing data into a FDC buffer, the number of bytes in the data buffer area must be written into the Data Size word in the header area. This count includes all message bytes plus those used in any sub headers.

Transmit channels (buffers) are passed to the Slot 0 Controller when they are empty. The Controller passes them back to the VXI module when they are full or have data in them. The amount of data in the buffer is up to the user and is specified in the FDC channel header. The only restrictions are that the amount of data cannot exceed the maximum buffer size and that the Controller must fill alternate buffers. The Slot 0 Controller must keep the module supplied with a new buffer before the old one is completely transmitted to maintain continuous data flow.

### A.2.4 Receive Channels

Receive channels are initialized with the FDC *Transfer to Commander* command to establish the data direction from servant to commander for the channel or channel pair. The Pair and Stream mode bits should be set in this command if these modes are required by the module. Once a receive channel is initialized with the *Channel Initialize* and *Transfer to Commander* commands, the buffer will be owned by the servant. At this point the receiver will start receiving data. When the number of bytes in the buffer reaches the switch point or if the *Switch Buffer* command is received, the buffer will be passed back to the commander and the receiver will switch to the other channel if it is available.

When a FDC buffer is passed, the number of bytes in the data buffer section is specified by the Data Size word in the header area. This includes all message bytes and any module sub header information.

The number of bytes for the switch point is set by the user with the SCPI *SYST:RECEIVE:BUF:SIZE* command. The switch point byte count includes the eight header area bytes plus all of those to be used in the data section when the buffer is 'full'. This command must be given before the *Transfer to Commander* command to be effective with the first buffer. If the other buffer is not ready then any new incoming data will be lost. A VXI event interrupt will occur when the buffers are switched, if interrupts were enabled. Use the *Go to Idle* command to terminate data transfer and to get last buffer with any remaining received data.

In addition to using FDC channels as A/B buffer pairs, individual FDC channels may be assigned to other uses such as fill buffers, or alternate program buffers. Consult your module's manual for specific information on its FDC channel usage.

## A.3 FAST DATA CHANNEL (FDC) MEMORY MAP

### A.3.1 Channel Memory Maps

A memory map for a typical FDC channel is shown in Tables A1 and A2. Both tables contain the same information but show it by VXIbus word width. Table A1 is organized as 16-bit words and Table A2 is organized as 32-bit words. The first eight bytes contain the FDC Channel Header information. The header contains the control bits for determining buffer ownership and the Data Size word that defines the space used in the data buffer. The FDC Channel data buffer starts with byte 8. Modules may use the data buffer space in the way that best fits their application. Byte numbers are in Motorola order for correlation with the VXI-10 Specification.

**TABLE A1 16-BIT WIDE FDC MEMORY MAP**

| Word Count | Byte 0 (MSB) | Byte 1 (LSB) |
|---|---|---|
| 0 | Rsvd Rsvd Rsvd Rsvd Rsvd Rsvd Rsvd TRIG | Rsvd Rsvd Rsvd Rsvd ABT RDY WDY END |
| 1 | Minor Revision Major Revision | Rsvd Rsvd Rsvd Rsvd Rsvd Rsvd Rsvd Rsvd |
| 2 | Data Size Bits 15-8 | Data Size Bits 7-0 |
| 3 | Data Size Bits 31-24 | Data Size Bits 23-16 |
| 4 | Buffer Byte 2 | Buffer Byte 3 |
| 5 | Buffer Byte 0 | Buffer Byte 1 |
| 6 | Buffer Byte 6 | Buffer Byte 7 |
| 7 | Buffer Byte 4 | Buffer Byte 5 |
| . | . . | . . |
| n | Buffer Byte i-1 | Buffer Byte i |

**TABLE A2 32-BIT WIDE FDC MEMORY MAP**

| Word Count | Byte 0 (MSB) | Byte 1 | Byte 2 | Byte 3 (LSB) |
|---|---|---|---|---|
| 0 | Revision | Rserved Bits | Reserved TRIG | Rsvd ABT RDY WDY END |
| 1 | Data Size Bits 31-24 | Data Size Bits 23-16 | Data Size Bits 15-8 | Data Size Bits 7-0 |
| 2 | Buffer Byte 0 | Buffer Byte 1 | Buffer Byte 2 | Buffer Byte 3 |
| 3 | Buffer Byte 4 | Buffer Byte 5 | Buffer Byte 6 | Buffer Byte 7 |
| 4 | Buffer Byte 8 | Buffer Byte 9 | Buffer Byte 10 | Buffer Byte 11 |
| . | . . | . . | . . | . . |
| n | Buffer Byte i-3 | Buffer Byte i-2 | Buffer Byte i-1 | Buffer Byte i |

Note: Bit definitions are listed in paragraph A3.2
Byte numbers are in Motorola byte order for correlation with the VXI-10 Specification.

.

### A.3.2 Fast Data Channel Buffer Definitions

The following definitions include definitions from the VXI-10 Fast Data Channel Specification.

**HEADER AREA :** First eight bytes of the FDC channel buffer. Contains the channel Revision and Data Size words.

**RSVD :** These bits are reserved and should be set to 0.

**MAJOR REVISION:** (Byte 0, bits 2-0). Must be a 2 (010)

**MINOR REVISION:** (Byte 0, bits 4-3). Must be 1 (01)

**TRIG:** (Byte 2, bit 0) The TRIG bit is utilized only within Message Transfer Protocol (MTP) to send the MTP Trigger command. It has no meaning outside of MTP and should be set to 0.

**END:** (Byte 3, bit 0) The END bit indicates whether this buffer of data is the last buffer of data in a data block. If the END bit is set to 1, this is the last buffer of data. If the END bit is set to 0 this is not the last buffer of data.

**WDY:** (Byte 3, bit 1) The WDY flag is utilized when data is transferred from the Commander to the Servant. If the WDY bit is set to 1, the Commander owns the FDC area. It can place a buffer of data into the FDC area and then set WDY to 0 to pass the buffer of data to the Servant. If the WDY bit is set to 0, the VXI Servant owns the FDC area. It may read the buffer of data and then set WDY high to pass the FDC area back to the Commander. When the channel is in the idle state, WDY is 0.

**RDY:** (Byte 3, bit 2) The RDY flag is utilized when data is transferred from the Servant to the Commander. If the RDY bit is set to 0, the VXI Servant owns the FDC area. It can place a buffer of data into the FDC area and set the RDY bit to 1 to pass the buffer of data to the Commander. If the RDY bit is set to 1, the Commander owns the FDC area. The Commander can read the buffer of data and then set the RDY bit to 0 to pass the FDC area back to the Servant. When the channel is in the idle state, RDY is 0.

**ABT:** (Byte 3, bit 3) The ABT bit indicates that an abort transfer is being requested for this block of data.

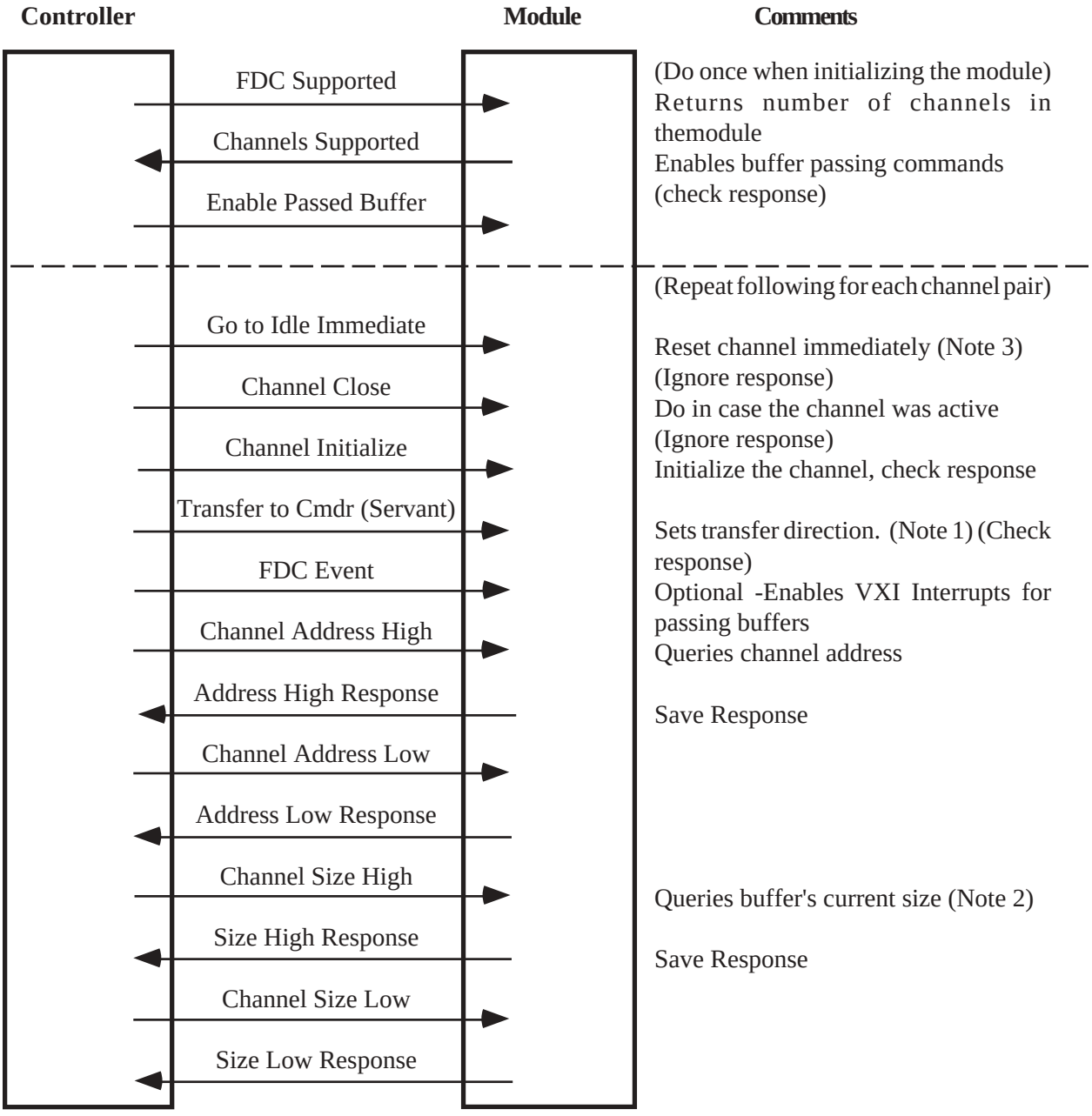
**DATA SIZE:** (Bytes 4-7) The DATA SIZE contains the number of bytes (i) contained in the FDC Data Buffer.

**FDC DATA BUFFER:** Memory area for the data in the FDC Channel Buffer. The Data Buffer starts with Byte 8 (Word 4 for 16-bit wide buffers or Word 2 for 32-bit wide buffers) and ends in Word n. The organization of the data buffer is module dependent.

Note: Refer to the Fast Data Channel Specification VXI-10 for additional information on the usage of the bits in the Channel Header.

## A.4 FAST DATA CHANNEL INITIALIZATION SEQUENCE

Figure A1 shows the recommended initialization sequence for a VXI module with a streaming channel pair. When initializing channel pairs, the commands are only sent to the lower channel of each pair. It is strongly recommended that the user include the Go-to-Idle-Immediate and Channel-Close commands in the initialization sequence so the sequence can initialize a previously opened channel.




Notes: 1. For Streaming mode, set the Buffer size with the SCPI *SYST:SIZE* command before isuing the *Transfer-to-Commander* command.
2. *Channel Size High* and *Channel Size Low* return the current buffer size (default setting or the SCPI *SYST:SIZE* command setting)
3. *Go-to-Idle Immediate* also resets channel specific hardware in most ICS modules.

**Figure A1 Recommended FDC Initialization Sequence**

## A.5 FAST DATA CHANNEL PASSING SEQUENCE

### A.5.1 Transfer to Servant

Figure A2 shows a sequence for swapping a streaming channel pair. In normal operation, buffers of data are passed from the data source to the data destination, alternating between the even and odd FDC channel. The END flag is used to indicate the end of a block of data, not the termination of the data transfer. The sequence in Figure A2 is for the Transfer to Servant direction. The steps below the dotted line are repeated until terminated by the Go-to-Idle command.

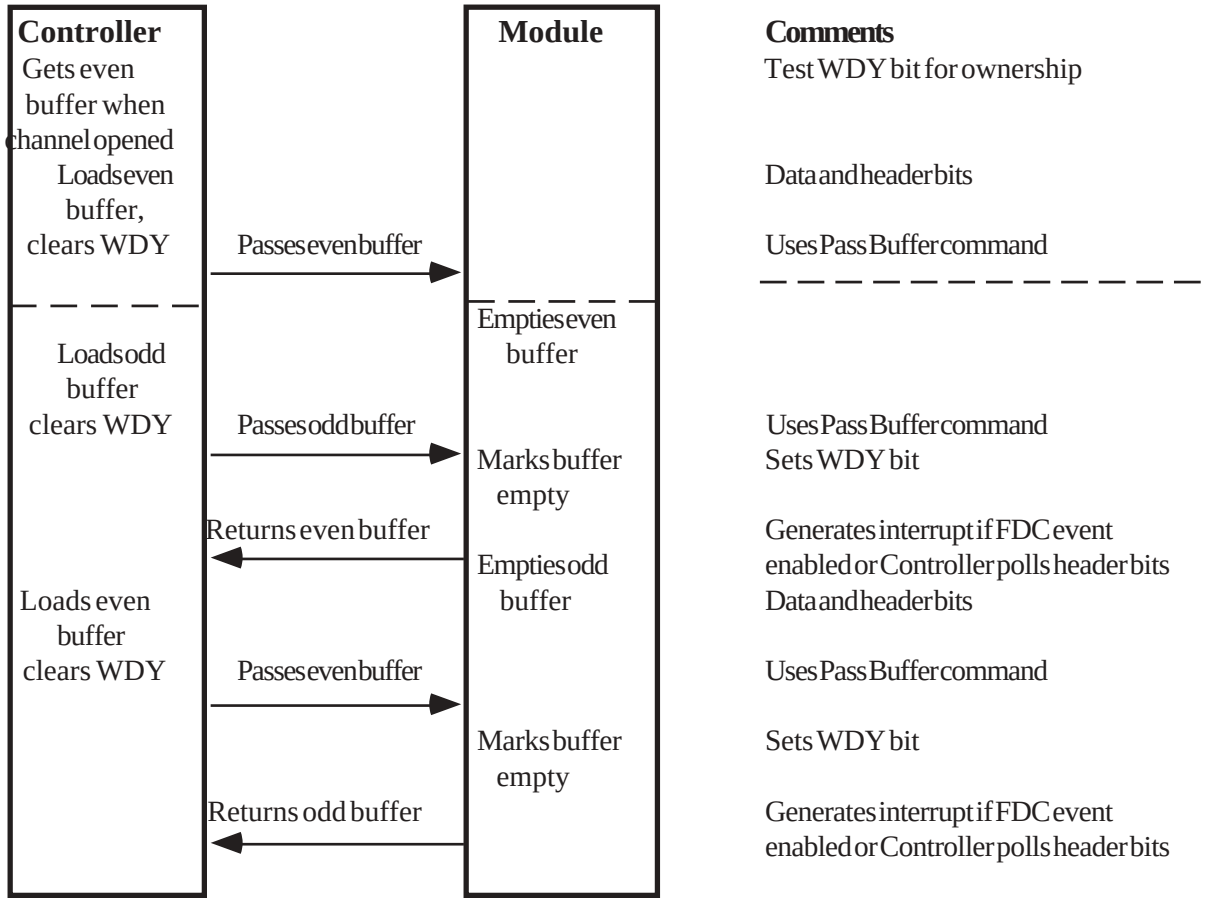



**Figure A2 Transfer to Servant Buffer Passing Sequence**

### A.5.2 Transfer to Commander

Figure A3 shows a sequence for swapping a streaming channel pair when the data direction is to the Commander. In normal operation, buffers of data are passed from the data source to the data destination, alternating between the even and odd FDC channel. Transfer takes place when the amount of data in the buffer reaches the switch point set by the *SYST:RECEIVE:BUF:SIZE* command. The steps below the dotted line are repeated until terminated by the Go-to-Idle command.

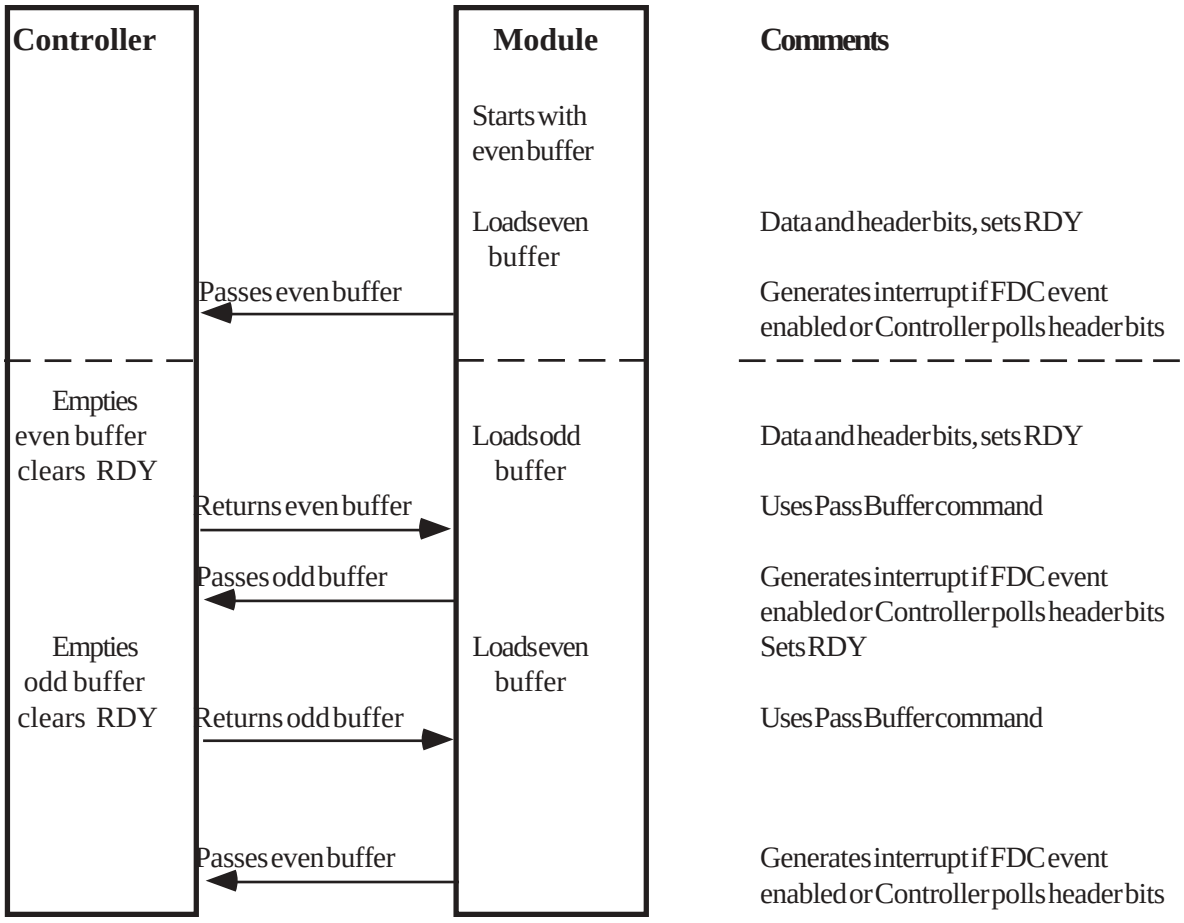



**Figure A3 Transfer to Commander Buffer Passing Sequence**

## A.6 FDC COMMAND REFERENCE

### A.6.1 Fast Data Channel (FDC) Command Summary

ICS's VXIbus modules with the Fast Data Channel option respond to the following FDC Standard commands and ICS expanded commands. The expanded commands are the same as the standard VXI-10 commands, except for the command codes and the channel number field which is expanded to 5 bits to allow for up to 32 channels. Both command sets are supported for software compatibility. To avoid conflicts, only one set of commands should be used in an application program. Refer to the Fast Data Channel Specification VXI-10, Rev. 2.10 for detailed information on the usage of these commands and explanation of their parameters. Table A1 lists the standard and ICS expanded FDC commands

**TABLE A3 FAST DATA CHANNEL STANDARD AND ICS EXPANDED COMMANDS**

| Command | Std. Code | Standard Binary Code | Exp. Code | Exp. Binary Code |
|---|---|---|---|---|
| Channel Address High (Q) | (0x9F80) | 1001 1111 1000 0ccc | (0x7E00) | 0111 1110 000c cccc |
| Channel Address Low (Q) | (0x9F00) | 1001 1111 0000 0ccc | (0x7C00) | 0111 1100 000c cccc |
| Channel Close (Q) | (0x9F98) | 1001 1111 1001 1ccc | (0x7E60) | 0111 1110 011c cccc |
| Channel Initialize (Q) | (0x9F90) | 1001 1111 1001 0ccc | (0x7E40) | 0111 1110 010c cccc |
| Channel Size High (Q) | (0x9F88) | 1001 1111 1000 1ccc | (0x7E20) | 0111 1110 001c cccc |
| Channel Size Low (Q) | (0x9F08) | 1001 1111 0000 1ccc | (0x7C20) | 0111 1100 001c cccc |
| Enable Passed Buffer (Q) | (0x9F18) | 1001 1111 0001 100e | (0x7C60) | 0111 1100 0110000e |
| FDC Event (Q) | (0x9FA0) | 1001 1111 1010 eccc | (0x7E80) | 0111 1110 10ec cccc |
| FDC Supported (Q) | (0x9F1F) | 1001 1111 0001 1111 | (0x7C7F) | 0111 1100 0111 1111 |
| Go to Idle (Q) | (0x9FB0) | 1001 1111 1011 iccc | (0x7EC0) | 0111 1110 11ic cccc |
| Passed Buffer | (0x9F10) | 1001 1111 0001 0ccc | (0x7C40) | 0111 1100 010c cccc |
| Switch Buffers (Pair) | N.S. | N.S. | (0x7C80) | 0111 1100 100c cccc |
| Transfer to Commander (Q) | (0x9FE0) | 1001 1111 111p sccc | (0x7F80) | 0111 1111 1psc cccc |
| Transfer to Servant (Q) | (0x9FC0) | 1001 1111 110p sccc | (0x7F00) | 0111 1111 0psc cccc |

(Q) = Query (command has a response)
N.S. = Not Supported
c = channel code
e = enable
i = immediate change
p = pair flag
s = stream flag

### A.6.2 Fast Data Channel (FDC) Command Descriptions

The following section provides a detailed description of each Fast Data Channel Command.

**Channel Initialize (0x9F90)**

This command is used to validate and initialize the FDC area.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | Channel # | | |

This response for this command is the same as for the following command.

**Expanded Channel Initialize (0x7E40)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | Channel # | | | | |

A single response word is placed in the Data Low register in the following format:

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Status | | | | 1 | ADDR | | | DATA | | | | PC | MODE | | |

Status:
- F - No Errors
- 7 - Channel already open
- 6 - No valid FDC area can be opened
- 5 - FDC Channel number not supported

**Channel Address Commands:**

These commands are used to retrieve the FDC area base address from the servant. The FDC address and size defines a memory area within the address space returned by the Channel Initialize command.

**Channel Address Low (0x9F00)**

The syntax of the Channel Address Low command is defined in the following table.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | Channel # | | |

This response for this command is the same as for the following command.

**Expanded Channel Address Low (0x7C00)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | Channel # | | | | |

This response for this command is the same as for the following command.

**Channel Address High (0x9F80)**

The syntax of the Channel Address High command is defined in the following table.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | Channel # | | |

This response for this command is the same as for the following command.

**Expanded Channel Address High (0x7E00)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | Channel # | | | | |

A single response data word is placed in the Data Low register for each command in the following format:

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FDC Area Address Low or High Word. | | | | | | | | | | | | | | | |

If no valid FDC area allotted, the returned value in the high and low response words is $HFFFF.

**Channel Size Commands:**

These commands are used to retrieve the FDC area size. The FDC size identifies the memory area allocated to this FDC channel starting at the Address returned by the Channel Address Low and Hi commands.

**Channel Size Low (0x9F08)**

The syntax of Channel Size Low command is defined in the following table.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | Channel # | | |

This response for this command is the same as for the following command.

**Expanded Channel Size Low (0x7C20)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | Channel # | | | | |

This response for this command is the same as for the following command.

**Channel Size High (0x9F88)**

The syntax of Channel Size High command is defined in the following table.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | Channel # | | |

This response for this command is the same as for the following command.

**Expanded Channel Size High (0x7E20)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | Channel # | | | | |

A single response word is placed in the Data Low register for each command in the following format:

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FDC Area Size Low or High Word. | | | | | | | | | | | | | | | |

If no valid FDC area can be allotted, the response in the high and low response words is $H0000.

**Go to Idle** **(0x9FB0)**

This command is used to terminate a Stream transfer. It may also be used to force termination of a Normal transfer.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | IM | Channel # | | |

This response for this command is the same as for the following command.

**Expanded Go to Idle** **(0x7EC0)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | IM | Channel # | | | | |

A single response word is placed in the Data Low register in the following format:

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Status | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Channel Close** **(0x9F98)**

This command is used to close the FDC channel.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | Channel # | | |

This response for this command is the same as for the following command.

**Expanded Channel Close** **(0x7E60)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | Channel # | | | | |

A single response word is placed in the Data Low register in the following format:

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Status | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Status: F - No Errors

7 - Channel is still active and will be returned to idle at the close of the current block of data or attempt to close a channel that is not open.

6 - Cannot idle a closed channel or close a channel that is not open.

5 - FDC Channel number not supported.

4 - Command not allowed.

**Transfer to Servant (0x9FC0)**

This command is used to initiate a data block transfer from the commander to the servant.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | PR | ST | Channel # | | |

This response for this command is the same as for the following command.

**Expanded Transfer to Servant (0x7F00)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | PR | ST | Channel # | | | | |

A single response word is placed in the Data Low register in the following format:

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Status | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Transfer to Commander (0x9FE0)**

This command is used to initiate a data block transfer from the servant to the commander.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | PR | ST | Channel # | | |

This response for this command is the same as for the following command.

**Expanded Transfer to Commander (0x7F80)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | PR | ST | Channel # | | | | |

A single response word is placed in the Data Low register in the following format:

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Status | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Status: F - No Errors, data transfer will commence
7 - Request with no valid FDC channel
6 - Request to send data when FDC channel is active
5 - PR bit not legal for this command
4 - Unable to utilize channel pair
3 - Unable to send/receive data for instrument specific reason(s)
2 - Unsupported mode( stream, normal or direction)

**FDC Event** **(0x9FA0)**

This command is used to control Standard FDC event generation. Use of this command will disable the Expanded FDC Events (if enabled).

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | EV | Channel # | | |

This response for this command is the same as for the following command.

**Expanded FDC Event** **(0x7E80)**

This command is used to control Expanded FDC event generation. Use of this command will disable the Standard FDC Events (if enabled).

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | EV | Channel # | | | | |

A single response word is placed in the Data Low register in the following format:

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Status | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Status: F - No Errors
7-Passed Buffer command not utilized.

**FDC Supported** **(0x9F1F)**

This command is used to determine FDC support.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

A single response word is placed in the Data Low register in the following format:

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C7 | C6 | C5 | C4 | C3 | C2 | C1 | C0 | MP | EX | RM | Min. Rev | | Maj. Rev | | |

**Expanded FDC Supported (0x7C7F)**

This command is used to determine Expanded FDC support.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

A single response word is placed in the Data Low register in the following format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | Max. Channel Number | | | | | MP | EX | RM | Min. Rev | | Maj. Rev | | |

**Enable Passed Buffer (0x9F18)**

This command requests that the passed buffer command be utilized by the servant.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | E |

This response for this command is the same as for the following command.

**Expanded Enable Passed Buffer (0x7C60)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | E |

A single response word is placed in the Data Low register in the following format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Status | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Status: F - No Errors
7 - Passed Buffer command not utilized.

**Passed Buffer (0x9F10)**

This command informs the servant that the commander has passed the buffer to the servant.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | Channel # | | |

This command does not have a response.

**Expanded Passed Buffer (0x7C40)**

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | Channel # | | | | |

This command does not have a response.

**Expanded Switch Buffers (0x7C80)**

This command is used to command the servant to switch input buffers and pass the current buffer to the commander. This command is not supported by the Standard FDC command set, but may be used with the Standard commands.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | Channel # | | | | |

This command does not have a response.

**Fast Data Channel Events (Interrupt Response)**

The Crypto generates FDC events when enabled by the Standard FDC Event command. The response word is generated when the channel passes the FDC area to the commander. The format of the return value for FDC events is described below.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | Channel # | | | Servants Logical Address | | | | | | | |

**Expanded Fast Data Channel Events (Interrupt Response)**

The Crypto generates Expanded FDC events (with User Defined protocol event format) when enabled by the Expanded FDC Event command. The response word is generated when the channel passes the FDC area to the commander. The format of the return value for expanded FDC events is described below.

Bit #

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | Channel # | | | | | Servants Logical Address | | | | | | | |

**Note:** Refer to the Fast Data Channel Specification VXIbus-10 for detailed information on the usage of the FDC Event command.

## A.7 FDC DEMO PROGRAM

### A.7.1 Program Description

The FDC Demo Program runs a fixed data pattern between two serial channels in an ICS Model VXI-5536 High Speed Serial Module. The program lets the user specify the transmit and receive channels and the serial parameters. It then transmits in groups of four messages. The message groups are repeated until interrupted by the user.

The purpose of the FDC Demo Program is to demonstrate how to initialize Fast Data Channel buffers and how to pass the buffers from the Controller to the module and back to the Controller. If the user wishes to run this program, he may down load the latest version of this program and its header files from ICS's Web Site at *www.icselect.com.*

The program uses ICS's VXI-554x VXI library. If you are using a different Slot 0 Controller, it will be necessary to change the function calls to those in the library supplied with your controller. This program also uses ICS"s expanded word serial commands for FDC channels. If your module uses 8 or less channels, you may change these commands back to the standard FDC commands in the VXI-10 specification. Although this program is written around ICS's VXI-5536 Quad High Speed Serial module, the general purpose functions contained herein may be used with any module that has FDC channels.

```
/* ********************************************************************** */
/*          VXI-5536 Demo Test                                            */
/*                                                                        */
/*          Filename:   5536demo.c                                        */
/*                                                                        */
/*          Demo test routines                                            */
/*                                                                        */
/*          Created:   Apr. 29, 1997  CB from 5536test program            */
/*          Revised:                                                      */
/* ********************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <io.h>

#include "..\vxi_lib\vxi_drv.h"                 // ICS VXI Library function prototypes

#include "..\vxi_lib\vxicmds.h"                 // VXI Std Word Serial Commands
#include "..\vxi_lib\fdc_cmds.h"                // FDC Word Serial Commands


int ModuleInit(void);
int DataTest(void);
void A32_WriteBufr(int LogicalAddr, int chan, int *buffer);
int setupChan(int *TxChan, int *RxChan, long *TxRate);
void XFDC_OpenWriteBufrPS(int LogicalAddr, int chan);
int XFDC_GetHeader(int LogicalAddr, int chan);
void XFDC_ChannelClose(int LogicalAddr, int chan);
void XFDC_OpenReadBufrPS(int LogicalAddr, int chan);
void XFDC_PassBuffer(int LogicalAddr, int chan);
int XFDC_GetChanAddr(int LogicalAddr, int chan, unsigned long *MemAddr);
```

**Figure A-4 FDC Demo Program Listing**

```
int Tx1Buffer[] = {44, 0, 4, 0, 0, 0,
        0x8005, 0x8003, 0x800b, 0x8009,
        0x8005, 0x8003, 0x800b, 0x8009,
        0x8005, 0x8003, 0x800b, 0x8009,
        0x8005, 0x8003, 0x800b, 0x8009};

int Tx2Buffer[] = {44, 0, 4, 0, 0, 0,
        0x8015, 0x8013, 0x801b, 0x8019,
        0x8015, 0x8013, 0x801b, 0x8019,
        0x8015, 0x8013, 0x801b, 0x8019,
        0x8015, 0x8013, 0x801b, 0x8019};

int LogicalAddr;

//extern long WrBusErr, RdBusErr, VErrCnt;

char title[]="VXI-5536 Demo Test Program, Version 1.0, 04/29/97";
char copywrt[]="Copyright (c) 1997, ICS Electronics ";


int debug = 0;
int run;

/* ****************************** main ********************************** */
/* ************************************************************************ */

main()
{
        puts(title);
        puts(copywrt);
        puts("\n\n");

        vxiOpen();                      // open ICS VXI-5543 VXI library

        if(!ModuleInit())               // initialize 5536 module
                DataTest();             // if OK run data test

        vxiClose();                     // close VXI library

        exit(0);
}

/* ************************** ModuleInit ****************************      */
/*      This function initializes the VXI-5536 module                     */
/* ************************************************************************ */

int ModuleInit(void)
{
        unsigned RtnValue;
        int Err_Code;
        long slot=0;

        puts("\n Searching commander table...");

        LogicalAddr = vxiGetAddr(536, 1);       // find 1st 5536
        if(LogicalAddr != -1)
        {
                Err_Code = vxiGetAttrib(LogicalAddr, VXI_SLOT, &slot);
                if(!Err_Code)
                {
                        printf(" VXI-5536 found in slot %d at Logical Address %d\n", (int)slot, LogicalAddr);
                }
        }
        else
        {
                printf("\n *** VXI-5536 Not Found ***, Run RES_MAN");
```

**Figure A-4 FDC Demo Program Listing Continued**

```
		return 1;
	}

	//		Clear Module

	Err_Code = vxiWScmd(LogicalAddr, Clear);
	if(Err_Code)
	{
		printf("*** ERROR %d sending VXI CLEAR to Logical Address: %d\n", Err_Code, LogicalAddr);
		vxiErrorDisp(Err_Code);
		return 1;
	}

	//		Send Enable Passed Buffer command

	Err_Code = vxiWSqry(LogicalAddr, XFDC_ENBL_PASS_BUFR, &RtnValue);
	if(!Err_Code)
	{
		printf("LA %d, XFDC Enable Passed Buffer response: %04X\n", LogicalAddr, RtnValue);
	}
	else
	{
		printf("*** ERROR %d sending XFDC Enable Passed Buffer to Logical Address: %d\n", Err_Code, LogicalAddr);
		vxiErrorDisp(Err_Code);
	}

	return 0;
}


/* ***************************** DataTest ******************************	*/
*/		VXI-5536 Data Test Function					*/
/* *********************************************************************	*/
int DataTest(void)
{
	unsigned RtnValue;
	int i, chan, intchan, tchan[4];
	long LoopCnt;
	long RxApass=0, RxBpass=0, TxApass=0, TxBpass=0;
	int Err_Code, TxChan, RxChan;
	int Tx1FDCbufr, Tx2FDCbufr, Rx1FDCbufr, Rx2FDCbufr;
	long TxRate;

	//		Setup module parameter per operator inputs

	Err_Code = setupChan(&TxChan, &RxChan, &TxRate);
	if(Err_Code)
		return 1;

	Rx1FDCbufr = (RxChan-1)*4 + 0;
	Rx2FDCbufr = (RxChan-1)*4 + 1;
	Tx1FDCbufr = (TxChan-1)*4 + 2;
	Tx2FDCbufr = (TxChan-1)*4 + 3;
	printf("FDC buffers: Rx A = %d, Rx B = %d, Tx A = %d, Tx B = %d\n", Rx1FDCbufr, Rx2FDCbufr ,Tx1FDCbufr, Tx2FDCbufr);

	tchan[0] = Tx1FDCbufr;
	tchan[1] = Tx2FDCbufr;
	tchan[2] = Rx1FDCbufr;
	tchan[3] = Rx2FDCbufr;

	XFDC_ChannelClose(LogicalAddr, Rx1FDCbufr);	// go to idle & close Rx A
	XFDC_ChannelClose(LogicalAddr, Tx1FDCbufr);	// go to idle & close Tx A

	// Init Output Buffer pair and write data

	XFDC_OpenWriteBufrPS(LogicalAddr, Tx1FDCbufr);
	A32_WriteBufr(LogicalAddr, Tx1FDCbufr, Tx1Buffer);
	A32_WriteBufr(LogicalAddr, Tx2FDCbufr, Tx2Buffer);
```

**Figure A-4 FDC Demo Program Listing Continued**

```
//              Init Input Buffer pair

XFDC_OpenReadBufrPS(LogicalAddr, Rx1FDCbufr);

puts("Transmitting Data...");

//              Check Tx buffer headers and pass 1st data buffers

RtnValue = XFDC_GetHeader(LogicalAddr, Tx1FDCbufr);
printf("XFDC Chan %d Header: %04X\n", Tx1FDCbufr, RtnValue);

if((RtnValue & 0x0002))
{
        XFDC_PassBuffer(LogicalAddr, Tx1FDCbufr);
}

RtnValue = XFDC_GetHeader(LogicalAddr, Tx2FDCbufr);
printf("XFDC Chan %d Header: %04X\n", Tx2FDCbufr, RtnValue);

if((RtnValue & 0x0002))
{
        XFDC_PassBuffer(LogicalAddr, Tx2FDCbufr);
}

do                      // loop until a key is hit
{
        for(i=0; i<4; i++)                      // poll all buffers
        {
                chan = tchan[i];
                RtnValue = XFDC_GetHeader(LogicalAddr, chan);
                if((RtnValue & 0x0002))                 // if Tx buffer passed
                {
                        if(chan == Tx1FDCbufr)
                                TxApass++;
                        else
                                TxBpass++;

                        XFDC_PassBuffer(LogicalAddr, chan);     // pass buffer back
                }
                else if((RtnValue & 0x0004))    // if Rx buffer passed
                {
                        if(chan == Rx1FDCbufr)
                                RxApass++;
                        else
                                RxBpass++;

                        XFDC_PassBuffer(LogicalAddr, chan);     // pass buffer back

                        printf("Passed Buffers: TA %ld, TB %ld, RA %ld, RB %ld\n",
                        TxApass, TxBpass, RxApass, RxBpass);

                        if(kbhit())             // check for key hit
                                break;
                }
        }
} while(!kbhit());
getch();                // clear key hit

XFDC_ChannelClose(LogicalAddr, Rx1FDCbufr);     // go to idle & close Rx A
XFDC_ChannelClose(LogicalAddr, Tx1FDCbufr);     // go to idle & close Tx A

return Err_Code;
}
```

**Figure A-4 FDC Demo Program Listing Continued**

```
/* ***************************** setupChan ******************************      */
/*      This function sets up the Rx & Tx channels per operator inputs                */
/* ************************************************************************   */

int setupChan(int *TxChan, int *RxChan, long *TxRate)
{
        unsigned RtnValue;
        int Err_Code = 0, Bits, TimOut, NumbWords;
        char command[20], DataSize[20], ExtClk[20], InvClk[20], GatedClk[20];
        char RxStart[20], MSBsel[20], OutStr[128];
        char InitStr[] = {"*cls; *ese 48"};

        Err_Code = vxiOutput(LogicalAddr, InitStr);
        if(Err_Code)
        {
                printf("\t*** ERROR, vxiOutput - %s, Err_Code = %d\n",

        InitStr, Err_Code);
                return Err_Code;
        }
        else
                printf("\nSetup commmands sent: %s\n", InitStr);

        do
        {
                puts("\nEnter parameters, [ENTER] for defaults, EXIT to quit");
                printf("\nEnter Transmitter channel number ([1]-4) > ");
                gets(command);                                  // get command from kybd
                if(!strcmpi(command,"EXIT"))
                {
                        Err_Code = 1;
                        break;
                }
                *TxChan = atoi(command);
                if(*TxChan == 0)
                {
                        *TxChan = 1;
                        printf("\tChannel set to default [%d]\n", *TxChan);
                }

                printf("\nEnter Receiver channel number (1-4 [%d]) > ", *TxChan);
                gets(command);                                  // get command from kybd
                if(!strcmpi(command,"EXIT"))
                {
                        Err_Code = 1;
                        break;
                }
                *RxChan = atoi(command);
                if(*RxChan == 0)
                {
                        *RxChan = *TxChan;
                        printf("\tRx Channel set to default [%d]\n", *RxChan);
                }

                printf("Select Data Bus Size (D16 | [D32]) > ");
                gets(DataSize);                                 // get command from kybd
                if(strlen(DataSize) == 0)
                {
                        strcpy(DataSize, "D32");
                        printf("\tData Bus Size set to default [%s]\n", DataSize);
                }

                printf("Select Words Per Message (1, [2], 4 > ");
                gets(command);                                  // get command from kybd
                NumbWords = atoi(command);
                if(NumbWords == 0)
```

**Figure A-4 FDC Demo Program Listing Continued**

```
{
        NumbWords = 2;
        printf("\tWords Per Message set to default [%d]\n", NumbWords);
}

printf("Select External Clock (ON | [OFF]) > ");
gets(ExtClk);                                   // get command from kybd
if(strlen(ExtClk) == 0)
{
        strcpy(ExtClk, "OFF");
        printf("\tExternal Clock set to default [%s]\n", ExtClk);
}

printf("Select Clock Inversion (ON | [OFF]) > ");
gets(InvClk);                                   // get command from kybd
if(strlen(InvClk) == 0)
{
        strcpy(InvClk, "OFF");
        printf("\tClock Invert set to default [%s]\n", InvClk);
}

printf("Select Tx Gated Clock ([ON] | OFF) > ");
gets(GatedClk);                                 // get command from kybd
if(strlen(GatedClk) == 0)
{
        strcpy(GatedClk, "ON");
        printf("\tTx Gated Clock mode set to default [%s]\n", GatedClk);
}

printf("Select Rx Start Mode ([GATE] | CLOCK) > ");
gets(RxStart);                                  // get command from kybd
if(strlen(RxStart) == 0)
{
        strcpy(RxStart, "GATE");
        printf("\tRx Start mode set to default [%s]\n", RxStart);
}

printf("Select MSB first ([ON] | OFF) > ");
gets(MSBsel);                                   // get command from kybd
if(strlen(MSBsel) == 0)
{
        strcpy(MSBsel, "ON");
        printf("\tMSB first set to default [%s]\n", MSBsel);
}

printf("Enter Number of bits (1 - [64]) > ");
gets(command);                                  // get command from kybd
Bits = atoi(command);
if(Bits == 0)
{
        Bits = 64;
        printf("\tBits set to default [%d]\n", Bits);
}

printf("Enter Time Out (0 - 500) [200] > ");
gets(command);                                  // get command from kybd
TimOut = atoi(command);
if(TimOut == 0)
{
        TimOut = 200;
        printf("\tTime Out set to default [%d]\n", TimOut);
}

printf("Enter Transmitter rate (153 - [1000000]) > ");
gets(command);                                  // get command from kybd
*TxRate = atol(command);
if(*TxRate == 0)
```

**Figure A-4 FDC Demo Program Listing Continued**

```
                {
                        *TxRate = 1000000;
                        printf("\tRate set to default [%ld]\n", *TxRate);
                }

                sprintf(OutStr,"*cls; c %d; bdw %s; bws %d; rck %s; str %s; bits %d; rsz 100; c %d; bits %d; bdw %s; bws %d; trt %ld;
                        ext %s; tgc %s; msb %s; tmo %d",
                *RxChan, DataSize, NumbWords, InvClk, RxStart, Bits, *TxChan, Bits, DataSize, NumbWords, *TxRate, ExtClk,
                        GatedClk, MSBsel, TimOut);

                Err_Code = vxiOutput(LogicalAddr, OutStr);
                if(Err_Code)
                {
                        printf("\t*** ERROR, vxiOutput - %s, Err_Code = %d\n", OutStr, Err_Code);
                        return Err_Code;
                }
                else
                        printf("VXI WS commmands sent: %s\n", OutStr);
                msDelay(100);

                vxiWSqry(LogicalAddr, ReadSTB, &RtnValue);
                if((RtnValue & 32))
                        puts("\t*** ERROR - Bad parameter ***");
                else
                        break;
        } while(1);

        return Err_Code;
}
/* ************************** A32_WriteBufr ************************** */
/*      This function writes a FDC buffer in A32 memory space.          */
/* ******************************************************************** */

void A32_WriteBufr(int LogicalAddr, int chan, int *buffer)
{
        int Err_Code, i, size;
        unsigned long MemAddr;

        size = *buffer;

        if(XFDC_GetChanAddr(LogicalAddr, chan, &MemAddr)) exit(Err_Code);

        for(i=0; i<size; i+=2)
        {
                if(vxiWrA32Reg(MemAddr, i+4, *buffer++)) break;
        }
}

/* ************************** XFDC_PassBuffer ************************** */
/*      This function passes a FDC channel buffer.                      */
/* ******************************************************************** */

void XFDC_PassBuffer(int LogicalAddr, int chan)
{
        int Err_Code;

        Err_Code = vxiWScmd(LogicalAddr, XFDC_PASSED_BUFR+chan);

        if(Err_Code)
        {
                printf("*** ERROR %d sending X Passed Buffer:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
        }
}
```

**Figure A-4 FDC Demo Program Listing Continued**

```
/* ********************** XFDC_ChannelClose ************************** */
/*     This function closes a XFDC channel.                            */
/* ********************************************************************* */

void XFDC_ChannelClose(int LogicalAddr, int chan)
{
        unsigned RtnValue;
        int Err_Code;

        Err_Code = vxiWSqry(LogicalAddr, XFDC_GO_TO_IDLE_IM+chan, &RtnValue);
        if(!Err_Code)
                printf("LA %d, Go to Idle IM:%d response: %04X\n", LogicalAddr, chan, RtnValue);
        else
        {
                printf("*** ERROR %d sending X Go to Idle IM:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
        }

        Err_Code = vxiWSqry(LogicalAddr, XFDC_CHAN_CLS+chan, &RtnValue);
        if(!Err_Code)
                printf("LA %d, X FDC Chan Close:%d response: %04X\n", LogicalAddr, chan, RtnValue);
        else
        {
                printf("*** ERROR %d sending X Chan Close:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
        }
}

/* ********************** XFDC_OpenWriteBufrPS ************************** */
/*     This function opens a FDC write buffer in Pair & Stream mode.       */
/* ********************************************************************* */

void XFDC_OpenWriteBufrPS(int LogicalAddr, int chan)
{
        unsigned RtnValue;
        int Err_Code;

        Err_Code = vxiWSqry(LogicalAddr, XFDC_CHAN_INIT+chan, &RtnValue);
        if(!Err_Code)
                printf("LA %d, X FDC Chan Init:%d response: %04X\n", LogicalAddr, chan, RtnValue);
        else
        {
                printf("*** ERROR %d sending X Chan Init:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
        }

        Err_Code = vxiWSqry(LogicalAddr, XFDC_XFER_TO_SRV_PS+chan, &RtnValue);
        if(!Err_Code)
                printf("LA %d, X FDC Xfer to Srv PS:%d response: %04X\n", LogicalAddr, chan, RtnValue);
        else
        {
                printf("*** ERROR %d sending X Xfer to Srv PS:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
        }
}
```

**Figure A-4 FDC Demo Program Listing Continued**

```
/* ***************************** XFDC_GetHeader ************************    */
/*      This function reads the X FDC buffer header bits.                    */
/* ********************************************************************    */

int XFDC_GetHeader(int LogicalAddr, int chan)
{
        long ChanAddr;
        unsigned RtnValue;
        int Err_Code;

        Err_Code = vxiWSqry(LogicalAddr, XFDC_CHAN_ADDR_HIGH+chan, &RtnValue);
        if(!Err_Code)
        {
                ChanAddr = (long)RtnValue << 16;
        }
        else
        {
                printf("*** ERROR %d sending X FDC Chan Addr High:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
                return -1;
        }

        Err_Code = vxiWSqry(LogicalAddr, XFDC_CHAN_ADDR_LOW+chan, &RtnValue);
        if(!Err_Code)
        {
                ChanAddr |= RtnValue;
        }
        else
        {
                printf("*** ERROR %d sending X FDC Chan Addr Low:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
                return -1;
        }

        Err_Code = vxiRdA32Reg(ChanAddr, 0, &RtnValue);
        if(Err_Code)
        {
                printf("*** ERROR %d reading from Address: %08lX\n", Err_Code, ChanAddr+2);
                vxiErrorDisp(Err_Code);
                return -1;
        }

        return RtnValue;
}
```

**Figure A-4 FDC Demo Program Listing Continued**

```
/* ************************ XFDC_OpenReadBufrPS ************************  */
/*      This function opens a FDC read buffer in Pair & Stream mode.                    */
/* ********************************************************************  */

void XFDC_OpenReadBufrPS(int LogicalAddr, int chan)
{
        unsigned RtnValue;
        int Err_Code;

        Err_Code = vxiWSqry(LogicalAddr, XFDC_CHAN_INIT+chan, &RtnValue);
        if(!Err_Code)
                printf("LA %d, X FDC Chan Init:%d response: %04X\n", LogicalAddr, chan, RtnValue);
        else
        {
                printf("*** ERROR %d sending X Chan Init:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
        }

        Err_Code = vxiWSqry(LogicalAddr, XFDC_XFER_TO_CMD_PS+chan, &RtnValue);
        if(!Err_Code)
                printf("LA %d, X FDC Xfer to Cmdr PS:%d response: %04X\n", LogicalAddr, chan, RtnValue);
        else
        {
                printf("*** ERROR %d sending X Xfer to Cmdr PS:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
        }
}

/* ************************ XFDC_GetChanAddr ************************  */
/*      This function get a XFDC buffer address.                        */
/* ********************************************************************  */

int XFDC_GetChanAddr(int LogicalAddr, int chan, unsigned long *MemAddr)
{
        unsigned RtnValue;
        int Err_Code;

        Err_Code = vxiWSqry(LogicalAddr, XFDC_CHAN_ADDR_HIGH+chan, &RtnValue);
        if(!Err_Code)
        {
                *MemAddr = (long)RtnValue << 16;
        }
        else
        {
                printf("*** ERROR %d sending XFDC Chan Addr High:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
        }

        Err_Code = vxiWSqry(LogicalAddr, XFDC_CHAN_ADDR_LOW+chan, &RtnValue);
        if(!Err_Code)
        {
                *MemAddr |= RtnValue;
        }
        else
        {
                printf("*** ERROR %d sending XFDC Chan Addr Low:%d to Logical Address: %d\n", Err_Code, chan, LogicalAddr);
                vxiErrorDisp(Err_Code);
        }
        return Err_Code;
}
```

**Figure A-4 FDC Demo Program Listing Continued**