ICS ELECTRONICS
division of Systems West Inc.

Better Solutions for GPIB and VXIbus Systems

VXIbus APPLICATION BULLETIN

VXI-1 Fast Data Channel Operation and ICS Expanded Command Set

INTRODUCTION

This Application Bulletin provides background information about the VXI-10 Fast Data Channel (FDC), about 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. While this Application Bulletin 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.

The VXI modules with Fast Data Channels are just becoming available and the majority of VXI module users are not familiar with the VXI Fast Data Channel protocols. Because of the newness of the VXI Fast Data Channel Specification, information about using the Fast Data Channel is not readily available. The purpose of this Application Bulletin is to provide the user with the necessary information to program the Fast Data Channels in ICS modules. However, a large portion of the information in this Application Bulletin is generic in nature and applies to any VXI module that uses Fast Data Channel.

FAST DATA CHANNEL ADVANTAGES

VXIbus message based modules have inherently slow data transfer rates because of the complex protocol used to transfer word serial messages. Data transfer with word serial messages is one byte of information for every 2 or 4 four bus transactions. This 'handshaking' holds the word serial message data transfer rate down to 10 to 20 kbytes/second which is too low for many real-time data transfer applications.

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. VXI modules with Fast Data Channel buffers have 'dual-port' memory that appears in the VXI Controller's A32 memory space. The FDC channels (buffers) are located in this shared memory space so they can be accessed by both modules.

Data transfer with the Fast Data Channel approaches the maximum transfer rate for the VXIbus, up to 32 Mbytes/sec. 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 channel word count and then makes one read per word. Each word is 32-bits wide and can contain four data bytes.

In ICS's VXI modules, the VXI-10 Fast Data Channel protocol is used as a way to improve the data transfer rate across the VXIbus and free up the VXI Controller for other functions besides transferring data. This is necessary for systems that have multiple data channels or for those systems with high-speed continuous data transfer requirements. ICS's modules use two large FDC buffers per data direction to prevent data loss and to buffer the incoming or outgoing data. The high-speed FDC data transfer rate coupled with data buffering greatly reduces the load on the VXI Controller and gives it time to do other computations.

DESCRIPTION

VXI Fast Data Channels occupy space in A32 memory and can be used to transfer data and/or commands between the Commander and the Servant module. The memory is physically part of the VXI module but it is mapped into the Controller's A32 address space. FDC channels (or buffers) and can be used singly or in pairs. The number, size and organization of the channels and their use are up to the module designer. 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. These control bits prevent data contention since only the channel owner can read or write data in the buffer.

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.

FAST DATA CHANNEL USAGE IN ICS'S VXI MODULES

In ICS VXI modules, the VXI-10 Fast Data Channel protocol is mainly used as a way to provide continuous data transfer capability while using a minimum of the VXIbus's data bandwidth and data transfer time. A pair of FDC channels are used for a single data transfer direction. The channels operate as *A/B buffer pair*s and in *Stream Transfer* mode for continuous data throughput. One module starts by filling the first buffer and then passing it to the other module. The second buffer is filled while the other module is emptying the first 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 buffers or FDC channels for a bidirectional data port such as a serial port or a GPIB bus interface. ICS's multiport VXI 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 listed in Table 1 and described in the Command Reference Section. These commands mirror the standard FDC command set.

The FDC channels must be initialized before they are used to establish data direction. The recommended initialization process is shown in Figure 3. The process shown in Figure 3 closes the channels at the start of the initialization process in case they had been previously initialized. Initializing an open channel will result in an error.

Transmit channels (buffers) are passed to the VXI Controller (Commander) when they are empty. The Commander 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 restriction is that the amount of data cannot exceed the maximum channel size. The Commander must keep the module supplied with a new buffer before the old one is used up to maintain continuous data flow.

Receive channels (buffers) alternate as they become full, or upon the receipt of a designated number of bytes in the buffer or upon receipt of the Switch Buffers command. The number of bytes for the switch point is set by the user with the SCPI *SYST:INP:BUF:SIZE* command. This command must be given before the *Transfer to Commander* command to be effective with the first buffer. When the buffer has filled enough to reach the switch point, the receiver will switch to the other buffer (if it has been returned as empty). 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 any extra FDC channel usage. The number of channels in a module can also be determined by querying the module with the *FDC Supported* command.

FAST DATA CHANNEL (FDC) MEMORY MAPS

Memory maps for a typical FDC channel are shown in Figures 1 and 2. Both figures contain the same information but show it by different VXIbus word widths. Figure 1 is organized as 16-bit words and Figure 2 is organized as 32-bit words. The first eight bytes in both figures contain the mandated FDC Channel Header information. The header contains the Control Bits for determining buffer ownership and defines the channel buffer size. The FDC Channel buffer space starts with Byte 8. Module designers may use the buffer space in the way that best fits their application. Byte numbers are in Motorola order for correlation with the VXI-10 Specification.

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

Figure 1 16-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.

Figure 2 32-Bit Wide FDC Memory Map

FAST DATA CHANNEL BUFFER DEFINITIONS

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

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.

FDC CHANNEL INITIALIZATION SEQUENCE

Figure 3 on the right shows the recommended initialization sequence for a streaming channel pair. The sequence can be used with minor changes to initialize any FDC channel. 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.

If streaming channel pairs are to be swapped when the buffer contents reach a preset byte count, the byte count needs to be set with the SCPI *SYST:INP:BUF:SIZE* command before issuing the *Transfer-to-Commander* command. The *Channel Size* commands will return the current buffer size setting. In ICS modules, the *Go-to Idle Immediate* command also resets the channel specific hardware.

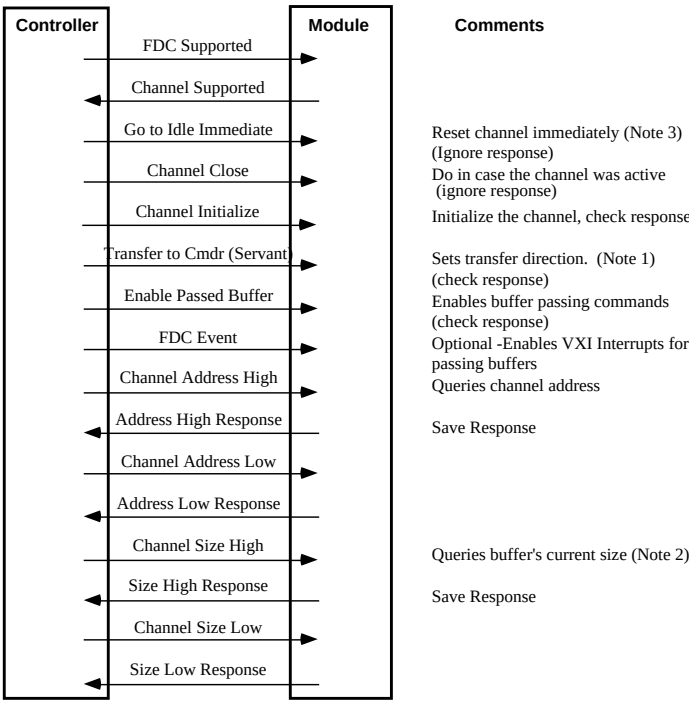

Figure 3 Recommended FDC Channel Initialization Sequence

FAST DATA CHANNEL PASSING SEQUENCE

Transfer to Servant

Figure 4 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 channels. 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 4 is for the Transfer to Servant direction. The sequence starts when the Commander loads the first (even) buffer and transfers it to the Servant. While the Servant empties the first buffer, the Commander fills the next buffer. The buffers are exchanged and the process repeats. The steps below the dotted line are repeated until terminated by the Go-to-Idle command.

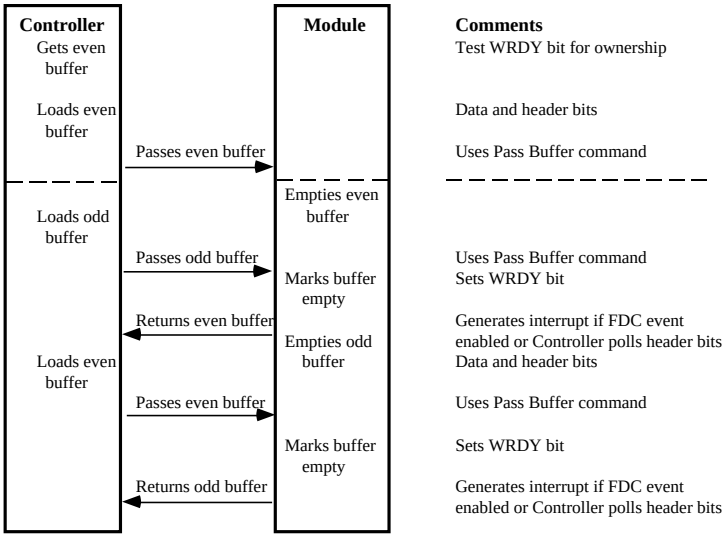

Figure 4 Transfer to Servant Buffer Passing Sequence

Transfer to Commander

Figure 5 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:INP:BUF:SIZE* command. The steps below the dotted line are repeated until terminated by the Go-to-Idle command.

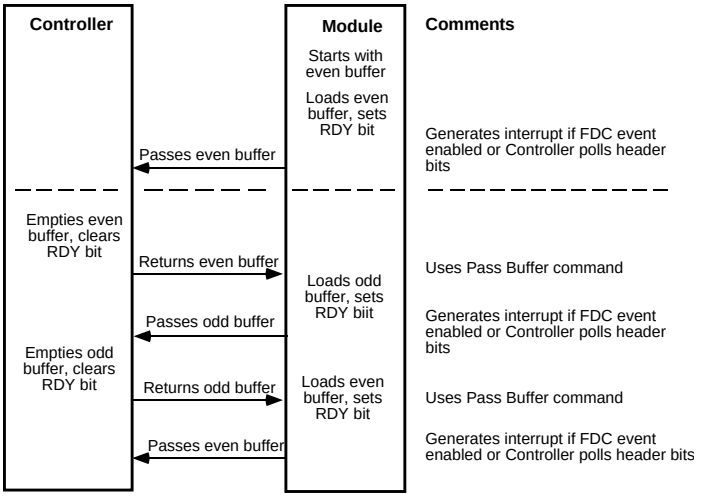

Figure 5 Transfer to Commander Buffer Passing Sequence

FDC Command Summary

VXIbus modules with the Fast Data Channel option respond to the FDC Standard commands listed in Table 1. ICS modules also respond to ICS's expanded FDC Command Set. The expanded commands are the similar to the standard VXI-10 commands, except they allow for 32 channels. The formats are the same except for the command codes and the channel number field which is expanded to 5 bits to allow for up to 32 channels. ICS modules support both command sets for software compatibility. However, 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 1 lists the standard and ICS expanded FDC commands.

Example Program

ICS has written an example program that can be used as a prototype for constructing your own FDC channel handling routines. The program is written in the C language so that it can be easily used with most VXI Controllers. The example program is well documented and covers FDC channel initialization and their use to transfer data. A .zip file (5536DEMO.ZIP) with the example program can be downloaded from ICS's website at http://www.icselect.com/ab_note.html.

The program was developed for ICS's VXI-5543/44 series Slot 0 VXI Controllers and uses ICS's own command set. When adapting the program to another VXI Controller, the user should replace ICS's commands with those from the other Controller. There are only a few commands which can be easily replaced with a good editor. The functions used in the example are described in the Addendum to this application note.

Selecting VXI Bus Controllers

When selecting a VXI bus Controller for use with a FDC capable module, the VXI Controller must be capable of accessing the A32 address space. Because of this requirement, most of the inexpensive GPIB-VXI Slot 0 Controllers on the market today cannot be used with FDC capable modules. The VXI Controller should also perform 32-bit data transfers to maximize the FDC data transfer rate, but 16-bit data transfers will also work. The Controller's software should have FDC and Word Serial Command functions to minimize your programming work. When in doubt, consult the manufacturer's application support engineers before buying.

Summary

This application note has shown that the use of the VXI Fast Data Channels in a VXI module greatly speeds up data transfer and reduces the load on the VXIbus. The FDC channels buffer a large amount of data and give the user a way of sending or receiving continuous data streams.

This application note also shows how a pair of channels can be initialized and opened for transferring data in either direction. An example FDC channel handling program is available for the user to use as a template for his own program.

TABLE 1 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 0110 000e
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) i = immediate change
N.S. = Not Supported p = pair flag
c = channel code s = stream flag
e = enable

FDC COMMAND REFERENCE

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

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 Numbe					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 Module 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 Module 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.

Addendum
VXI-5543 Command Reference

A.1 Introduction

This addendum reprints ICS's VXI Command definitions so the user can find the equivalent commands when converting the 5536DEMO program over to another VXI Controller

The VXI-5543's VXI Driver Library contains all of the high and low level functions for the control of the VXI bus. These commands are all ICS's proprietary functions. The advantage of ICS's Driver library is that it contains many high level functions that simplify the user's programming effort and are easy to use.

A.2 Common Command Parameters

Table A-1 lists the Common Command Parameters used in the VXI-5543's VXI library.

TABLE A-1 VXI LIBRARY COMMON COMMAND PARAMETERS

PARAMETER	DESCRIPTION
LA	VXI Logical Address integer value, 0 to 255
RegOffset	VXI Register Offset integer value, even number 0 to 30
In_String	Input string, character pointer to string buffer for input
Max_Len	Maximum length, maximum number of characters to input to pointer location
Output_String	Output String, character pointer to string for output
Wscmd	Word Serial Command, unsigned integer with standard 16 bit VXI Word Serial Command code
RtnValue	Return Value, unsigned integer pointer to location for 16 bit word
Value	Output Value, unsigned integer value to output
Trigger_bits	Trigger Bit Pattern, integer with trigger enable mask word
Intr_bits	Interrupt Bit Pattern, integer with interrupt enable mask word
IRQ_lines	VXI IRQ Lines, integer with VXI IRQ enable mask word
Trigger_line	TTL Trigger Line, integer with TTL Trigger line number, 0 to 7
ModelCode	VXI Device Model Code, unsigned integer with manufacturers model code
IRQ_line	VXI IRQ Line, integer with VXI IRQ line number 1 to 7
MODID_line	VXI MODID Line, integer with MODID line number, 0 to 12
Status_ID	VXI Status ID, unsigned integer with device Status ID code
Display_Flag	Display Flag, integer with display enable flag, 0 to 1
Error_Code	Error Code, integer with VXI library error code, 0 to 10
void	no parameters required

Note: Function descriptions which end with a (Q) are queries and return a response which must be read before sending the device another command. Use vxiWSqry with these commands.

A.3 VXI Word Serial Commands

VXI Word Serial Commands are single word commands that contain binary coded information. The VXI Specification provides a list of standard commands but not all instruments are required to respond to all commands. When in doubt check the device's manual before using a particular command. In addition, the VXI Specification also allows for user defined Word Serial Commands to accomplish a function unique to that instrument. For this reason, it is a good idea to check a device's manual before programming it to learn what commands it responds to.

The VXI-5543 outputs Word Serial Commands that **do not require a response** with **vxiWScmd** function. Word Serial Commands that **expect a response** should be outputted with the **vxiWSqry** function. For convenience sake and for better program documentation, the VXICMDS.H include file defines the Word Serial Commands with English like names. Table A-2A lists the Word Serial Commands, their HEX value and English name. Table A-2B lists the Word Serial Command error response codes.

TABLE A-2A VXI WORD SERIAL COMMANDS

WORD SERIAL COMMAND	CODE	FUNCTION
AbortNormOp	0xc8ff	Abort Normal Operation (Q)
AssgnHandLine	0xa900	Assign Handler Line (Q)
AssgnIntrLine	0xaa00	Assign Interrupter Line (Q)
AysncModeCntl	0xa800	Asynchronous Mode Control (Q)
BeginNormOp	0xfcff	Begin Normal Operation (Q)
BeginNormOpT	0xfdff	Begin Normal Operation Top (Q)
ByteAvail	0xbc00	Byte Available
Byte AvailEnd	0xbc00	Byte Available End of Mesg
ByteReq	0xdeff	Byte Request (Q)
Clear	0xfff	Clear
ClearLock	0xefff	Clear Lock
ControlEvent	0xaf00	Control Event (Q)
ControlResponse	0x8f00	Control Response (Q)
EndNormOp	0xc9ff	End Normal Operation (Q)
GrantDev	0xbf00	Grant Device
IndentCmdr	0xbe00	Identify Commander
ReadHandlers	0xc7ff	Read Handlers (Q)
ReadHandLine	0x8c00	Read Handler Line (Q)
ReadIntrLine	0x8d00	Read Interrupter Line (Q)
ReadIntrpters	0xcaff	Read Interrupters (Q)
ReadMODID	0xccff	Read MODID (Q)
ReadProto	0xdfff	Read Protocol (Q)
ReadProtoErr	0xcdeff	Read Protocol Error (Q)
ReadSTB	0xcfff	Read STB (Q)
ReadServArea	0xceff	Read Servant Area (Q)
ReleaseDev	0x8e00	Release Device (Q)
SetLock	0xeeff	Set Lock
SetLowerMODID	0xae00	Set Lower MODID (Q)
SetUpperMODID	0xad00	Set Upper MODID (Q)
Trigger	0xedff	Trigger

TABLE A-2B WORD SERIAL COMMAND RESPONSE CODES

WORD SERIAL COMMAND	CODE	FUNCTION
WScmdOK	0xfffe	Word Serial command OK
WScmdFAIL	0x7ffe	Word Serial command FAILED

4.4 Command Quick Reference List

The Quick Reference List is divided into the followings sections:

Table A-3 lists the VXIbus Commands and Functions
Table A-4 lists the Returned Error Codes

TABLE A-3 VXIBUS LIBRARY COMMANDS

COMMAND	DESCRIPTION
VXI CONTROL FUNCTIONS	
void vxiArmTrig(int Trigger_bits)	Arm TTL VXI Trigger line(s) for output
void vxiClrIRQ(void)	Clear all VXI IRQ lines
void vxiClrMODID(void)	Clear all VXI MODID lines
int vxiGetAddr(int ModelCode, int Occurrence)	Get Logical Address of device with occurrence of Model Code
int vxiGetAttrib(int LogicalAddr, int AttribName,long *Value)	Get Device Attribute
int vxiOpen(void)	Read Initialization and Commander table files
void vxiPlsTrig(void)	Pulse armed VXI TTL Trigger lines
void vxiSetAttrib(int LogicalAddr, int AttribName, long Value)	Set Device Attribute
void vxiSetIRQ(int IRQ_line)	As a device, set VXI IRQ line true
void vxiSetMODID(int MODID_Line)	As a device, set VXI MODID line
void vxiSysReset(int Display_Flag)	Send SysReset to VXI chassis
void vxiTrigger(int Trigger_Line)	Pulse VXI TTL Trigger line
VXI REGISTER FUNCTIONS	
int vixRdReg(int LA, int RegOffset, unsigned *RtnValue)	Read a hex value from VXI register (RegOffset)
vxiRdA24Reg(long BaseAddr, long RegOffset, unsigned int *RtnValue)	Reads a hex value from A24 VXI address space
vxiRdA32Reg(long BaseAddr, long RegOffset, unsigned int *RtnValue)	Reads a hex value from A32 VXI address space
int vxiWrReg(int LA, int RegOffset, unsigned Value)	Write hex value to VXI register (RegOffset)
vxiWrA24Reg(long BaseAddr, long RegOffset, unsigned int Value)	Writes a hex value to A24 VXI address space

COMMAND	DESCRIPTION
vxiWrA32Reg(long BaseAddr. long RegOffset,unsigned int Value)	Writes a hex value to A32 VXI address space
WORD SERIAL FUNCTIONS	
int vxiEnter(int LA, char *In_String, int Max_Len)	Read Word Serial Message string from VXI device
int vxiEnterB(int LogicalAddr, char *MsgPtr, long MaxLen, long *ByteCount)	Read Word Serial Binary Message
int vxiOutput(int LA, const char* Output_String)	Send Word Serial Message string to VXI device
int vixOutputB(int Logical Addr, char *MsgPtr,long Count, long *SentCount)	Output Word Serial Command
int vxiWScmd(int LA, unsigned WScmd)	Send hex Word Serial Command
int vxiWSqry(int LA, unsigned Wscmd, unsigned *RtnValue)	Send hex Word Serial Command and get Response
INTERRUPT FUNCTIONS	
void vxiCloseIntr(void)	Remove ISR vector and clear Intr and IRQ masks
void vxiEnaIntr(int Intr_Bits)	Enable summary interrupt register with mask
void vxiEnaIRQ(int IRQ_lines)	Enable VXI IRQ line detector
void vxiEnaTrig(int Trig_line)	Enable VXI TTL Trigger line detector for line
int vxiGetIntr(void)	Get status of summary interrupt register
int vxiGetIRQ(void)	Get status of VXI IRQ lines
int vxiGetTrig(void)	Get status of VXI TTL Trigger detector
void vxiInitIntr(int Intr_mask, int IRQ_mask)	Initialize ISR and set Intr and IRQ masks
int vxiSrvIntr(int *LogicalAddr, int *Status, int *Intr_Status)	Test IRQ interrupt and service VXI IRQ
int vxiSrvIRQ(int IRQ_numb, unsigned *Status_ID)	Service VXI IRQ and get STATUS/ID word
DISPLAY and LED FUNCTION	
void vxiDispCmdrTable(void)	Display commander table information
void vxiDispSysInfo(void)	Display system information (S/N etc.)
void xiErrorDisp(int Error_Code)	Display VXI error message on console device

COMMAND	DESCRIPTION
void ClrAccessLED(void)	Clears Access LED
void SetAccessLED(void)	Sets Access LED
TIME UTILITIES	
void far msDelay(unsigned ms)	Suspends program execution for ms milliseconds.
unsigned long far icsTimer(void)	Reads high resolution system timer. Returns number of 215 µsec periods since midnight.
bool is TimeOut(msMark)	Sets background time mark when first called. Later calls check to see if system time is greater than the time mark. msMark is in milliseconds.

TABLE A-4 RETURNED ERROR CODES

ERROR CODE	ERROR NUMBER	ERROR MESSAGE - vxiErrorDisp()
No_Error	0	No Errors detected
Buss_Err	1	***BUS ERROR (*BERR) ***
WrtRdy_Err	2	***WRITE Ready Time Out ERROR ***
RdRdy_Err	3	*** READ Ready Time Out ERROR ***
DIR_Err	4	*** DIR Time Out ERROR ***
DOR_Err	5	*** DOR Time Out ERROR ***
Proto_Err	6	*** PROTOCOL ERROR (*ERR)
BusGrnt_Err	7	*** Bus Not Granted ERROR ***
RspRead_Err	8	*** Response Not Read ERROR***
LA_Err	9	*** Logical Address ERROR ***
Reg_Err	10	*** Register Offset ERROR ***
undefined	undefined	*** Undefined ERROR code ***