VMEbus Extensions for Instrumentation

Fast Data Channel Specification
VXIbus-10

Revision 2.10
May 18, 1995

Table of Contents

A. Introduction

Fast Data Channel (FDC) protocol provides a mechanism for transferring data between a VXIbus Commander and its Servants. The primary goals of FDC are:

> Provide an efficient mechanism for transferring large blocks of data between VXIbus Commanders and their Servants.
>
> Utilize standard, existing (VME compatible) hardware structures.
>
> Minimize the complexity of the protocol, making software driver development easy and driver execution fast.

The complexity of the software driver has been kept to a minimum to make implementation easy and execution efficient. Memory which is accessible by both devices, a Message Based interface and a software driver on the Commander and Servant are the only requirements for implementation of the protocol.

A.1 Description

The *FDC protocol* supports the transfer of *data blocks* between a VXIbus Commander and its VXIbus Servant. The data can be moved from the Commander to the Servant or from the Servant to the Commander. The media used to move the data is electronic memory which is accessible to both the commander and the servant. VXIbus Word Serial commands are used to determine the size and location of the memory. Simple flags are used to handshake *buffers of data* between the commander and the servant, allowing the transfer of data blocks larger than the available Memory.

The FDC protocol requires that a software driver exist on both the commander and the servant. This driver establishes an FDC *Channel* and manages data transfers. The VXIbus commander is responsible for FDC channel management and initiating any data transfers. The commander first sends the *Channel Initialize* command to the servant. This creates a valid FDC channel. It then sends the *Channel Address* and *Channel Size* Word Serial commands to determine the location and size of the FDC Area. At this point a valid FDC channel exists between the commander and the servant.

Once an FDC channel has been created, it can be used to transfer data blocks. To transfer data from the commander to the servant, the commander sends a *Transfer to Servant* command to the servant. The servant responds by setting the appropriate flags in the Channel Header and returns a success response. Buffers of data are passed from the commander to the servant until the entire data block has been transferred.

To transfer data from the servant to the commander, the *Transfer to Commander* command is sent to the servant. The servant responds by setting the appropriate flags in the Channel Header and returns a success response. Buffers of data are passed from the servant to the commander.

Four flags are used to manage the transfer of data blocks. The *Write Ready* and *Read Ready* flags are used to pass buffers of data back and forth between the commander and servant.

Either the commander or the servant *owns* the FDC area at any point in time. The owner of the FDC area has both read and write privileges to the Channel Header and Channel Buffer. The non-owner has only read privileges for the FDC header only. Ownership is identified by the *Write Ready* and *Read Ready* flags.

The *End* flag informs the data receiver that this buffer of data is the last buffer in the transfer of the data block.

The fourth bit in the FDC Header is the *Abort* bit. If either the commander or servant chooses to prematurely terminate the transfer, it can set the abort bit when it receives ownership of the FDC Area.

The device setting the *Abort* bit is the *abort generator*. Once the abort generator sets the *Abort* bit, it passes the FDC area to the *abort receiver*. The abort receiver acknowledges the abort by clearing the *Abort* bit and terminating the transfer. When the abort generator detects this acknowledgment it also terminates the transfer. The data block being transferred is invalid if the transmission is terminated by the *Abort* bit. Reporting of the abort to the user application is an instrument specific function.

An FDC channel can be in one of two states: *active* or *idle*. When a channel is not engaged in data transfer, it is idle. This is its state immediately after creation. While a data transfer is in progress the channel is active. A channel may transition between the active and idle states many times during its lifetime.

Two classes of FDC commands are defined: *Standard* and *Extended*. Standard commands are 16 bit Word Serial Commands. They are limited to controlling no more than 8 channels. Extended commands are 32 bit long Word Serial commands providing up to 2^{16} channels.

Channels are formed only between commanders and their direct servants. Each of these channels are independent: They each have unique FDC areas.

Enhanced throughput may be achieved by utilizing a *pair* of FDC channels in tandem. Pairs are formed using adjacent even-odd channels i.e. 0-1, 2-3, 4-5, ... The lower numbered channel always transfers the first buffer of data in a block of data. The data source switches back and forth between the channels as it sends each buffer of data. The data receiver tracks this action, switching back and forth between the two channels as it reads each buffer of data. Using FDC channel pairs allows the sender to write a buffer of data to one channel while the receiver is reading a buffer of data from the other channel.

An FDC channel can be one of two types: *Normal* or *Stream*. A Normal channel transfers blocks of data and transitions between the idle and active states. Each transfer is initiated by *Transfer to Commander* or *Transfer to Servant* command. When the last buffer of data is transferred the channel will transition to the idle state.

A Stream channel is always in the active state. It is established in the same manner as a normal channel except the *Stream* flag is set in the *Transfer to Servant/Commander* command. A *Transfer to Servant* or *Transfer to Commander* command is sent only once, when the channel is formed, to establish the direction of the stream. Buffers of data are passed in the regular fashion. The *End* bit is used to separate data blocks but does not terminate the transfer. Stream channels are terminated by sending the *Go to Idle* command.

Three modes of access to the FDC buffer are supported to allow efficient implementation of instrument hardware: random, linear, and FIFO access. Random allows data access in any sequence any number of times. Linear requires a single sequential data access through the buffer. FIFO places the entire data buffer behind a single address and requires repeated access of that single location. Supported FDC data access modes are indicated by the response from the *Channel Initialize* command.

The format and type of data transferred via FDC channels is not defined by the FDC protocol. It is the responsibility of both the module manufacturer and the user to ensure the data transferred is usable by the VXIbus commander or servant device. Where appropriate, identification information may be included in the data block so that error checking can be made.

A.2 Vocabulary

FDC: Fast Data Channel - A VXIbus protocol which supports the transfer of data between Commanders and Servants.

FDC Area: A portion of memory which is accessible to both the VXIbus commander and servant which is allocated to the FDC Channel. This memory contains the Channel Header and Channel Data Buffer.

Channel Header: The first 8 bytes of the FDC area. The header contains the FDC Version, data buffer management flags and the data size value.

Data Buffer: The portion of the FDC area directly following the channel header and extending to the end of the FDC area. The data buffer is used to temporarily store data being transferred.

FDC Channel: The memory and protocol which provides the data transport mechanism for FDC.

Data Block: A well defined array of data in an application specific format.

Buffer of Data: A buffer of data is a logical construct which contains two elements. The first element is all or a portion of the data in a data block to be transferred using FDC protocol. The second element is a 32 bit unsigned integer which reflects the number of bytes or the number of FIFO accesses contained in this particular portion of the data block.

Idle Channel: An FDC channel which has been initialized but is not currently engaged in transferring data.

Active Channel: An FDC channel which is currently engaged in transferring data.

Abort Generator: An FDC device which, during this transfer, has set the *Abort* bit.

Abort Receiver: An FDC device which, during this transfer, will acknowledge the *Abort* bit which was set by the other FDC device.

FDC Driver: A software or firmware algorithm which implements the FDC protocol.

Normal Transfer: An FDC transfer mode which can be used to transfer discrete data blocks. A normal channel transitions between the active and idle state for each data block transfer.

Stream Transfer: An FDC transfer mode which is always active and is able to pass a continuous flow of data.

FDC Command: An FDC command which uses the 16 bit Word Serial commands for channel management.

Extended FDC Command: An FDC command which uses the 32 bit Word Serial commands for channel management.

BC Command: A standard FDC command which has bit 15 set to 0. BC commands are provided for backwards compatibility to prior versions of the FDC standard.

Data Source: The VXIbus device which is providing the data being transferred. During a transfer to servant, the commander is the data source. During a transfer to commander, the servant is the data source.

Data destination: The VXIbus device which is accepting the data being transferred. During a transfer to servant, the servant is the data destination. During a transfer to commander, the commander is the data destination.

Local Memory: Memory which is directly owned by the FDC servant. This can be memory which physically resides either on the servant or an VXIbus memory device which is a direct servant of the FDC servant. To utilize memory which does not physically reside on the FDC servant, the servant must be a bus master.

Remote Memory: Memory which is not directly owned by the FDC servant. A remote memory pool must be allocated to the servant by the FDC commander. The FDC commander and servant then negotiate an FDC area within this memory pool to create an FDC channel.

A.3 FDC Memory Allocation Map

The FDC area contains the FDC Channel Header information and the FDC Data Buffer.

BIT	7 - 4	3	2	1	0

FDC Channel Header					
BYTE 0	Minor Revision		Major Revision		
BYTE 1	RSVD	RSVD	RSVD	RSVD	RSVD
BYTE 2	RSVD	RSVD	RSVD	RSVD	**TRIG**
BYTE 3	RSVD	**ABT**	**RDY**	**WDY**	**END**
BYTE 4	**DATA SIZE** BITS 31-24				
BYTE 5	**DATA SIZE** BITS 23-16				
BYTE 6	**DATA SIZE** BITS 15-8				
BYTE 7	**DATA SIZE** BITS 7-0				

FDC Channel Buffer	
BYTE 8	**DATA BUFFER** BYTE 0 (FIFO BITS 31-24)
BYTE 9	**DATA BUFFER** BYTE 1 (FIFO BITS 23-16)
BYTE 10	**DATA BUFFER** BYTE 2 (FIFO BITS 15-8)
BYTE 11	**DATA BUFFER** BYTE 3 (FIFO BITS 7-0)
. . . BYTE N	**DATA BUFFER** BYTE N-8

Notes

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 *Write Ready* flag is utilized when data is transferred from the commander to the servant. If the *Write Ready* 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 *Write Ready* to 0 to pass the buffer of data to the servant. If the *Write Ready* bit is set to 0, the VXIbus servant owns the FDC area. It may read the buffer of data and then set *Write Ready* high to pass the FDC area back to the commander. When the channel is in the idle state, *Write Ready* is 0.

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

ABT: The *Abort* bit indicates that an abort transfer is being requested for this block of data. The current owner of the FDC area (abort generator) may set the *Abort* bit to a 1 to request that a data block transfer be aborted. The abort generator then passes ownership of the FDC area to the abort receiver.

In a Normal channel the abort receiver sets the *Abort* bit to 0 (zero) and returns the channel to the Idle state. It discards any data which was received during the aborted transfer. The abort generator returns the channel to the idle state when the *Abort* bit is returned to 0 by the abort receiver.

In a Stream channel the abort receiver sets the *Abort* bit to 0 but does not change the state of the channel; the channel remains in the Active state. It discards any data which was received during the aborted block transfer.

DATA SIZE: The FDC DATA SIZE contains the number of bytes contained in this buffer of data.

DATA BUFFER: Memory area where the data to be transferred is placed. There are 3 modes of access supported for the data buffer: Random, Linear, and FIFO. Random access allows the buffer to be accessed in any order any number of times. Linear requires the buffer to be accessed from start to end incrementing to the next address on each access. Each location may be accessed only once. FIFO mode requires that the FDC Buffer is read repeatedly at the defined offset until the specified number of bytes have been read.

B. FDC Devices General Requirements

FDC devices manage the FDC protocol with VXIbus Word Serial commands. These commands are 16 or 32 bit values which are interpreted by the VXIbus servant and executed. Some of the commands provide a response word which indicates execution status and returns information about the state of the FDC channel. VXIbus Word Serial commands are supported only by VXIbus message based devices.

RULE B.1.1 :

An FDC device **SHALL** be a message based device.

To keep the FDC protocol simple and efficient, the allowed communications is limited to VXIbus commanders and their direct servants. A FDC channel may not be shared by more than two devices.

RULE B.1.2 :

An FDC channel **SHALL NOT** be shared by more than two VXIbus devices.

RULE B.1.3 :

An FDC channel **SHALL NOT** be established other than between a VXIbus servant and its VXIbus commander.

RULE B.1.4 :

When FDC channels are used in pairs then the pairs **SHALL** be formed between adjacent channels using the algorithm 2*N and (2*N) + 1 where N = 0, 1, 2, ... e.g.: 0 and 1, 2 and 3, 4 and 5 or any valid even odd pair.

RULE B.1.5 :

When FDC channels are used in pairs, the first buffer of data **SHALL** be sent on the lower numbered pair. The second buffer of data **SHALL** be sent on the higher numbered pair. Data buffer transmission **SHALL** continue to switch back and forth between the pair until the entire block of data is transferred or the transfer is terminated by an abort.

RULE B.1.6 :

The FDC area base address **SHALL** be aligned on a double long word (8 bytes) boundary.

RULE B.1.7 :

An FDC device **SHALL NOT** write to the FDC area unless it has ownership of the FDC area.

OBSERVATION B.1.1 :

If the *Write Ready* and *Read Ready* bits are both 0 then the servant owns the FDC area. If either of the *Write Ready* or *Read Ready* bits is 1 then the commander owns the FDC area

RULE B.1.8 :

An FDC device **SHALL NOT** read the FDC DATA area unless it has ownership of the FDC area. It may read the HEADER area at any time

RULE B.1.9 :

An FDC device **SHALL** read or write the FDC HEADER area only using VMEbus D16 accesses.

RULE B.1.10 :

When a device receives ownership of the FDC area during an FDC transfer, it **SHALL** first check the *Abort* flag. If the *Abort* flag is set, the device **SHALL** clear the *Abort* flag and if it is a normal channel, return the channel to the idle state.

OBSERVATION B.1.2 :

Because the FDC data buffer is required to be aligned on a double long word (8 byte) boundary and each buffer of data (except the last one) must have an even double long word length, the software algorithms which move data into or out of the data buffer can be optimized for efficiency and speed.

B.2 FDC Buffer Passing

During an FDC transfer, the commander and servant must pass ownership of FDC buffers back and forth. To determine that a buffer has been passed, the commander or servant may either poll the FDC header or utilize the buffer passing notification mechanisms provided by FDC.

The FDC *Passed Buffer* command provides notification to the servant that the commander has passed an FDC buffer to the servant. The channel number of the passed buffer is included in the Passed Buffer command. To utilize this mechanism the servant must indicate support for the *Passed Buffer* command in its response to the *Channel Initialize* command. For channels established with standard 16 bit FDC commands, the commander must enable the servant to accept the *Passed Buffer* command by sending the *Enable Passed Buffer* command. Channels established with Extended FDC commands are enabled by default. Once enabled, the servant is no longer required to poll the FDC channel header when waiting to receive a buffer. The commander will send the *Passed Buffer* command to the servant after a buffer is passed on any channel.

RULE B.2.1 :

An FDC commander **SHALL NOT** send the FDC *Passed Buffer* command to an FDC servant unless the servant indicates support for the *Passed Buffer* command in its response to the FDC *Channel Initialize* command and is enabled to receive the *Passed Buffer* command.

RULE B.2.2 :

If an FDC commander has enabled the FDC *Passed Buffer* command for its FDC servant, it **SHALL** send the FDC *Passed Buffer* command to that FDC servant each time it passes a buffer to the servant. The *Passed Buffer* command **SHALL** be sent after the buffer is passed.

OBSERVATION B.2.1 :

Extended channels which indicate support for the FDC *Passed Buffer* command are enabled by default. The Passed Buffer command must always be sent to an extended FDC servant which indicates support for the *Passed Buffer* command.

OBSERVATION B.2.2 :

The enabling of the FDC *Passed Buffer* command applies to all channels supported by the FDC servant. The *Passed Buffer* command can not be enabled or disabled for individual channels.

FDC events provide notification to the commander that its servant has passed it an FDC buffer. The servant must be enabled to send FDC events for each channel which the commander expects to receive events. The *FDC Event* command is used to enable or disable event generation. Once enabled, the commander need not poll the FDC channel header to determine when it is passed a buffer for that channel. Once the event is enabled, the servant will send the FDC event each time it passes a buffer on that channel.

When the commander receives an FDC event it must determine which channel the event is for. FDC events for standard 16 bit FDC channels contain the channel number in the event. Extended channel events do not contain the channel number. For extended channels, the commander must send the *FDC Event Query* command to the servant. The servant maintains a queue of extended events and returns the next entry in this queue. The *FDC Event Query* response indicates which channel sent the event. The commander should continue to send the *FDC Event Query* command until the servant's extended event queue is emptied. When the queue is empty, the servant will respond with *no events* in the Extended FDC Event Queue.

RULE B.2.3 :

If an FDC servant has been enabled to send the FDC event on a particular channel, it **SHALL** send the FDC event to that FDC commander each time it passes a buffer to the commander. The FDC Event **SHALL** be sent after the buffer is passed.

RULE B.2.4 :

If an extended FDC servant has been enabled to send the FDC event on a particular channel, it **SHALL** maintain a queue of events which can contain one event for each channel supported. Each time this event queue becomes not empty, the servant **SHALL** send an extended FDC event.

RULE B.2.5 :

If an FDC commander has enabled events on an extended FDC servant and receives an extended FDC event, it **SHALL** send the *FDC Event Query* command to the FDC servant to retrieve the channel number of the FDC event. The commander **SHALL** repeat this operation until all events have been retrieved from the servant.

PERMISSION B.2.1 :

An FDC commander which retrieves extended FDC events using the *FDC Event Query* command **MAY** process each event as it is retrieved or retrieve all events and then process them.

B.3 Normal Data Transfer Termination

Normal channel transfer termination is achieved by setting the *End* flag in the channel header. When the *End* flag is set, it indicates that this is the last buffer of data to transfer.

Setting the *End* flag in a Stream channel defines the end of a data block but does not terminate the data transfer. Stream channels are always active. To terminate a stream, the *Go to Idle* command must be sent.

RULE B.3.1 :

The data source **SHALL** set the *End* flag in the channel header of the last buffer of data, even if the transfer involves only one buffer of data. The transfer is not complete until the data destination clears the *End* bit to 0.

RULE B.3.2 :

The data destination **SHALL** clear the *End* bit on the successful receipt of the last buffer of data.

RULE B.3.3 :

When a Normal channel data destination successfully receives a buffer of data with the *End* flag set, it **SHALL** clear the *End* bit and then return its channels to the idle state.

RULE B.3.4 :

When an FDC transfer terminates and the channel is returned to the idle state, if the commander owns the FDC area, it **SHALL** return the FDC area to the servant by clearing all of the channel header flags.

B.4 Aborted Data Transfer Termination

Normal FDC transfers may be aborted by setting the *Abort* flag in the channel header. The abort flag may be set only by the owner of the FDC area. The abort generator must pass the FDC area to the other FDC device, the abort receiver, to request that the transfer be aborted. The abort receiver clears the *Abort* flag and terminates the transfer. When the abort generator detects the negation of the *Abort* flag it terminates the transfer. The channels are then idled and the FDC areas are be returned to the servant. Aborting a transfer invalidates the data already transferred.

For FDC channel pairs, the abort generate/acknowledge cycle occurs on one of the two channels. Once the abort is acknowledged, both channels are idled and ownership of the buffers is returned to the servant.

RULE B.4.1 :

The data destination device **SHALL NOT** clear the *End* bit to 0 on receipt of the last buffer of data if the receipt of the last buffer of data is not successful. It **SHALL** instead set the *Abort* flag and pass the FDC area back to the data source to indicate a transfer abort.

RULE B.4.2 :

When a Normal channel pair data destination device receives a buffer of data with the *Abort* flag set, it **SHALL** acknowledge the abort by clearing the *Abort* flag and pass the buffer to the data source device. It then returns both channels of the pair to the idle state.

RULE B.4.3 :

When a Normal transfer which uses a channel pair is aborted, both the data sender and the data receiver **SHALL** return both channels of the pair to the idle state after the abort has been sent and acknowledged.

OBSERVATION B.4.1 :

During a Normal FDC transfer, ownership of the flags and data buffers is passed back and forth between the sender and the receiver. The *Abort* flag can be set only by the current owner of the FDC area. Once the *Abort* flag has been set, ownership must be passed to the other side for acknowledgment.

Aborting Stream Channel Pairs

In a normal stream pair the channels, buffers are loaded and passed in a specific order. First the even buffer is loaded and passed, followed by the odd buffer. Both the data source and data destination must follow this sequence regardless of the *End* or *Abort* flags. In Stream FDC channel pairs, a particular set of actions are specified for the commander and servant to ensure correct sequencing of the channel pair is maintained.

When the *data destination* device rejects a buffer in a stream pair transfer, the following buffer may already have been sent by the data source. In general, it can not be known if the following buffer has been sent. To ensure deterministic operation, the data destination is required to always disregard the buffer following the buffer it aborted. If the data source acknowledges an aborted buffer and it has not yet sent the following buffer, it must send the following buffer. Because this buffer may have no useful purpose except to maintain synchronization, the data source may elect to send an empty buffer. The data destination is not allowed to access or modify the data contents of this buffer.

When the data destination aborts the last buffer in a data block, the following buffer is the first buffer of the next data block. Because this following buffer is returned unmodified, it can be reused if proper synchronization is maintained. When the data destination aborts the last buffer in a data block, it sets the *Abort* bit leaving the *End* bit set and returns the buffer. The data source acknowledges the aborted buffer and ensures the following buffer has been sent. The data destination returns this following buffer unmodified. This returned following buffer is the first buffer of the next data block. It cannot be sent because it is not the next channel in the even/odd sequence. To maintain synchronization, an empty buffer can be sent on the other even/odd channel to maintain the sequence. The returned following buffer can then be resent in proper sequence.

Example: Stream Channel Abort

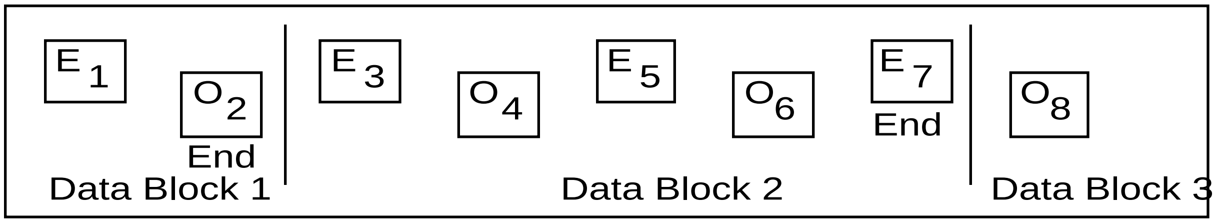


Figure B.4a

In figure B.4a, when the data destination device has received buffer **E5** (even 5) and decides to abort the transfer of data block 2. It sets the *Abort* flag in the channel header of buffer **E5** and returns it to the data source. It must also disregard the next buffer received after **E5** to maintain correct synchronization, in this case **O6** (odd 6).

If the data source has already passed buffer **O6** when it detects the abort, it need not take any further actions. However, if the data source has not sent **O6**, it must send the **O6** buffer to the data destination to ensure synchronization. The buffer may be empty, partially full or a complete buffer of data. The data destination device must pass the **O6** buffer of data back to the data source unmodified.

RULE B.4.4 :

When a Stream FDC channel pair data destination device aborts the transfer it **SHALL**

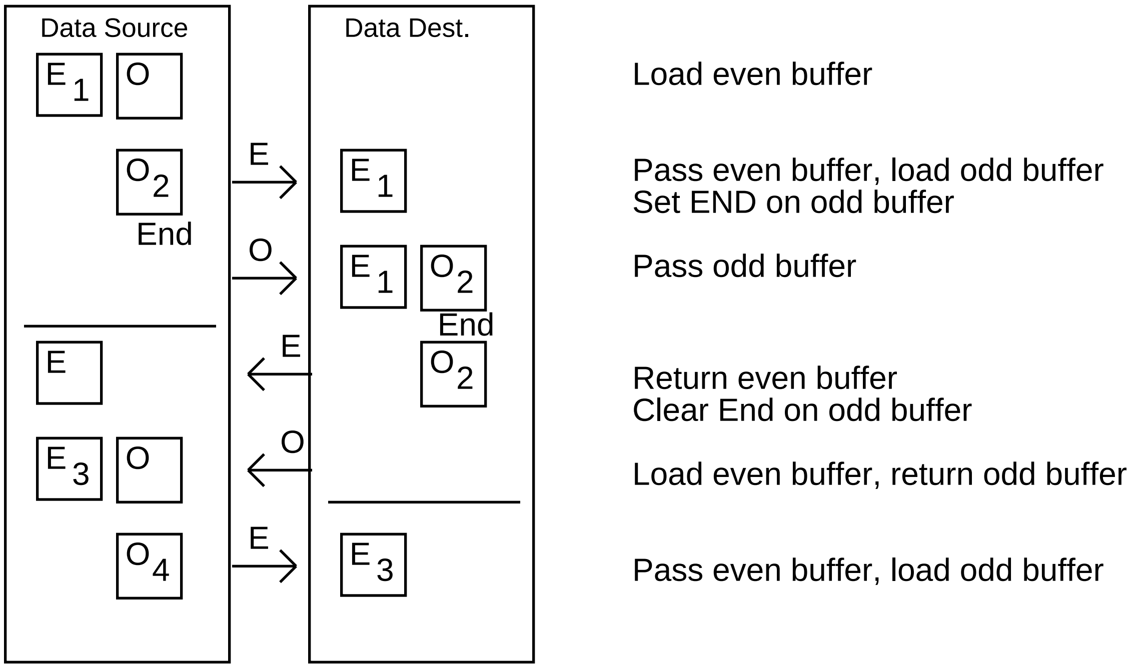
1. set the *Abort* flag in the channel header
2. return the FDC area to the data source
3. return the next FDC buffer received without accessing the data

RULE B.4.5 :

When a Stream FDC channel pair data source receives an FDC area with the *Abort* flag set and the *End* flag cleared and it has not yet sent the next FDC buffer in the current sequence, it **SHALL** send the next FDC buffer in the current sequence. The data in this buffer will not be accessed by the data destination.

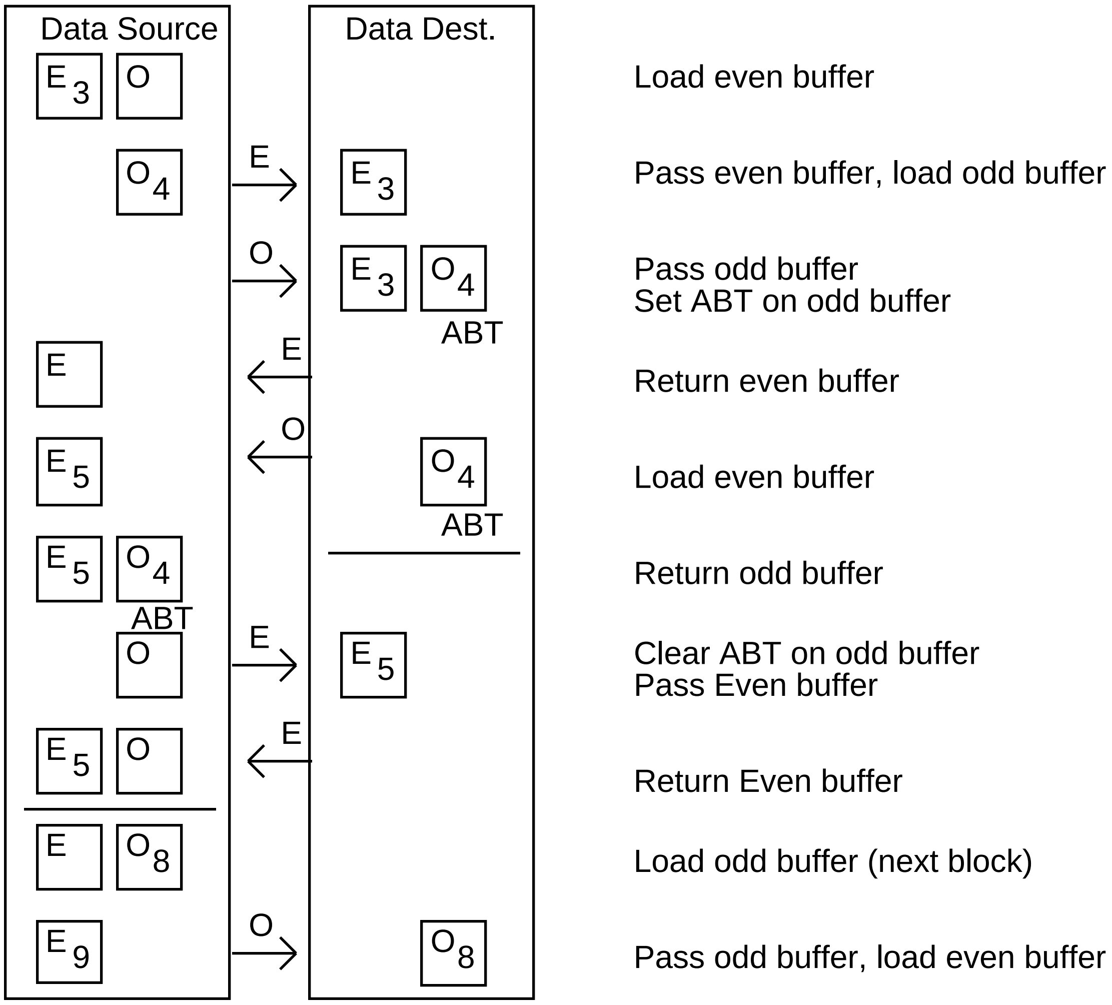
Example: Stream Channel Pair Normal Sequence (no abort)

During an FDC transfer which utilizes the stream pair mode, buffers of data are passed from the data source to the data destination, alternating between the even channel (**E**) and odd channel (**O**). The End flag does not terminate the transfer but indicates the end of a block of data (dashed line).



Example: Stream Channel Pair Aborted by the Data Destination

When a data destination device determines that it must abort the data transfer, it must wait for ownership of one of the FDC buffers. In this example, the data destination sets the Abort flag on **O4** and returns the buffer to the data source. To maintain synchronization, the data destination is required to return the buffer following the aborted buffer without accessing its data contents, in this case **E5**. Buffer **E5** is returned to the data source, completing the abort cycle. A new data block transfer is then started beginning with data buffer 8 which will be transferred on the odd channel (refer to Figure B.4a).

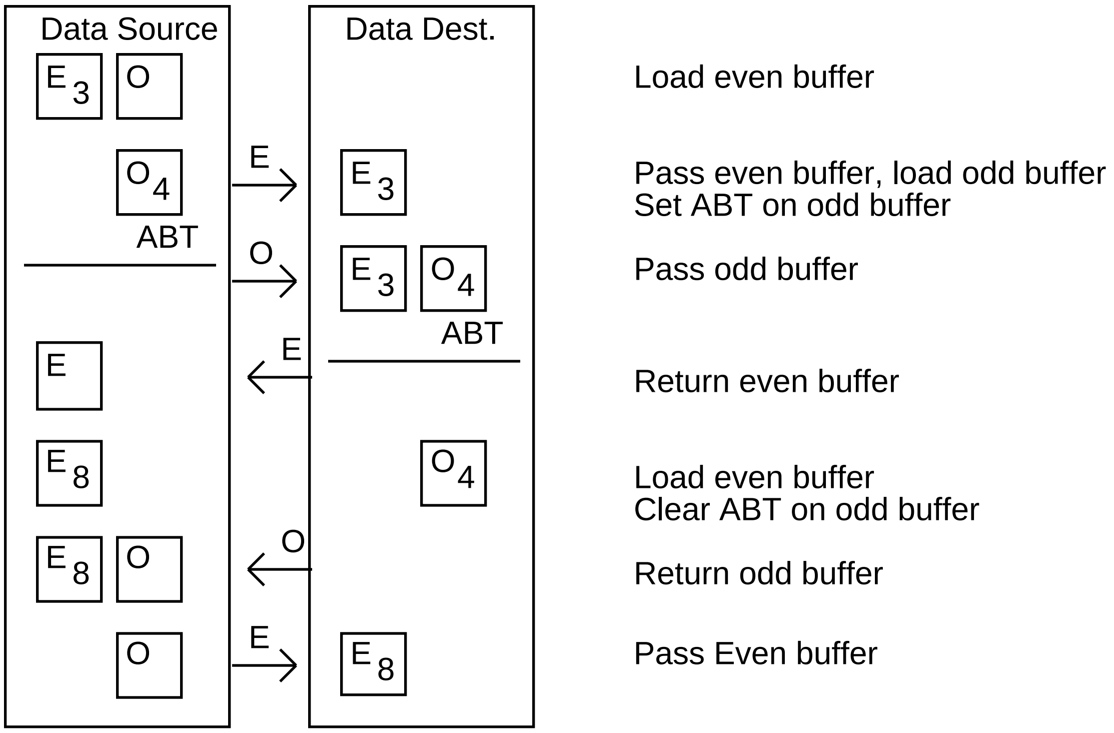


Example: Stream Channel Pair Aborted by the Data Source

When the data source aborts a block transfer, the status of both the even and odd channels are known. The data source is required to follow the even/odd sequence of sending buffers. By definition, when the data source sets the *Abort* flag on a buffer, the following buffer has not yet been sent Because there is no ambiguity, there is no need to send the buffer following the aborted buffer to ensure synchronization. In this case, a normal abort acknowledge cycle is executed on the aborted buffer. A new data block transfer begins on the following buffer.

If the aborted buffer is the last buffer in the data block, the *End* flag may or may not be sent with the *Abort* flag. Because the *Abort* flag takes precedence over the *End* flag, the *End* flag has no significance.

In this example, the data source determines that buffer **O4** should be aborted. It sets the *Abort* flag on **O4** and passes the buffer to the data destination. The data destination acknowledges the *Abort* flag and returns the buffer **O4** to the data source. The data source then begins transferring the next data block using the next channel using the even/odd sequence on **E8** (refer to figure B.4a). In this example the *End* flag is not sent.



B.5 VXIbus Servant Requirements

RULE B.5.6 :

A VXIbus servant which implements the FDC protocol **SHALL** implement all Standard FDC Word Serial Protocol Commands.

OBSERVATION B.5.2 :

A VXIbus servant must implement all commands. However, this does not require that the servant be capable of both sending and receiving data. The *Transfer To Servant* or *Transfer To Commander* command may return a *not supported* response.

PERMISSION B.5.2 :

A VXIbus FDC servant device **MAY** support only the capability to send data or to receive data on a particular channel. The servant is not required to do both.

RULE B.5.7

A VXIbus servant which implements the FDC protocol and indicates support for remote memory in the *FDC Supported* command response **SHALL** implement all FDC Memory Management commands.

RULE B.5.8 :

A VXIbus servant which implements the FDC Extended commands **SHALL** implement all defined extended commands.

B.6 VXIbus Commander Requirements

In addition to the standard FDC commands, a set of Backwards Compatibility (BC) commands is defined. These commands are used by a FDC commander to support servants implementing an earlier version of the FDC protocol. The BC commands are implemented only within some devices which have Manufacturer ID codes of 4092 or 4093. BC commands have the same command codes as the corresponding standard commands except for the value of bit 15. Bit position 15 is a 1 for the standard commands and a 0 for the BC commands.

Similarly, the FDC event has a standard code and a BC code. The standard FDC event has bit 14 set to 1 and the BC event has bit 14 set to 0.

RULE B.6.1 :

Commanders which provide user-accesible FDC services **SHALL** support both the standard and BC command codes and events.

A VXIbus commander may discover support for the FDC protocol by its direct servants by sending the *FDC Supported* Word Serial command. If the device supports FDC it will provide an appropriate response. If FDC is not supported an unsupported command error will be generated by the Word Serial Protocol.

RULE B.6.2 :

A VXIbus commander is required to support both the standard command set and the BC command set. To determine which command set to utilize, the commander **SHALL** first send the standard *FDC Supported* Word Serial command. If the standard command fails and the VXIbus Manufacturer ID is 4092 or 4093, then the BC *FDC Supported* Word Serial command **SHALL** be tested. The success of one of these commands **SHALL** set the command type for all following commands.

RULE B.6.3 :

A VXIbus commander which implements the FDC protocol **SHALL** implement all FDC Memory Management commands.

RULE B.6.4 :

A VXIbus commander which implements the FDC protocol **SHALL NOT** send any FDC Memory Management commands to the servant if the servant does not indicate support for remote memory in its response from the *FDC supported* command.

RULE B.6.5 :

A VXIbus commander which implements the FDC protocol **SHALL NOT** send any FDC Extended commands to the servant if the servant does not indicate support for Extended commands in its response from the *FDC supported* command.

B.7 VXIbus LINEAR and FIFO Buffer Requirements

The Linear and FIFO modes are provided in the FDC protocol to allow hardware implementations of the FDC data buffer which are state machine controlled. This type of interface can provide throughput improvement over software managed data transfer.

To ensure the interoperability of these interfaces the access method and sequencing are restricted to a greater degree than for Random access mode. The smallest unit of information in these modes is a single data register in the FIFO.

RULE B.8.1 :

In FIFO or Linear access modes, the data register size **SHALL** be the largest data width indicated in the *Channel Initialize* DATA field in the response word.

PERMISSION B.8.1 :

A FIFO or Linear data buffer **MAY** allow other data widths of smaller size than the data register size as indicated in the *Channel Initialize* DATA field. These are identified as fractional register data elements.

RULE B.8.2 :

In FIFO or Linear access modes, fractional register data elements **SHALL** be of equal size and **SHALL** be accessed by the FDC commander in ascending order starting at the lowest address. The sum of the partial register data elements **SHALL** be equal to the total register width (example: 2 bytes = 1 word).

RULE B.8.3 :

In FIFO or Linear access modes, the data size indicated in the header **SHALL** be in bytes. The number of commander accesses to the data buffer shall not exceed the number of bytes divided by the register width in bytes.

RULE B.8.4 :

In FIFO or Linear access modes, unless the *End* bit is set in the FDC header, the data size for a buffer of data **SHALL** be an even multiple of the register size.

The buffer of data with the *End* bit set indicates the actual remaining byte count to be transferred in the block allowing any arbitrary number of bytes to be transferred. If the byte count is not an even multiple of the register size, the remainder indicates the number of partial register bytes which are valid within the last FIFO location transferred in this block. These are packed into the lowest address locations within the register in ascending order.

C. Standard Word Serial Commands

Channel Initialize:

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

The syntax of the *Channel Initialize* 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	1	0	Channel #		

Channel # : number 0 - 7 selecting a particular channel

*BC = 0

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: This field indicates status of the *Channel Initialize* command.

F_{16} - The Buffer Initialize command executed with no errors.
7_{16} - ERROR - Channel already open.
6_{16} - ERROR - No valid FDC area can be allocated.
5_{16} - ERROR - This FDC channel number not supported.

ADDR : Supported address space
1 - A16 space
2 - A24 space
4 - A32 space

DATA : Allowed Data Buffer Access Sizes
1 - D08
2 - D16
4 - D32
8 - D64

PC : *Pass Command* Supported
0 - Supported - Commander must send *Pass Command*
1 - Not Supported - Commander must not send *Pass Command*

MODE : Data Buffer mode
  1 - Random access
  2 - Linear access
  4 - FIFO access

The following restrictions apply to the above flags:

1. The ADDR and MODE fields may contain only one of the listed values.
2. The DATA field value may be any bit-wise combination of the listed values. The DATA field value applies to the FDC Data Area only.
3. The FDC header is restricted to D16 accesses.

The servant receiving the *Channel Initialize* command must initialize the FDC header area for the selected channel before returning a success response. The header RSVD, flag bits, and data size should be set to zero. The version number should be set appropriately. It should place the channel in the idle state.

The standard *Channel Initialize* command can only initialize channels 0 through 7. The *Extended Channel Initialize* command must be used to create channels 8 through 65,535.

Channel Address Low: , Channel Address High:

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.

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 #		

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 #		

Channel # : number 0 - 7 selecting a particular channel

*BC = 0

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

Response Word

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 can be allocated, the value returned in the low and high response words should be $HFFFF.

Channel Size Low: , Channel Size High:

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.

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 #		

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 #		

Channel # : number 0 - 7 selecting a particular channel

*BC = 0

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

Response Word

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 allocated, the value returned in the first and second response words should be 0000_{16}.

The channel size defines the total memory allocated to the FDC protocol for this channel. It is the sum of the FDC Channel Header and Channel Data Buffer.

Go to Idle:

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

The syntax of the *Go to Idle* 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	1	1	IM	Channel #		

Channel # : number 0 - 7 selecting a particular channel
IM : Immediate Flag

0 - Transition to idle state at the end of the current block transfer
1 - Transition to idle immediately and re-initialize the buffer header

*BC = 0

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: This flag indicates status of the *Go to Idle* command.

F_{16} - The channel has returned to the idle state.
7_{16} - PENDING - The channel is still active and will be returned to idle at the end of the current block of data.
6_{16} - ERROR - Can't idle a closed channel.
5_{16} - ERROR - This FDC channel number not supported.
4_{16} - ERROR - Command not allowed.

The *Go to Idle* command is used to terminate a stream transfer. If the immediate flag is zero, the current data block transfer should be completed before terminating the stream and returning the channel to the idle state.

For a channel established as transfer to servant, the current block transfer is complete when the servant has successfully accepted a buffer of data with the end bit set and no longer has ownership of the FDC area. For a channel established as transfer to commander, the current block transfer is complete when the servant has successfully transferred a buffer of data with the end bit set to the commander and the servant has ownership of the FDC area.

If the PENDING response is returned, the servant will transition to the idle state at the completion of the current block transfer. To determine that the transfer is complete, the commander may send the *Go to Idle* command as many times as necessary, evaluating the response.

If the *Immediate* flag is set to one, the servant must clear all of the header flags to zero and return the channel to the idle state before returning its response. Once *Go to Idle Immediate* is sent, the controller must not access the channel's FDC area until it requests another transfer.

Sending *Go to Idle* to a normal and active channel has no effect and returns the PENDING response. However, sending *Go to Idle Immediate* will cause the normal channel to terminate in the same manner as a stream channel. This may be done to abort a transfer in progress and will result in loss of data. Sending the *Go to Idle* command to a channel pair causes both channels to transition to the idle state.

Channel Close:

This command is used to close the FDC channel.

The syntax of the *Channel Close* 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	1	1	Channel #		

Channel # : number 0 - 7 selecting a particular channel

*BC = 0

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: This flag indicates status of the *Channel Close* command.

F_{16} - The Channel Close command executed with no errors.
7_{16} - ERROR - Attempt to close a channel that is not open.
6_{16} - ERROR - Attempt to close a channel that is active.

Transfer to Servant:

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

The syntax of the *Transfer to Servant* 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	1	0	PR	ST	Channel #		

Channel # : number 0 - 7 selecting a particular channel

ST : Stream Flag
    0 - Transfer is a Normal data block transfer
    1 - Transfer is a Stream transfer

PR : Channel Pair Flag
    0 - Channel is not to be used as a pair
    1 - Channel is to be used as a pair.

*BC = 0

Note: The PR bit may be set only for channels 0,2,4, and 6. If this bit is set, the corresponding channel pair is made active.

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: This flag indicates status of the Transfer to Commander command.
    F_{16} - No errors detected. Data transfer will commence.
    7_{16} - ERROR - Request to send data with no valid FDC channel.
    6_{16} - ERROR - Request to send data when FDC channel is already active.
    5_{16} - ERROR - PR bit not legal for this channel.
    4_{16} - ERROR - Unable to utilize channel pair.
    3_{16} - ERROR - Unable to receive data for instrument specific reasons.
    2_{16} - ERROR - Unsupported mode (stream, normal or direction)

The *Transfer to Servant* command should clear the *Abort*, *Read Ready*, and *End* bits to zero, and *Write Ready* to one before returning the response. It should transition the channel state to active.

Transfer to Commander:

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

The syntax of the *Transfer to Commander* 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	1	1	PR	ST	Channel #		

Channel # : number 0 - 7 selecting a particular channel

ST : Stream Flag
 0 - Transfer is a normal data block transfer
 1 - Transfer is a Stream transfer

PR : Channel Pair Flag
 0 - Channel is not to be used as a pair
 1 - Channel is to be used as a pair.

*BC = 0

Note: The PR bit may be set only for channels 0,2,4, and 6. If this bit is set, the corresponding channel pair is made active.

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: This flag indicates status of the Transfer to Commander command.
 F_{16} - No errors detected. Data transfer will commence.
 7_{16} - ERROR - Request to send data with no valid FDC channel.
 6_{16} - ERROR - Request to send data when FDC channel is already active.
 5_{16} - ERROR - PR bit not legal for this channel.
 4_{16} - ERROR - Unable to utilize channel pair.
 3_{16} - ERROR - Unable to send data for instrument specific reasons.
 2_{16} - ERROR - Unsupported mode (stream, normal or direction)

The *Transfer to Commander* command should clear the *Abort*, *Read Ready*, *Write Ready*, and *End* bits to zero before returning the response. It should transition the channel state to active.

FDC Event:

This command is used to control FDC event generation.

The syntax of the *FDC Event* 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	1	0	EV	Channel #		

Channel # : number 0 - 7 selecting a particular channel

EV: Event Enable Bit
0 - No events are generated by FDC protocol
1 - The Servant will send an event when it passes the FDC area to the Commander.

*BC = 0

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: This flag indicates status of the Event command.
F_{16} - No errors detected.
7_{16} - ERROR - Events not supported.

Events return a 16 bit value to the commander which uniquely identifies the event. The format of the return value for FDC events is described below.

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							

Channel #: The channel selected by the command
Servants Logical Address: The logical address of this module
*BC = 0

When the FDC event is enabled for a particular FDC channel, the event will be sent to the commander each time FDC buffer ownership is passed back to the commander. FDC event generation is controlled by the *FDC Event* command and is not affected by the VXIbus *Control Event* command. The default state of FDC events is disabled (0).

To ensure correct operation of events, an event/interrupt handler must be installed on the commander. Failure to install the handler may cause poor performance or communication failure.

FDC Supported:

This command is used to determine FDC support.

The syntax of the *FDC Supported* 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	1	1	1	1	1

*BC = 0

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
C7	C6	C5	C4	C3	C2	C1	C0	MP	EX	RM	Min. Rev		Maj. Rev		

Maj. Rev: FDC major revision level

Min. Rev: FDC minor revision level

RM: Remote Memory Supported
- 0 - Remote memory is not supported
- 1 - Remote memory is supported

EX: Extended Commands
- 0 - Not Supported
- 1 - Supported

MP: Message Transfer Protocol
- 0 - Not Supported
- 1 - Supported

C0..C7: Available channels
- 0 - not available
- 1 - available

Enable Passed Buffer:

This command requests that the *Passed Buffer* command be utilized by the servant.

The syntax of the *Enable passed Buffer* 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	1	1	0	0	E

E: Enable Passed Buffer Command = 1, disable = 0

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_{16} - No Errors Detected.
7_{16} - Passed Buffer Command Not Utilized.

This command must be sent to modules which indicate support for the *Passed Buffer* command in their response word to the *Channel Initialize* command. It was not supported on prior versions of the FDC specification. It should not be sent if the servant does not indicate support for the *Passed Buffer* command in the *Channel Initialize* response.

Passed Buffer:

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

The syntax of the *Passed Buffer* 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	1	0	Channel #		

Channel # : number 0 - 7 selecting a particular channel

The *Passed Buffer* command eliminates the requirement for the FDC servant to continuously poll the FDC header bits to determine when the commander passes the FDC buffer. The *Passed Buffer* command should be sent only to servants which have enabled to use the command via the *Enable Passed Buffer* command.

When a FDC servant has enabled the *Passed Buffer* command, the commander is *required* to send the *Passed buffer* command to the servant each time it passes a buffer in the normal manner (after clearing the *Read Ready* or *Write Ready* bits).

The FDC commander must not send the *Passed Buffer* command to servants which do not indicate utilization of the command or which have not been enabled to utilize the command.

D. Memory Management FDC Commands

FDC Protocol utilizes electronic memory which is accessible to both the commander and the servant to share data. This memory may either be *local* or *remote*.

Local memory is owned by the servant. The servant can allocate local memory without regard to other system components. Remote memory is owned or managed by the commander. For the servant to utilize remote memory, the memory must first be granted to the servant. The servant must also verify that it can access the granted memory.

Local Memory

- Dual-Port memory which physically resides on the servant
- A VXIbus memory device which is a direct servant of the FDC servant

Remote Memory

- Dual-Port memory which physically resides on the commander
- A VXIbus memory device which is a direct servant of the FDC commander
- VME memory which is managed by the commander

Memory management FDC commands provide a negotiating mechanism for remote memory. The commander identifies a pool of remote memory which may be granted to the servant with the *Use address*, *Use Size* and *Use Access* commands. The commander then grants that memory to the servant by sending the *Memory Pool Allocate* command. If the servant can utilize the remote memory pool, it accepts the allocation. Otherwise, the allocation fails and the memory pool remains with the commander.

The remote memory pool provided to the servant remains valid until another *Memory Pool Allocate* command is received. If a new remote memory pool is being granted, the prior pool is freed back to the commander, except for memory allocated to existing channels. As these channels are closed, their memory is freed to the commander. The initial remote pool is a NULL pool which has address = 0, size = 0, Access = 0.

The servant allocates memory to FDC channels from its local or remote memory pools. The memory pool to be used by the *Channel Initialize* and *Extended Channel Initialize* commands is selected by the *Memory Pool Select Command*. The selected pool is used until another *Memory Pool Select* command is received. The default memory pool is the local pool.

The memory allocated to each FDC channel is determined by the servant. The *Channel Address* and *Channel Size* commands return the actual memory allocated to each channel from the pool. If the pool of memory is exhausted, further *Channel Initialize* commands will fail.

Memory Management commands are implemented as standard and Longword Serial commands. Standard Word Serial commands are 16 bits in length. Longword Serial commands are 32 bits in length.

Memory Pool Select:

This command selects the memory pool from which the servant may allocate memory during the *Channel Initialize* and *Extended Channel Initialize* commands. The selection is between the local and remote memory pools.

The syntax of the *Memory Pool Select* 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	1	1	0	1	RM

RM : Selected Memory Pool
- 0 - local memory
- 1 - remote memory

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_{16} - No Errors Detected
7_{16} - Unable to switch to remote memory
6_{16} - Unable to switch to local memory
5_{16} - No remote memory pool allocated

The default value if this command is not sent is local memory. Local memory is memory which is owned by the servant. Remote memory is memory which is owned by the commander. To use remote memory a memory pool must first be allocated for use and accepted by the servant.

Memory Pool Allocate:

This command allocates a memory pool identified by the *Use Address*, *Use Size* and *Use Access* commands.

The syntax of the *Memory Pool Allocate* 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	1	1	1	0	AP

AP : Allocate Identified Memory Pool
0 - De-allocate current remote memory pool
1 - allocate identified remote memory pool

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_{16} - No Errors Detected
7_{16} - ERROR - Unable to utilize remote memory
6_{16} - ERROR - Unable to utilize address access mode
6_{16} - ERROR - Unable to utilize data access mode
6_{16} - ERROR - Use Memory Pool Size = 0

To utilize remote memory, a memory pool must be identified with the *Use Address*, *Use Size* and *Use Access* commands. The Memory Pool Allocate command is then sent to the servant. Successful execution of this command grants the memory pool to the servant for allocation to FDC channels.

The servant can have only one remote memory pool at a time. Any prior pool is freed back to the commander when the new pool is accepted. Existing FDC channel areas are not affected by this command.

This command can also de-allocate the current remote memory pool. The remote memory pool size and address are returned to the default values of 0.

Use Address:

This command is used to specify the address of the remote memory pool. Both the low and high values should be sent to set the complete 32 bit address value.

The syntax of the *Use Address* command is defined in the following table.

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	HI

Bit #

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

HI: Set High or Low word of 32 bit address
- 0 - Set Low Word, bits 15..0
- 1 - Set High Word, bits 31..16

size: value to set

There is no response to this command. It sets a software register value which is used later by the *Memory Pool Allocate* command.

Use Size:

This command is used to specify the size of a remote memory pool. Both the low and high values should be sent to set the complete 32 bit size value.

The syntax of the *Use Size command* are defined in the following tables.

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	HI

Bit #

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

HI: Set High or Low word of 32 bit size
    0 - Set Low Word, bits 15..0
    1 - Set High Word, bits 31..16

size: value to set

There is no response to this command. It sets a software register value which is used later by the *Memory Pool Allocate* command.

Use Access:

This command is used to specify the type of access for the remote memory pool.

The syntax of the *Use Access* command is defined in the following table.

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	0	A 32	A 24	A 16	D 64	D 32	D 16	D 8

Only one address type may be selected. Other types must be disabled. Any combination of data types may be selected.

0 - disabled
1 - selected

There is no response to this command. It sets a software register value which is used later by the *Memory Pool Allocate* command.

E. Extended FDC Commands

Extended FDC commands utilize VXIbus reserved Longword Serial commands. The extended commands support up to 2^{16} channels. Extended channel data transfer management commands are not allowed to be sent to channels 0 - 7.

The extended FDC commands are implemented as Longword serial commands which contain 32 bits. If the VXIbus Data High and Data Low registers are written 16 bits at a time, the register values which contain bits 31-16 are first written to the Data High register. Next, Register values which contain bits 15-0 are next written to the Data Low register. Both registers may be written simultaneously with a 32 bit access. These registers are read with the same sequence restrictions.

Extended Channel Initialize:

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

The syntax of the *Extended Channel Initialize* command is defined in the following table.

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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: This flag indicates status of the *Extended Channel Initialize* command.

F_{16} - The Buffer Initialize command executed with no errors.
7_{16} - ERROR - Channel already open.
6_{16} - ERROR - No valid FDC area can be allocated.
5_{16} - ERROR - This FDC channel number not supported.
4_{16} - ERROR - Unable to use external memory

See standard *Channel Initialize* command for field definitions.

Extended Channel Address:

This command is used to retrieve the FDC area base address from the servant.

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

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bit #

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

A single response data long word is placed in the Data Hi and Low registers in the following format:

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FDC Area Address Hi Word															

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FDC Area Address Lo Word															

If no valid FDC area can be allocated, the value returned in the response word should be $HFFFFFFFF.

Extended Channel Size:

This command is used to retrieve the FDC area size.

The syntax of the *Extended Channel Size* command is defined in the following table:

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Bit #

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

A single response data long word is placed in the Data Hi and Low registers in the following format:

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FDC Area Size Hi Word															

Bit #

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FDC Area Size Lo Word															

If no valid FDC area can be allocated, the value returned in the first and second response words should be 00000000_{16}.

The channel size is defines the total memory allocated to the FDC protocol. It is the sum of the FDC Channel Header and Channel Data Buffer.

Extended Go to Idle:

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

The syntax of the *Extended Go to Idle* command is defined in the following table:

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	IM

Bit #

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

IM : Immediate Flag

0 - Transition to idle state at the end of the current block transfer
1 - Transition to idle immediately and re-initialize the buffer header

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

See standard command *Go To Idle* for description

Extended Channel Close:

This command is used to close an Extended FDC channel.

The syntax of the *Extended Channel Close* command is defined in the following table:

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Bit #

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

See standard *Channel Close* command for description

Extended Transfer to Servant:

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

The syntax of the *Extended Transfer to Servant* command is defined in the following table:

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	0	1	0	PR	ST

Bit #

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

ST : Stream Flag
    0 - Transfer is a Normal data block transfer
    1 - Transfer is a Stream transfer

PR : Channel Pair Flag
    0 - Channel is not to be used as a pair
    1 - Channel is to be used as a pair.

Note: The PR bit may be set only for even channels 8,10,12,... If this bit is set, the corresponding channel pair is made active.

A single response long word is placed in the Data Hi and Low registers 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

See standard *Transfer to Servant* command for description

Extended Transfer to Commander:

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

The syntax of the *Extended Transfer to Commander* command is defined in the following table.

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	0	1	1	PR	ST

Bit #

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

ST : Stream Flag
- 0 - Transfer is a normal data block transfer
- 1 - Transfer is a Stream transfer

PR : Channel Pair Flag
- 0 - Channel is not to be used as a pair
- 1 - Channel is to be used as a pair.

Note: The PR bit may be set only for even channels 8,10,12,... If this bit is set, the corresponding channel pair is made active.

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

See standard *Transfer to Commander* command for description

Extended FDC Event:

This command is used to control FDC event generation.

The syntax of the *Extended FDC Event* command is defined in the following table.

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	EV

Bit #

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

EV: Event Enable Bit

0 - No events are generated by FDC protocol

1 - The Servant will send an event when it passes the FDC area to the Commander.

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: This flag indicates status of the Event command.

F_{16} - No errors detected.

7_{16} - ERROR - Events not supported.

Events return a 16 bit value to the commander which uniquely identifies the event. The format of the return value for FDC events is described below.

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

Extended FDC events do not include the extended channel number as part of the event. To determine the source channel of the event the *Extended FDC Event Query* command must be sent to the servant. The servant should provide an Extended FDC event queue. Each channel may generate a single event so the queue must be able to accommodate as many events as it allows channels.

Extended FDC Event Query:

This command is used to identify the channel source of an extended event.

The syntax of the *Extended FDC Event Query* command is defined in the following table.

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Bit #

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

A single response long word is placed in the Data Hi and Low registers in the following format:

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Status				1	1	1	1	1	1	1	1	1	1	1	1

Bit #

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

Status: This flag indicates status of the Event command.

F_{16} - No errors detected.
7_{16} - ERROR - Not a valid channel.
6_{16} - ERROR - Events not enabled for this channel.
5_{16} - ERROR - No events in the Extended FDC Event Que.

When a commander requests extended FDC events from the servant, it should continue to request additional events until the "No events in the Extended FDC Event Queue" error is encountered to ensure all events are delivered.

Extended Passed Buffer:

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

The syntax of the *Extended Passed Buffer* command is defined in the following table.

Bit #

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Bit #

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

The *Extended Passed Buffer* command must be sent to the FDC servant if the servant indicates support for the passed buffer command in the response from the *Extended Channel Initialize* command. The *Passed Buffer* command is sent each time the FDC area is passed to the servant. The *Write Ready* or *Read Ready* flag is first cleared and then the command is sent.

F. Examples

F.1 Channel Initialization Procedure

Only the VXIbus commander can initiate a FDC channel, the VXIbus servant can not initiate a FDC channel.

FDC channel initialization is used by FDC devices to establish a channel between a VXIbus commander and its servant. The FDC channel establishment commands are: *Channel Address*, *Channel Size*, and *Channel Initialize*. These commands setup the FDC channel.

The FDC channel initialization is described in the following table.

VXIbus Servant	VXIbus Commander
	Send *Channel Initialize* command
Return response word	
	Send *Channel Address* command.
Return the FDC area base address	
	Send *Channel Size* command
Return the FDC area size	
If no errors have been reported a valid channel exists.	If no errors have been reported a valid channel exists.

F.2 Data transfer from Commander to Servant

The following table describes how blocks of data are transferred from a VXIbus commander to its servant. It is assumed that the FDC channel has been initialized and is in the idle state. A normal transfer will be requested.

VXIbus Servant	VXIbus Commander
	Send *Transfer to Servant* command.
Receive *Transfer to Servant command*. If no error set *Write Ready* bit to 1, set *Abort*, *Read Ready*, and *End* to 0. Return the response value. If error, terminate the transfer and return to the idle state.	
	Get the response from *Transfer to Servant* command. If error, terminate the transfer and return to the idle state.
	Wait for *Write Ready* to become 1 Check the *Abort* bit Put data into FDC data buffer Set FDC data size Set *Write Ready* bit 0
Wait for *Write Ready* bit to become 0 Check the *Abort* and *End* bits Read data from FDC data buffer Set *Write Ready* bit to 1	
. . .	. . .
	Wait for *Write Ready* to become 1 Check the *Abort* bit Put last data block to FDC data buffer Set FDC data size **Set *End* bit to 1 for last data block** Set *Write Ready* bit 0
Wait for *Write Ready* bit to become 0 Check the *Abort* and ***End*** bits Read last data block from FDC data buffer Set *End* bit to 0	
Return to idle state	
	Wait for *End* bit to become 0 Return to idle state

During a transfer to servant, after the VXIbus commander has sent the last data buffer to the servant and has set the *End* bit, the commander must wait until the VXIbus servant clears the *End* bit back to 0, then the VXIbus commander will return to idle state for normal transfers. When the servant clears the *End* bit to 0, it will return to idle state for normal transfers.

F.3 Data transfer from Servant to Commander

The following table describes the data transfer from VXIbus servant to VXIbus commander. It is assumed that the FDC channel has been initialized. A normal transfer will be requested.

VXIbus Servant	VXIbus Commander
	Send *Transfer to Commander* command.
Receive *Transfer to Commander* command. Set *End*, *Abort*, *Read Ready*, and *Write Ready* to 0 then return response value. If error, terminate the transfer and return to the idle state.	
	Get the response from *Transfer to Commander* command. If error, terminate the transfer and return to the idle state.
Put a buffer of data into the FDC data buffer Set the FDC data size Set *Read Ready* bit to 1	
	Wait for *Read Ready* to become 1 Check the *Abort* and *End* bits Read data from FDC data buffer Set *Read Ready* to 0
. . .	. . .
Wait for *Read Ready* to become 0 Check the *Abort* bit Put the last data block to FDC data buffer Set the FDC data size **Set *End* bit to 1 for last data block** Set *Read Ready* bit to 1	
	Wait for *Read Ready* to become 1 Check the *Abort* and ***End*** bits Read last data block from FDC data buffer Set *Read Ready* and *End* to 0
	Return to idle state
Wait for *End* to become 0 Return to idle state	

During a transfer to commander, when the VXIbus servant has sent the last data buffer to the VXIbus commander, the servant will wait until the VXIbus commander clears the *Read Ready* and *End* bit back to 0 before returning to idle state during a normal transfer. When the VXIbus commander clears the *End* and *Read Ready* bit to 0, the commander will return to idle state during a Normal transfer.

F.4 Transfer Data using a Channel Pair

The following table describes how blocks of data are transferred from a VXIbus commander to its servant using Channel Pair 0 and 1. It is assumed that FDC channels 0 and 1 have been initialized and they are in the idle state.

VXIbus Servant ch0	VXIbus Servant ch1	VXIbus Commander ch0	VXIbus Commander ch1
		Send *Transfer to Servant* command to ch0 with PR bit set	
Receive *Transfer to Servant* command with PR bit set. If no error set *Abort*, *Read Ready*, and *End* to 0 and *Write Ready* to one for both ch0 and ch1. Return response value. If error, terminate the transfer and return both channels to the idle state.			
		Get the response from *Transfer to Servant* command. If error, terminate the transfer and return both channels to the idle state.	
		Wait for *Write Ready* = 1 Check *Abort* bit Put data into FDC area Set *Write Ready* bit to 0	
Wait for *Write Ready* = 0 Check *Abort* and *End* bits. Read data from FDC area Set *Write Ready* to 1			Wait for *Write Ready* = 1 Check *Abort* bit Put data into FDC area Set *Write Ready* bit to 0
	Wait for *Write Ready* = 0 Check *Abort* and *End* bits. Read data from FDC area. Set *Write Ready* to 1	Wait for *Write Ready* = 1 Check *Abort* bit Put data into FDC area Set *Write Ready* bit to 0	
Wait for *Write Ready* = 0 Check *Abort* and *End* bits. Read data from FDC area Set *Write Ready* to 1			Wait for *Write Ready* = 1 Check *Abort* bit Put data into FDC area Set *Write Ready* bit to 0
. . .	. . .	. . .	. . .
	Wait for *Write Ready* = 0 Check *Abort* and *End* bits. Read data from FDC area Set *Write Ready* to 1	Wait for *Write Ready* = 1 Check *Abort* bit Put data into FDC area Set *Write Ready* bit to 0 Set *End* bit to 1	
Wait for *Write Ready* = 0 Check *Abort* and *End* bits. Read last data from FDC area Set *End* to 0			

Return to Idle	Return to Idle	Wait for *End* = 0	
		Return to Idle	Return to Idle

The *End* bit is set by the commander when the last buffer of data is transferred. When the *End* bit is received by the servant, both channels of the pair must go to the idle state even though no explicit termination semaphore is received for the second channel. When the *End* acknowledge is returned, both of the commander channels transition to the idle state.

F.5 Stream Data transfer from Commander to Servant

The following table describes how blocks of data are transferred from a VXIbus commander to its servant using streams. It is assumed that the FDC channel has been initialized and is in the idle state. A stream transfer will be requested.

VXIbus Servant	VXIbus Commander
	Send *Transfer to Servant* command.
Receive *Transfer to Servant* command. If no error set *Write Ready* bit to 1, set *Abort*, *Read Ready*, and *End* to 0. Return the response value. If error, terminate the transfer and return to the idle state.	
	Get the response from *Transfer to Servant* command. If error, terminate the transfer and return to the idle state.
	Wait for *Write Ready* to become 1 Check the *Abort* bit Put data into FDC data buffer Set FDC data size Set *Write Ready* bit 0
Wait for *Write Ready* bit to become 0 Check the *Abort* bit Read data from FDC data buffer Set *Write Ready* bit to 1	
. . .	. . .

NOTE: A Stream channel operates identically to a normal channel except that the *End* bit does not terminate the data transfer. The *End* bit is used to separate blocks of data in the same way as the GPIB EOI or CR-LF. Once the sender sets the *End* bit, the receiver is required to acknowledge it by clearing the *End* bit. Blocking of data in streams is dependent on the application and is not specified beyond the requirement to acknowledge. To terminate a stream, the *Go to Idle* command is used.

F.6 Abort Flag Usage

It is assumed that a normal transfer is requested.

Error Generator	Error Detector
Wait for ownership of the FDC buffer	
Set *Abort* bit to 1 Pass FDC area to error detector with *Read Ready* or *Write Ready*	
	Poll *Read Ready* or *Write Ready* for buffer ownership transfer Check *Abort* bit for 1 If *Abort* = 1 then set *Abort* to 0. If this is a Normal channel also set *Read Ready*, *Write Ready* and *End* bits to 0
Wait for *Abort* bit to become 0 Return to idle state Report error through normal instrument specific mechanisms	Return to idle state

The FDC transfer can be aborted by either the VXIbus servant or VXIbus commander by setting the *Abort* bit to 1 during a normal transfer. When the VXIbus servant or VXIbus commander is passed the FDC area, it should check the *Abort* bit to determine whether an error has occurred. Once the *Error Generator* has set the *Abort* bit to 1, it is required to pass the buffer to the Error Detector. The *Error Detector* acknowledges detection of *Abort* bit by clearing it to 0. This completes the abort cycle.

G. Message Transfer Protocol

G.1 Introduction

Message transfer protocol (MTP) provides an alternate mechanism for transporting commands and responses into and out of VXIbus modules. It does this by standardizing the usage of an input and an output FDC channel for communication. Modules which support MTP switch from Byte Transfer Protocol (BTP) to MTP when the *MTP Activate* Word Serial command is successfully executed.

After MTP is activated, communication which would normally have used Byte Transfer Protocol is directed through MTP. If the MTP channel is terminated, the module switches back to Byte Transfer Protocol. MTP provides a more efficient mechanism to communicate messages within a VXIbus system.

MTP utilizes two FDC channels. Channel 4 is defined as the command channel over which instrument commands are sent. The command channel utilizes a Stream transfer to servant FDC channel. Channel 6 is defined as the response channel over which instrument responses are sent. The response channel utilizes a Normal transfer to commander FDC Channel. MTP may also use channel pairs. When channel pairs are used the command channel is on channels 4 and 5, the response channel is on channels 6 and 7.

G.2 MTP Establishment and Termination

VXIbus servant support for Message Transfer Protocol is indicated by the response from the *FDC supported* command. MTP can be established using either local memory channels or remote memory channels.

MTP is established by initializing the MTP channels and then sending the *MTP Activate* command to the servant. The *MTP Activate* command requests that the servant disable Byte Transfer Protocol (DIR & DOR = 0), initialize the MTP FDC channels and activate MTP communications.

The response word from *MTP Activate* informs the commander of the servants success. If the servant is successful, the commander initializes its MTP FDC interface and routes any pending or future messages through MTP. If MTP establishment fails BTP remains enabled and the MTP channels are closed.

MTP termination returns the communications back to Byte Transfer Protocol. It occurs when the servant receives and successfully executes a *MTP Terminate* command. If successful, the servant re-enables the Byte Transfer Protocol, closes the MTP channels, and returns a success response.

The MTP terminate will fail if there is output data which has been delivered to the commander but has not yet been completely read. The Word Serial *Clear* command may be sent to ensure that there is no pending output data before the *MTP Terminate* command is sent.

G.3 MTP Communication

Once MTP is established, all commands and responses are communicated through MTP. All VXIbus Word serial commands are functional except the *Byte Available* (BAV), *Byte Request* (BRQ) and the *Trigger* command. Trigger commands are sent using the *MTP Trigger* command.

The Word Serial *Clear* command performs the same interface functions in MTP as it does in BTP. However, the *Clear* command also causes any pending output in the MTP FDC buffers to be discarded. The MTP servant must return ownership of the FDC area(s) for the command channel back to the commander. The MTP driver on the commander must return ownership of the response channel FDC areas to the servant. This returns the interface to a ready state

Transferring instrument commands to the servant is accomplished in the normal manner for stream channels. The commander places a message in the FDC buffer, sets the data size in the channel header, sets the *End* bit appropriately and passes the FDC area to the servant. The servant retrieves this information from the FDC area and passes the FDC area back to the commander.

The MTP response channel utilizes a Normal FDC *Transfer to Commander* command to retrieve servant response data. When the commander requests a response, the *Transfer to Commander* command is sent on the MTP response channel. This makes the response channel active and allows the servant to pass buffers of data to the commander until the entire response is retrieved. The Transfer terminates normally and the response channel returns to the idle state.

Because the VXIbus *Trigger* command must be sent synchronously to the command stream, MTP defines a trigger command which is sent through the MTP command channel. The *MTP trigger* command is sent by setting the *trigger* bit in the FDC header to a one in the command channel and passing the FDC area to the servant. The data buffer contents and size parameters have no meaning in the *MTP Trigger* command. The servant executes the trigger command and passes the command channel FDC area back to the commander. The commander must ensure that the trigger bit is cleared before sending subsequent commands.

G.4 MTP Error Conditions

The commander should maintain a time-out timer on both the command and response channels. The command channel might time out if the servant has not returned the FDC area within the time-out period. The response channel might time out if the response FDC area is not passed to the commander within the time-out period.

The *Abort* bit in the MTP FDC Header may not be set by either the commander or the servant on the MTP command channel. The servant may set the *Abort* bit on the MTP response channel in order to discard its output buffer. This may occur if a multiple query error is detected or a Word Serial *Clear* command is received.

If the commander requests a response when no servant query has generated any response data, the MTP servant returns a NULL response to the commander. A NULL response is a single FDC buffer with a zero buffer size and the *End* bit set in the FDC channel header. The commander should interpret a NULL response as a IEEE 488 talked with nothing to say error.

G.5 Automatic MTP Activation

Controllers which support FDC and MTP protocol may automatically activate MTP communications at system initialization. MTP servant support can be determined by executing the FDC Supported command. If MTP is supported the commander can request activation of MTP any time after the servant enters the VXIbus NORMAL Operation state.

MTP activation may also be requested by the application program or instrument driver. An MTP commander should respect the request to activate MTP to ensure the highest level of perceived interoperability.

G.6 MTP General Requirements

RULE G.6.1

MTP **SHALL** utilize only the Random mode of access for the FDC buffer area.

The Word Serial *Clear* command may be used to return the MTP interface to a ready state. The *Clear* command discards any pending commands or responses and returns the MTP FDC areas to their data sources.

RULE G.6.2

The *Abort* bit in the FDC header of an MTP command channel **SHALL NOT** be set for any reason.

RULE G.6.3

The *Abort* bit in the FDC header of an MTP response channel **SHALL NOT** be set by the commander.

To ensure that the VXIbus servant has been correctly initialized prior to entering MTP, MTP activation and termination is restricted to the NORMAL Operation state. When an instrument transitions to the CONFIGURE state, I/O buffers are flushed and MTP terminates. When the instrument is returned to the NORMAL state, Byte Transfer Protocol will be active.

RULE G.6.4

The transition between MTP and Byte Transfer Protocol **SHALL** only occur when the VXIbus device is in the NORMAL Operation state. MTP **SHALL** terminate on entry of the CONFIGURE state.

G.7 MTP Commanders

MTP commanders are required to provide support for all MTP modes. This includes support for local and remote memory and single and paired channels. This ensures that MTP compliant components provide a high level of interoperability.

RULE G.7.5

MTP commanders **SHALL**

1. support single channel MTP control and response channels
2. support paired channel MTP control and response channels
3. support local memory MTP control and response channels
4. support remote memory MTP control and response channels

MTP commanders must ensure that the MTP channels are initialized before sending the *MTP Activate* command. MTP channels are established utilizing either local memory or remote memory. The *MTP Supported* command indicates which type of memory must be used. If remote memory is used, the requested size of the command and response channel buffers are also indicated. If local memory is used, the servant allocates the appropriate memory from its own memory pool.

RULE G.7.6

An MTP commander **SHALL** initialize FDC channels 4 and 6 before sending the *MTP Activate* command. If MTP channel pairs are used, the commander **SHALL** also initialize the FDC channels 5 and 7 before sending the *MTP Activate* command.

RULE G.7.7

A commander **SHALL NOT** send the *MTP Activate* command to the servant if the VXIbus Response Register *Read Ready* bit is asserted.

OBSERVATION G.7.1

While MTP is active, the VXIbus response register DIR and DOR bits remain false. The commander is not allowed to send the Byte Transfer Protocol commands BAV, BRQ or the Word Serial command Trigger to the servant.

RULE G.7.8

While MTP is active, a commander **SHALL NOT** send the FDC *Goto Idle* command to channels engaged in MTP protocol.

RULE G.7.9

A commander engaged in MTP **SHALL** send commands only on channel 4 or channel 4 and 5 for channel pairs.

G.8 MTP Servants

To provide a basic level of functionality within MTP, a minimum level of support is required of the MTP servant. This ensures that the commander will be successful in switching from Byte Transfer Protocol to MTP.

RULE G.8.10

If a servant claims conformance to MTP, it **SHALL NOT** utilize the MTP command and response channels for any other purpose. If the device supports MTP channel pairs then channels 4,5,6 and 7 are reserved. If channel pairs are not supported then channels 4 and 6 are reserved.

RULE G.8.11

MTP servants **SHALL**

1. support single channel MTP control and response channels
2. support either local memory or remote memory MTP control and response channels

The order of actions taken by the MTP servant as it transitions from Byte Transfer Protocol to MTP is defined to ensure the lowest chance of communication error.

RULE G.8.12

When a MTP servant executes the *MTP Activate* command, it **SHALL**

1. transition the MTP command channels to stream transfer to servant and pass ownership of the MTP command channels FDC areas to the commander
2. Ensure that the MTP response channels are initialized and idle
3. If steps 1 or 2 failed, return an error status
4. un-assert the DIR and DOR bits in the VXIbus response register and disable Byte Transfer Protocol
5. Return a success status

RULE G.8.13

When a servant receives a Word Serial *Clear* command, it **SHALL** return the MTP FDC area(s) for the command channel to the commander. It **SHALL NOT** set the VXIbus response register DIR bit to 1.

To send a Trigger command, the MTP commander sets the *Trigger* bit in the MTP command channel FDC header and passes the FDC area to the servant. The servant must check the *Trigger* bit each time it receives a command channel FDC area. When the *Trigger* bit is set, the buffer size and buffer data have no meaning and must be ignored. The commander must ensure the *Trigger* bit in the FDC header is correctly set for subsequent communications.

RULE G.8.14

When a MTP servant receives ownership of a command channel FDC area, it **SHALL** check the *Trigger* bit in the FDC header. If the *Trigger* bit is set, the servant **SHALL** execute the VXIbus trigger function as defined by the Word Serial *Trigger* command, clear the *Trigger* bit and pass the FDC area back to the commander.

The MTP Terminate command returns the command and response channels back to the Byte Transfer protocol. To succeed, there must not be any response data which has not been read by the commander. The servant can not recover this data so it would be lost.

RULE G.8.15

When a MTP servant receives the *MTP Terminate* command it **SHALL**

1. if it does not own all response channel FDC areas return a failed response
2. enable Byte Transfer Protocol setting DIR and DOR appropriately
3. ensure that any pending command or response data is transferred to the Byte Transfer Protocol interface
4. close all channels which were allocated to the MTP
5. return a success response

H. MTP Commands

MTP Supported:

This command is used to determine MTP support.

The syntax of the *MTP Supported* 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	1	0	0	1	0	0

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				Rsp Buf Size				Cmd Buf Size				1	1	PR	RM

RM: Remote Memory
- 0 - Use local memory only
- 1 - Use remote memory only

PR: Channel Pairs
- 0 - Channel Pairs are NOT supported
- 1 - Channel Pairs are supported

Cmd Buf Size: Requested remote MTP command data buffer size

Rsp Buf Size: Requested remote MTP response data buffer size

The requested size fields are only valid when the RM bit is set to 1. These fields are provided to advise the commander of the minimum memory pool that should be offered during the remote memory negotiation. The commander is not required to respect the request however.

The size is calculated as follows:

$1024 * 2^{size} + 8$

For example: if size = 2 then the requested size for the FDC area is 4104 bytes.

Status: This flag indicates status of the MTP Supported command.

F_{16} - MTP supported.
7_{16} - ERROR - MTP not supported.

MTP Activate:

This command is used to initiate MTP transferring communications from Byte Transfer Protocol to Message Transfer Protocol.

The syntax of the *MTP Activate* 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	1	0	0	0	0	PR

PR : use channel pairs

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: This flag indicates status of the MTP Initiate command.
F_{16} - No errors detected.
7_{16} - ERROR - remote memory not supported.
6_{16} - ERROR - pairs not supported.
5_{16} - ERROR - MTP channels not available
4_{16} - ERROR - MTP already active.

Channels used by MTP must be initialized before the *MTP Activate* command is sent. Channels 4 and 6 must always be initialized. Channels 5 and 7 must be initialized if the PR bit is set to 1.

MTP Terminate:

This command is used to terminate MTP returning communications to Byte Transfer Protocol.

The syntax of the *MTP Terminate* 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	1	0	0	1	0	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
Status				1	1	1	1	1	1	1	1	1	1	1	1

Status: This flag indicates status of the MTP Terminate command.
F_{16} - MTP successfully terminated
7_{16} - ERROR - Pending output data
6_{16} - ERROR - MTP not currently active

MTP Terminate is the only provided method for returning communication to Byte Transfer protocol. *MTP Terminate* will fail if there is pending output data. Pending output data is data which has been sent to the commander which has not been completely read. The indicator of pending output data is one of the MTP response FDC areas is owned by the commander. Once the commander reads all of the data it must return ownership of the FDC area to the servant.

Pending output data can be cleared by reading the remaining response data or sending a Word Serial *Clear* command.