USER MANUAL

6075

4 / 8 Channel Serial Communications Module

PUBLICATION NO. 980870

RACAL INSTRUMENTS

Racal Instruments, Inc.
4 Goodyear St., Irvine, CA 92618-2002
Tel: (800) RACAL-ATE, (800) 722-2528, (949) 859-8999;
FAX: (949) 859-7139

Racal Instruments, Ltd.
480 Bath Road, Slough, Berkshire, SL1 6BE, United Kingdom
Tel: +44 (0) 1628 604455; FAX: +44 (0) 1628 662017

Racal Systems Electronique S.A.
18 Avenue Dutartre, 78150 LeChesnay, France
Tel: +33 (1) 3923 2222; FAX: +33 (1) 3923 2225

Racal Systems Elettronica s.r.l.
Strada 2-Palazzo C4, 20090 Milanofiori Assago, Milan, Italy
Tel: +39 (0)2 5750 1796; FAX +39 (0)2 5750 1828

Racal Elektronik System GmbH.
Technologiepark Bergisch Gladbach, Friedrich-Ebert-Strasse,
D-51429 Bergisch Gladbach, Germany
Tel.: +49 2204 8442 00; FAX: +49 2204 8442 19

Racal Instruments, Ltd.
Unit 5, 25F., Mega Trade Center, No 1, Mei Wan Road, Tsuen Wan,
Hong Kong, PRC Tel: +852 2405 5500, FAX: +852 2416 4335

http://www.racalinstruments.com

PUBLICATION DATE: February 20, 2003

THANK YOU FOR PURCHASING THIS RACAL INSTRUMENTS PRODUCT.

For this product, or any other Racal Instruments product that incorporates software drivers, you may access our web site to verify and/or download the latest driver versions. The web address for driver downloads is:

http://www.racalinstruments.com/downloads

You will be asked to register one time only to gain access to the driver and product manual downloads sections. At registration a cookie will be placed on your computer if you choose to accept it. This is done to facilitate your use of these sections on future visits. You may refuse to accept the cookie and still have complete access to the software driver database but will have to re-register every time you visit. This cookie is for ease of use only and no information is gathered for, sold, or reported to, any third party organization.

If you have any questions about software driver downloads or our privacy policy, please contact us at info@racalinstruments.com.

WARRANTY STATEMENT

All Racal Instruments, Inc. products are designed and manufactured to exacting standards and in full conformance to Racal's ISO 9001 procedures.

For the specific terms of your standard warranty, or optional extended warranty or service agreement, contact your Racal customer service advisor. Please have the following information available to facilitate service.

1. Product serial number
2. Product model number
3. Your company and contact information

You may contact your customer service advisor by:

E-Mail:	Helpdesk@racalinstruments.com	
Telephone:	+1 800 722 3262	(USA)
	+44(0) 8706 080134	(UK)
	+852 2405 5500	(Hong Kong)
Fax:	+1 949 859 7309	(USA)
	+44(0) 1628 662017	(UK)
	+852 2416 4335	(Hong Kong)

RETURN of PRODUCT

Authorization is required from Racal Instruments before you send us your product for service or calibration. Call your nearest Racal Instruments support facility. A list is located on the last page of this manual. If you are unsure where to call, contact Racal Instruments, Inc. Customer Support Department in Irvine, California, USA at 1-800-722-3262 or 1-949-859-8999 or via fax at 1-949-859-7139. We can be reached at: helpdesk@racalinstruments.com.

PROPRIETARY NOTICE

DISCLAIMER

Buyer acknowledges and agrees that it is responsible for the operation of the goods purchased and should ensure that they are used properly and in accordance with this handbook and any other instructions provided by Seller. RII products are not specifically designed, manufactured or intended to be used as parts, assemblies or components in planning, construction, maintenance or operation of a nuclear facility, or in life support or safety critical applications in which the failure of the RII product could create a situation where personal injury or death could occur. Should Buyer purchase RII product for such unintended application, Buyer shall indemnify and hold RII, its officers, employees, subsidiaries, affiliates and distributors harmless against all claims arising out of a claim for personal injury or death associated with such unintended use.

FOR YOUR SAFETY

Before undertaking any troubleshooting, maintenance or exploratory procedure, read carefully the **WARNINGS** and **CAUTION** notices.

This equipment contains voltage hazardous to human life and safety, and is capable of inflicting personal injury.

If this instrument is to be powered from the AC line (mains) through an autotransformer, ensure the common connector is connected to the neutral (earth pole) of the power supply.

Before operating the unit, ensure the conductor (green wire) is connected to the ground (earth) conductor of the power outlet. Do not use a two-conductor extension cord or a three-prong/two-prong adapter. This will defeat the protective feature of the third conductor in the power cord.

Maintenance and calibration procedures sometimes call for operation of the unit with power applied and protective covers removed. Read the procedures and heed warnings to avoid “live” circuit points.

Before operating this instrument:

1. Ensure the proper fuse is in place for the power source to operate.
2. Ensure all other devices connected to or in proximity to this instrument are properly grounded or connected to the protective third-wire earth ground.

If the instrument:

- fails to operate satisfactorily
- shows visible damage
- has been stored under unfavorable conditions
- has sustained stress

Do not operate until, performance is checked by qualified personnel.

Racal Instruments

EC Declaration of Conformity

We

Racal Instruments Inc.
4 Goodyear Street
Irvine, CA 92718

declare under sole responsibility that the

6075-4, 4 Channel Serial Communications Module
P/N 407840-004
and
6075-8, 8 Channel Serial Communications Module
P/N 407840-008

conform to the following Product Specifications:

Safety: EN61010-1:1993+A2:1995

EMC: EN61326:1997+A1:1998

Supplementary Information:

The above specifications are met when the product is installed in a Racal Instruments certified mainframe with faceplates installed over all unused slots, as applicable

The product herewith complies with the requirements of the Low Voltage Directive 73/23/EEC and the EMC Directive 89/336/EEC.

Irvine, CA, December 23, 2002 Karen L Evensen

Karen Evensen, Engineering Director

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
GETTING STARTED

What's In This Chapter

This chapter contains a general description of the VXIbus Models 6075-4, 4-Channel and 6075-8, 8-Channel Serial Communications Modules along with an overall functional description of the instruments. It lists and describes various options available for this model. It also describes the Model 6075 front panel connectors and indicators.

NOTE:

This manual provides a complete description of all features and options available with the Model 6075 however, some items described in the following paragraphs may not be installed in your instrument.

Figure 1-1, Model 6075

Introduction

The Model 6075 is a single slot, C-size VXI module with four or eight independent serial channels for transmitting and receiving serial messages or data. Any or all of the serial channels can be active at the same time. Data transfer between the 6075 and the VXI controller is via VXI Fast Data Channels to minimize VXIbus transfer time and assure continuous data flow when operating at high baud rates. Each channel has two modes of operation: Asynchronous and SDLC.

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

The SDLC mode transmits data bytes in packets with a CRC checksum. SDLC characters have 8 data bits and no start or stop bits. The 6075's SDLC packet can hold from 2 to 30 data bytes. The packet size is set by the Frame:Size command. The SDLC packet format is:

Start byte	Data Bytes	CRC Bytes	End byte

SDLC data rates can be up to 1 MB/s. The transmit 1X clock is normally transmitted with SDLC data. The data from each received packet is placed in the FDC data buffer and is immediately followed by the 16-bit status byte from the channel's UART. The status byte contains information about the packet and identifies the packet as being a good or bad packet. The 6075 module is a Message Based I4 class VXI instrument with interrupter capability and supports all of the required VXIbus Word Serial Commands under the VXIbus Specification VXIbus-1, Rev. 1.4. Control and setup commands are Word Serial Messages and Word Serial Commands. Serial data is normally sent and received via Fast Data Channel buffers in A32 address space. Test messages can be sent and received as Word Serial Messages. The 6075 should only be used with a Slot 0 Controller that supports the A32 address space.

The data input and output FDC channels operate as A/B buffer pairs and in Stream Transfer mode for enhanced throughput. Each channel uses two FDC buffers for transmit and two for receive. This makes a total of thirty-two FDC channels used to transfer the serial data over the VXIbus. The 6075 module supports Fast Data Channel (FDC) data transfer per VXIbus Specification VXIbus-10, Rev. 2.10 using the Standard FDC

Word Serial Commands and an expanded command set that handles up to 32 FDC channels. In addition the module supports SCPI commands (1994.0) and IEEE Std 488.2-1987 commands.

VXIbus Description

VXIbus Objectives

The goal of the VXIbus Consortium is to create an open industry standard for modular instruments by defining interoperability between vendors, mechanical and environmental requirements, EMC compatibility, system initialization and software communication protocols. The physical portion of the VXIbus specification was adapted from the existing VME bus specification (IEEE-STD 1014). VXI is an acronym for “VME bus Extensions for Instrumentation.”

The VXIbus specification details the technical requirements of VXIbus compatible components such as mainframes, backplanes, power supplies and modules. The specification also provides for interconnecting and operating different manufacturers’ products within the same chassis. At the time of this publication, the current revision of the specification is version 2.0. The success of the specification is evidenced by over 300 manufactures who make over 800 different VXIbus products and hundreds of users .

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

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

Advantages of VXIbus Based Systems

The VXIbus provides the user with the following advantages:

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

VXIbus System Configurations

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

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

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

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

Data Transfer Methods

VXI modules can be register or message based . Register based modules are typically controlled by direct reads or writes to registers in the module. Message based modules communicate with word serial messages that are strings of ASCII or binary bytes. Word transfer uses the VXI word serial protocol that examines bits in the modules's response register to maintain an orderly data transfer. Because of the word serial protocol, register based modules are typically faster than message based modules but they lack the intelligence of message based modules. A new Fast Data Channel specification, VXI-1, provides for direct transfer of data from a module's memory to the Slot 0 Controllers at rates up to 32 Mbytes per second. This allows smart message based modules to have the same high data transfer rates as do register based modules.

Additional information about the VXIbus

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

Commander: A VXIbus device that has VME bus master capability and may have VXIbus servants under it in the system hierarchy. A Commander may act as a Servant to another Commander. A Commander must be message based.
Servant: A VXIbus device (with or without VME bus master capability) that is under control of a Commander in the VXIbus system hierarchy. A Servant may also be a Commander to other Servants. A Servant may be either message or register based.
Interrupt Handler: The module in the VXIbus system that generates the hardware interrupt acknowledge for a particular VME interrupt level. In VXIbus, the software interrupt handler may or may not be on the same module as the hardware interrupt handler.
Logical Address: A unique 8 bit number (0-255) which identifies each VXIbus device in a system. It defines the device's A16 register addresses.
Resource Manager: A message based commander located at logical address 0 which provides configuration management services, including address map configuration, Commander/Servant mapping, self test, and diagnostic management.
VXI Message Based Instrument: An intelligent instrument that implements the defined VXIbus registers and, at a minimum, word serial protocol.
VXI Word Serial Messages: The simplest required communication protocol supported by Message Based devices in a VXIbus system. It utilizes the A16 communications registers to transfer data or commands as a series of characters on the VXIbus backplane. The end bit is asserted on the last character of the message. Uses Word Serial Commands: Byte Available and Byte Request.
VXI Word Serial Commands: Single word, 16-bit commands sent from the commander to its servants. Some Word Serial Commands have a response word. The VXI specification defines a number of standard Word Serial commands that are reserved for use by the Slot 0 Controller or the Resource Manager. The specification also allows instrument designers to define their own Word Serial Commands.
VXI Commands: These are commands passed from a Commander to a Servant within the VXIbus environment. There are three broad categories of commands: VXIbus Instrument Protocols, IEEE 488.2 Common Commands, SCPI commands, and Device specific commands. A command may or may not be stimulated by an external event. For example an IEEE-488 Group Execute Trigger will generate a trigger command to all addressed devices. However, a Begin Normal Operations command is generated by the VXIbus resource manager and has no external source.
VXI Events: VXIbus Events are passed from a Servant to a Commander. They may be generated by the Servant either in response to a command (e.g., an invalid command error), or due to an external condition (e.g., data ready or status change).
VXI Fast Data Channel: A method for exchanging data between a commander and a servant module that utilizes a minimum of handshaking to transfer data so that the data transfer rate approaches the theoretical VME bus transfer rate. Allows a commander access to portions of the device's memory or a registers in the A32 address space. Data transfer is unidirectional for each channel and can be D16 or D32 bit words. Multiple channels may be opened for each device using A/B channel pairs for continuous data transfer.
488-VXIbus Interface Device: An IEEE-488 to VXIbus Interface Device is a message based device which provides communication between the IEEE-488 bus and VXIbus instruments. Typically this function is included in the Slot 0 card for external control of the VXI chassis.

Table 1-1, VXIBus Glossary

Safety Considerations

The Model 6075 has been manufactured according to international safety standards.

WARNING

Do not remove instrument covers when operating or when the chassis power cord is connected to the mains.

Any adjustment, maintenance and repair of an opened, powered-on instrument should be avoided as much as possible, but when necessary, should be carried out only by a skilled person who is aware of the hazard involved.

Supplied Accessories

The following accessories are supplied with each 6075 module.

Qty.	Part No.	Description
1	980870	6075 Instruction Manual.
2	TBD	DC-37S Mating Connector (6075-8 only)
2	TBD	DC Hood (6075-8 only)

Specifications

Instrument specifications are listed in the paragraphs below. These specifications are the performance standards or limits against which the instrument is tested.

VXIbus Capabilities

The 6075 has the following VXIbus capabilities:

Addressing	Static configured addresses 1-254 or Dynamic configuration
Manufacturer ID	4091 decimal (Racal Instruments)
Device Class	Message based I4 class VXI instrument
Address Space	A16 (D16) / A32 (D32) 64 bytes in A16 space 1 or 4 Mbytes in A32 space
Model Code	12 bits, 607 decimal for Model 6075-4 608 decimal for Model 6075-8
Interrupter	D16, Programmable Interrupter
Event Generator	Programmable, Interrupts only
FDC Event Generator	Programmable, Interrupts only
Response Generator	Not Supported
VMEbus Master	Not Supported
Commander	Not Supported
Signal Register	Not Supported
Handshake	Normal handshake only
Fast Data Channel	A32 (D32) Random Access, 16 Channels: I/O Pair & Stream Transfer mode, Standard FDC Word Serial Commands and Expanded FDC Word Serial Commands per FDC Addendum
FDC Transfer Rate	approximately 4 Mbytes per second
Primary Classification	I4

VXIbus Instrument Protocols

The 6075 is a message based instrument that supports the VXIbus instrument protocols for operation as an I4 class VXI instrument and Fast Data Channels, using the VXIbus word serial protocol and 488.2 common commands. As such it supports the following instrument protocol commands as described below and Racal Instruments' expanded FDC command set in the FDC Addendum.

Command	Description
Byte Available	Sends Word Serial Messages to the 6075.
Byte Request	Reads command responses or data from the 6075.
Clear	The clear command causes the 6075 to clear its internal buffers, reset its command parser to its initial state. Does not affect the Fast Data Channel buffers or other buffers.
Trigger	The Trigger command causes the unit to execute commands in the program buffer specified by the Arm Trigger command.
Set and Clear Lock	Not supported by the 6075, no front panel controls.
Read STB	Response is a byte equivalent to the serial poll and *STB? response in IEEE Standard 488.2.
Abort Normal operation	Standard response.
Begin Normal operation	Standard response.
End Normal operation	Standard response.
Read Protocol	Standard response.
Read Protocol Error	Standard response.
Assign Interrupter Line	Standard response.
Read Interrupter Line	Standard response.
Read Interrupters	Standard response.
Asynchronous mode control	Standard response.
Control Event	Standard response.
Arm Trigger	A Racal Instruments defined command: 0x02*bbbbettt* used to specify the TTL Trigger line that will be used to run the selected buffer when the 6075 is sent a TTL Trigger or a Word Serial Trigger command. The buffer is one of the 6075's program storage buffers. Where: *bbbb* = Bits 4-7 are the selected program storage buffer number. *e* = Bit 3 is the enable bit. *ttt* = Bits 0-2 specify the TTL Trigger line. There is no response to this command.
Channel Address High	Standard response.
Channel Address Low	Standard response.
Channel Close	Standard response.
Channel Initialize	Standard response.

Channel Size High	Standard response.
Channel Size Low	Standard response.
Enable Passed Buffer	Standard response.
Passed Buffer	Standard response.
FDC Event	Standard response.
FDC Supported	Standard response.
Go to Idle	Standard response.
Transfer to Commander	Standard response.
Transfer to Servant	Standard response

.

Table 1-2 Supported Instrument Protocols

VXIbus Fast Data Channels

The 6075 supports up to thirty-two VXI Fast Data Channels for transferring data from the 6075's serial channels to the Slot 0 Controller. The channels are used as A/B pairs in stream mode for continuous data transfer. Even numbered channels are the A channel. The channel assignments and their maximum size is shown in **Table 1-3**. The user can set the receiver buffer size to a value less than the maximum shown in the Table. Buffer starting addresses are pre-assigned and do not change if the buffer size is changed.

The 6075 supports the standard VXI Fast Data Channel Commands and Racal Instruments' expanded Fast Data Channel command set for setting up the FDC Channels and controlling their operation. Both commands sets are described in the Fast Data Channel Addendum. Racal Instruments' expanded command set is preferred because the standard VXI FDC Commands only support eight FDC channels and have limited usefulness with the 6075.

FDC Channel	Function	Buffer Sizes	
		6075-4	6075-8
0	Ch 1 Receive	128 Kbytes	64 Kbytes
1	Ch 1 Receive	128 Kbytes	64 Kbytes
2	Ch 1 Transmit	128 Kbytes	64 Kbytes
3	Ch 1 Transmit	128 Kbytes	64 Kbytes
4	Ch 2 Receive	128 Kbytes	64 Kbytes
5	Ch 2 Receive	128 Kbytes	64 Kbytes
6	Ch 2 Transmit	128 Kbytes	64 Kbytes
7	Ch 2 Transmit	128 Kbytes	64 Kbytes
8	Ch 3 Receive	128 Kbytes	64 Kbytes
9	Ch 3 Receive	128 Kbytes	64 Kbytes
10	Ch 3 Transmit	128 Kbytes	64 Kbytes
11	Ch 3 Transmit	128 Kbytes	64 Kbytes
12	Ch 4 Receive	128 Kbytes	64 Kbytes
13	Ch 4 Receive	128 Kbytes	64 Kbytes
14	Ch 4 Transmit	128 Kbytes	64 Kbytes
15	Ch 4 Transmit	128 Kbytes	64 Kbytes

16	Ch 5 Receive		64 Kbytes
17	Ch 5 Receive		64 Kbytes
18	Ch 5 Transmit		64 Kbytes
19	Ch 5 Transmit		64 Kbytes
20	Ch 6 Receive		64 Kbytes
21	Ch 6 Receive		64 Kbytes
22	Ch 6 Transmit		64 Kbytes
23	Ch 6 Transmit		64 Kbytes
24	Ch 7 Receive		64 Kbytes
25	Ch 7 Receive		64 Kbytes
26	Ch 7 Transmit		64 Kbytes
27	Ch 7 Transmit		64 Kbytes
28	Ch 8 Receive		64 Kbytes
29	Ch 8 Receive		64 Kbytes
30	Ch 8 Transmit		64 Kbytes
31	Ch 8 Transmit		64 Kbytes

Table 1-3 FDC Channels

IEEE 488.2 Common Commands

The 6075 supports the following VXIbus IEEE 488.2 Common Commands to perform functions such as service request enable, event status register query and status byte query, etc. IEEE 488.2 Common Commands begin with an asterisk (*), and may include one or more parameters. Detailed descriptions of the commands can be found in Chapter 3 and in the IEEE Std 488.2-1987 Standard.

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

STB and ESR Response

The 6075 responds to the Read STB and ESR commands by sending a byte equivalent to the serial poll and *STB? or *ESR? response in IEEE Standard 488.2. The 6075's Read STB and ESR response bytes have the following bit assignments:

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

Table 1-4 STB and ESR response Bytes

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

SCPI Commands

The 6075 supports SCPI commands that allow the user to configure the module, set the data buffer sizes and control the Questionable and Conditional registers in the 6075's Status Reporting Structure. The SCPI commands conform to the SCPI 1995.0 Standard and are listed in **Table 3-5**. **Table 1-5** lists the 6075's configurable parameters and the factory settings.

Operational Register

The 6075 provides an Operational Register that reports the status of the Waiting for Trigger bit. The register structure is a four register set with a Condition register, Transition register, Event register and an Enable register. Refer to section 3.5 for a description of the registers' operation. Bit assignments are:
Bit 5 Waiting for Trigger

Questionable Register

The SCPI Questionable Register is not used in the 6075 module. The register structure is in place and the commands are supported by the 6075's parser so their use will not report a command error. The Questionable Register commands have no operational effect on the 6075 module.

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

Table 1-5 6075 Default Parameter Settings

Serial Interfaces

The 6075 provides four or eight serial interfaces for Asynchronous and SDLC communication with external devices. Each interface supports RS-232, RS-422 or RS-485 serial links.

Modes

Asynchronous

A character oriented protocol where each character has a start bit, data bits, an optional parity bit and one or more stopbit(s). There is no limit to the number of characters in a message nor the time between characters or messages. Use RS-232, RS-422 or RS-485 signals. Maximum module data rate for all channels is 460,000 baud.

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

SDLC

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

The SDLC packet format is:

Start byte	data bytes	CRC	End byte

RS-232 Signals

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

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

RS-422 Signals

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

RS-485 Signals

Signals	TX and RX on a single balanced signal pair Signal ground is SG. Shield is chassis ground.

Transmit Levels	+4 ± 1 VDC differential for binary '0' or on -4 ± 1 VDC differential for binary '1' or off
Receive Levels	± 0.2 VDC minimum ± 10 VDC maximum differential or between any signal and SG.
Mode	Half duplex only
Termination	Switchable on to TX lines

Baud Rates

Baud rates for asynchronous characters can be generated from the internal 7.3728 MHz oscillator or from and eternal RXCLK input. Standard internally generated rates start at 50 Hz and include: 150, 300, 600, 1200, 2.4 K, 4.8 K, 7.2 K, 9.6 K, 19.2 K, 38.4 K, 57.6 K, 76.8 K, 115.2 K, 153.6 K, 230.4 K and 460.8 Kbaud. Baud rate accuracy is ± 0.02 %. Unit selects closest integer value for nonstandard rates by dividing down from 460,800 Hz. Maximum module rate for simultaneous use of all channels is > 460,000 baud.

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

Serial Buffers

See FDC buffer specifications in **Table 1-3**. DATA and DATAT messages limited to 1024 bytes.

Termination Resistor Network

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

Diagnostic Capabilities

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

Timing Pulse Output

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

Period	0.0001 to 3.0000 seconds
Pulse Width	0.0001 to 3.0000 seconds

Front Panel Indicators

The 6075 has a four character alpha numeric display on the front panel which displays module status, logical device addresses and diagnostic messages. Normal mode display messages are listed below. The complete message set is listed in Section 3.

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

Physical

Size	C size VXI module
Dimensions	352.43 mm long x 233.35 mm high x 30.18 mm wide
Weight	1.79 kg (3.94 lbs) including RF shields
Power	5 VDC at 3.00 A max ±24 VDC at 0.002 A -2 VDC at 0.009 A -5.2 VDC at 0.050 A
Cooling Requirements	1.26 Liters/Second at 0.095 mm H_2O for a 10°C internal rise
Temperature	0 °C to 55 °C operating -20 °C to 70 °C storage
Connectors	Four Channel Units: Four 9-pin DE-9P Connectors with lock studs Eight Channel Units: Two 37-pin, DC 37P Connectors with lock studs
MTBF	39,817 Hours

Functional Description

A detailed functional description is given in the following paragraphs. The description is divided into logical groups: input and output connectors, operating modes, output type, output state, synchronization, filters and front panel indicators.

Chapter 2
CONFIGURING THE INSTRUMENT

Installation Overview

This chapter contains information and instructions necessary to prepare the 6075 for operation. Details are provided for initial inspection, grounding safety requirements, repacking instructions for storage or shipment, logical address selection and installation information.

Unpacking and Initial Inspection

Unpacking and handling of the synthesizer requires only normal precautions and procedures applicable to handling of sensitive electronic equipment. The contents of all shipping containers should be checked for included accessories and certified against the packing slip to determine that the shipment is complete.

Safety Precautions

The following safety precautions should be observed before using this product and associated computer. Although some instruments and accessories would normally be used with non-hazardous voltages, there are situations where hazardous conditions may be present.
This product is intended for use by qualified personnel who recognize shock hazards and are familiar with the safety precautions required to avoid possible injury. Read the operating information carefully before using the product.
Exercise extreme caution when a shock hazard is present. Lethal voltage may be present on cables, connector jacks, or test fixtures. The American National Standard Institute (ANSI) states that a shock hazard exists when voltage levels greater than 30V RMS, 42.4V peak or 60 VDC are present.

WARNING

For maximum safety, do not touch the product, test cables, or any other instrument parts while power is applied to the circuit under test. ALWAYS remove power from the entire test system before connecting cables or jumpers, installing or removing cards from the computer, or making internal changes such as changing the module address.

WARNING

Do not touch any object that could provide a current path to the common side of the circuit under test or power line (earth) ground. Always keep your hands dry while handling the instrument.

When using test fixtures, keep the lid closed while power is applied to the device under test. Safe operation requires that the computer lid be closed at all times during operation. Carefully read the “Safety Precautions” instructions that are supplied with your computer. Before performing any maintenance, disconnect the line cord and all test cables.

Maintenance should only be performed by qualified service personnel.

Performance Checks

The instrument has been inspected for mechanical and electrical performance before shipment from the factory. It is free of physical defects and in perfect electrical order. Check the instrument for damage in transit and perform the electrical procedures outlined in the section entitled **Unpacking and Initial Inspection.**

Grounding Requirements

To insure the safety of operating personnel, the U.S. O.S.H.A. (Occupational Safety and Health Administration) requirement and good engineering practice mandate that the instrument panel and enclosure be "earth" grounded. Although connector housings are isolated from the front panel, the metal part is connected to earth ground.

Long Term Storage or Repackaging For Shipment

If the instrument is to be stored for a long period of time or shipped immediately, proceed as directed below. If you have any questions, contact your local Racal Instruments Representative or the Racal Instruments Customer Service Department.

1. Repack the instrument using the wrappings, packing

Shipment

material and accessories originally shipped with the unit. If the original container is not available, purchase replacement materials.

2. Be sure the carton is well-sealed with strong tape or metal straps.
3. Mark the carton with the model and serial number. If it is to be shipped, show sending and return address on two sides of the box.

NOTE

If the instrument is to be shipped to Racal Instruments for calibration or repair, attach a tag to the instrument identifying the owner. Note the problem, symptoms, and service or repair desired. Record the model and serial number of the instrument. Show the work authorization order as well as the date and method of shipment. ALWAYS OBTAIN A RETURN MATERIAL AUTHORIZATION (RMA) NUMBER FROM THE FACTORY BEFORE SHIPPING THE INSTRUMENT TO RACAL INSTRUMENTS.

Preparation For Use

Preparation for use includes removing the Model 6075 from the container box, selecting the required logical address and installing the module in a VXIbus chassis.

The 6075 Quad Serial Module is ready for RS-232 and RS-422/RS-485 operation when shipped.

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

Logical Address Selection

The VXIbus Chassis Resource Manager identifies modules in the system by the module's address. VXIbus logical addresses can range from 0 to 255, however, only addresses 1 to 254 are available for VXIbus modules. Logical address 0 is reserved for the Resource Manager. Logical address 255 permits the Resource Manager to dynamically configure the module logical address.

The 6075 has an internal address switch for setting its logical address. The switch is accessible from the side of the module

when the unit is outside the VXI chassis. Set the 6075's address switch to any unused value between 1 to 254 for static addressing or to 255 for dynamic address assignment**. Figure 2-1** shows the switch layout and rocker bit weights. The address switch is set to logical address 16 (HEX 10) by the factory.

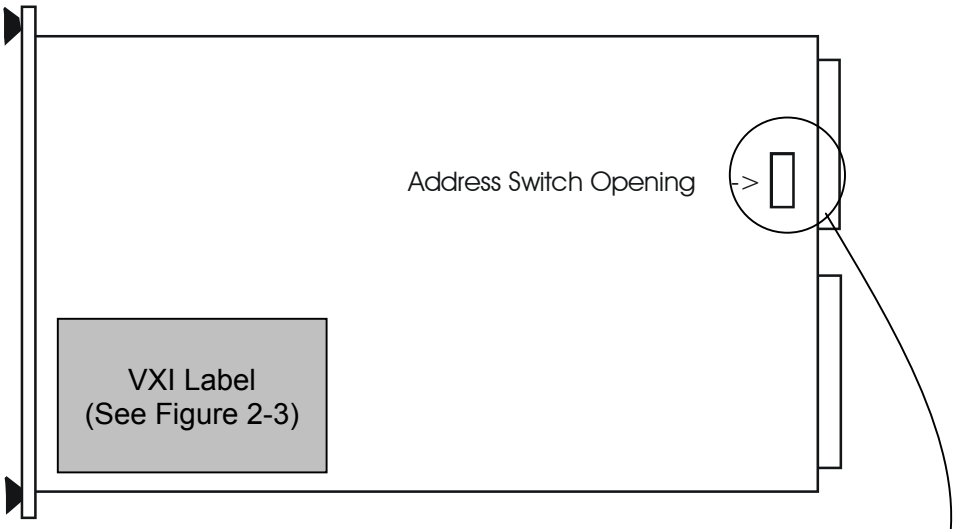
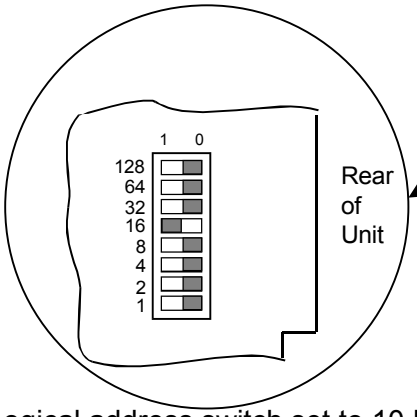


Figure 2-1 Address Switch Location

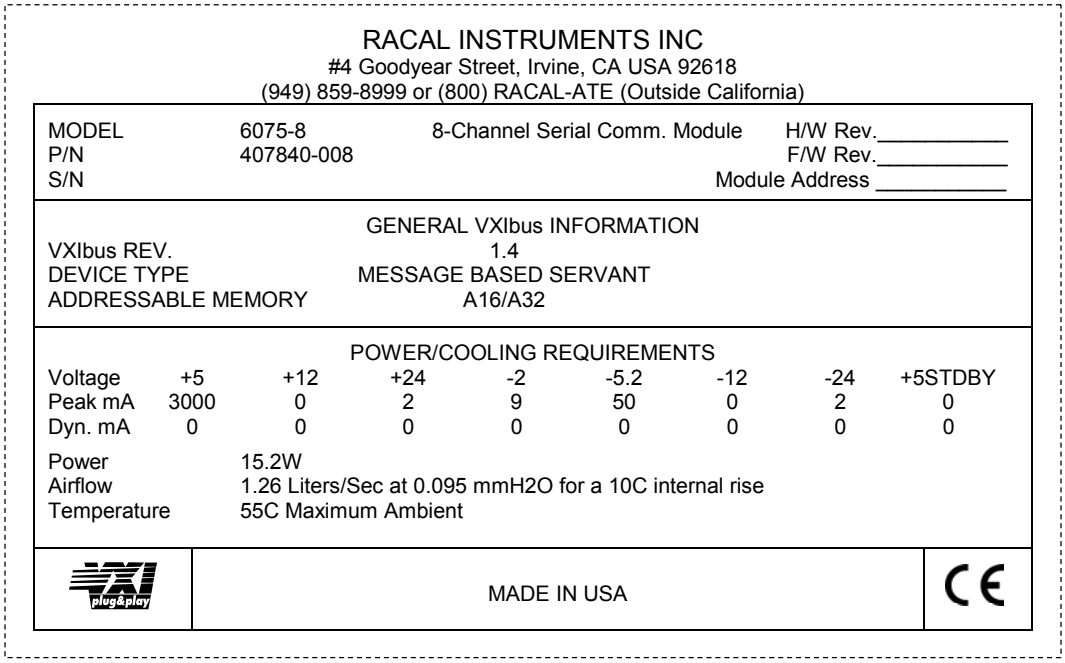


Logical address switch set to 10 Hex

Figure 2-2 Address Switch Layout

VXI Labeling

The identification label on the side of the module contains important information about the 6075 module including power and cooling requirements, the module serial number and the module's hardware and firmware revisions. The label is located in the lower left corner of the module's right cover as shown in **Figure 2-1**. The label is shown in **Figure 2-3.**



RACAL INSTRUMENTS INC
#4 Goodyear Street, Irvine, CA USA 92618
(949) 859-8999 or (800) RACAL-ATE (Outside California)

MODEL	6075-8	8-Channel Serial Comm. Module	H/W Rev._____
P/N	407840-008		F/W Rev._____
S/N			Module Address _____

GENERAL VXIbus INFORMATION

VXIbus REV.	1.4
DEVICE TYPE	MESSAGE BASED SERVANT
ADDRESSABLE MEMORY	A16/A32

POWER/COOLING REQUIREMENTS

Voltage	+5	+12	+24	-2	-5.2	-12	-24	+5STDBY
Peak mA	3000	0	2	9	50	0	2	0
Dyn. mA	0	0	0	0	0	0	0	0

Power	15.2W
Airflow	1.26 Liters/Sec at 0.095 mmH2O for a 10C internal rise
Temperature	55C Maximum Ambient

MADE IN USA

Figure 2-3 6075 VXI Identification Label

Checking the 6075's Logical Device Address

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

Serial Interface

Four Channel Units

The 6075-4 Quad Serial Module has four 9-pin DE-9P connectors with lock studs that independently provide either RS-232, RS-422 or RS-485 signals. **Table 2-1** shows the serial signal assignments and the signal direction relative to the 6075.

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

Note: Signal direction arrows ← = in to 6075, → = out from 6075

Table 2-1 Serial Interface Pin Assignments for Four Channel Units

RS-232 Connections

When RS-232 signals are selected, the signal pinouts are compatible with those on the 9-pin COM port of an IBM type computer. The 6075's RS-232 signals can be expanded out to a 25 pin connector with a 9 pin to 25 pin adapter. The RS-232 column in **Table 2-1** above shows the RS-232 signal pin assignments and their direction relative to the 6075. **Figure 2-4** shows the wiring diagram for a typical 9 pin to 25 pin adapter. For minimum RS-232 connections, use the serial signals on pins 2, 3 and 5. Jumper the unused RS-232 control lines back to themselves as follows: DTR to CTS and DCD. Implement either the hardware or software flow control if the serial messages could overflow either devices' receive buffer.

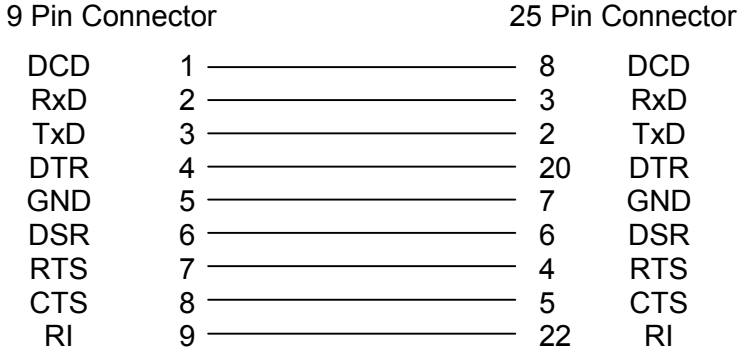


Figure 2-4 DTE 9 Pin to 25 Pin Adapter

RS-422 Connections

When RS-422 signals are selected, the pinouts appear as shown in the RS-422 FD/HD column in **Table 2-1**. For asynchronous RS-422 data transfer, the TXCLK and RXCLK signals are not used and may be left open. Use the 6075's internal oscillator to generate baud rates as listed in the Baud Rates Specifications provided in the **Getting Started** Chapter of the manual. For SDLC operation, TXCLK may be an input or output signal and RXCLK can be an input or it can be not used.

In half-duplex mode, the transmitter is tristated between messages. The 6075's internal termination network can be used to bias the TX data lines in the 'mark' state when the transmitter is not active.

RS-485 Connections

When RS-485 signals are selected, data is transmitted and received on the Tx/Rx signal pair in the RS-485 HD column in **Table 2-1**. The RXCLK and TXCLK are not used and maybe ignored. The 6075's internal termination network can be used to bias the TX/RX data lines in the 'mark' state when the transmitter is not active.

Eight Channel Units

The 6075-8 Octal Serial Module has two 37-pin DE-9P connectors that each provide four serial channels. Channels 1, 2, 5, 6 and the Pulse Output are on J5. Channels 3, 4 7 and 8 are on J6. The signal definitions and functions are the same as for the four channel unit described in the preceding sections. **Table 2-2** shows the serial signal assignments and the signal direction relative to the 6075.

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

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

Note: Signal direction arrows ← = in to 6075, → = out from 6075

Table 2-2 Serial Interface Pin Assignments for Eight Channel Units

Chapter 3
USING THE INSTRUMENT

Overview

This chapter contains information about how to operate the Model 6075. The instrument can be programmed using a set of SCPI commands. A list of SCPI. The following paragraphs describe the various modes of operation and give examples on how to program the 6075.

This section includes directions for using the Front Panel Display, on how to configure the 6075 interfaces and on using the 6075 interfaces to control GPIB devices.

Front Panel Displays and Controls

The 6075 has a four character alphanumeric display on the top of its front panel and a Data LED for each port. The Data LED blinks when data is being transferred on the channel interface.

Alphanumeric Display

At power turn-on or when reset, the alphanumeric display operates in its normal mode and displays the 6075's status and addressed interface. The normal power-on display sequence is:

Display	Meaning
SysF	On during selftest while SYSFAIL is asserted. If on after 4.9 seconds, the module could be defective or SYSFAIL is still being held on.
RAC	Momentary on after selftest is passed.
6075	Momentary on after selftest is passed.
FAIL	On only if selftest failed or the module was commanded into the failed state.
Init	On after selftest passed while module is waiting for a Begin-Normal-Operation command. Module is in configure sub-state.
Rdy	Unit ready for Word Serial commands.

If the unit fails selftest, FAIL will be displayed for 5 seconds and then the display will show the appropriate failure error code (ERxx). The error codes are listed in **Table 3-5**.

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

Display Button

The Display Button is a recessed push button located below the display as shown in **Figure 3-1**. The Display Button selects the 6075 display and diagnostic modes described in **Table 3-1**. **Tables 3-2** through **3-5** list the messages for the various display modes.

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

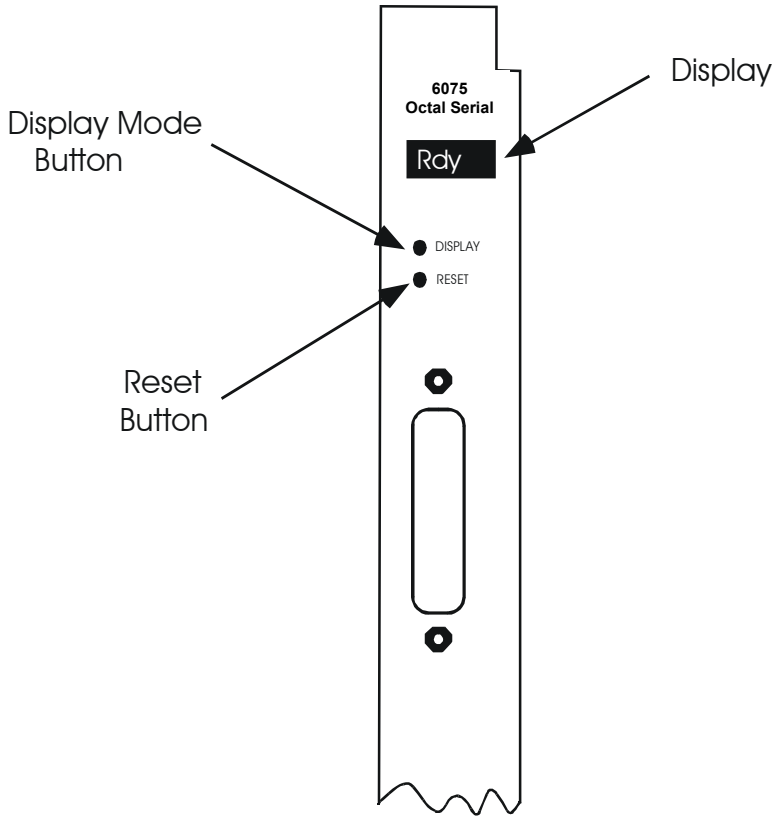


Figure 3-1 Display Mode Button Location

Reset Button

The Reset Button is a recessed push button located below the Display Button as shown in **Figure 3-1**. Pressing the Reset Button resets the 6075 processor and internal logic. The Resource Manager must be run or the Begin-Normal-Operation command must be issued to resume normal operation after a reset.

Display Modes

The 6075 display has alternate display modes to give the user more information about the module and its status. **Table 3-1** describes the module's display modes. **Tables 3-2** through **3-5** list the displayed parameters.

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

Table 3-1 6075 Display Modes

e	Meaning
6075	Power on ID message
SAVE	Data written to the Flash memory
FAIL	Self Test failed
RAC	Power on ID message
Init	Initialized, waiting for begin normal operation command
Rdy	Normal ready message, begin normal operation received
SysF	SystFail Asserted
Chn	Serial Channel n selected (n = 1 to 8)

Table 3-2 Normal Display Mode Messages

e	Meaning
BNO	Begin Normal command received.
ERnn	Error message - see **Table 3-5**
Errn	Unsupported command error
IDCr	Identify commander
SfGP	Soft reset

Table 3-3 Extended Mode Messages

Display	Meaning
FAIL	Self test failed
Pass	Selected slave test passed

Table 3-4 TEST Mode Messages

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

Table 3-5 ERROR Messages

General Operating Instructions

This section provides the user with the necessary information to operate the 6075. It is required reading for new 6075 users.

Pre-Operation Setup

Before using the 6075, set the 6075's address switch and install the module in a VXI chassis as directed in the Installation Section.

Power Turn-on

Turn the VXI chassis power on. The 6075's display should cycle through its identification sequence while the unit does its power-on selftest. When the display shows Init, the module has passed its self test and is waiting for the Begin-Normal-Operation Command. After the Resource Manager has run, the display shows Rdy, which indicates that the module is good and has received the Begin-Normal-Operation command. For the meaning of other messages, refer to the previous sections and to **Tables 3-2** and **3-3**.

The 6075 can also be Soft Reset by asserting the Reset bit (bit 0) in the VXIbus Control Register for a minimum period of 100 microseconds. This causes a software reset of the module. All settings revert to their power turn-on values. The 6075 remains inactive after a Soft Reset until activated by a Begin-Normal-Operation command. The Begin-Normal-Operation command is typically sent by the Resource Manager but can be sent by the user after a reset. If interrupts are used, the Assign Interrupt Line command will have to be sent also.

Interface Programming Concepts

Communication with the 6075 over the VXIbus is done with either Word Serial Commands and Word Serial Messages or via the Fast Data Channel buffers.

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

The applicable Word Serial Commands for the 6075 are described in subsequent sections and are listed in **Table 3-6**. Racal Instruments' expanded Word Serial Commands for FDC Channels are described in the FDC Addendum. Note that some Word Serial Commands are queries and return a response. The response must be read before sending the module a new command.

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

The Fast Data Channel buffers are the normal way serial data is transferred between the VXI Commander and the 6075. The Fast Data Channel buffers are used as pairs to pass serial data in one direction. The controller places data in one buffer while the 6075 empties the other channel's buffer. The Controller can switch buffers when ready. The 6075 places received data in a FDC buffer until it reaches the full point. The buffers are then switched and the 'full' buffer is passed to the Controller. The Controller learns of the buffer switch either by a VXI interrupt or by polling the buffer header.

Four fast data channels are required for each 6075 serial channel, two for transmit and two for receive. Bits in the Fast Data Channel header define the number of bytes in the FDC buffer and buffer ownership. The FDC buffer space is allocated by the Resource Manager function in the Slot 0 Controller. The Fast Data Channel buffers are initialized by the user's program. Refer to Section **Fast Data Channel Usage** later in this chapter for information about the FDC buffers and their organization in the 6075. Also, refer to **Appendix A** for more detailed information on how to setup and use the FDC buffers.

The 6075 generates VXI Interrupts. The 6075 is a single VXI interrupter. The VXI interrupts can be FDC events or normal VXI events. The normal VXIbus interrupts are enabled by setting the appropriate bits in the 488.2 Status Enable Register and Event Enable Register in the 6075's Status Reporting Structure. The FDC channels can be set to interrupt when the Receive buffer has a message, has received a preset byte count or is full. FDC interrupts can also occur when there is a full buffer that needs to be emptied or an empty buffer that needs filling. The VXIbus interrupt line is assigned when the Resource Manager is run. When an interrupt occurs, examine the upper 8 bits of the interrupt response word to determine the interrupt type. If the interrupt is a normal VXI event interrupt, use the Word Serial Read STB command to read the interface(s) Status Byte Register to determine the cause of the interrupt. If the interrupt is a FDC interrupt, the FDC buffer number is specified in the upper byte of the interrupt response word.

The 6075 has a VXI Status Register (at an offset of four bytes above its base A16 address) which can be read by the VXI Slot 0 Controller. Bits 6-13 of the VXI Status Register are device dependent bits that the device designer can use to simplify device programming. In the 6075, these bits are used to pass status back to the user. **Figure 3-2** shows the register bit assignments and their channel usage.

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

Figure 3-2 VXI Status Register Bit Assignments

Program Recommendations

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

1. Do not continuously poll the 6075 with the VXI Read STB command. This command requires internal processor time and continuous polling degrades the interface's performance.

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

Controlling the 6075 with Multi-Tasking Operating Systems

Outputting data to the 6075 using Word Serial Messages from a task in a Multi-Tasking environment requires careful programming to avoid confusing the module or accidentally changing interface channels. These concerns are not relevant when passing data in FDC channel buffers.

If multiple tasks are used to control the 6075, the user needs to preface each Word Serial command with the Channel selection command to be sure of sending the command to the correct channel. Do not rely upon the channel setting being left in your channel number. A Channel selection command is not necessary when using FDC buffers to send and receive data as the FDC channels are dedicated to a specific serial channel.

Word Serial Commands

The 6075 supports the standard Word Serial Commands defined in the VXI Specification, the VXI FDC commands and Racal Instruments' expanded FDC commands. Word Serial Commands are single word, 16-bit binary commands and are coded differently from the data words used in Word Serial Messages. The 6075 always responds to VXI Word Serial Commands regardless of the mode of the interface. The Word Serial Commands are used to configure the 6075's VXI interface, alter its operation or control its FDC buffers. A list of the VXI Word Serial commands is provided in **Table 3-6**.

The user should consult his VXI Slot 0 Controller manual for specific directions on sending Word Serial Commands to a VXIbus module. The following example uses the VISA I/O library to send a 16-bit word serial command (0xFCFF = Begin Normal Operation) to the data low register (offset 0xE) of an instrument. Note that the instrument handle (hdl) must be previously opened using the `viOpen()` function from VISA.

```
ViStatus err;

err = viOut16(hdl,VI_A16_SPACE,0xE,0xFCFF);
```

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

Table 3-6 6075 Word Serial Commands

Fast Data Channel Usage

6075 FDC Buffer Organization

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

In the 6075, each 16-bit word contains two serial characters. The first or most significant character is in byte 0. The second of later character is in byte 1. Serial messages can be an even or odd number of characters. The value in the Data Buffer Size Word, Word #1 of the Header in **Figure 3-3**, is the message byte count.

As part of an application, the user can put multiple, concatenated messages in the buffer. This multiple message will be treated by the 6075 as a single message in the buffer. If a message has an odd number of characters (bytes), the first character of the subsequent message is packed in the word with the last character of the previous message. The byte count is then set to the total number of characters in the buffer.

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

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

Notes: Bit definitions are listed in the next section (**FDC Buffer Definitions**).
Data area starts below the dashed line.
Byte numbers are in Intel byte order as seen by the Slot 0 Controller.

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

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

Notes: Bit definitions are listed in the next section (**FDC Buffer Definitions**).
Data area starts below the dashed line.
Byte numbers are in Intel byte order as seen by the Slot 0 Controller
Data sample contains n characters.

Figure 3-4 FDC Buffer Layout for 32 Bit Words

FDC Buffer Definitions

FDC BUFFER: Total buffer area

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

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

MAJOR REVISION: Must be 2 (3 bit code)

MINOR REVISION: Must be 1 (2 bit code)

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

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

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

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

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

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

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

DATA BUFFER ORGANIZATION: In the 6075 the whole data buffer section is used for data. Transmit data is right justified with the first (most significant) character or byte in byte 0. Received data is placed in the buffer in the same manner.

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

FDC Buffer Initialization

The FDC buffers need to be initialized before they can be used to transfer data. The initialization sequence and commands are described in the FDC Addendum. The buffers should be initialized as streaming pairs where they automatically switch as they become full (receive) or empty (transmit). The 6075 only supports Pair and Stream mode. The FDC mode selection is made by setting flag bits in the Transfer to Commander or Transfer to Servant initialization command.

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

FDC Buffer Operation

In the 6075 module, the Fast Data Channel (FDC) buffers are used as A/B buffer pairs in stream mode to transfer data over the VXIbus backplane at a high data rate. For a transmit pair, the Controller first loads the first (lower numbered) buffer and passes it to the 6075. The buffer can contain as little as one message. The 6075 transmits the data from the first buffer while the Controller places any additional data in the second buffer. The 6075 returns the empty buffers to the Controller. This operation continues, with the Controller loading data into alternate buffers and passing them to the 6075 as long as there is data to send. The Controller should not reuse the same buffer twice in a row, even when both buffers are empty.

Data is received in the opposite fashion. The 6075 loads the received data into alternate buffers and passes them to the Controller. A receive buffer is 'full' when it has been filled to its assigned size as set by the SYST:COMM:SER:BUFF:SIZe command. When a receive buffer is 'full', the module switches to the other buffer and continues receiving data. The 6075 can generate a VXIbus interrupt to notify the controller that it has a buffer ready to pass. This operation continues with the 6075 filling alternate receive buffers as it receives data.

The user should consider the message length and VXI access word size when setting the buffer size. For multiple messages, the buffer size should be a multiple of the received message size to avoid splitting messages. Otherwise, the result could be very confusing data in the buffer. The user must read out

complete words when reading the buffer, even if the full count ended in the middle of a VXI bus word.

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

488.2 Compliance

The 6075 is an IEEE-488.2 compliant VXIbus instrument. As an IEEE-488.2 compliant VXIbus instrument, the 6075 responds to the IEEE-488.2 commands and queries listed in **Table 3-7**.

488.2 Status Reporting Structure

The 6075 includes the IEEE-488.2 Status Reporting structure shown in **Figure 3-5**. The expanded status reporting structure conforms to the SCPI 1994.0 Specification and builds on the IEEE 488.2 Standard with the addition of the Questionable and Operational registers. The Event and Status registers are controlled and queried with the IEEE-488.2 common commands. The added Questionable and Operational registers are controlled and queried with SCPI commands.

As shown in **Figure 3-5**, IEEE 488.2 service request generation is a multilevel function and is determined by the occurrence of an event that has its corresponding enable bit set to '1'. A service request uses a VXI interrupt to signal the bus controller that an event has occurred and/or that the 6075 needs service. There are four major sources of service requests, each of which is summarized in a bit in the Status Byte Register. Three of the sources are event registers with their own enabling bits and the fourth is the Output Queue. The Event registers and the Output Queue are cleared when read or by the *CLS command.

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

WARNING - Continuously polling the 6075 with the Read STB command ties up the internal processor and degrades module performance. The recommended method to test for data in the receive buffer is to enable the VXIbus interrupt and then execute a Read STB command when the interrupt is received.

The 6075 logical devices respond to the Read STB command and *STB? query with a response byte equivalent to the serial poll response in IEEE Standard 488.2. The 6075's STB response byte is shown in **Figure 3-5**.

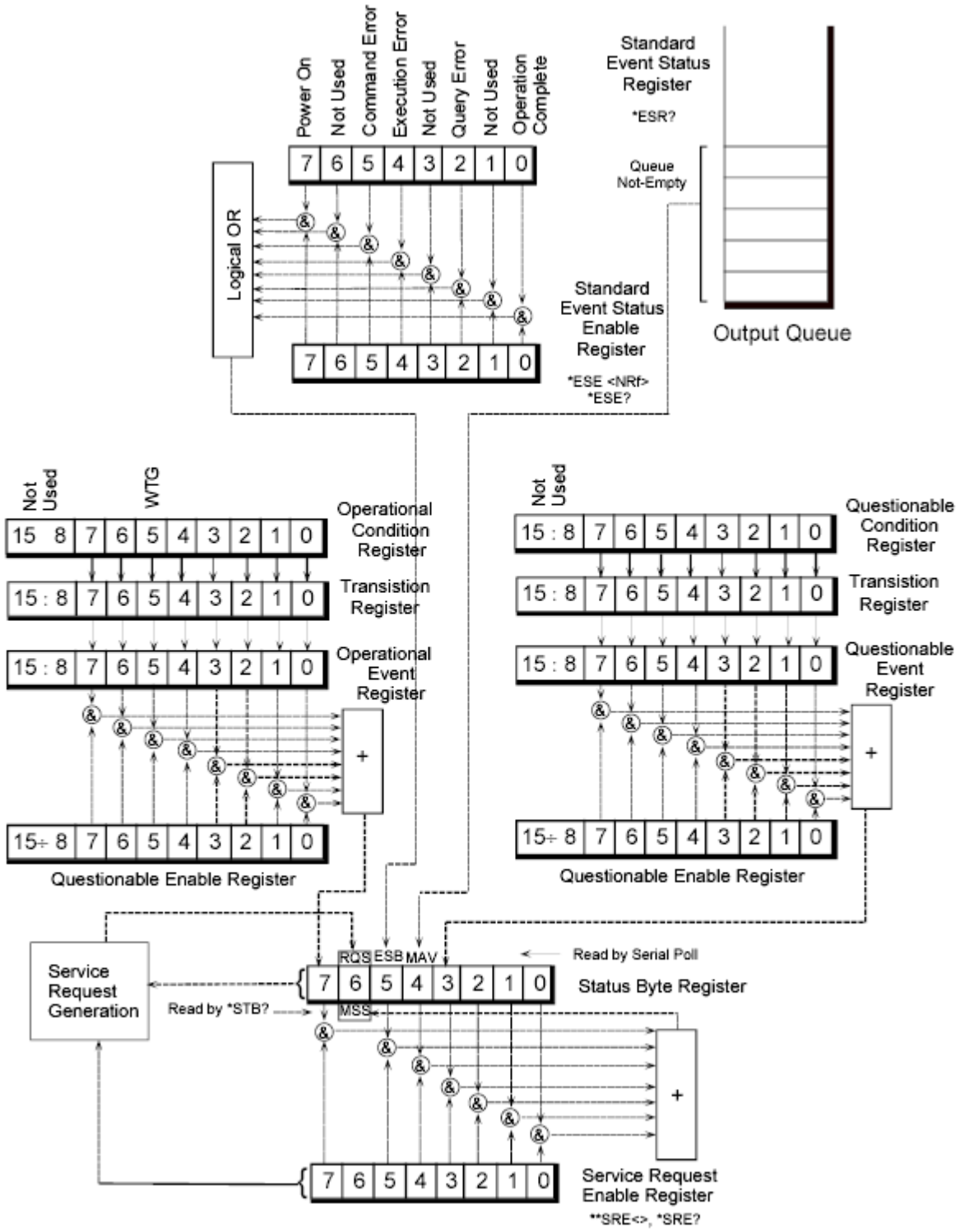


Figure 3-5 Status Reporting Structure

Event Registers

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

Each event register contains eight or sixteen bits. When the register is read, its response is a decimal number that is the sum of the binary bit weights of the bits that are logical 1s.
e.g., 23 decimal = 0001 0111 binary

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

At power turn-on or when the 6075 is reset, all of the enabling registers are set to their saved values if *PSC (Power-on-clear) is not enabled. The *ESE and *SRE commands are used to change the enabling register values. New values are saved with the 488.2 *SAV 0 command. **Table 3-8** lists the recommended settings for the Event Status and Status Byte enable registers.

Standard Event Status Register

The Standard Event Status Register reports events that are common to all 488.2 devices. This includes events such as self test errors, command errors, execution errors, power on and operation complete. The Power-on event occurs at power turn-on and can be used to signal a power off-on occurrence.

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

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

Questionable Registers

The Questionable Registers has no use in the 6075 module. This register is implemented so that the module will conform to the SCPI requirements. If used, the Questionable Condition Register reports the status of its input signals. A logic 1 means that the signal is asserted. The Questionable Transition Register filters the inputs and passes only the enabled state changes to the Questionable Event Register. The Questionable Event Register bits becomes true (1) when the positive transition bit is enabled and the associated condition register bit makes a 0 to 1 transition or when the negative transition bit is enabled and the associated condition register bit makes a 1 to 0 transition. When both transitions are selected for the same bit, the corresponding Questionable Event Register bit sets whenever the digital input changes state.

The Questionable Enable Register enables set Event bits to be summarized in the Status Byte Register. The following example enables bit 0:

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

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

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

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

Operational Condition Registers

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

In the 6075, the Operational Condition Register only reports the WTG (Waiting for Trigger) status. The WTG bit is true when the 6075 has been armed and is waiting for a trigger. The following command will query the Operational Condition Register:

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

Output Queue

The Output Queue is used by the 6075 to send IEEE 488.2 messages back to the bus controller. These messages are device responses and 6075 responses to queries sent to the unit by the bus controller. The Output Queue reports a '1' in bit 4 of the Status Byte Register when it contains a message(s) to be read by the bus controller. Reading the contents of the Output Queue clears its summary bit. The Output Queue is read by a Word Serial message. The Output Queue can be tested for presence of data by testing the DOR bit in the response register or by testing the MAV bit in the STB register. If the Output Queue is not read before sending another query, its contents will be lost and an error reported.

Status Byte Register

The 6075 generates a service request whenever any of the enabled bits in the Status Byte Register become true. The Status Byte Register may be read by the *STB? query. The Status Byte Register is enabled by setting the corresponding bits in the Service Request Enable Register with the *SRE command. e.g.

*SRE 40 'Sets the SRE Register to 0010 1000 which enables just the Event Status and Questionable summary bits to generate a service request

Command	488.2 Function
*CLS	CLEAR STATUS - Clears all event registers summarized in the status byte, except for "Message Available," which is cleared only if *CLS is the first message in the command line.
*ESE <value>	EVENT STATUS ENABLE - Sets "Event Status Enable Register" to <value>, an integer between 0 and 255. <value> is an integer whose binary equivalent corresponds to the state (1 or 0) of bits in the register. If <value> is not between 0 and 255, an Execution Error is generated. EXAMPLE: decimal 16 converts to binary 00010000 which sets bit 4 to a logical 1. This would enable the “Execution Error” condition in the Standard Event Status Register to propagate to set bit 5 (ESB) in the Status Byte Register
*ESE?	EVENT STATUS ENABLE QUERY - Returns the <value> of the "Event Status Enable Register" set by the *ESE command. <value> is an integer whose binary equivalent corresponds to the state (1 or 0) of bits in the register.
*ESR?	EVENT STATUS REGISTER QUERY - Returns the <value> of the "Event Status Register" and then clears it. <value> is an integer whose binary equivalent corresponds to the state (1 or 0) of bits in the register.
*IDN?	IDENTIFICATION QUERY – Returns its identification code as four fields separated by commas. These fields are: manufacturer, model, six-digit serial number and hardware-firmware version and date. A typical IDN response is: "Racal Instruments, 6075-4, S/N 00101, Rev. 00.00 Version 00.04.12".

Command	488.2 Function
*OPC	OPERATION COMPLETE COMMAND - Causes the Unit to generate the operation complete message in the Standard Event Status Register when all pending selected Unit operations have been finished.
*OPC?	OPERATION COMPLETE QUERY - Places an ASCII character 1 into the Unit's Output Queue when all pending selected Unit operations have been finished.
*PSC<value>	POWER-ON STATUS CLEAR - Controls the automatic power-on clearing of the SRE and ESE registers. *PSC 0 allows devices to restore the saved SRE and ESE values and to assert SRQ upon power turn-on. *PSC 1 enables the power-on clear and disallows a SRQ at power turn-on. The PSC commands save the SRE and ESE values in the E2PROM. Not implemented in Version 00.00.
*PSC?	POWER-ON STATUS CLEAR QUERY - Querys the PSC flag value. A returned value of 0 indicates the registers will retain their saved values, a returned value of 1 indicates the registers will be cleared. Not implemented in Version 00.00.
*RCL <value>	RECALL - Restores the state of Unit from a copy stored in its E2PROM by *SAV command. *RCL 0 restores the power on setting. <value> = 0. Not implemented in Version 00.00.
*RST	RESET - Unit restores its power-up state except that the state of IEEE-488 interface is unchanged, including: instrument address, Status Byte and ESR Register. Disables the trigger function.
*SAV <value>	SAVE - Saves current Unit configuration in the E2PROM. *SAV 0 saves the current setting as the new power on setting. <value>=0. Not implemented in Version 00.00.
*SRE <value>	SERVICE REQUEST ENABLE - Sets the "Service Request Enable Register" to <value>, an integer between 0 and 255. The value of bit six is ignored because it is not used by the Service Request Enable Register. <value> is an integer whose binary equivalent corresponds to the state (1 or 0) of bits in the register. If <value> is not between 0 and 255, an Execution Error is generated.
*SRE?	SERVICE REQUEST ENABLE QUERY - Unit returns the <value> of the "Service Request Enable Register" (with bit six set to zero). <value> is an integer whose binary equivalent corresponds to the state (1 or 0) of bits in the register.
*STB?	READ STATUS BYTE - Unit returns the <value> of the "Status Byte" with bit six as the "Master Summary" bit. <value> is an integer whose binary equivalent corresponds to the state (1 or 0) of bits in the register.
*TRG	TRIGGER - Unit initiates an action determined by the manufacturer. In the 6075, the action taken is to run the commands stored in a Program Buffer. The ArmTrigger command must be sent first to select which buffer will be run when triggered.
*TST?	SELF-TEST QUERY - Causes the Unit to run internal self-test. Test takes about 1 second and clears all internal buffers. Unit will not respond to any GPIB inputs while performing its internal self-test. A zero response indicates no failures. Other responses are listed in **Table 5-1**.

Command	488.2 Function
*WAI	WAIT-TO-CONTINUE - Prevents the Unit from executing any further commands or queries until the No-Operation-Pending flag is TRUE.

Table 3-7 IEEE 488.2 Command List

Saving the Enable Register Values

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

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

Note that the ESE and SRE commands must be on the same line or set prior to the *PSC 0 command to be saved. A later *PSC 1 command sets the PSC flag which will cause the ESE and SRE registers to be cleared at the next power turn-on and prevent a power on SRQ.

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

Table 3-8 Recommended ESE And SRE Bit Values

SCPI Conformance Information

The 6075 module accepts SCPI commands and command extensions to configure its digital interface, to set the data formats and to transfer data. The SCPI commands conform to SCPI Standard 1994.0 and provide an industry standard, self-documenting form of code that makes it easy for the programmer to maintain the application program.

Table 3-9 shows the 6075 SCPI command tree. The 6075 uses portions of the SCPI SYSTEM, STATUS, INSTRUMENT, SOURCE, and CALIBRATE subsystems. The commands follow SCPI's hierarchical 'tree like' structure which starts with a root keyword and branches out to the final action keyword. Each command can be used as a query except where noted. The SCPI commands in the 6075 are not case sensitive. The portion of the command shown in capitals denotes the abbreviated form of the keyword. Either the abbreviated ("short form") or whole keyword ("long form") may be used when entering a complete command. Bracketed keywords are optional and may be omitted. There must be a space between the command and the parameter or channel list.

e.g., SYSTEM:COMMUNICATE:SERIAL:BAUD 9600

is the same as

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

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

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

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

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

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

Short Form Commands

The 6075 also accepts short hand commands which invoke the same action as do the corresponding SCPI commands. The short form commands are one to five characters long and are not case sensitive. The short form commands have the advantage of reduce the typing load on the programmer when operating the interface from a terminal or from a terminal emulation program. In a time critical program they reduce VXI bus traffic and the 6075's parser execution time.

Table 3-9 shows the short hand commands alongside the SCPI commands. Their parameter form is the same as that of the SCPI commands. A space is required between the short form command and its parameter. The SCPI command descriptions in **Table 3-10** also apply to the appropriate short form commands.

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

e.g. C 3
is the same as
INST:NSEL 3

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

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

Keyword	Parameter	Short Hand	Notes
SYSTem			
:COMMunicate			
:SERial			
[:RECeive]			
:PROTocol	[ASYNC] | SDLC	PROT	
:BAUD	50-460800 [9600]	BAUD	
:BITS	5-8 [8]	BITS	
:SBITs	[1] | 2	SBITS	
:PARity			
[:TYPE]	EVEN | ODD | [NONE]	PAR	
:ECHO	ON | [OFF]	ECHO	
:LOOPback	[OFF] | INT | MON	LOOP	
:MODE	[232] | 422FD | 422HD | 485	MODE	
:HANDshake	ON | [OFF]	HAND	
:RXCLK	[INT] | EXT	RXCLK	
:TXCLKOUT	[IN] | OUT	TXCLK	
:TERMination	ON | [OFF]	TERM	
:DMA	[ON] | OFF	DMA	
:BUFfer			
:SIZe	8 - [65535]		
:DATA	D16 | [D32]		
:CLOCK			
:FREQuency	value		
:DATA	string	DATA	Send data
:DATA?	string	DATA?	Read data
:DATAT	string	DATAT	Send termintd data
:UPdate		UP	Update UART
:ERRor?	(0, “No error”)		Reads error
:VERSion?	(1994.0)		Reads version
INSTrument			
:NSELect	1 - 8 [1]	C	Channel select
STATus			
:OPERational		WTG	status in bit 5
[:EVENt]?			
:CONDitional?			
:ENABle	0-7FFF [0]		
:PTRansistion	0-7FFF [All 1s]		
:NTRansistion	0-7FFF [0]		
:OPERational			
[:EVENt]?			
:CONDitional?			
:ENABle	0-7FFF [0]		
:PTRansistion	0-7FFF [All 1s]		
:NTRansistion	0-7FFF [0]		
:PRESet			

SOURCE			Pulse Output
:PULSe			
:PERiod	.0001-3. [.01] sec		
:WIDTh	.0001-3. [.0025] sec		
INITate			Trigger Control
[:IMMeditate]		TI	Enable 1 trigger
:CONTinuous	ON | [OFF]	TC	Enable continuous triggers
ABORt		TA	Disable triggers
CALibrate			Setup
:DATe	mm/dd/yyyy		
:DEFault			
:IDN	string		
:LOCK	ON | [OFF]		
:MANufacturer	0-4095 [4073]		VXI Mfgr Number
:MODel	0-4095 [539]		VXI Model Number

Table 3-9 6075 SCPI Command Tree - Asynchronous Mode

Notes:

1. Parameter enclosed by [ ] - denotes factory default
2. Parameter enclosed by ( ) - denotes power on default
3. SCPI name ends with ? - denotes query only
4. Unless otherwise noted SCPI command is also a query
5. Keyword enclosed by [ ] - denotes optional use
6. Only a configuration command that has one of its parameters enclosed by [ ] can change its parameter setting and have this setting stored in the 4803's Flash (with the *SAV command).
7. The format for a SCPI list is (@1,2, n) or (@ 1:n). There must be a space between the @ and the first number and parenthesis are required. A list of numbers is separated by commas or uses a colon to denote a range of numbers.
8. Numeric entries conform to IEEE-488.2 section 7.7.2.4 for decimal numeric parameters.
9. ASCII formatted data is a series of decimal values (0-255) for each byte separated by commas. e.g. 64, 132, 8
10. The CAL:DATe commands stores the CAL:IDN and CAL:DATe parameters in the 4803's Flash.
11. The CAL:DEFault command resets the Flash memory to it factory settings. Caution - All user settings will be overridden by this command.
12. Most parameters can be output in various numeric formats (radix). The parameters with decimal 0-255 value ranges may also be output as HEX using #h00-#hFF or Binary using #b00000000-#b11111111. Conversely, the parameters shown with HEX (#h) values can also be output in Decimal

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

:RXCLK	INT	Selects internal 7.3728 MHz clock for ASYNC baud rates or external RX clock. The EXT selection is valid only when RS-422 or RS-485 signals are selected. RXCLK is 16x for Asynchronous mode, 1x for Synchronous mode. Values are INT and EXT.
:TXCLKOUT	IN	Enables TX clock transmitter Valid only when RS-422 or RS-485 signals are selected. TXCLK is generated from the RXCLK signal. TXCLK is 16x for Asynchronous mode, 1x for Synchronous mode. Values are IN and OUT.
:TERMination	OFF	ASYNC command connects RS-422 RX lines and RS-485 TX/RX lines to a termination network. Values are OFF and ON.
:DMA	OFF	ASYNC command enables DMA transfer of received data to the FDC receive buffers. Use DMA only for high-speed, continuous data transfer. DMA on disables EOM character checking and Message counting. Value is ON and OFF.
:BUFfer	-	Identifies FDC buffer settings.
:DATA	D32	Sets FDC buffer access word size. Values are D16 and D32.
:SIZe	65535	Sets the number of bytes the FDC receive buffer can hold before the 6075 switches buffers (switch point). Will override MESSage setting. Values are 8 to 131072 bytes (for 6075-4, 65536 for the 6075-8) and should include the 8 FDC header bytes.
:CLOCK		Identifies Clock settings.
:FREQuency	7372800	Sets oscillator frequency in Hz for baud rate calculations. Default is the 6075's internal clock frequency. The clock setting must be changed if an external RXCLK source is selected or if the internal oscillator is changed. Values are 100 000 to 10 000 000. (spaces shown for clarity)
:DATA		Diagnostic command that sends a test serial data message without adding any termination characters to the message. Uses the VXI Word Serial Message to send or receive serial data. Maximum data size is 1024 characters. The DATA command will time out if it cannot place data in the UART's FIFO, set the Execution Error bit in the ESR Register and display Err on the front panel. Value is a string of 8-bit characters.
:DATA?		DATA? is an ASYNC mode query that enables the selected channel's receiver. A query when no data is present returns a 'no data available' response. The serial channel should be re-initialized before using it with the FDC buffers. DATA? does not check for EOM character.
:DATAT		ASYNC diagnostic command that sends a test serial data message and automatically appends a CR LF termination sequence to the message. Uses the VXI Word Serial Message to send or receive serial data.. Maximum data size is 1024 characters. The DATAT command will time out if it cannot place data in the UART's FIFO, set the Execution Error bit in the ESR. and display Err on the front panel. Value is a string of 8-bit characters. DATAT? query is identical to DATA?

:ERRor?		Queries SCPI error value. The standard response for no error is 0, "No error"
:VERsion?		Queries SCPI version. Response is 1995.0
INSTrument		Starts Instrument Subsystem
:NSELect	1	Selects channel number. Values are 1 to 8 for Octal units and 1-4 for Quad units.
STATus		Starts Status Reporting Subsystem
:OPERational		Identifies Operational registers.
:QUEStionable		Identifies Questionable registers.
[:EVENt?]		Returns contents of the event register associated with the command.
:CONDition?		Returns contents of the condition register associated with the command.
:ENABle	0	Sets the enable mask which allows the true conditions in the associated event register to be reported in the summary bit.
:PTRansition	32767	Sets the positive transition register. Value is 0 to 32767.
:NTRansition	0	Sets the negative Transition register. Value is 0 to 32767.
:PREset		Sets the selected Enable Register, PTR and NTR registers to their default values (0, 32767 and 0 respectively) so the VXI module detects positive changes.
SOURce		Starts Pulse Subsystem
:PULSe		Identifies Pulse Output.
:PERiod	0.01	Sets Pulse period in seconds. Value is 0.0001 (100 microseconds) to 3.000 seconds.
:WIDTh	0.0025	Sets Pulse width in seconds. Value is 0.0001 (100 microseconds) to 3.000 seconds.
CALibrate		Starts calibrate branch
:IDN <string>		Sets user IDN message. String is up to 72 characters and consists of four fields (manufacturer, model code, serial number and firmware revision) separated by commas. e.g. Racal Instruments, 6075, S/N 012345, Rev 00.00 Ver 00.04.17.
:DATe <date>		Saves IDN message and date. Date is in mm/dd/yyyy format.
:DATe?		Queries the calibration date. The response is 00/00/0000 for factory default settings.
:DEFault		Sets Flash to factory settings.
:LOCK	0	Disables configuration commands when On. Disabled commands are not recognized as valid SCPI commands. Values = 0|1 or OFF|ON. **Table 1-4** indicates which commands can be locked.
:MANufacturer	4091	Sets VXI Manufacturer Number in the VXI ID Register. The Manufacturer Number is obtained from the SCPI Consortium. Default value is 4091 (Racal Instruments). Value is 0 to 4095.
:MODel	607	Sets module Model Number in the VXI Device Type Register. Default is 607 for the 6075-4, and 608 for the 6075-8. Value is 0 to 4095.

Table 3-10 SCPI Command Reference

Programming Guidelines

Overview

The 6075 has four or eight serial channels that can independently transmit and/or receive data. Each channel has its own baud rate generator, transmit and receive circuits. To send a channel command, the user must first select the channel with the INST subsystem command. This will allow the SYSTEM configuration commands to go to the correct channel. Next set the channel protocol to select Asynchronous or SDLC operation. Next set the baud rate, mode, signal type, character format and other parameters as required to configure the channel. When done, use the *SAV 0 command to save the current configuration for all of the channels as the module's power-on default setting. Note that changing any serial parameter resets the channel's UART to the idle state.

As an example, the following commands would set channel 3 to 19,200 baud rate and saves the change:

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

VXI Programming Requirements

It is important that the programmer understand the difference between a Word Serial Command and a Word Serial Message when working with the 6075. Refer **to Interface Programming Concepts** and **Word Serial Commands** sections earlier in this chapter for the definitions and command examples. Consult your Slot 0 Controller's instruction manual or contact the manufacturer's application support people for the directions on how to send VXI Word Serial Commands.

Check your VXI Slot 0 Controller for A32 Address Space Capability. This capability is required to address the 6075's FDC buffers. Most GPIB-VXI modules do not have this capability.

Testing an Asynchronous Channel

Connect a serial device or terminal to the channel to be tested. A PC running Hyperterminal can also be used as the terminal. Use an interactive control program to send the setup commands to the 6075. Setup the channel for the desired baud rate, mode etc.

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

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

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

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

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

```
*CLS
INST:NSEL 1
SYST:COMM:SER:LOOP OFF
SYST:COMM:SER:DATAT?          'read 'no data available'
                              response
SYST:COMM:SER:DATAT test message
                              'sends a message
                              to the device
                              'device sends a
                              response to the VXI-6075
SYST:COMM:SER:DATAT?          'read device response
                              message
```

Transmitting Data

Fast Data Channel buffers are the normal way serial data is transferred between the VXI Commander and the 6075. The Fast Data Channel buffers are used as pairs to pass serial data in one direction. The VXI controller places data in one buffer while the 6075 owns the other buffer. Bits in the Fast Data Channel header define the number of bytes in the FDC buffer and who 'owns' the buffer.

Four fast data channels are required for each 6075 serial channel. The Fast Data Channel buffers are initialized by the Resource Manager function in the Slot 0 Controller. Refer to the **Fast Data Channel Usage** section earlier in this chapter for information about the FDC buffers and their organization in the 6075. Refer to Fast Data Channel Addendum for detailed information on how to setup and use the FDC buffers.

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

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

The sample program in the Fast Data Channel Addendum includes the function XFDC_OpenWriteBuffers which shows how to use these commands. Refer to **Figure A-1** in the Fast Data Channel Addendum for detailed instructions on initiating the FDC buffers.

To transmit data, the user writes data into the first transmit FDC buffer and sets the header bit to RDY. Transmission of the first message starts as soon as a buffer with data is passed to the 6075. The user can 'fill' the second FDC transmit buffer while the 6075 is transmitting data from the first FDC buffer. When the transmitter empties the first FDC buffer, its header WDY bit is set and the user can again write data into it. The FDC buffers must be used in their A/B sequence to avoid getting out of sync with the 6075 transmitter.

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

Receiving Data

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

The SYST:COMM:SER:BUFF:SIZE command sets the point (in bytes) at which the receive buffer becomes 'full' and the receiver switches to the other buffer. This command only sets the buffer's usable area, it does not change the number of bytes allocated to the buffer. When the data in the receive buffer reaches the 'full' point, the 6075 can generate a VXIbus interrupt to notify the user that the buffer is 'full'. The user should set this point so that the receive buffers are swapped when a fixed number of messages have been received. The byte count in the Size Command is all of the bytes in the FDC buffer including the FDC header.

The following commands will set the 6075's channel 2’s FDC receive buffers to receive 48 characters.

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

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

Receiving Messages with an EOM Character

The 6075 can receive data as messages rather than as bytes. This method is easier to implement than byte counting when the messages are asynchronous and of varying lengths. The EOM character recognition increments the Message counter. The FDC buffers are swapped when the desired number of messages have been received.

To use EOM character detection, leave the buffer size set to the default value of 65,536 bytes. Set the EOM character to the desired termination character. For multiple messages in the FDC buffer, set the MESSage parameter to the desired number. DMA must be turned OFF for the EOM character to function properly.

For example, the following commands set the 6075 to terminate messages with a carriage return (13) character and to accumulate 10 messages in the buffers.

```
INST:SEL 2
SYST:COMM:SER:DMA 0
SYST:COMM:SER:EOM 13
SYST:COMM:SER:MESS 10
```

SDLC Mode Operation

In the SDLC mode, the 6075 sends and receives data as packets of bytes. To use the SDLC mode, the baud rate and frame size need to be set. The frame size is the number of bytes transmitted. Frame size is an even number between 2 and 30. The transmit bufers should be loaded with exact multiple of the frame size.

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

When receiving SDLC data, the 6075 will save a frame of received data in the FDC buffer. Next the 6075 appends a 16-bit status word from the Serial Interface controller to the data in the FDC buffer. The bit pattern for the Status Word is shown in **Figure 3-6**. This process repeats until the buffer reaches the switch point. The buffer size for setting the switch point is :

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2nd BE	1st BE			Rx Residue			ShortF CV Type	Exited Hunt	Idle Rcvd	Break /Abort	Rx Bnd	CRCE /FE	Abort /PE	Rx Over	Rx Avail

Figure 3-6 Serial Controller Status Word

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

Clock Selection

A 6075 serial channel can use several sources for its transmit and receive clocks. The default setting is to use the internal 7.3728 MHz oscillator as the clock source for the channel's baud rate generator. In the RS-232 mode, this is the only source that you can use. In the RS-422 and RS-485 modes, an external RX clock and/or an external TX clock can be used in place of the internal clock. Also in the RS-422 and RS-485 modes, the TX clock transceiver can be set to provide a clock output for an external device.

The following commands will set the 6075's channel 3 for external clock inputs.

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

Timing Pulse Output

Eight channel 6075s have an RS-422 Pulse output that can be programed from 10 kHz down to once every three seconds. The pulse width is also programmable. Program pulse width first when reducing the period. Program period first when increasing the pulse width.

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

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

This page intentionally left blank

Chapter 4

THEORY OF OPERATION

What's In This Chapter

This chapter provides a description of the design and theory of operation for the 6075 modules. It is intended to help in the understanding of the module operation and as an aid in using the module

Basic Operation

The 6075 is a VXIbus module with four or eight programmable serial interfaces and clock pulse output for interfacing with devices that use asynchronous or synchronous serial data. The serial interfaces are independently programmable and can be set to operate with RS-232, RS-422 or RS-485 signals. The 6075 has an internal oscillator for generating all common baud rates and can work with an external clock for synchronous data. The 6075 uses VXI Fast Data Channels to achieve continuous operation on all channels without tying up the VXI backplane.

As mentioned, the 6075 module is available in two versions. The four channel version has four 9-pin DE connectors on the front panel. The eight channel version has two 37-pin connectors on the front panel. The eight channel version includes a programmable pulse output that can be used to trigger events or other test functions. The module's FDC transmit algorithm has been modified to match the end user's software.

6075 Block Diagram Description

A simplified Block Diagram of the 6075 is shown in **Figure 4-1**. The 6075 is assembled from two printed circuit board assemblies, a VXI Interface Card and a Serial Interface card. The Model VXI Interface Card is a 386EX processor based interface with 4 Mbyte of DRAM memory that contains the FDC buffers. The Serial Interface Card contains the serial channel drivers, UARTs and FIFOs. A 16-bit 386EX expansion bus is used to communicate with Serial Interface Card.

Communication with the VXIbus is via a 16 or 32 bit wide data bus. Word serial messages are handshaked in on a word-by-word basis, parsed and executed. The Word serial messages are used to configure and test the serial channels, to initialize or close the FDC channels and to control the pulse output. Transmit data is placed in the FDC buffers located in the VXI Interface Card's DRAM memory. FDC buffer accesses place the 386EX processor in a hold condition while the commander directly accesses buffers the VXI Interface Card's memory.

Access and buffer ownership is controlled by management bits in the FDC buffer headers. When the FDC buffer is passed to the module, the 386 transfers data from the FDC buffer into the TX FIFO in the UART. It keeps the TX FIFO full until the message is completely transmitted.

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

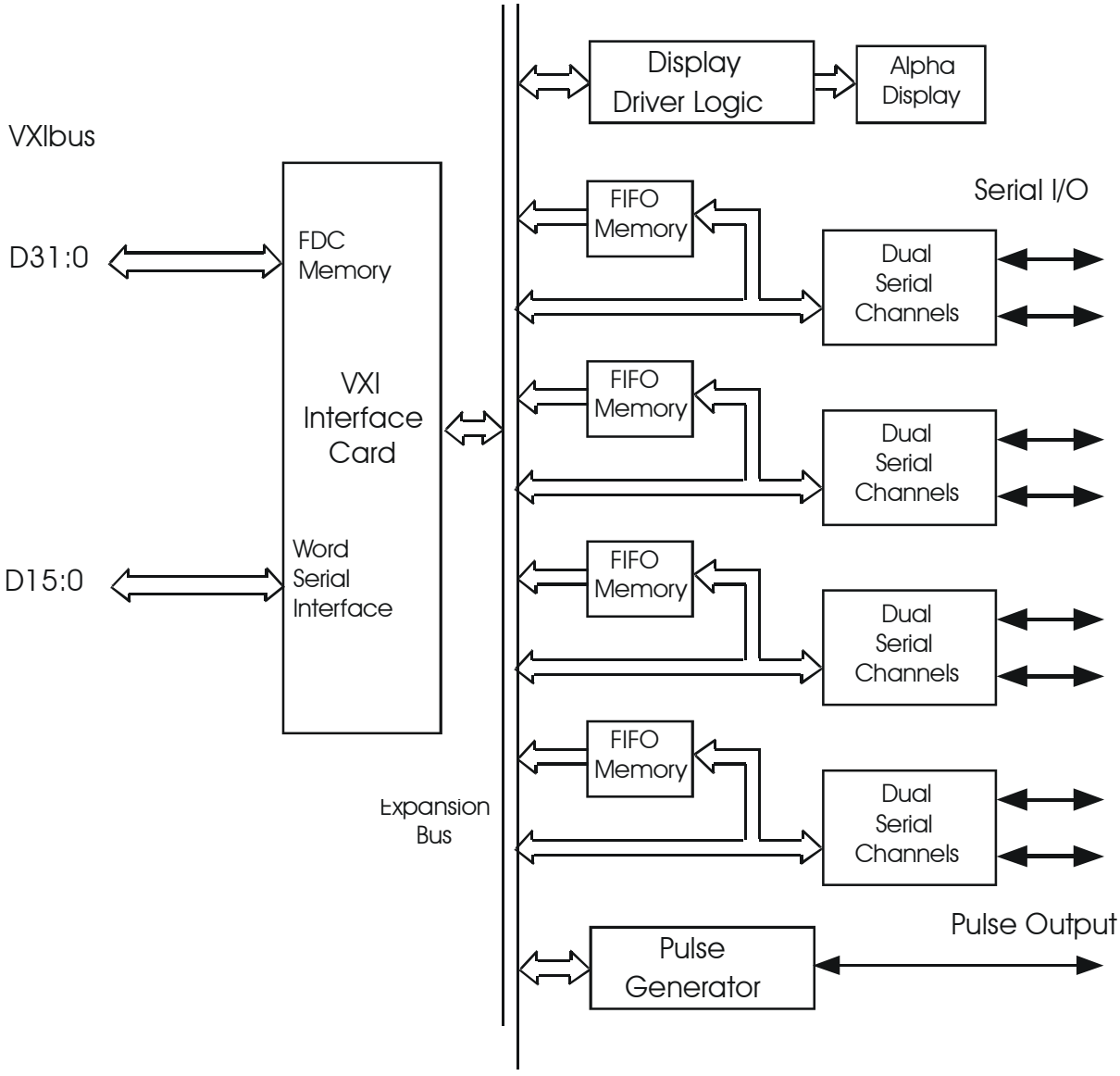


Figure 4-1, 6075 Simplified Block Diagram

VXI-Interface Card Description

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

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

The VXI Interface Card also contains two 16-bit registers that provide static digital I/O signals for the Serial Interface Card. The static signals can be programmed as inputs or outputs. In the VXI Interface Card's application within the 6075, one register outputs higher order address bits to select the serial channel, and to control the display generator logic. The second register is used to read the serial channel's interrupt status. The VXI Interface Card communicates with most of the logic on the Serial Interface Card via a 16-bit expansion bus. The expansion bus lets the program read or write to the Serial Interface at I/O addresses 300 HEX to 37E HEX.

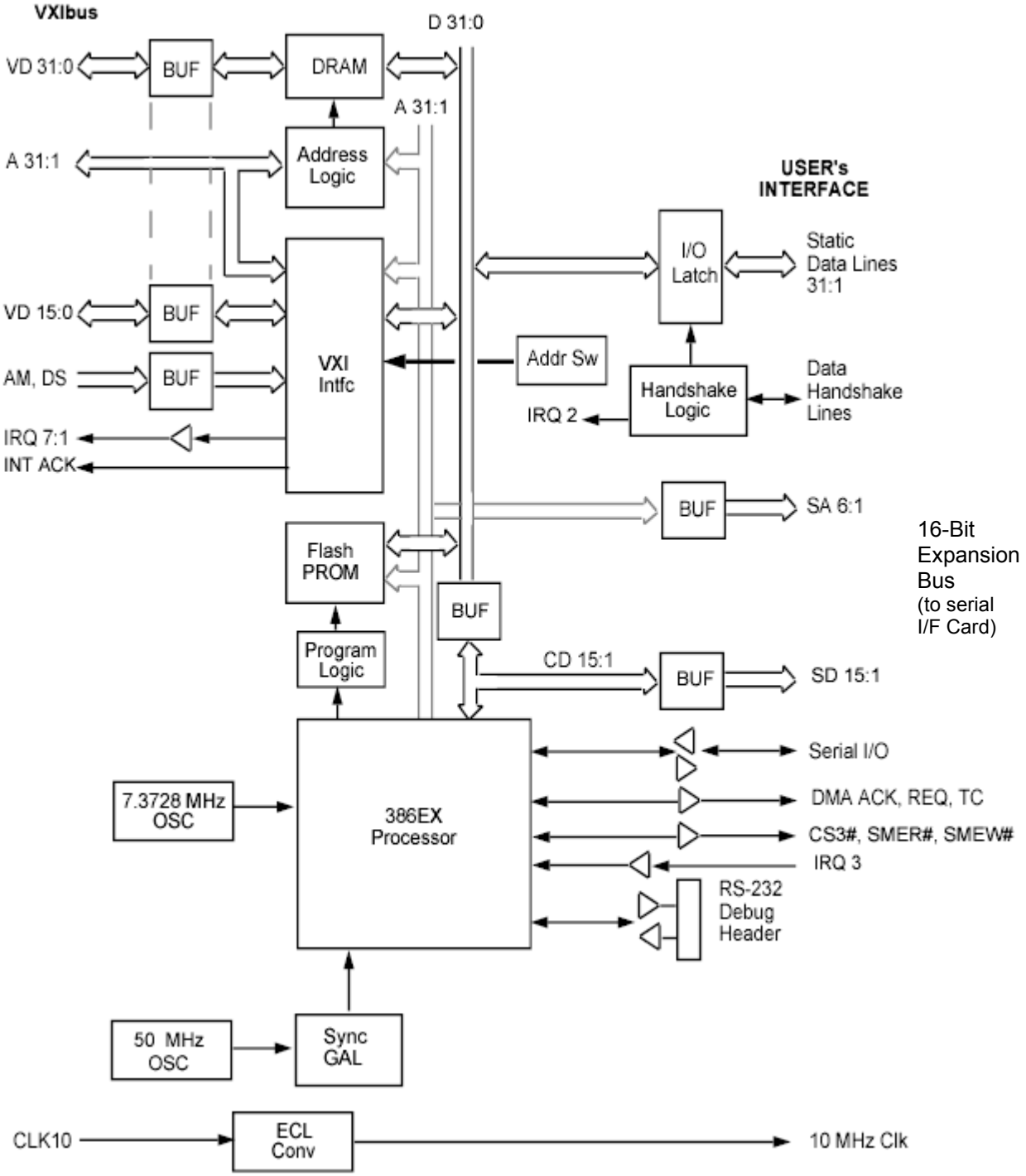


Figure 4-2, VXI Interface Card Block Diagram

Serial Interface Board

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

typical Dual Serial Channel.

The standard value for the reference oscillator is 7.3728 MHz. This frequency is a 16x clock for 460,800 baud and can be divided down into all of the standard rates from 230,400 to 50 baud. The 6075's parser accepts any baud rate up to 460,800 and selects the closest divider ratio to produce the commanded baud rate. At high baud rates, the divider steps are very coarse and the resulting frequency may not be very close to the commanded value. At lower baud rates (below 10,000), the divider steps are much finer and the resulting frequency is very close to the commanded value. For an exact nonstandard baud rate, replace the 6075's reference oscillator with one that can be divided down to the desired baud rate and change the oscillator frequency setting.

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

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

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

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

reference oscillator or from an external source.

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

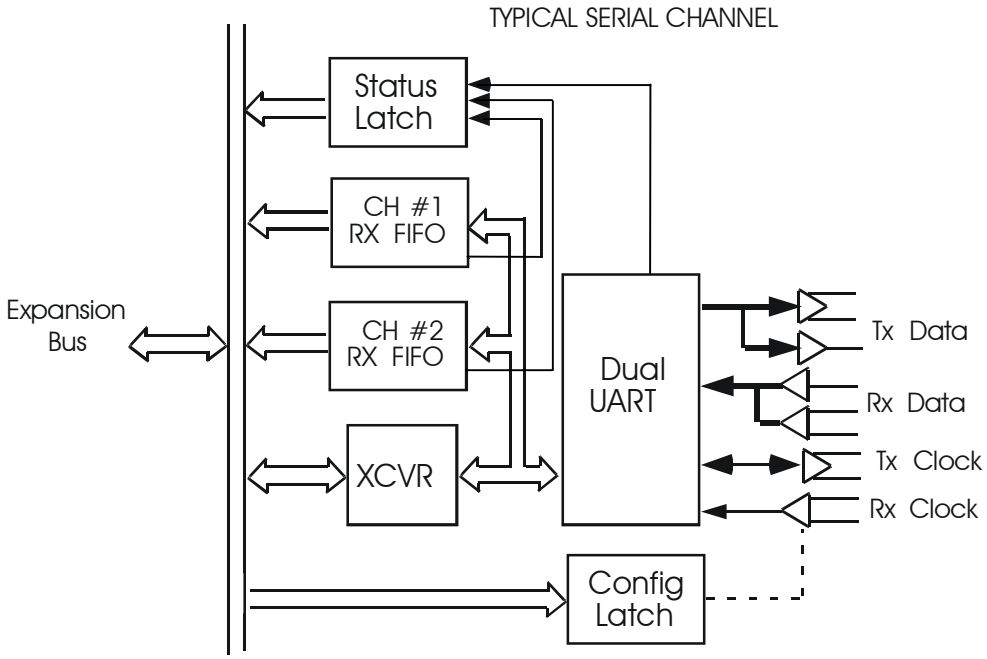


Figure 4-3, VXI 6075 Serial Interface Board Block Diagram

Chapter 5

MAINTENANCE AND PERFORMANCE CHECKS

Maintenance Overview

This section describes the maintenance, troubleshooting and repair procedures for the 6075 Serial Interface card. The 6075 does not require periodic calibration and has no internal adjustments.

WARNING

The procedures described in this section are for use only by qualified service personnel. Many of the steps covered in this section may expose the individual to potentially lethal voltages that could result in personal injury or death if normal safety precautions are not observed

CAUTION

ALWAYS PERFORM DISASSEMBLY, REPAIR AND CLEANING AT A STATIC SAFE WORKSTATION.

Troubleshooting Procedures

Self Test Failures

The 6075 has a built in self test routine which is designed to detect basic circuit faults. The self test routine checks the Flash, DRAM and to some extent, the interface chips. Any self test failure is indicated by a blank front panel display or by a FAIL indication on the front panel display.

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

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

Troubleshooting Guide

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

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

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

Table 5-1 Troubleshooting Guide

This page was Left Intentionally Blank

Chapter 6
PRODUCT SUPPORT

Product Support

Racal Instruments has a complete Service and Parts Department. If you need technical assistance or should it be necessary to return your product for repair or calibration, call 1-800-722-3262, or call 949-859-8999 and ask for Customer Support. You may also contact Customer Support via E-Mail at:

helpdesk@racalinstruments.com

If you require parts to repair the product at your facility, call 1-800-722-3262, or 949-859-8999 and ask for the Customer Service Department.

When sending your instrument in for repair, complete the form in the back of this manual. For worldwide support and the office closest to your facility, refer to the Support Offices section on the following page.

Reshipment Instructions

Use the original packing material when returning the instrument to Racal Instruments for servicing. The original shipping carton and the instrument's plastic foam will provide the necessary support for safe reshipment.
If the original packing material is unavailable, wrap the chassis in ESD shielding material and use foam to surround and protect the instrument.
Reship in either the original or a new shipping carton.

Support Offices

Racal Instruments, Inc.
4 Goodyear St., Irvine, CA 92618-2002
Tel: (800) RACAL-ATE, (800) 722-2528, (949) 859-8999; FAX: (949) 859-7139

Racal Instruments, Ltd.
480 Bath Road, Slough, Berkshire, SL1 6BE, United Kingdom
Tel: +44 (0) 1628 604455; FAX: +44 (0) 1628 662017

Racal Systems Electronique S.A.
18 Avenue Dutartre, 78150 LeChesnay, France
Tel: +33 (1) 3923 2222; FAX: +33 (1) 3923 2225

Racal Systems Elettronica s.r.l.
Strada 2-Palazzo C4, 20090 Milanofiori Assago, Milan, Italy
Tel: +39 (0)2 5750 1796; FAX +39 (0)2 5750 1828

Racal Elektronik System GmbH.
Technologiepark Bergisch Gladbach, Friedrich-Ebert-Strasse, D-51429 Bergisch Gladbach, Germany
Tel.: +49 2204 8442 00; FAX: +49 2204 8442 19

Racal Instruments, Ltd.
Unit 5, 25F., Mega Trade Center, No 1, Mei Wan Road, Tsuen Wan, Hong Kong, PRC
Tel: +852 2405 5500, FAX: +852 2416 4335

REPAIR AND CALIBRATION REQUEST FORM

To allow us to better understand your repair requests, we suggest you use the following outline when calling and include a copy with your instrument to be sent to the Racal Repair Facility.

Model_____ Serial No._____ Date_____

Company Name_____ Purchase Order #_____

Billing Address_____

City

State/Province Zip/Postal Code Country

Shipping Address_____

City

State/Province Zip/Postal Code Country

Technical Contact_____ Phone Number ( )_____

Purchasing Contact _____ Phone Number ( )_____

1. Describe, in detail, the problem and symptoms you are having. Please include all set up details, such as input/output levels, frequencies, waveform details, etc.

2. If problem is occurring when unit is in remote, please list the program strings used and the controller type.

3. Please give any additional information you feel would be beneficial in facilitating a faster repair time (i.e., modifications, etc.)

4. Is calibration data required? Yes No (please circle one)

Call before shipping Ship instruments to nearest support office.

Note: We do not accept "collect" shipments.

This page intentionally left blank

Appendix A
VXIBUS FDC ADDENDUM

Introduction

This addendum provides background information about the VXI-10 Fast Data Channel, features of the FDC that are specific to the 6075 and its use of Fast Data Channel transfer protocol and the Expanded FDC Command Set. Several sample programs are also included which show how to fully utilize the FDC architecture in a real application.

While this addendum provides recommended channel initialization and data passing procedures, it is not a replacement for the FDC Specification. Users of the Fast Data Channel transfer protocol are urged to read the VXI-10 Fast Data Channel Specification.

Description

Depending upon the module type and usage, commands and/or data may be transferred between the Slot 0 Controller and the module via Fast Data Channel buffers in the module's A32 memory space. In most of Racal Instruments VXIbus modules, the FDC channels operate as *A/B buffer pair*s and in *Stream Transfer* mode for enhanced throughput. In this mode, two FDC channels (Buffers A and B) are used for a data transfer direction and four channels for a bidirectional data port. i.e. a serial port or a GPIB Interface. The number of FDC channels in a module is determined by multiplying the number of interface channels or ports in the module by four. Other uses of the FDC protocol, such as passing static data, may require different numbers of FDC channels or different modes so it is best to check the exact FDC channel count in the specifications section of your module. Currently, Racal Instruments 6075 VXIbus modules support Fast Data Channel (FDC) data transfer per VXIbus Specification VXI-10, Rev. 2.10 using the Standard FDC Word Serial Commands and Racal Instruments' expanded command set that handles up to 32 FDC channels. Refer to the VXIbus Specification VXI-10, Rev. 2.10 for detailed information on the standard commands and other Fast Data Channel configurations.

Background Information

Fast Data Channel Advantages

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

VXI Shared Memory concept was the VXIbus Consortium's first attempt to provide high speed data transfer. The Shared Memory protocols were too complex and it was not a practical solution. In 1995, the VXIbus Consortium defined the new VXI Fast Data Channel specification. The Fast Data Channel concept has the speed advantages of shared memory without the software overhead. Data transfer with the Fast Data Channel approaches the maximum transfer rate for the VXIbus. With Word Serial messages, reading a data character requires the controller to make a minimum of four VXIbus accesses - read the response register, send byte request, read the response register again and then read the data byte. To read a register, the Controller must make a read to see that the data is present and then read the data. With the Fast Data Channel, the Controller reads the word count and then makes one read per word. Each word can contain four data bytes which again quadruples the data transfer rate.

VXI Fast Data Channels can be used to transfer data and/or commands between the Commander and the Servant module. The number, size and organization of the channels and their use are up to the module designer. Channels (or buffers) can be used singularly or in pairs. Control bits in the FDC channel header (see **Tables A-1** and **A-2**) define who 'owns' the buffer and if the buffer is full or empty. The control bits prevent data contention since only the channel owner can read or write data in the buffer.

Fast Data Channel Usage on the 6075

On the 6075 VXI module, the VXI-10 Fast Data Channel protocol is used as a way to improve the data transfer rate across the VXIbus. Because this module has serial data ports, FDC channels are organized to maintain a continuous flow of high-speed data. A pair of FDC channels is used for each data transfer direction. The channel pairs usually operate as A/B type buffer pairs in the FDC Stream Transfer mode to shuttle data between the module and the Slot 0 Controller. One device fills the first buffer while the other module is emptying the second buffer. The buffers are exchanged and the process repeats until all of the data has been transferred.

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

The number and usage of the FDC channels in a module is established when the module is designed. The number of channels in a module can be determined by checking the module's specifications or by querying the module with the *FDC Supported* command. It is important to note that FDC channel ownership is indicated by the *RDY* and *WDY* bits in the FDC header. If both bits are cleared, the channel belongs to the servant and should not be written to or read by the commander. Reading the buffer will not cause an error, but the data in the buffer may not be valid. If the *RDY* bit is set, the commander owns the buffer and it contains data to be read. If the *WDY* bit is set the commander also owns the buffer and may write data into it. FDC channels are passed to the commander in two ways. When a buffer is ready to be passed, the *RDY* or *WDY* bit is set in the header. If FDC Events are enabled for the channel, an interrupt is generated to the commander containing the FDC channel number being passed. If the FDC Events are not enabled, the commander must poll the FDC header and check the bits to detect the passed buffer.

If the *Channel Initialize* command response indicates that the *Pass Command* is supported, the *Enable Passed Buffer* command must be sent to the servant and FDC channels are passed to the servant with the *Pass Buffer* command. If the *Pass Buffer* command is not supported, the channels are passed by clearing the *RDY* or *WDY* bit and the servant must poll the bits to detect the channel pass. Most modules require the *Pass Buffer* command to minimize the internal processor overhead. The FDC channels must be initialized before they are used to establish data direction for the Slot 0 Controller. The recommended initialization process is shown in **Figure A-1**. **Figure A-1** resets the channels at the beginning of the initialization in case they had been previously used.

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

Transmit Channels

Transmit channels are initialized with the FDC *Transfer to Servant* command to establish the data direction from commander to servant for the channel or channel pair. The Pair and Stream mode bits should be set in this command if these modes are required by the module. Once a transmit channel is initialized with the *Channel Initialize* and *Transfer to Servant* commands the buffer will be owned by the commander. The commander should check the FDC header for ownership before writing to the buffer. Then the first data message may be written to the buffer and passed to the servant with the *Passed Buffer* command. When the buffer is received by the servant, it will transmit the data. When writing data into a FDC buffer, the number of bytes in the data buffer area must be written into the Data Size word in the header area. This count includes all message bytes plus those used in any sub headers.

Transmit channels (buffers) are passed to the Slot 0 Controller when they are empty. The Controller passes them back to the VXI module when they are full or have data in them. The amount of data in the buffer is up to the user and is specified in the FDC

channel header. The only restrictions are that the amount of data cannot exceed the maximum buffer size and that the Controller must fill alternate buffers. The Slot 0 Controller must keep the module supplied with a new buffer before the old one is completely transmitted to maintain continuous data flow.

Receive Channels

Receive channels are initialized with the FDC *Transfer to Commander* command to establish the data direction from servant to commander for the channel or channel pair. The Pair and Stream mode bits should be set in this command if these modes are required by the module. Once a receive channel is initialized with the *Channel Initialize* and *Transfer to Commander* commands, the buffer will be owned by the servant. At this point the receiver will start receiving data. When the number of bytes in the buffer reaches the switch point or if the *Switch Buffer* command is received, the buffer will be passed back to the commander and the receiver will switch to the other channel if it is available. When a FDC buffer is passed, the number of bytes in the data buffer section is specified by the Data Size word in the header area. This includes all message bytes and any module sub header information. The number of bytes for the switch point is set by the user with the SCPI *SYST:RECEIVE:BUF:SIZE* command. The switch point byte count includes the eight header area bytes plus all of those to be used in the data section when the buffer is 'full'. This command must be given before the *Transfer to Commander* command to be effective with the first buffer. If the other buffer is not ready then any new incoming data will be lost. A VXI event interrupt will occur when the buffers are switched, if interrupts were enabled. Use the *Go to Idle* command to terminate data transfer and to get last buffer with any remaining received data.

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

Fast Data Channel (FDC) Memory Map

Channel Memory Maps

A memory map for a typical FDC channel is shown **in Tables A-1** and **A-2**. Both tables contain the same information but show it by VXIbus word width. **Table A-1** is organized as 16-bit words and **Table A-2** is organized as 32-bit words. The first eight bytes

contain the FDC Channel Header information. The header contains the control bits for determining buffer ownership and the Data Size word that defines the space used in the data buffer. The FDC Channel data buffer starts with byte 8. Modules may use the data buffer space in the way that best fits their application. Byte numbers are in Motorola order for correlation with the VXI-10 Specification.

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

Table A-1 16-Bit Wide FDC Memory Map

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

Table A-2 32-Bit Wide FDC Memory Map

Notes:

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

Fast Data Channel Buffer Definitions

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

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

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

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

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

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

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

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

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

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

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

FDC DATA BUFFER: Memory area for the data in the FDC Channel Buffer. The Data Buffer starts with Byte 8 (Word 4 for 16-bit wide buffers or Word 2 for 32-bit wide buffers) and ends in Word n. The organization of the data buffer is module dependent. Note: Refer to the Fast Data Channel Specification VXI-10 for additional information on the usage of the bits in the Channel Header.

Fast Data Channel Initialization Sequence

Figure A-1 shows the recommended initialization sequence for a VXI module with a streaming channel pair. When initializing channel pairs, the commands are only sent to the lower channel of each pair. It is strongly recommended that the user include the Go-to-Idle-Immediate and Channel-Close commands in the initialization sequence so the sequence can initialize a previously opened channel.

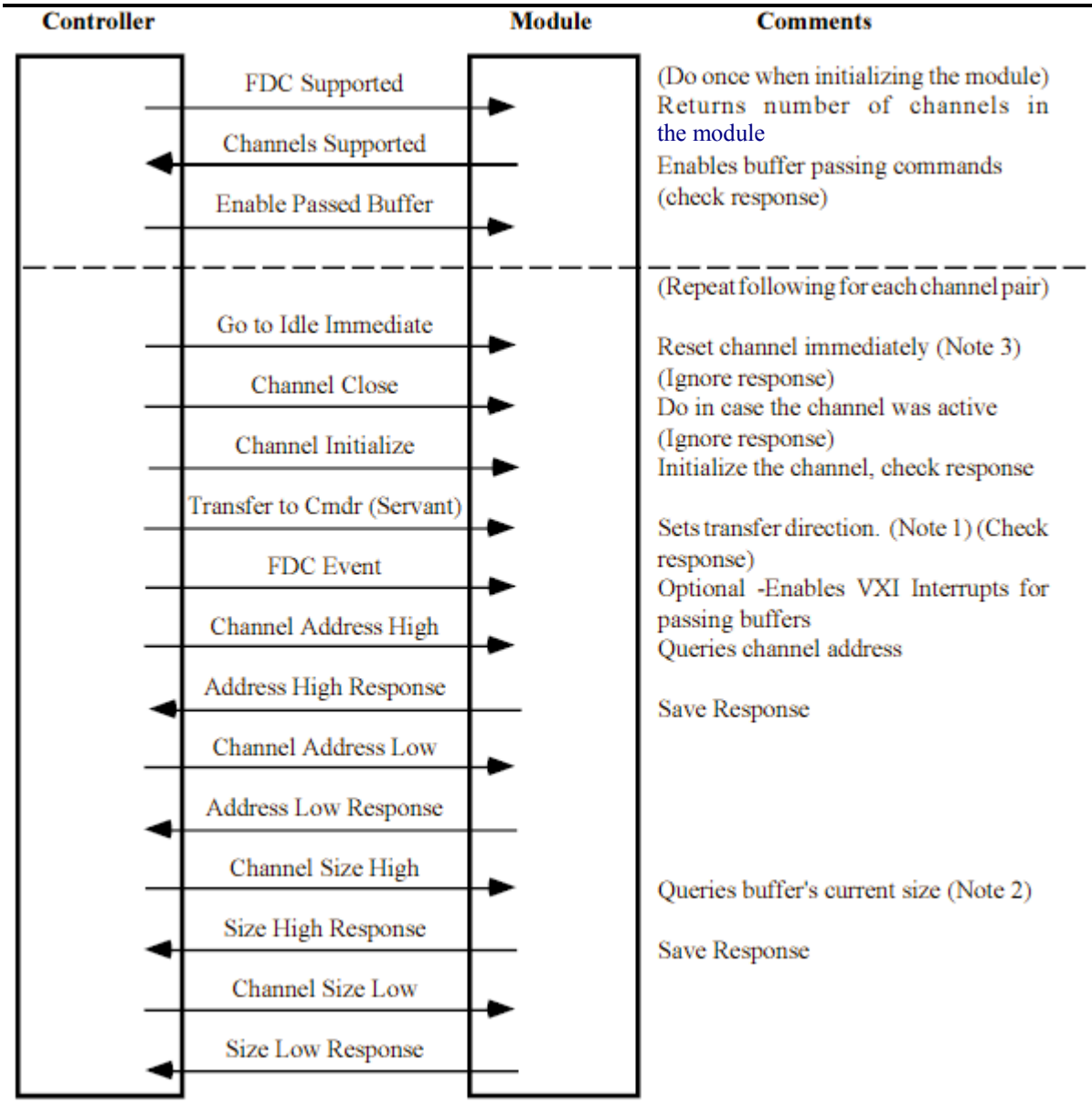


Figure A-1 Recommended FDC Initialization Sequence

Notes:

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

Fast Data Channel Passing Sequence

Transfer to Servant

Figure A-2 shows a sequence for swapping a streaming channel pair. In normal operation, buffers of data are passed from the data source to the data destination, alternating between the even and odd FDC channel. The END flag is used to indicate the end of a block of data, not the termination of the data transfer. The sequence in **Figure A-2** is for the Transfer to Servant direction. The steps below the dotted line are repeated until terminated by the Go-to-Idle command.

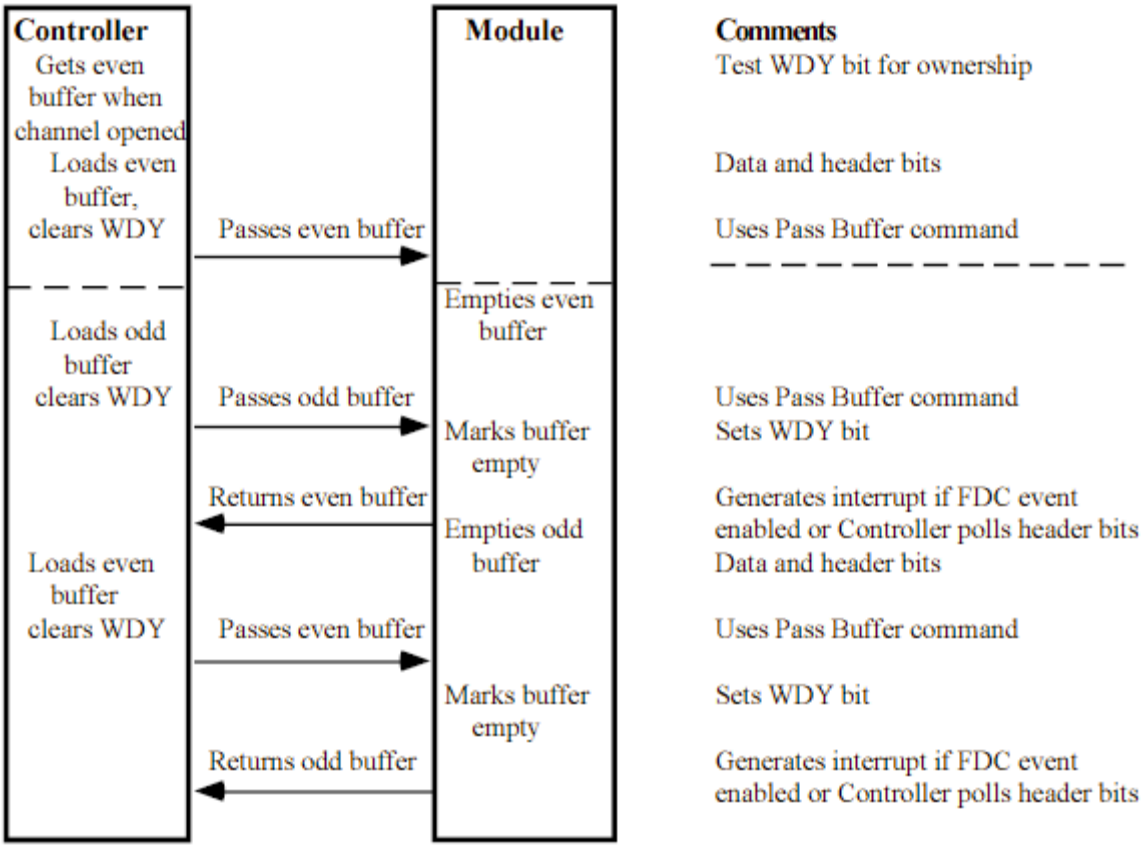


Figure A-2 Transfer to Servant Buffer Passing Sequence

Transfer to Commander

Figure A-3 shows a sequence for swapping a streaming channel pair when the data direction is to the Commander. In normal operation, buffers of data are passed from the data source to the data destination, alternating between the even and odd FDC channel. Transfer takes place when the amount of data in the buffer reaches the switch point set by the S*YST:RECEIVE:BUF:SIZE* command. The steps below the dotted line are repeated until terminated by the Go-to-Idle command.

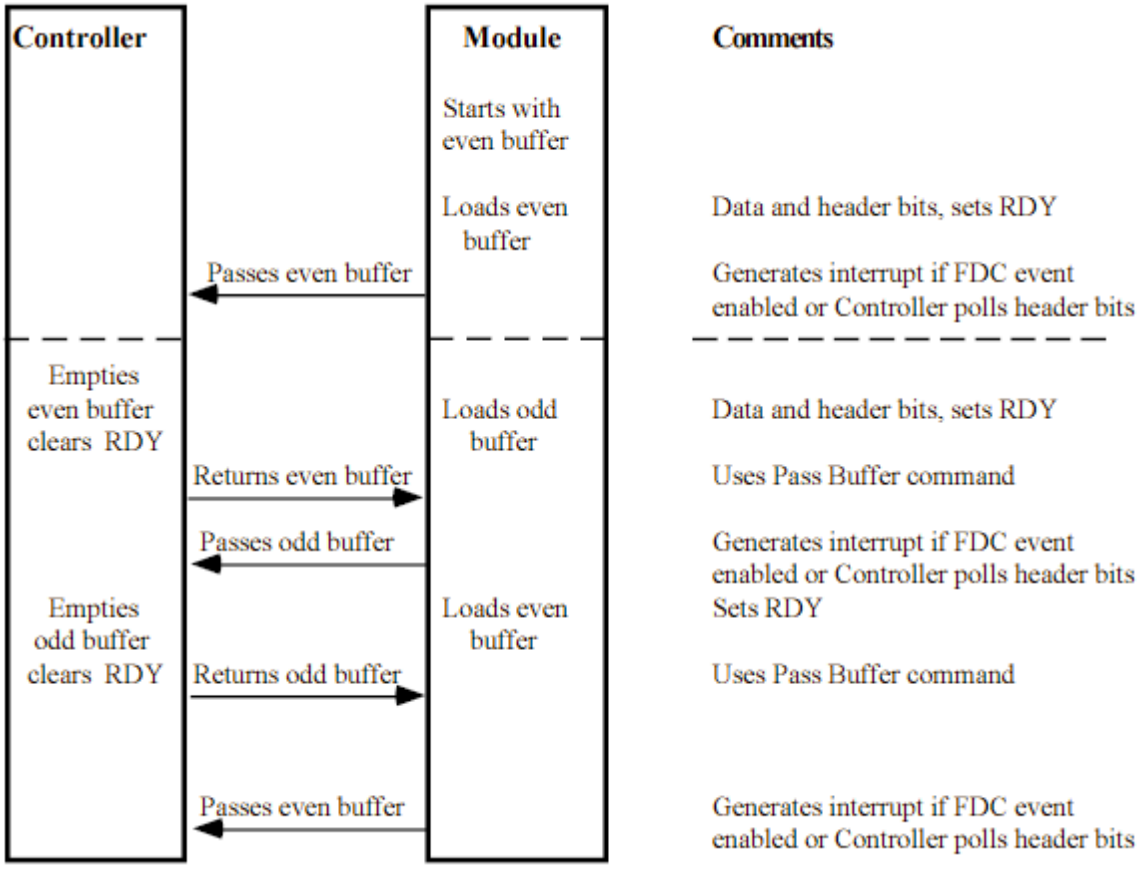


Figure A-3 Transfer to Commander Buffer Passing Sequence

FDC Command Reference

Fast Data Channel (FDC) Command Summary

The 6075 with Fast Data Channel respond to the following FDC Standard commands and expanded commands. The expanded commands are the same as the standard VXI-10 commands, except for the command codes and the channel number field which is expanded to 5 bits to allow for up to 32 channels. Both command sets are supported for software compatibility. To avoid conflicts, only one set of commands should be used in an application program. Refer to the Fast Data Channel Specification VXI-10, Rev. 2.10 for detailed information on the usage of these commands and explanation of their parameters. **Table A-1** lists the standard and Racal Instruments expanded FDC commands

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

Table A-3 Fast Data Channel Standard and Racal Instruments Expanded Commands

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

Fast Data Channel (FDC) Command Descriptions

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

Channel Initialize (0x9F90)

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

Bit #

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

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

Expanded Channel Initialize (0x7E40)

Bit #

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

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

Bit #

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

Status:

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

Channel Address Commands

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

Channel Address Low (0x9F00)

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

Bit #

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

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

Expanded Channel Address Low (0x7C00)

Bit #

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

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

Channel Address High (0x9F80)

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

Bit #

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

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

Expanded Channel Address High (0x7E00)

The syntax of the Expanded 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
0	1	1	1	1	1	1	0	0	0	0	Channel #				

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

Bit #

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

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

Channel Size Commands

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

Channel Size Low (0x9F08)

The syntax of the 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 response for this command is the same as for the following command.

Expanded Channel Size Low (0x7C20)

The syntax of the Expanded 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
0	1	1	1	1	1	0	0	0	0	1	Channel #				

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

Channel Size High (0x9F88)

The syntax of the 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 #		

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

Expanded Channel Size High (0x7E20)

The syntax of the Expanded 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
0	1	1	1	1	1	0	0	0	0	1	Channel #				

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

Bit #

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

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

Go to Idle (0x9FB0)

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

Bit #

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

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

Expanded Go to Idle (0x7EC0)

This command is used to terminate a Stream transfer on an Expanded FDC channel. It may also be used to force termination of a Normal transfer transfer on an Expanded FDC channel.

Bit #

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

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

Bit #

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

Channel Close (0x9F98)

This command is used to close an FDC channel.

Bit #

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

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

Expanded Channel Close (0x7E60)

This command is used to close an Expanded FDC channel.

Bit #

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

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

Bit #

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

Status:

F - No Errors
7 - Channel is still active and will be returned to idle at the close of the current block of data or attempt to close a channel that is not open.
6 - Cannot idle a closed channel or close a channel that is not open.
5 - FDC Channel number not supported.
4 - Command not allowed.

Transfer to Servant (0x9FC0)

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

Bit #

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

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

Expanded Transfer to Servant (0x7F00)

This command is used to initiate a data block transfer from the commander to the servant on an Expanded FDC channel.

Bit #

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

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

Bit #

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

Transfer to Commander (0x9FE0)

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

Bit #

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

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

Expanded Transfer to Commander (0x7F80)

This command is used to initiate a data block transfer from the servant to the commander on an expanded FDC channel.

Bit #

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

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

Bit #

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

Status:

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

FDC Event (0x9FA0)

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

Bit #

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

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

Expanded FDC Event (0x7E80)

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

Bit #

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

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

Bit #

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

Status:

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

FDC Supported (0x9F1F)

This command is used to determine FDC support.

Bit #

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

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

Bit #

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

Expanded FDC Supported (0x7C7F)

This command is used to determine Expanded FDC support.

Bit #

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

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

Bit #

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

Enable Passed Buffer (0x9F18)

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

Bit #

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

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

Expanded Enable Passed Buffer (0x7C60)

This command requests that the passed buffer command be utilized by the servant for an Expanded FDC channel.

Bit #

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

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

Bit #

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

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

Passed Buffer (0x9F10)

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

Bit #

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

This command does not have a response.

Expanded Passed Buffer (0x7C40)

This command informs the servant that the commander has passed the buffer to the servant for an Expanded FDC channel.

Bit #

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

This command does not have a response.

Expanded Switch Buffers (0x7C80)

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

Bit #

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

This command does not have a response.

Fast Data Channel Events (Interrupt Response)

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

Bit #

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

Expanded Fast Data Channel Events (Interrupt Response)

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

Bit #

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

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

Example Software Using FDC

This appendix includes several examples from the XIplug&play driver which illustrate how to use the FDC channels to transmit and receive characters.

The example functions shown in this appendix are taken from the VXIplug&play driver which ships with the module. This driver is also available for download from the Racal Instruments web site at http://www.racalinstruments.com/downloads.

The example driver functions make use of other calls in the VXIplug&play driver. The support functions are listed following the set of examples for reference. The entire set of example and support functions are contained in the VXIplug&play driver for the 6075.

Example #1: Transmit Data Using FDC

This function may be used directly or as an example. This function opens the FDC channel pair for the transmit channel specified. It then loads the data and passes control of the FDC buffer to the instrument, at which time the 6075 will actually transmit the data.

```
/*=========================================================================*/
/* Function: transmit data                                                 */
/* Purpose:  This function may be used directly or as an example           */
/*           This function opens the FDC channel pair for the transmit     */
/*           channel specified.  It then loads the data and passes control */
/*           of the FDC buffer to the instrument, at which time the 6075   */
/*           will actually transmit the data.                              */
/*=========================================================================*/
ViStatus _VI_FUNC ri6075_transmit_data (ViSession vi, ViInt16 channel,
                                        ViChar _VI_FAR tx_buffer[],
                                        ViInt32 tx_len)
{
    ViStatus error;
    ViInt16 xmit_fdc_buffer;
    ViInt32 id;
    ViInt32 xmit_fdc_buffer_size;
    ViInt32 xmit_blk_len;
    ViChar *tx_buf_ptr;

    error = viGetAttribute (vi, VI_ATTR_USER_DATA, &id);
    if (error < 0) return( error );

    error = ri6075_int32_range (id, 0, RI6075_MAX_INSTR, RI6075_USER_DATA_ERROR);
    if (error < 0) return error;

    error = ri6075_int_range (channel, RI6075_CHAN_1,
                              RI6075_CHAN_8, RI6075_CHANNEL_PARAM);
    if(error < 0) return error;
```

```
if (tx_len < 1)
    return( RI6075_TRANSMIT_LENGTH_PARAM );

/* check for attempt to use channels 5 through 8 with 6075-4 model */
if (max_channels[id] < channel)
    return RI6075_INCORRECT_MODEL_FOR_CHANNEL;

// the even (lower-numbered) channel is all that is needed since
// the odd channel always = even channel + 1
xmit_fdc_buffer = 2 + ((channel - 1) * 4);

// open the write buffer as pair/streaming mode
// this will open up the pair of FDC channels
error = ri6075_XFDC_OpenWriteBufrPS (vi, xmit_fdc_buffer);
if (error < 0) return error;

// get the transmit FDC buffer size (this should be 64K or 128K)
// the buffer size depends on the hardware installed
error = ri6075_XFDC_GetChanSize(vi, xmit_fdc_buffer, &xmit_fdc_buffer_size);
if (error < 0) return error;

// leave space for the 8 byte FDC header
xmit_fdc_buffer_size -= RI6075_FDC_HEADER_LENGTH;

// ensure the number of bytes to transmit will fit into 2 FDC buffers
if (tx_len > (2 * xmit_fdc_buffer_size))
    return( RI6075_XFDC_BUFFER_TOO_SMALL_ERR );

// if the buffer is too small, you must follow an example such as the
// double buffer one below.  That one provides an example of how to
// switch between the 2 FDC buffers for the transmit channel to output
// as much data as needed


// output as much data as will fit into the first FDC buffer
tx_buf_ptr = tx_buffer;

if (tx_len > xmit_fdc_buffer_size) {

    // load the first data into the first FDC buffer
    error = ri6075_XFDC_TransmitBufr(vi, xmit_fdc_buffer, tx_buf_ptr,
                                         xmit_fdc_buffer_size);
    if (error < 0) return error;

    // use the odd FDC buffer as well
    xmit_fdc_buffer += 1;

    xmit_blk_len = tx_len - xmit_fdc_buffer_size;
    tx_buf_ptr += xmit_fdc_buffer_size;

    // load the rest of the data into the 2nd FDC buffer
    error = ri6075_XFDC_TransmitBufr(vi, xmit_fdc_buffer, tx_buf_ptr, xmit_blk_len);
    if (error < 0) return error;
}
else { // it all fits into 1 FDC buffer
```

```
        // load all the data into the first FDC buffer
        error = ri6075_XFDC_TransmitBufr(vi, xmit_fdc_buffer, tx_buffer, tx_len);
        if (error < 0) return error;
    }


    return( error );
}
```

Example #2: Open a Receive Channel to Begin Receiving Characters

This function may be used directly or as an example. This function opens the FDC channel pair for the receive channel specified. Opening the FDC channel pair must be done in order to collect receive data. After this function has been executed, the specified channel can begin to collect data.

```
/*=============================================================================*/
/* Function: open receive channel                                              */
/* Purpose:  This function may be used directly or as an example               */
/*           This function opens the FDC channel pair for the receive          */
/*           channel specified.  Opening the FDC channel pair must be done     */
/*           in order to collect receive data.  After this function has        */
/*           been executed, the specified channel can begin to collect         */
/*           data.                                                             */
/*=============================================================================*/
ViStatus _VI_FUNC ri6075_open_recv_channel (ViSession vi, ViInt16 channel,
                                            ViInt32 bufferSize)
{
    ViStatus error;
    ViInt32 id;
    ViInt16 recv_fdc_channel;

    error = viGetAttribute (vi, VI_ATTR_USER_DATA, &id);
    if (error < 0) return( error );

    error = ri6075_int32_range (id, 0, RI6075_MAX_INSTR, RI6075_USER_DATA_ERROR);
    if (error < 0) return error;

    error = ri6075_int_range (channel, RI6075_CHAN_1, RI6075_CHAN_8, RI6075_CHANNEL_PARAM);
    if(error < 0) return error;


    error = ri6075_int32_range(bufferSize, 1, 131072, RI6075_BUFFER_SIZE_PARAM);
    if (error < 0) return error;


    error = ri6075_set_recv_buffer_size (vi, channel, bufferSize);
    if (error < 0) return error;

    // open the receive channel's FDC buffers in paired/stream mode
    recv_fdc_channel = (channel - 1) * 4;

    error = ri6075_XFDC_OpenReadBufrPS (vi, recv_fdc_channel);

    return( error );
}
```

Example #3: Transmit Using Double Transmit Buffers

This function is an example that shows how to use the driver to use double-buffered transmit mode. This example uses the FDC paired/stream mode to output a set of messages. Each message is loaded into alternating buffers of the card. This demonstrates how to load one buffer while the other is being transmitted. This example uses 38400 baud, no parity, 8 data bits, one stop bit. Each message is variable in length. The number of messages to output is set by the parameter.

```
/*=============================================================================*/
/* Function: Example Transmit double buffer                                    */
/* Purpose:  This function is an example that shows how to use the driver  */
/*           to use double-buffered transmit mode.  This example uses the  */
/*           FDC paired/stream mode to output a set of messages.  Each     */
/*           message is loaded into alternating buffers of the card.  This */
/*           demonstrates how to load one buffer while the other is being  */
/*           transmitted.  This example uses 38400 baud, no parity, 8 data */
/*           bits, one stop bit.  Each message is variable in length.  The */
/*           number of messages to output is set by the parameter          */
/*=============================================================================*/
ViStatus _VI_FUNC ri6075_example_transmit_buffers(ViSession vi, ViInt16 channel,
                                                  ViInt16 num_messages)
{
    ViStatus error;
    ViInt16 xmit_fdc_buffer, fdc_buffer_this_time;
    ViChar msg_text[100];
    ViChar append_text[90];
    ViBoolean am_loading_even_buffer;
    ViInt16 msg_count;
    ViInt16 chars_this_time;
    ViInt16 char_index;
    time_t ref_time, time_now;
    ViBoolean fdc_buffer_available;
    ViInt16 fdc_header;

    // ensure number of messages is valid
    error = ri6075_int_range (num_messages, 1, 1024, RI6075_NUM_MESSAGES_PARAM);
    if (error < 0) return error;

    // set parity = none
    error = ri6075_set_parity (vi, channel, RI6075_PARITY_NONE);
    if (error < 0) return error;

    // set data bits = 8
    error = ri6075_set_data_bits (vi, channel, RI6075_BITS_8);
    if (error < 0) return error;

    // set baud rate = 9600
    error = ri6075_set_xmit_baud_rate (vi, channel, 38400);
    if (error < 0) return error;

    // set stop bits = 1
    error = ri6075_set_stop_bits (vi, channel, 1);
    if (error < 0) return error;
```

```
    // enable end-of-message = linefeed (ASCII 10)
    error = ri6075_set_eom_character (vi, channel, (ViInt16) '\n');
    if (error < 0) return error;

    // use 1 message
    error = ri6075_set_message_count (vi, channel, 1);
    if (error < 0) return error;

    // enable DMA
    error = ri6075_set_DMA (vi, channel, RI6075_HANDSHAKE_OFF);
    if (error < 0) return error;


    // determine which FDC buffer will be associated with the channel
    // the even (lower-numbered) channel is all that is needed since
    // the odd channel always = even channel + 1
    xmit_fdc_buffer = 2 + ((channel - 1) * 4);

    // open the write buffer as pair/streaming mode
    // this will open up the pair of FDC channels
    error = ri6075_XFDC_OpenWriteBufrPS (vi, xmit_fdc_buffer);
    if (error < 0) return error;

    // now form the messages and output
    // always start loading the EVEN buffer first
    am_loading_even_buffer = VI_TRUE;

    for (msg_count = 1;  msg_count <= num_messages;  ++msg_count) {

        // form the message (for demo, make variable length)
        sprintf(msg_text, "Message # %4d: ", msg_count);

        // form a variable number of characters in the message (10 to 26)
        chars_this_time = 10 + (msg_count % 17);

        for (char_index = 0;  char_index < chars_this_time;  ++char_index)
            append_text[char_index] = 'A' + char_index;

        // terminate message with carriage return + linefeed (EOM character)
        append_text[char_index++] = '\r';
        append_text[char_index++] = '\n';
        append_text[char_index] = '\0';

        // concatentate the messages to form the final data
        strcat(msg_text, append_text);

        // form the FDC buffer number we are using this time
        fdc_buffer_this_time = xmit_fdc_buffer;
        if (am_loading_even_buffer == VI_FALSE)
            fdc_buffer_this_time += 1;

        // wait (up to 5 seconds) to ensure we have access to the FDC buffer
        ref_time = time (&time_now);

        fdc_buffer_available = VI_FALSE;

        while ((difftime (time_now, ref_time) < 10)
              && (fdc_buffer_available == VI_FALSE)) {
```

```
            // update current time
            time_now = time(&time_now);

            // get the FDC header and check for WDY bit, meaning buffer is available
            error = ri6075_XFDC_GetHeader (vi, fdc_buffer_this_time, &fdc_header);
            if (error < 0) return error;

            if (fdc_header & RI6075_XFDC_WDY_BIT)
                fdc_buffer_available = VI_TRUE;
        }

        // check for time-out
        if (fdc_buffer_available != VI_TRUE)
            return RI6075_XFDC_WDY_ERR;

        // load the buffer with the data
        error = ri6075_XFDC_TransmitBufr(vi, fdc_buffer_this_time, msg_text,
                                            (ViInt32) (strlen(msg_text)));
        if (error < 0) return error;


        // change the indication of which buffer we are loading for next time
        am_loading_even_buffer = !am_loading_even_buffer;
    }


    // Wait for the WDY bit to be set, meaning we can close the buffer
    fdc_buffer_available = VI_FALSE;

    while ((difftime (time_now, ref_time) < 10) && (fdc_buffer_available == VI_FALSE)) {

        // get the FDC header and check for WDY bit, meaning buffer is available
        error = ri6075_XFDC_GetHeader (vi, fdc_buffer_this_time, &fdc_header);
        if (error < 0) return error;

        if (fdc_header & RI6075_XFDC_WDY_BIT)
            fdc_buffer_available = VI_TRUE;
    }

    // we are done with the FDC channel(s), so close them
    error = ri6075_XFDC_ChannelClose (vi, xmit_fdc_buffer);
    if (error < 0) return(error);

    // if we get here, we are done
    return( error );
}
```

Example #4: Transmit and Receive From One or Two Channels

This function is an example that shows how to use the driver to use both a transmit channel and a receive channel. This example assumes the transmit channel is connected to the receive channel via a "null modem."

```
/*=========================================================================*/
/* Function: Example Transmit and Receive                                  */
/* Purpose:  This function is an example that shows how to use the driver  */
/*           to use both a transmit channel and a receive channel          */
/*                                                                         */
/*           This example assumes the transmit channel is connected to the */
/*           receive channel via a "null modem"                            */
/*=========================================================================*/
ViStatus _VI_FUNC ri6075_example_xmit_recv(ViSession vi,
                                           ViInt16 tx_chan,
                                           ViChar _VI_FAR *tx_buffer,
                                           ViInt32 tx_len,
                                           ViInt16 rx_chan,
                                           ViChar _VI_FAR *rx_buffer,
                                           ViPInt32 rx_len,
                                           ViInt16 max_wait_secs)
{
    ViStatus error;
    ViBoolean am_loading_even_buffer;
    ViBoolean am_reading_even_buffer;
    ViInt16 fdc_header;
    ViInt16 rx_fdc_buffer_even, rx_fdc_buffer_odd, rx_buffer_number;
    ViInt32 rx_fdc_buf_size;
    ViInt16 tx_fdc_buffer_even, tx_fdc_buffer_odd, tx_buffer_number;
    ViInt32 tx_fdc_buf_size;
    ViInt32 total_bytes_received, bytes_received_this_time;
    ViInt32 total_bytes_transmitted, bytes_transmitted_this_time;
    ViInt32 bytes_left_to_receive;
    ViChar *tx_buf_ptr;
    ViChar *rx_buf_ptr;
    ViInt16 delay_time;
    time_t ref_time, time_now;
    ViInt16 firstFDCchannel;
    ViBoolean bEvenFull, bOddFull, bEvenAvail, bOddAvail;
    ViInt32 nEvenCnt, nOddCnt;



    // ensure transmit channel is valid
    error = ri6075_int_range (tx_chan, RI6075_CHAN_1, RI6075_CHAN_8, RI6075_CHANNEL_PARAM);
    if(error < 0) return error;

    // ensure number of bytes to transmit is valid
    // allow multiple buffers
    error = ri6075_int32_range (tx_len, 1, (65536 * 10), RI6075_TRANSMIT_LENGTH_PARAM);
    if(error < 0) return error;

    // ensure receive channel is valid
    error = ri6075_int_range (rx_chan, RI6075_CHAN_1, RI6075_CHAN_8, RI6075_CHANNEL_PARAM);
```

```
if(error < 0) return error;

// ensure wait time is valid
error = ri6075_int_range (max_wait_secs, 1, 3600, VI_ERROR_PARAMETER8);
if(error < 0) return error;

// reset the module to get a known state
error = ri6075_reset(vi);
if (error < 0) return error;

// set parity = none
error = ri6075_set_parity (vi, tx_chan, RI6075_PARITY_NONE);
if (error < 0) return error;

error = ri6075_set_parity (vi, rx_chan, RI6075_PARITY_NONE);
if (error < 0) return error;

// set data bits = 8
error = ri6075_set_data_bits (vi, tx_chan, RI6075_BITS_8);
if (error < 0) return error;

error = ri6075_set_data_bits (vi, rx_chan, RI6075_BITS_8);
if (error < 0) return error;

// set baud rate = 9600
error = ri6075_set_xmit_baud_rate (vi, tx_chan, 38400);
if (error < 0) return error;

error = ri6075_set_xmit_baud_rate (vi, rx_chan, 38400);
if (error < 0) return error;

// set stop bits = 1
error = ri6075_set_stop_bits (vi, tx_chan, 1);
if (error < 0) return error;

error = ri6075_set_stop_bits (vi, rx_chan, 1);
if (error < 0) return error;

// turn DMA off to ensure odd byte count
error = ri6075_set_DMA(vi, tx_chan, 0);
if (error < 0) return error;

error = ri6075_set_DMA(vi, rx_chan, 0);
if (error < 0) return error;


// calculate the FDC buffers based on the channel numbers used
tx_fdc_buffer_even = 2 + ((tx_chan - 1) * 4);
tx_fdc_buffer_odd = tx_fdc_buffer_even + 1;

rx_fdc_buffer_even = (rx_chan - 1) * 4;
rx_fdc_buffer_odd = rx_fdc_buffer_even + 1;

// open the FDC read buffers first
error = ri6075_XFDC_OpenReadBufrPS (vi, rx_fdc_buffer_even);
if (error < 0) return error;

// now open the FDC write buffers
```

```
error = ri6075_XFDC_OpenWriteBufrPS (vi, tx_fdc_buffer_even);
if (error < 0) return error;

// get the transmit FDC buffer size (this should be 64K or 128K)
// the buffer size depends on the hardware installed
error = ri6075_XFDC_GetChanSize(vi, tx_fdc_buffer_even, &tx_fdc_buf_size);
if (error < 0) return error;

// leave space for the 8 byte FDC header
tx_fdc_buf_size -= RI6075_FDC_HEADER_LENGTH;


// now form the messages and output
// always start loading the EVEN buffer first
am_loading_even_buffer = VI_TRUE;
total_bytes_transmitted = 0;
tx_buf_ptr = tx_buffer;

am_reading_even_buffer = VI_TRUE;
total_bytes_received = 0;
rx_buf_ptr = rx_buffer;

ref_time = time(&time_now);

while ((difftime(time_now,ref_time) < max_wait_secs)
   && (total_bytes_transmitted < tx_len)) {

    // update the current time stamp
    time(&time_now);

    // now load up the transmit buffer and send the data
    tx_buffer_number = (am_loading_even_buffer) ? tx_fdc_buffer_even :
                                                  tx_fdc_buffer_odd;

    // check if the present transmit buffer can be written to
    error = ri6075_XFDC_GetHeader(vi, tx_buffer_number, &fdc_header);
    if (error < 0) return error;

    if (fdc_header & RI6075_XFDC_WDY_BIT) {

        // break the data into blocks that can fit into the FDC buffer
        bytes_transmitted_this_time = tx_len - total_bytes_transmitted;

        if (bytes_transmitted_this_time > tx_fdc_buf_size)
            bytes_transmitted_this_time = tx_fdc_buf_size;

        // load the data and pass to the instrument to xmit
        error = ri6075_XFDC_TransmitBufr(vi, tx_buffer_number, tx_buf_ptr,
                                                 bytes_transmitted_this_time);
        if (error < 0) return error;

        // update our counts
        total_bytes_transmitted += bytes_transmitted_this_time;
        tx_buf_ptr += bytes_transmitted_this_time;

        // NOT the flag so we ping-pong to the opposite transmit FDC buffer
        am_loading_even_buffer = !am_loading_even_buffer;
    }
```

```
    // check if the present read buffer is ready
    rx_buffer_number = (am_reading_even_buffer) ? rx_fdc_buffer_even :
                                                  rx_fdc_buffer_odd;

    error = ri6075_XFDC_GetHeader(vi, rx_buffer_number, &fdc_header);
    if (error < 0) return error;

    if (fdc_header & RI6075_XFDC_RDY_BIT) {

        // read buffer ready, read the data
        rx_fdc_buf_size = 131072;
        error = ri6075_XFDC_ReadBufr(vi, rx_buffer_number, rx_buf_ptr,
                                          rx_fdc_buf_size, &bytes_received_this_time);
        if (error < 0) return error;

        // update the data to indicate the total number of bytes read thus far
        total_bytes_received += bytes_received_this_time;
        rx_buf_ptr += bytes_received_this_time;

        // NOT the flag so we ping-pong to the opposite receive FDC buffer
        am_reading_even_buffer = !am_reading_even_buffer;
    }

}

// ensure all bytes have been transmitted
if (total_bytes_transmitted < tx_len)
    return( -9999 );    // not a real driver error code:  this is an example

// read the recieve block length
error = ri6075_XFDC_GetChanSize(vi, rx_fdc_buffer_even, &rx_fdc_buf_size);
if (error < 0) return error;

rx_fdc_buf_size -= RI6075_FDC_HEADER_LENGTH;

// we are done transmitting
// now we have to get all of the receive bytes

while ((difftime(time_now,ref_time) < max_wait_secs)
   && (total_bytes_received < tx_len)) {

    // update the current time stamp
    time(&time_now);

    // check if the present read buffer is ready
    rx_buffer_number = (am_reading_even_buffer) ? rx_fdc_buffer_even :
                                                  rx_fdc_buffer_odd;

    error = ri6075_XFDC_GetHeader(vi, rx_buffer_number, &fdc_header);
    if (error < 0) return error;

    if (fdc_header & RI6075_XFDC_RDY_BIT) {

        // read buffer ready, read the data
        error = ri6075_XFDC_ReadBufr(vi, rx_buffer_number, rx_buf_ptr,
                                         rx_fdc_buf_size, &bytes_received_this_time);
        if (error < 0) return error;
```

```
            // update the data to indicate the total number of bytes read thus far
            total_bytes_received += bytes_received_this_time;
            rx_buf_ptr += bytes_received_this_time;
            am_reading_even_buffer = !am_reading_even_buffer;


            // now send the "Switch FDC Buffers" command to the last receive buffer
            // and read what data is in it (if anything)
            rx_buffer_number = (am_reading_even_buffer) ? rx_fdc_buffer_even :
                                                          rx_fdc_buffer_odd;
        }

        // check for "not-full" buffer condition
        // this will happen when the receive buffer size is not evenly divisible
        // by the transmision length
        bytes_left_to_receive = total_bytes_transmitted - total_bytes_received;

        if (bytes_left_to_receive < rx_fdc_buf_size) {

            error = ri6075_read_channel_status (vi, rx_chan, &firstFDCchannel,
                                                    &bEvenFull, &nEvenCnt,
                                                    &bOddFull, &nOddCnt,
                                &bEvenAvail, &bOddAvail);


            if (bEvenFull || bOddFull
                    || ((nEvenCnt + nOddCnt) >=  bytes_left_to_receive)) {

                error = ri6075_XFDC_SwitchBuffers(vi, rx_buffer_number);
                if (error < 0) return error;

                error = ri6075_XFDC_GetHeader(vi, rx_buffer_number, &fdc_header);
                if (error < 0) return error;

                if (fdc_header & RI6075_XFDC_RDY_BIT) {

                    // read buffer ready, read the data
                    error = ri6075_XFDC_ReadBufr(vi, rx_buffer_number,
                                                          rx_buf_ptr,
                                                          rx_fdc_buf_size,
                                                          &bytes_received_this_time);
                    if (error < 0) return error;

                    // update the data to indicate the total number of bytes
                                // read thus far
                    total_bytes_received += bytes_received_this_time;
                    rx_buf_ptr += bytes_received_this_time;

                    // since we retrieved the last of the bytes, we are done
                    break;
                }
            }
        }

    }

    if (total_bytes_received == 0) {
```

```
        error = ri6075_read_recv_channel (vi, rx_chan, rx_buf_ptr,
                                    bytes_left_to_receive,
                                  &total_bytes_received);

        if (error < 0) return error;
    }

    *rx_len = total_bytes_received;

    // we are done with the FDC channel(s), so close them
    error = ri6075_XFDC_ChannelClose (vi, tx_fdc_buffer_even);
    if (error < 0) return(error);

    error = ri6075_XFDC_ChannelClose (vi, rx_fdc_buffer_even);
    if (error < 0) return(error);

    // if we get here, we are done
    return( error );
}
```

Example #5. Transmit and Receive With SDLC Format

This function is an example that shows how to use the to set up the channels in SDLC format and transmit data one channel and receive the data from another channel. This example assumes the transmit channel is connected to the receive channel via a "null modem."

```
/*=========================================================================*/
/* Function: Example SDLC transmit and receive                             */
/* Purpose:  This function is an example that shows how to use the driver  */
/*           to set up the channels in SDLC format and transmit data from  */
/*           one channel and receive the data from another channel.        */
/*                                                                         */
/*           This example assumes the transmit channel is connected to the */
/*           receive channel via a "null modem"                            */
/*=========================================================================*/

ViStatus _VI_FUNC ri6075_example_sdlc_xmit_recv (ViSession vi,
                                                 ViInt16 frameSize,
                                                 ViInt16 transmitChannel,
                                                 ViChar _VI_FAR transmitData[],
                                                 ViInt32 transmitCount,
                                                 ViInt16 receiveChannel,
                                                 ViChar _VI_FAR receiveData[],
                                                 ViInt32 maxReceiveCount,
                                                 ViPInt32 receiveCount)
{
    ViStatus error;
    ViInt16 tx_fdc_channel;
    ViInt16 rx_fdc_channel;

    // reset the unit to a known state
    error = ri6075_reset(vi);
    if (error < 0)     return error;

    // select the SDLC protocol for the two channels
    error = ri6075_set_protocol (vi, transmitChannel, RI6075_PROTOCOL_SDLC);
    if (error < 0)     return error;

    error = ri6075_set_protocol (vi, receiveChannel, RI6075_PROTOCOL_SDLC);
    if (error < 0)     return error;

    // now set up RS422FD as the protocol for the send and receive channels
    error = ri6075_set_interface_type (vi, transmitChannel,RI6075_MODE_422FD);
    if (error < 0)     return error;

    error = ri6075_set_interface_type (vi, receiveChannel, RI6075_MODE_422FD);
    if (error < 0)     return error;

    // now set the frame sizes for the transmit and receive channels
    error = ri6075_set_frame_size (vi, transmitChannel, frameSize);
    if (error < 0)     return error;

    error = ri6075_set_frame_size (vi, receiveChannel, frameSize);
    if (error < 0)     return error;
```

```
    // now set the baud rate for the transmit and receive channel
    // we will use 1000000 for this example
    error = ri6075_set_xmit_baud_rate (vi, transmitChannel, 1000000);
    if (error < 0)      return error;

    error = ri6075_set_xmit_baud_rate (vi, receiveChannel, 1000000);
    if (error < 0)      return error;

    // now open the transmit FDC channel for the transmit channel
    tx_fdc_channel = (transmitChannel - 1) * 4 + 2;
    error = ri6075_XFDC_OpenWriteBufrPS (vi, tx_fdc_channel);
    if (error < 0) return error;

    rx_fdc_channel = (receiveChannel - 1) * 4;
    error = ri6075_XFDC_OpenReadBufrPS (vi, rx_fdc_channel);
    if (error < 0) return error;

    // load and send the transmit data
    error = ri6075_XFDC_TransmitBufr (vi, tx_fdc_channel, transmitData, transmitCount);
    if (error < 0) return error;

    // Wait 5 seconds to get the data
    ri6075_delay( 5 );

    // read the data in the receive channel
    error = ri6075_read_recv_channel (vi, receiveChannel, receiveData, maxReceiveCount,
                                          receiveCount);
    if (error < 0) return error;


    return error;
}
```