VM6068

HIGH-PERFORMANCE SERIAL INTERFACE MODULE

USER'S MANUAL

P/N: 82-0027-000
Released February 19, 2007

VXI Technology, Inc.

2031 Main Street
Irvine, CA 92614-6509
(949) 955-1894

TABLE OF CONTENTS

INTRODUCTION

CERTIFICATION

VXI Technology, Inc. (VTI) certifies that this product met its published specifications at the time of shipment from the factory. VTI further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology (formerly National Bureau of Standards), to the extent allowed by that organization's calibration facility, and to the calibration facilities of other International Standards Organization members.

WARRANTY

The product referred to herein is warranted against defects in material and workmanship for a period of three years from the receipt date of the product at customer's facility. The sole and exclusive remedy for breach of any warranty concerning these goods shall be repair or replacement of defective parts, or a refund of the purchase price, to be determined at the option of VTI.

For warranty service or repair, this product must be returned to a VXI Technology authorized service center. The product shall be shipped prepaid to VTI and VTI shall prepay all returns of the product to the buyer. However, the buyer shall pay all shipping charges, duties, and taxes for products returned to VTI from another country.

VTI warrants that its software and firmware designated by VTI for use with a product will execute its programming when properly installed on that product. VTI does not however warrant that the operation of the product, or software, or firmware will be uninterrupted or error free.

LIMITATION OF WARRANTY

The warranty shall not apply to defects resulting from improper or inadequate maintenance by the buyer, buyer-supplied products or interfacing, unauthorized modification or misuse, operation outside the environmental specifications for the product, or improper site preparation or maintenance.

VXI Technology, Inc. shall not be liable for injury to property other than the goods themselves. Other than the limited warranty stated above, VXI Technology, Inc. makes no other warranties, express or implied, with respect to the quality of product beyond the description of the goods on the face of the contract. VTI specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

VXI Technology, Inc.
2031 Main Street
Irvine, CA 92614-6509 U.S.A.

DECLARATION OF CONFORMITY

Declaration of Conformity According to ISO/IEC Guide 22 and EN 45014

MANUFACTURER'S NAME	VXI Technology, Inc.
MANUFACTURER'S ADDRESS	2031 Main Street Irvine, California 92614-6509-6509
PRODUCT NAME	High-Performance Serial Interface Module
MODEL NUMBER(S)	VM6068
PRODUCT OPTIONS	All
PRODUCT CONFIGURATIONS	All

VXI Technology, Inc. declares that the aforementioned product conforms to the requirements of the Low Voltage Directive 73/23/EEC *and the* EMC Directive 89/366/EEC *(inclusive* 93/68/EEC*) and carries the "CE" mark accordingly. The product has been designed and manufactured according to the following specifications:*

SAFETY	EN61010 (2001)
EMC	EN61326 (1997 w/A1:98) Class A CISPR 22 (1997) Class A VCCI (April 2000) Class A ICES-003 Class A (ANSI C63.4 1992) AS/NZS 3548 (w/A1 & A2:97) Class A FCC Part 15 Subpart B Class A EN 61010-1:2001

The product was installed into a C-size VXI mainframe chassis and tested in a typical configuration.

I hereby declare that the aforementioned product has been designed to be in compliance with the relevant sections of the specifications listed above as well as complying with all essential requirements of the Low Voltage Directive.

February 2007

Steve Mauga, QA Manager

GENERAL SAFETY INSTRUCTIONS

Review the following safety precautions to avoid bodily injury and/or damage to the product. These precautions must be observed during all phases of operation or service of this product. Failure to comply with these precautions, or with specific warnings elsewhere in this manual, violates safety standards of design, manufacture, and intended use of the product.

Service should only be performed by qualified personnel.

TERMS AND SYMBOLS

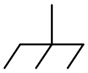
These terms may appear in this manual:

WARNING Indicates that a procedure or condition may cause bodily injury or death.

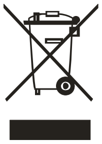
CAUTION Indicates that a procedure or condition could possibly cause damage to equipment or loss of data.

These symbols may appear on the product:

ATTENTION - Important safety instructions



Frame or chassis ground



Indicates that the product was manufactured after August 13, 2005. This mark is placed in accordance with *EN 50419, Marking of electrical and electronic equipment in accordance with Article 11(2) of Directive 2002/96/EC (WEEE)*. End-of-life product can be returned to VTI by obtaining an RMA number. Fees for take-back and recycling will apply if not prohibited by national law.

WARNINGS

Follow these precautions to avoid injury or damage to the product:

Use Proper Power Cord To avoid hazard, only use the power cord specified for this product.

Use Proper Power Source To avoid electrical overload, electric shock, or fire hazard, do not use a power source that applies other than the specified voltage.

Use Proper Fuse To avoid fire hazard, only use the type and rating fuse specified for this product.

WARNINGS (CONT.)

Avoid Electric Shock

To avoid electric shock or fire hazard, do not operate this product with the covers removed Do not connect or disconnect any cable, probes, test leads, etc. while they are connected to a voltage source. Remove all power and unplug unit before performing any service. ***Service should only be performed by qualified personnel.***

Ground the Product

This product is grounded through the grounding conductor of the power cord. To avoid electric shock, the grounding conductor must be connected to earth ground.

Operating Conditions

To avoid injury, electric shock or fire hazard:

- Do not operate in wet or damp conditions.
- Do not operate in an explosive atmosphere.
- Operate or store only in specified temperature range.
- Provide proper clearance for product ventilation to prevent overheating.
- DO NOT operate if any damage to this product is suspected. ***Product should be inspected or serviced only by qualified personnel.***

Support Resources

Support resources for this product are available on the Internet and at VXI Technology customer support centers.

VXI Technology
World Headquarters

VXI Technology, Inc.
2031 Main Street
Irvine, CA 92614-6509

Phone: (949) 955-1894
Fax: (949) 955-3041

VXI Technology
Cleveland Instrument Division

5425 Warner Road
Suite 13
Valley View, OH 44125

Phone: (216) 447-8950
Fax: (216) 447-8951

VXI Technology
Lake Stevens Instrument Division

VXI Technology, Inc.
1924 - 203 Bickford
Snohomish, WA 98290

Phone: (425) 212-2285
Fax: (425) 212-2289

Technical Support

Phone: (949) 955-1894
Fax: (949) 955-3041
E-mail: support@vxitech.com

Visit http://www.vxitech.com for worldwide support sites and service plan information.

SECTION 1

INTRODUCTION

INTRODUCTION

The VM6068 is a high-performance serial interface module that has been designed for high data throughput, multiple serial protocols, and flexible electrical interfacing. The instrument uses the message-based word-serial interface for programming and data movement and allows direct register access for very high-speed data input and retrieval. The VM6068 command set conforms to the SCPI standard for consistency and ease of programming.

The VM6068 is a member of the VXI Technology VMIP™ (*VXI Modular Instrumentation Platform*) family and is available as a 4-, 8-, or 12-channel single-wide VXIbus instrument. Figure 1-2 shows the 12-channel version of the VM6068. The 8-channel version would not have J200 and its associated LEDs and nomenclature, and the 4-channel version would have neither J200 nor J202. In addition to these three standard configurations, the VM6068 may be combined with any of the other members of the VMIP family to form a customized and highly integrated instrument (see Figure 1-1). This allows the user to reduce system size and cost by combining the VM6068 with two other instrument functions in a single-wide C-size VXIbus module.

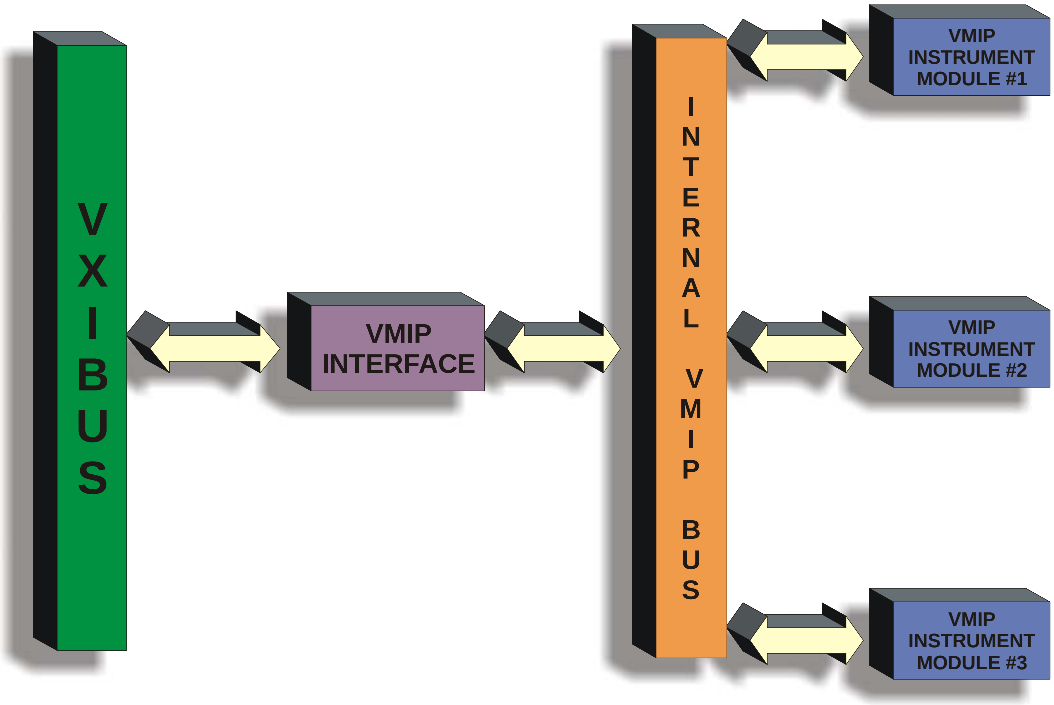
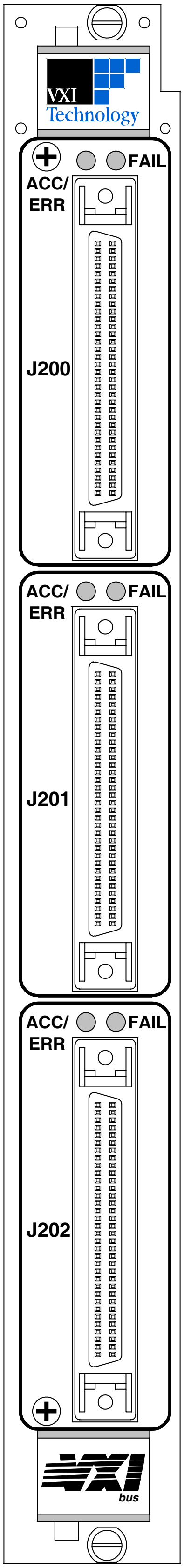

VXIBUS
VMIP INTERFACE
INTERNAL VMIP BUS
VMIP INSTRUMENT MODULE #1
VMIP INSTRUMENT MODULE #2
VMIP INSTRUMENT MODULE #3

FIGURE 1-1: VMIP™ PLATFORM



Regardless of whether the VM6068 is configured with other VM6068 modules or with other VMIP modules, each group of four channels is treated as an independent instrument in the VXIbus chassis and, as such, each group of four channels has its own FAIL and ACCESS light.

Description

The VM6068 is a high-performance VXIbus serial interface utilizing the Motorola MC68360 QUICC™ (*Quad Integrated Communication Controller*) integrated microprocessor and peripheral combination. The MC68360 provides four highly flexible serial communication channels along with a CPU32+ processor core, four baud rate generators, two DMA channels, four timers, a dynamic RAM controller, a dedicated communication RISC controller with fourteen serial DMA channels and two TDM (time division multiplexers).

The QUICC microcontroller's serial interface is brought to the front panel via four programmable interface driver/ receiver ICs. These driver/receivers may be programmed to operate at RS-232, RS-422, RS-449, RS-485, V.35, and EIA-530 levels. Each channel may be programmed independently of the other channels.

The VXIbus is handled by the VMIP module and it in turn passes parsed data through to the VM6068 module via a bi-directional FIFO interface for maximum performance (see Figure 1-3). In addition to passing parsed SCPI command and data streams, the VMIP module may be configured to allow direct hardware access from the VXIbus to the FIFOs for maximum data throughput. The VMIP module has its own MC68340 microcontroller to handle the VXIbus traffic relieving the VM6068 from this activity.

The bi-directional mailbox interface between the VMIP module and the VM6068 provides a way for the VMIP module to pass instructions and mode information to and from the VM6068. With the addition of interrupt capability, each side of the interface can be notified of a pending message. This reserves the FIFOs for large blocks of data and allows data streams to be stopped midstream.

Both the VMIP module and the VM6068 store their embedded program in FLASH ROM. This allows for easy field updates and upgrades. The firmware may be distributed on diskette or BBS and updated via the VXIbus and the associated slot 0 controller.

Figure 1-2: Front Panel Layout

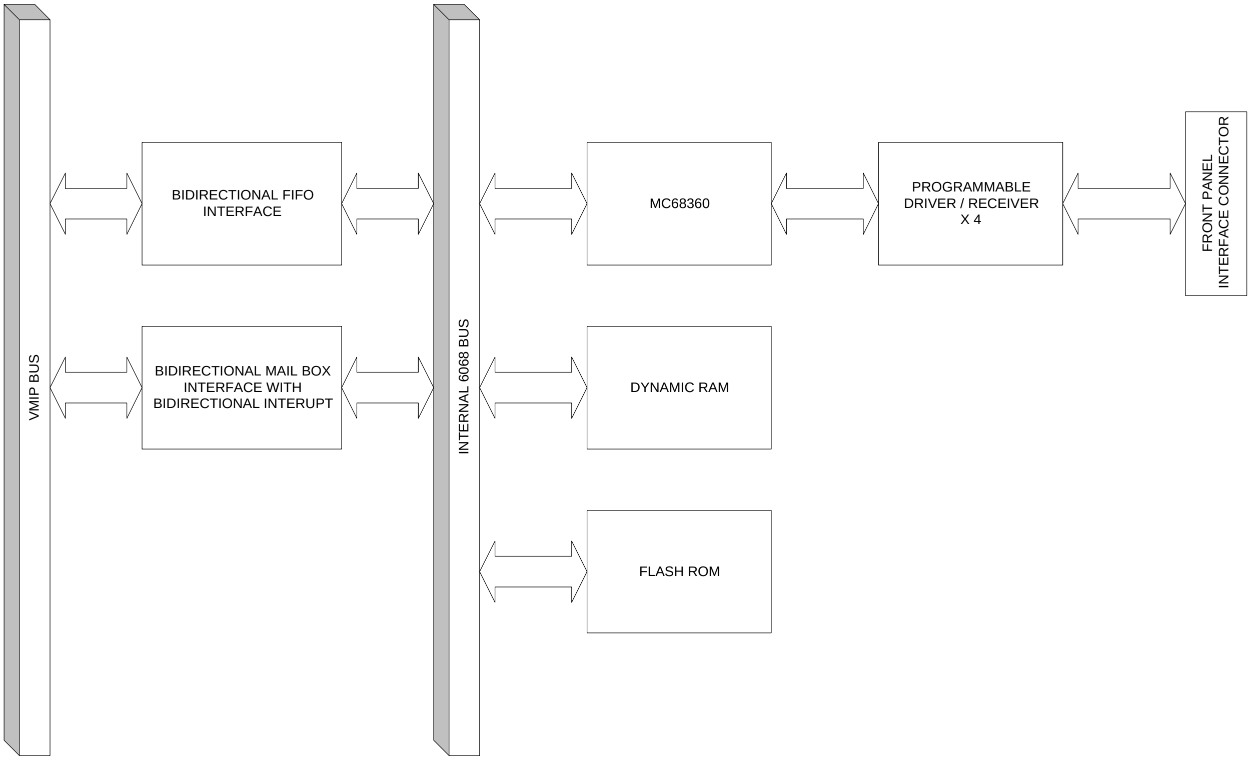

VMIP BUS
BIDIRECTIONAL FIFO INTERFACE
BIDIRECTIONAL MAIL BOX INTERFACE WITH BIDIRECTIONAL INTERUPT
INTERNAL 6068 BUS
MC68360
DYNAMIC RAM
FLASH ROM
PROGRAMMABLE DRIVER / RECEIVER X 4
FRONT PANEL INTERFACE CONNECTOR

FIGURE 1-3: VM6068 BLOCK DIAGRAM

Serial Interface Protocols

The VM6068 supports a variety of both bit-oriented and byte-oriented serial protocols. These protocols are supported via the microcode in the RISC controller, which supervises the activities of the four serial channels. The standard protocols supported by the VM6068 are:

HDLC/SDLC™	***High-Level/Synchronous Data Link Control***. This is one of the most commonly used layer 2 protocols in the OSI seven-layer model. HDLC uses a zero insertion/deletion process known as bit-stuffing to ensure that the bit pattern of the delimiter flag does not occur in the fields between flags. The HDLC frame is synchronous and therefore relies on the physical layer to provide a method of clocking and synchronizing the transmitter and receiver. SDLC™ is IBM's specific version of HDLC.
UART	***Universal Asynchronous Receiver Transmitter***. This protocol provides the standard asynchronous character-oriented UART serial interface with features such as appending a start bit, a parity bit and two or less stop bits to each character sent. The receiver typically over-samples the incoming data by a factor of 16 although the UART mode also supports a 1x clock in synchronous mode.

VM6068 Specifications

General Specifications	
Number of Channels	
VM6068-1	4
VM6068-2	8
VM6068-3	12
VXI Communication	
	Message-based Word Serial Interface Direct Register Access, A16 memory space
Protocols	
	HDLC/SDLC, UART
Data Transmission Modes	
	Block Mode, Character Mode
Buffer RAM[1]	
Standard	2 MB (megabytes)per group of four channels
Option 1	4 MB per group of four channels
Data Throughput[1,2]	
1 HDLC	5.0 Mb/s (megabits per second) (I/O limited, QUICC will do 8 Mb/s) [3]
2 HDLC	4.0 Mb/s
3 HDLC	2.6 Mb/s
4 HDLC	2.048 Mb/s
4 UART	625 kb/s (kilobits per second)
Physical Interface Standards[4]	
	RS-232, RS-422, RS-449, RS-485, V.35, EIA-530
Trigger Source	
	Word Serial Message VXIbus TTL Trigger 0 through 7 Internal Timer
Internal Timer Range	
	1 µs to 2.147 s
Power Requirements	
VM6068-1	+5.0 V @ 1.70 A, +12.0 V @ 0.10 A, -12.0 V @ 0.10 A
VM6068-2	+5.0 V @ 2.66 A, +12.0 V @ 0.20 A, -12.0 V @ 0.20 A
VM6068-3	+5.0 V @ 3.62 A, +12.0 V @ 0.30 A, -12.0 V @ 0.30 A
Cooling Requirements	
VM6068-1	1.0 L/s @ 0.7 mm H_2O for 10°C rise
VM6068-2	1.5 L/s @ 0.7 mm H_2O for 10°C rise
VM6068-3	2.0 L/s @ 0.7 mm H_2O for 10°C rise
Operating Temperature	
	0°C to 50°C
Non-Operating Temperature	
	-55°C to 75°C
Humidity (Non-Condensing)	
	≤ 95% relative humidity from 0°C to 30°C ≤ 75% relative humidity to 40°C ≤ 45% relative humidity to 50°C
Size	
	10.309” H x 1.188” W x 14.469” D (261.849 mm x 30.175 mm x 367.513 mm)
Manufacturer’s ID	
	3915
Module Model Code	
	261

Driver/Receiver Specifications

RS-485 Driver	
High Level Output	+6.0 V max.
Low Level Output	-0.3 V min.
Differential Output	±1.5 V min., ±5.0 V max.
Open Circuit Voltage	±6.0 V max.
Transition Time	120 ns max.
Transmission Rate	5 Mb/s max.
RS-485 Receiver	
High Threshold	+0.2 V min., +12.0 V max. (a)-(b)
Low Threshold	-7.0 V min., -0.2 V max. (a)-(b)
Common Mode Range	-7.0 V min., +12.0 V max.
Receiver Sensitivity	±0.2 V over the common mode range
V.35 Driver	
Differential Output	±0.44 V min., ±0.66 V max., 100 Ω Load
Transition Time	40 ns max.
Transmission Rate	5 Mb/s max.
V.35 Receiver	
High Threshold	+0.2 V min., +12.0 volts max. (a)-(b)
Low Threshold	-7.0 V min., -0.2 V max. (a)-(b)
Common Mode Range	-7.0 V min., +12.0 V max.
Receiver Sensitivity	±0.2 V over the common mode range
RS-422 Driver	
Differential Output	±2.0 V min., ±5.0 V max.
Open Circuit Voltage	±6.0 V max.
Balance	±0.4 V max.
Offset	+3.0 V max.
Short Circuit Current	±150 mA max.
Transition Time	60 ns
Transmission Rate	5 Mb/s max.
RS-422 Receiver	
High Threshold	+0.2 V min., +6.0 V max. (a)-(b)
Low Threshold	-6.0 V min., -0.2 V max. (a)-(b)
Common Mode Range	-10.0 V min., +10.0 V max.
Receiver Sensitivity	±0.2 V over the common mode range
Input Impedance	4 kΩ min.
RS-232 Driver	
High Level Output	+5.0 V min., +15.0 V max.
Low Level Output	-5.0 V min., -15.0 V max.
Short Circuit Current	±100 mA max.
Open Circuit Voltage	±15.0 V max.
Power Off Impedance	300 Ω min.
Slew Rate	30.0 V/µs max. (R_L = 3 kΩ, C_L = 15 pF)
Transition Time	1.56 µs max.
Transmission Rate	120 kb/s max.
RS-232 Receiver	
High Threshold	1.7 V typ., +2.4 V max. (a)-(b)
Low Threshold	0.8 V min., 1.2 V min. (a)-(b)
Open Circuit Bias	0.0 V min., +2.0 V max.
Input Impedance	3 kΩ min., 7 kΩ max., 5 kΩ typ.

Notes

1. These numbers apply to a single VM6068 VMIP module that has 4 channels. Note that an 8-channel system has two VM6068 VMIP modules that are treated as two distinct and separate instruments.
2. These performance specifications apply to a single VM6068 VMIP module. These specifications are preliminary and are subject to change. All specifications assume full duplex operation. If half duplex is used, the throughput is approximately doubled.
3. The serial throughput is limited to 5 Mb/s by the programmable drivers and receivers.
4. See the section in Section 2 (Installation) discussing the driver combinations forming the different physical interface standards.

SECTION 2

PREPARATION FOR USE

INSTALLATION

When the VM6068 is unpacked from its shipping carton, the contents should include the following items:

(1) VM6068 VXIbus module
(1) VM6068 High-performance Serial Interface Module User's Manual (this manual)

All components should be immediately inspected for damage upon receipt of the unit.

Once the VM6068 is assessed to be in good condition, it may be installed into an appropriate C-size or D-size VXIbus chassis in any slot other than slot 0. The chassis should be checked to ensure that it is capable of providing adequate power and cooling for the VM6068. Once the chassis is found be adequate, the VM6068's logical address and the chassis' backplane jumpers should be configured prior to the VM6068's installation.

CALCULATING SYSTEM POWER AND COOLING REQUIREMENTS

It is imperative that the chassis provide adequate power and cooling for this module. Referring to the chassis user's manual, confirm that the power budget for the system (the chassis and all modules installed therein) is not exceeded and that the cooling system can provide adequate airflow at the specified backpressure.

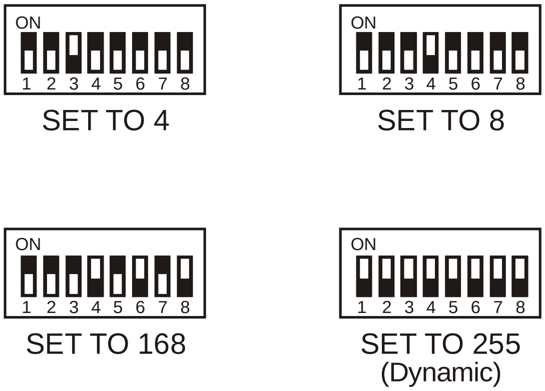
⚠ It should be noted that if the chassis cannot provide adequate power to the module, the instrument may not perform to specification or possibly not operate at all. In addition, if adequate cooling is not provided, the reliability of the instrument will be jeopardized and permanent damage may occur. Damage found to have occurred due to inadequate cooling would also void the warranty of the module.

SETTING THE CHASSIS BACKPLANE JUMPERS

Please refer to the chassis' User's Manual for further details on setting the backplane jumpers.

SETTING THE LOGICAL ADDRESS

The logical address of the VM6068 is set by a single 8-position DIP switch located near the module's backplane connectors (this is the only switch on the module). The switch is labeled with positions 1 through 8 and with an ON position. A switch pushed toward the ON legend will signify a logic 1; switches pushed away from the ON legend will signify a logic 0. The switch located at position 1 is the least significant bit while the switch located at position 8 is the most significant bit. See Figure 2-1 for examples of setting the logical address switch.



Switch Position	Switch Value
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128

FIGURE 2-1: LOGICAL ADDRESS SWITCH SETTING EXAMPLES

The VMIP may contain three separate instruments and will allocate logical addresses as required by the VXIbus specification (revisions 1.3 and 1.4). The logical address of the instrument is set on the VMIP carrier. The VMIP logical addresses must be set to an even multiple of 4 *unless dynamic addressing is used*. Switch positions 1 and 2 must always be set to the OFF position. Therefore, only addresses of 4, 8, 12, 16, ... 252 are allowed. The address switch should be set for one of these legal addresses and the address for the second instrument (the instrument in the center position) will automatically be set to the switch set address plus one; while the third instrument (the instrument in the lowest position) will automatically be set to the switch set address plus two. If dynamic address configuration is desired, the address switch should be set for a value of 255 (All switches set to ON). Upon power-up, the slot 0 resource manager will assign the first available logical addresses to each instrument in the VMIP module.

If dynamic address configuration is desired, the address switch should be set for a value of 255. (All switches set to ON). Upon power-up, the slot 0 resource manager will assign the first available logical addresses to each instrument in the VMIP module.

FRONT PANEL INTERFACE WIRING

The VM6068's serial interface is made available on the front panel of the instrument. The 4-channel version (VM6068-1) will have J201 that contains all signals for this instrument. The 8-channel version (VM6068-2) will have J201 and J202 provided, while the 12-channel version (VM6068-3) will have J200, J201 and J202. The wiring for each of these connectors is identical and since each group of four channels is treated as a separate instrument, the module will have three Channel 1s, three Channel 2s, three Channel 3s, and three Channel 4s.

The connector used in the VM6068 is a 68-pin high-density type commonly known as a 68-pin version of the SCSI-2 connector. The mating connector is an IDC (Insulation Displacement Connector) component and is available from a variety of sources. The connector attaches to two 34-conductor 0.050 centers ribbon cable and the pin out has been selected to allow for using the twisted pair type of ribbon cable. Some manufacturers also allow the use of discrete 30 gauge stranded wire.

TABLE 2-1: J200, J201, AND J202 PIN OUT

J200, J201, and J202 PIN OUT for V.35, RS-422, RS-485, RS-449, and EIA530

Signal Name	Type	Channel 1 Pin	Channel 2 Pin	Channel 3 Pin	Channel 4 Pin
TXD-	Output	1	19	35	53
TXD+	Output	2	20	36	54
RXD-	Input	3	21	37	55
RXD+	Input	4	22	38	56
RTS-	Output	5	23	39	57
RTS+	Output	6	24	40	58
CTS-	Input	7	25	41	59
CTS+	Input	8	26	42	60
DTR-	Output	9	27	43	61
DTR+	Output	10	28	44	62
DSR-	Input	11	29	45	63
DSR+	Input	12	30	46	64
TXC-	Output	13	31	47	65
TXC+	Output	14	32	48	66
RXC-	I/O	15	33	49	67
RXC+	I/O	16	34	50	68
GROUND	Power	17	18	51	52

J200, J201, and J202 PIN OUT for RS-232

Signal Name	Type	Channel 1 Pin	Channel 2 Pin	Channel 3 Pin	Channel 4 Pin
TXD-	Output	1	19	35	53
RXD-	Input	3	21	37	55
RTS-	Output	5	23	39	57
CTS-	Input	7	25	41	59
DTR-	Output	9	27	43	61
DSR-	Input	11	29	45	63
TXC-	Output	13	31	47	65
RXC-	I/O	15	33	49	67
GROUND	Power	17	18	51	52

RS-485 WIRING TERMINATION DIAGRAM

In order to communicate between VM6068 channels with a RS-485 connection, an impedance must be applied on the transmitting line to adjust the tri-state. Please refer to the EIA-485 standards manual for further information. The following is an example of pin connection and termination for Channel 1 / Channel 3 communication.

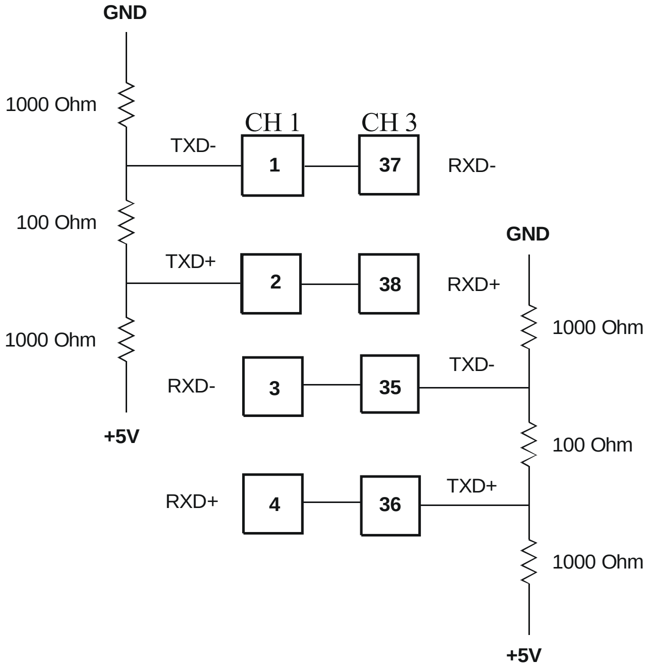


FIGURE 2-2: RS-485 CONNECTION TERMINATION

The mating connector to J200, J201, or J202 is available from the following companies:

AMP, Inc.

P/N 749621-7	Connector
P/N 749195-2	Back Shell
P/N 82208	Catalog Covering This Series of Connectors

Circuit Assembly

P/N CA-68NDP-12GT	Connector
P/N CA-68NDBS-1M	Back Shell
P/N DG01	Catalog covering this series of connectors

The pin locations for J200, J201, and J202 (located on the front panel) are shown in Figure 2-2.

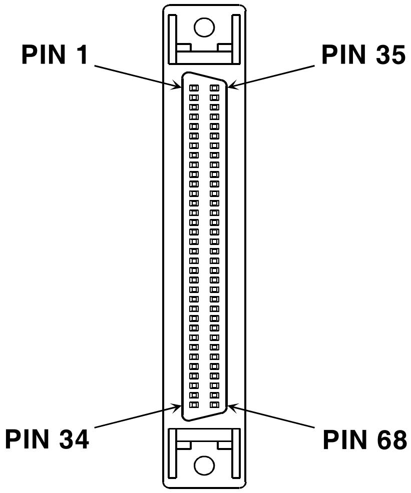


FIGURE 2-3: VM6068 - J200, J201, AND J202 PIN LOCATIONS

SECTION 3

PROGRAMMING

INTRODUCTION

The VM6068 module is a VXIbus message-based device whose command set is compliant with the Standard Command for Programmable Instruments (SCPI) programming language. See the Sample Program later in this section for specific programming examples and command usage. Also refer to individual command descriptions.

All module commands are sent over the VXIbus backplane to the module. Commands may be in upper, lower, or mixed case. All numbers are sent in ASCII decimal unless otherwise noted.

The module recognizes SCPI commands. SCPI is a tree-structured language based on IEEE-STD-488.2 Specifications. It utilizes the IEEE-STD-488.2 Standard command and the device dependent commands are structured to allow multiple branches off the same trunk to be used without repeating the trunk. To use this facility, terminate each branch with a semicolon. As an example, **RECeive:CLOCk:DIVide**, **RECeive:CODE**, and **RECeive:PARity** are all branches off the **SERial:RECeive** trunk. This makes it possible to combine several commands as follows:

```
SER2:REC:CLOC:DIV 1;CODE MANC;PAR EVEN
```

The above command is the same as the following:

```
SER2:REC:CLOC:DIV 1
SER2:REC:CODE MANC
SER2:REC:PAR EVEN
```

Note that each command separated by the semi-colons must be from the same branch level otherwise an error would occur. The commands **CLOCk:DIVide 1**, **CODE MANC**, and **PARity EVEN** all start at the same branch level.

See the *Standard Command for Programmable Instruments (SCPI) Manual, Volume 1: Syntax & Style, Section 6* for more information.

The SCPI commands in this section are listed in upper and lower case. Character case is used to indicate different forms of the same command. Keywords can have both a short form and a long form (some commands only have one form). The short form uses just the keyword characters in uppercase. The long form uses the keyword characters in uppercase plus the keyword characters in lowercase. Either form is acceptable. Note that there are no intermediate forms. All characters of the short form or all characters of the long form must be used. Short forms and long forms may be freely intermixed. The actual commands sent can be in upper case, lower case or mixed case (case is only used to distinguish long form and short form for the user). As an example, these commands are all correct and all have the same effect:

```
SER2:rec:par even
SER2:receive:par even
SER2:rec:parity even
SER2:receive:parity even
SERIAL2:REC:PAR EVEN
SERIAL2:RECEIVE:PAR EVEN
SERIAL2:REC:PARITY EVEN
SERIAL2:RECEIVE:PARITY EVEN
```

The following command is **not** correct because it uses part of the long form of SERial, but not all letters of the long form.

SERI2:REC:PARITY EVEN – incorrect syntax (additional "i")

All of the SCPI commands also have a query form unless otherwise noted. Query forms contain a question mark (?). The query form allows the system to ask what the current setting of a parameter is. The query form of the command generally replaces the parameter with the question mark. Query responses do not include the command header. This means only the parameter is returned; no part of the command is returned.

When character data is used for a parameter, both short and long forms are recognized. If the command has a query form with character response data, the short form is always returned in upper case. As an example, to find out what the current receive parity mode is on Channel 2, use the following command:

```
SER2:REC:PAR?
```

The response could be:

```
EVEN
```

This tells the user that the Channel 2 receive parity mode is set to EVEN.

Multiple commands can also be combined on one line. To do this, terminate one command with a semicolon and start the next command with a colon. As an example, Channel 2 format and receive parity mode could be set as follows:

```
FORM:DATA 2 INT;:SER2:REC:PAR EVEN
```

When combining commands, keep in mind the size of the input buffer. Command lines that are too long will generate an error and not be used.

The IEEE-STD-488.2 Common Commands can be placed anywhere set off from the rest of the command by a semicolon. They can also be placed alone on a line. For example, place the ***RST** command in front of an initialization string as follows:

```
*RST;SER2:REC:CLOC:DIV 1;:CODE MANC;:PAR EVEN
```

Note that the **SER2:REC:CLOC:DIV 1** command **did not** require a leading colon because there was no prior trunk of the SCPI tree.

NOTATION

Keywords or parameters enclosed in square brackets ([ ]) are optional. If the optional part is a keyword, the keyword can be included or left out. Omitting an optional parameter will cause its default value to be used.

Parameters are enclosed by angle brackets (< >). Braces ({ }) are used to enclose one or more parameters that may be included zero or more times. A vertical bar (|), read as "or", is used to separate parameter alternatives.

EXAMPLES OF SCPI COMMANDS

FORMat:DATA

The format data command sets the data format for retrieving received characters. The data formats supported are ASCII, interger, hexadecimal, octal, binary, and packed.

FORMat[:DATA] <channel> <type>

Where <channel> is 1, 2, 3 or 4

Where <type> is ASC | INT | HEX | OCT | BIN | PACK

ASCii Data is transferred in NR1 format with 1, 2 or 3 significant digits. Multiple numbers are separated by commas. The string "ABC" is output as 65,66,67.

INTeger Received data is transferred as an indefinite block.
Note*: Data can be transmitted in either a definite or an indefinite block. (See IEEE-STD-488.2 Sections 8.7.9 and 8.7.10). The indefinite length arbitrary block is terminated with a combination of a LF (Line Feed) character and an END indication.*

HEXadecimal Data is encoded as a non-decimal numeric, using base 16 and preceded by a #H as specified in IEEE-488.2. The length is fixed at 2 digits. The string "ABC" is output as #H41, #H42, #H43.

OCTal Data is encoded as a non-decimal numeric using base 8 and preceded by a #Q as specified in IEEE-488.2. The length is fixed at 3 digits. The string "ABC" is output as #Q101, #Q102, #Q103.

BINary Data is encoded as a non-decimal numeric using base 2 and preceded by a #B as specified in IEEE-488.2. The length is fixed at 8 digits. The string "ABC" is output as #B01000001, #B01000010, #B01000011.

PACKed Data is the same as INTeger data as described above.

EXAMPLES

`FORM 3 ASC`	*Sets the data format for Channel 3 to ASCII*
`FORM 4 OCT`	*Sets the data format for Channel 4 to OCT*
`FORM 2 INT`	*Sets the data format for retrieving receivedcharacters to INTeger on Channel 2*

`FORM:DATA? 2`
`INT`
	Indicates that the data format for Channel 2 is set to INTerger.

BAUD

The Baud command sets the baud rate for one of the four generators available in the VM6068. Generator 1 is used for Channel 1, Generator 2 for Channel 2, etc. Each baud rate generator is a series of programmable dividers, driven by the CPU clock operating at 24 MHz. The programmed baud rate is rounded to the nearest available baud rate. Because the generator's output may be divided by a receive or transmit channel, a divisor parameter is allowed which will take into account this clock division in calculating the desired baud rate.

[SYSTem:][COMMunicate:]BAUD <generator> <baud_rate>,<divisor>

Where <generator> is 1 | 2 | 3 | 4

Where <baud_rate> is a numeric ASCII value from 367 to 3e6

Where <divisor> is 1 | 8 | 16 | 32

EXAMPLES

`BAUD 2 19200,16`	*Sets the baud rate for generator 2 to 19,200 Baud*

`BAUD? 2`
`19200,16`
	Returns the baud rate for generator 2

SERial:BITS

The serial bits command sets the number of transmit or receive data bits on the selected channel. It is important to note that the command is valid only in UART mode. In the non-UART mode, the query response is always 8.

[SYSTem:][COMMunicate:]SERial[[<channel>]]:BITS<bits>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <bits> is 5 | 6 | 7 | 8

EXAMPLES

`SER4:BITS7`	*Sets Channel 4 to 7 bits*

`SER4:BITS?`
`7`
	Querying number of bits for Channel 4

SERial:CLOCk

This command sets the direction of the bi-directional clock.

SERial[<channel>]:CLOCk <direction>	***Where <channel> is the channel whose corresponding clock's direction is to be configured (default is Channel 1)*** ***Where <direction> is IN | OUT, the direction in which the clock is to be driven***

EXAMPLES

`SER1:CLOC IN`	*Drives Channel 1's corresponding clock IN*
`SER2:CLOC OUT`	*Drives Channel 2's corresponding clock OUT*
`SER3:CLOC IN`	*Drives Channel 3's corresponding clock IN*
`SER3:CLOC?` `INT`	*Queries the direction in which Channel 3's clock is being driven*

SERial:CONTrol:CTS

This command enables or disables the CTS handshaking on the specified serial channel.

SERial [<channel>]:CONTrol:CTS <boolean>	***Where <channel> is the serial channel for which CTS handshaking is to be enabled or Disabled (default is Channel 1)*** ***Where <boolean> specifies whether CTS handshaking is to be enabled or disabled***

EXAMPLES

`SER1:CONT:CTS ON`	*Enables CTS handshaking on Channel 1*
`SER2:CONT:CTS OFF`	*Disables CTS handshaking on Channel 2*
`SER3:CONT:CTS ON`	*Enables CTS handshaking on Channel 3*

`SER3:CONT:CTS?`
`1`
	Queries whether CTS handshaking is enabled/disabled on Channel 3

SERial:CRC

This command selects the CRC generation in HDLC mode.

[SYSTem:][COMMunicate:]SERial[<channel>]:CRC <type>

	Where <channel> specifies the serial channel for which the CRC generation mode is to be configured in HDLC mode (default is Channel 1) ***Note that the command generates an error if the channel is not operating in HDLC mode*** ***Where <type> specifies the CRC generation mode***

EXAMPLES

`SER1:CRC CCITT16`	*Configures Channel 1's CRC generation mode as CCITT16* *It is assumed that Channel 1 is operating in HDLC mode*
`SER2:PROT HDLC`	*Configures Channel 2's serial interface protocol as HDLC*
`SER2:CRC CCITT32`	*Configures Channel 2's CRC generation mode as CCITT32*
`SER3:PROT HDLC`	*Configures Channel 3's serial interface protocol as HDLC*
`SER3:CRC CCITT16`	*Configures Channel 3's CRC generation mode as CCITT16*
`SER3:CRC?` `CCITT16`	*Queries the CRC generation mode used by Channel 3*

SERial:PROTocol

The serial protocol command sets the serial interface OSI layer 2 protocol. The UNKNOWN response is what is returned if not in one of the other known protocols.

[SYSTEM:][COMMunicate:]SERial [<channel>]:PROTocol <type>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <type> is BIS | HDLC | LOC | TRAN | UART | UJNKNOWN

EXAMPLES

`SER 2:PROT HDLC`	*Sets the serial interface protocol for Channel 2 as HDLC*

`SER 2:PROT?`
`HDLC`
	Returns the serial interface protocol type which is currently configured as HDLC

`SER1:PROT UART`	*Sets UART as the protocol for Channel 1*

NOTE The VM6068 defaults to the UART protocol. The following settings are protocol specific where the correct protocol must be set first in order for these to function properly.

UART	HDLC
`SERial:BITS`	`SERial:CRC`
`SERial:RECeive:IDLe`	`SERial:RECeive:HADDress`
`SERial:RECeive:PARity`	`SERial:RECeive:HMASK`
`SERial:TRANsmit:PARity`	
`SERial:TRANsmit:SBITs`	

SERial:RECeive:CODE

The Serial Receive Code command sets the data decoding method for a receive channel. Each channel contains a digital phase locked loop (DPLL) that can be programmed to decode a variety of different coding methods: non-zero return, NRZI mark, NRZI space, FM0, FM1, Manchester and differential Manchester.

[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:CODE <decode>

Where <channel> = 1 | 2 | 3 | 4 (default is Channel 1)

Where <decode> = NRZ | NRZM | NRZS | FM0 | FM1 | MANChester | DMANchester

NRZ Non-Return to Zero - A 1 is represented by a high data level for the entire bit time. A 0 is represented by a low data level for the entire bit time.

NRZM NRZI Mark - A 1 is represented by no transition at the beginning of the bit. A 0 is represented by a transition at the beginning of the bit. This is the reverse of NRZI Space.

NRZS NRZI Space - A 1 is represented by a transition at the beginning of the bit. A 0 is represented by no transition at the beginning of the bit. This is the reverse of NRZI Mark.

FM0 A 1 is represented by a transition at the beginning of the bit and no transition at the center of the bit. A 0 is represented by a transition at the beginning of the bit and a transition at the center of the bit. This is the reverse of FM1.

FM1 A 1 is represented by a transition at the beginning of the bit and a transition at the center of the bit. A 0 is represented by a transition at the beginning of the bit and no transition at the center of the bit. This is the reverse of FM0.

MANC Manchester - A 1 is represented by a high to low transition at the center of a bit. A 0 is represented by low to high transition at the center of the bit. In either case there may be a transition at the beginning of the bit to achieve the required polarity.

DMAN Differential Manchester (a.k.a. Differential Biphase-L) - A 1 is represented by a transition at the center of the bit with the opposite direction from the transition at the center of the preceding bit. A 0 is represented by a transition at the center of the bit with the same polarity as the transition at the center of the preceding bit.

Note: *Here, when "levels" are mentioned, it refers to logical levels. Different electrical standards produce different voltage levels on the signal lines.*

EXAMPLES

```
SER3:REC:CODE MANC
```

```
SER3:REC:CODE?
MANC
```

Note: A query response of NONE would indicate an unrecognized code type

SERial:RECeive:ERRor:MASK

The Serial Receive Error MASK command masks selected error types that will be reported. A “1” bit allows the error to be reported and a “0” bit masks it. (The default setting is a “1” bit for all errors.) The mask is a direct correlation to the Buffer Descriptor. The default mask is #H3B for UART and #HBF for HDLC.

HDLC & UART:

```
#define  BD_BUSY_BIT             0x4000      // unused bit in the BD
```

This is an input overrun error indicating that the receive buffer is full.

UART:

```
#define  U_CDLOST_ERR           0x0001
#define  U_OV_ERR               0x0002
#define  U_PARITY_ERR           0x0008
#define  U_FRAME_ERR            0x0010
#define  U_BREAK_ERR            0x0020
```

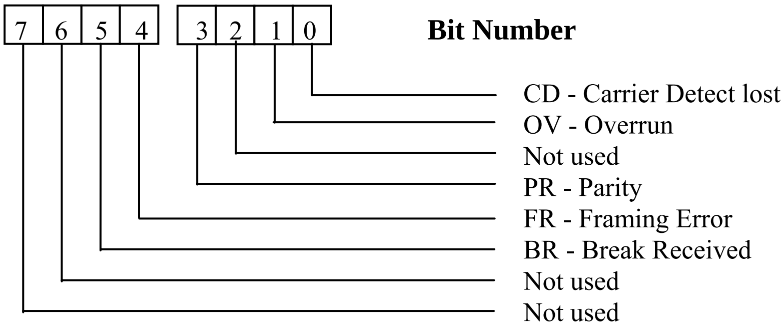


FIGURE 3-1: UART ERROR MASK

CD - Carrier Detect Lost The carrier detect signal was negated during message reception.

OV - Overrun A receiver overrun occurred during message reception.

PR - Parity Error A character with a parity error was received and is located in the last byte of this buffer. A new receive buffer will be used for further data reception.

FR - Framing Error A character framing error was received and is located in the last byte of this buffer. A framing error is a character without a stop bit. A new receive buffer will be used for further data reception.

BR - Break Received A break sequence was received while receiving data into this buffer.

HDLC:

```
#define  H_CDLOST_ERR          0x0001
#define  H_OV_ERR              0x0002
#define  H_CRC_ERR             0x0004
#define  H_ABORT_ERR           0x0008
#define  H_NONOCTET_ERR        0x0010
#define  H_FRAME_ERR           0x0020
#define  H_DPLL_ERR            0x0080
```

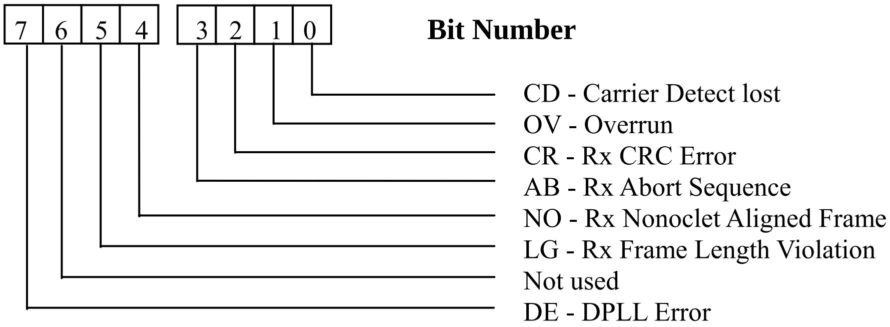


FIGURE 3-2: HDLC ERROR MASK

CD - Carrier Detect Lost
The carrier detect signal was negated during frame reception.

OV - Overrun
A receiver overrun occurred during frame reception.

CR - Rx CRC Error
This frame contains a CRC error. The received CRC bytes are always written to the receive buffer.

AB - Rx Abort Sequence
A minimum of seven consecutive ones was received during frame reception.

NO - Rx Nonoctet Aligned Frame
A frame that contained a number of bits not exactly divisible by eight was received.

LG - Rx Frame Length Violation
A frame length greater than the maximum defined for this channel was recognized (only the maximum-allowed number of bytes (MFLR) is written to the data buffer). This event will not be reported until the Rx BD is closed and the RXF bit is set, after receipt of the closing flag. The actual number of bytes received between flags is written to the data length field of this BD.

DE - DPLL Error
This bit is set by the HDLC controller when a DPLL error has occurred during the reception of his buffer. In decoding modes where a transition is promised every bit, the DE bit will be set when a missing transition has occurred.

SERial:RECeive:PARity

The Serial Receive Parity command sets a channel's parity type. It is only valid in the UART mode. The following modes are selected: even, odd, none, ignore, zero or one.

[SYSTem:][COMMunicate:]SERial[<channel>][:RECeive]:PARity <type>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <type> is EVEN | ODD | NONE | IGN | ZERO

EVEN - Received characters are checked for even parity.

ODD - Received characters are checked for odd parity.

NONE - No parity is checked on received characters. If a parity bit is sent to the receiver, it may cause a framing error. This also turns off the parity for the transmitter.

IGNore - All parity errors on received data are ignored.

ZERO - Received characters are checked for a 0 parity bit.

ONE - Received characters are selected for a 1 parity bit.

EXAMPLES

`SER1:PROT UART`	*Setting Channel 1 protocol to UART*
`SER2:REC:PARITY EVEN`	*Sets Channel 2's parity type to EVEN*
`SER2:REC:PAR?` `EVEN`	*Return the EVEN parity type used on a selected receive Channel 2*
`SER1:REC:PAR ONE`	*Sets the parity of Channel 1 to one parity bit*

SERial:STANdard

The serial standard command sets the desired physical interface standard for the selected channel. The available standards are RS-232, RS-422, RS-449, RS-485, V.35, and EIA-530. This command controls both transmit and receive hardware.

[SYSTem:][COMMunicate:]SERial[<channel>]:STANdard <standard>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <standard> is 232 | 422 | 449 | 485 | V.35 | EIA-530

EXAMPLES

`SER2:STAN 422`	*Sets the physical interface standard as 422 for Channel 2*

`SER2:STAN?`
`422`
	Returns the physical interface standard for Channel 2 which is currently set to 422

`SER1:STAN 530`	*Setting Channel 1's standard interface to EIA-530*

`SER1:STAN?`
`530`
	Querying Channel 1's standard interface

`SER2:STAN 449`	*Setting Channel 2's standard interface to RS449*

`SER2:STAN?`
`449`
	Querying Channel 2's standard interface

SERial:TRANsmit:PARity

The serial transmit parity command sets the transmit channel's parity. The following modes are supported: even, odd, none, zero, one or unknown.

[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:PARity <type>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <type> is EVEN | ODD | NONE | ZERO | ONE | UNKNOWN

EVEN	Transmitted characters are sent with an even parity.
ODD	Transmitted characters are sent with an odd parity.
NONE	No parity bit is sent on transmitted characters.
ZERO	Transmitted characters are sent with a 0 parity bit.
ONE	Transmitted characters are sent with a 1 parity bit.
UNKNOWN	This is what is returned in the non-UART mode.

Enabling parity for the transmitter (EVEN | ODD | ONE | ZERO) also enables the parity for the receiver. Turning parity off (NONE) also disables parity for the receiver. It is important to note that this command is only valid in UART mode.

EXAMPLES

`SER2:PROT UART`	*Setting Channel 2 protocol to UART*
`SER2:TRAN:PAR ONE`	*Sets the transmit parity for Channel 2 to one*
`SER2:TRAN:PAR?` `ONE`	*Returns the transmit Channel 2's parity which is currently set to ONE*

SERial:TRANsmit:CLOCk:DIVide

This command configures the baud clock divide ratio used by the transmitter.

SERial[<channel>]:TRANsmit:CLOCk:DIVide <ratio>

Where <channel> specifies the serial channel whose transmitter baud clock divide ratio is to be configured (default is Channel 1)

Where <ratio> specifies the divide ratio to be configured

EXAMPLES

`SER1:TRAN:CLOC:DIV 32`	*Configures the baud clock divide ratio of Channel 1 as 32*
`SER2:TRAN:CLOC:DIV 8`	*Configures the baud clock divide ratio of Channel 2 as 8*

`SER2:TRAN:CLOC:DIV?`
`8`
	Queries the baud clock divide ratio of Channel 2

SERial:TRANsmit:CLOCk:SOURce

This command configures the baud rate clock source for a serial channel's transmitter.

SERial[<channel>]:TRANsmitter:CLOCk:SOURce <source>

Where <channel> specifies the serial channel whose baud rate clock source is to be configured (default is Channel 1)

Where <source> specifies the baud rate clock source to be configured

Note that Channel 1 and 2 can be only connected to EXT1 or EXT2 if an external source is to be selected. Similarly, Channel 3 and 4 can be only connected to EXT3 or EXT4 if an external source is to be selected else an instrument error is generated.

EXAMPLES

`SRE1:TRAN:CLOC:SOUR INT1`	*Configures INT1 as the baud rate clock source for Channel 1*
`SER2:TRAN:CLOC:SOUR EXT1`	*Configures EXT1 as the baud rate clock source for Channel 2*
`SER2:TRAN:CLOC:SOUR?` `EXT1`	*Queries the baud rate clock source for Channel 2*
`SER3:TRAN:CLOC:SOUR EXT4`	*Configures EXT4 as the baud rate clock source for Channel 3*
`SER3:TRAN:CLOC:SOUR?` `EXT4`	*Queries the baud rate clock source for Channel 3*

SERial:TRANsmit:CODE

This command configures the data encoding method for a transmit channel. See **SERial:RECeive:CODE**.

SERial[<channel>]:TRANsmit:CODE <encode>	***Where <channel> specifies the serial channel whose data encoding method is to be configured (default isChannel 1)*** ***Where <encode> specifies the data encoding method to be configured***

EXAMPLES

`SER1:TRAN:CODE DMAN`	*Configures Differential Manchester as the data encoding method for Channel 1*
`SER2:TRAN:CODE FM1`	*Configures FM1 as the data encoding method for Channel 2*
`SER2:TRAN:CODE?` `FM1`	*Queries the data encoding method for Channel 2*

SERial:TRANsmit:SBITs

The serial transmit sbits command sets the number of stop bits on the selected transmit channel. It is important to note that this command is not applicable for RECEIVE channels. This command is only valid in the UART mode. For non-UART mode, this command is ignored.

[SYSTem:][COMMunicate:]SERial[<channel>][:TRANsmit]:SBITs <bits>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <bits> is either 1 or 2

EXAMPLES

`SER4:TRAN:SBIT 1`	*Sets the number of stop bits to one for Channel 4*

`SER4:TRAN:SBIT?`
`1`
	Returns the number of stop bits for Channel 4, which is currently set to 1

TRACe:DATA

The trace data command is used to load or retrieve data to or from the transmit or receive queues using the word serial interface. Data may be loaded into a transmit queue using the block format or by using a series of comma-separated values. The Trace Data query is used to retrieve received data in the format determined by FORMat:DATA command.

TRACe:DATA <trace_name>, (<block> | <NRf> {,<NRf>})

Where <trace_name> is TCH1 | TCH2 | TCH3 | TCH4 for transmit queues

Where <block> is as defined in IEEE-488.2

Where <NRf> is as defined in IEEE-488.2

Note: When the query for the above command is used, the trace names are RCH1, RCH2, RCH3 and RCH4 (for receive queues).

EXAMPLES

`TRAC:DATA TCH1,65,66,67`	*Loads data to the transmit queue using word serial interface*

`TRAC:DATA? RCH1`
`#13ABC`
	Retrieves data from the receive queue using word serial interface, in the format determined by FORMAT:DATA command

`TRAC:DATA TCH2,#18ABCDEFGH`	*<block> = <Definite Length Arbitrary Block Response Data> - see section 8.7.9 of IEEE-STD-488.2*

TRACe:DATA:FEED

The trace data feed command is used to establish a hardware FIFO-based data path for a specified queue.

This command sets up all the necessary hardware to move data written directly to the VXI device dependent register at offset 20_{16} into the desired queue. This command also allows the user to retrieve data through the hardware FIFO data path in a similar fashion to loading the queues.

TRACe:DATA:FEED <trace_name>,<data_handle>

Where <trace_name> is TCH1 | TCH2 | TCH3 | TCH4 for transmit queues and RCH1 | RCH2 | RCH3 | RCH4 for receive queues

Where <data_handle> is FIFO | NONE. If FIFO is selected, the connection is established NONE breaks the connection

EXAMPLES

`TRAC:DATA:FEED TCH2,FIFO`	*Selects transmit Channel 1 and established a connection*

`*OPC?`
`1`
	*As the instrument must set the internal registers and initialize a DMA channel, *OPC? Command is used to determine whether the connection is properly established prior to sending data to the VM6068*

`TRAC:DATA:FEED?`
`NONE,TCH1`
	Returns NONE, which is the trace name of receive queues and TCH1which is the trace name of transmit queues

TRACe:FREE?

This command queries the amount of memory that is unused in a queue.

TRACe:FREE? <trace_name>

Where <trace_name> specifies the queue whose amount of unused memory is to be queried

EXAMPLES

```
TRAC:FREE? TCH1
100
```

Queries the amount of unused memory in transmit queue TCH1

```
TRAC:FREE? RCH2
1000
```

Queries the amount of unused memory in receive queue RCH2

TRACe:LENGth?

This command queries the number of characters in the specified queue.

TRACe:LENGth? <trace_name>	***Where <trace_name>specifies the queue whose number of characters present is to be queried***

EXAMPLES

`TRAC:LENG? TCH1`
`100`
	Queries the number of characters present in transmit queue TCH1

`TRAC:LENG? RCH3`
`1002`
	Queries the number of characters present in receive queue RCH3

TRACe:POINTs

This command sets the size of a transmit or receive queue.

TRACe:POINTs <trace_name>,<points>	***Where <trace_name> specifies the queue size is to be configured*** ***Where <points> specifies the size of the queue***

EXAMPLES

`TRAC:POIN TCH1,2500`	*Configures the size of transmit queue TCH1 as 2500 bytes*
`TRAC:POIN RCH4,4500`	*Configures the size of receive queue RCH4 as 4500 bytes*

`TRAC:POIN?`
`RCH4,4500`
	Queries the size of receive queue RCH4

XON/XOFF FUNCTIONALITY

SERial:RECeive:PACE

This command is used to pace the receiver. A channel's receive queue THReshold is monitored. When a THReshold limit has been exceeded, the appropriate action will be taken.

[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:PACE XON | NONE | IRQ | TRIGGER <trigline>

NONE	Self evident.
XON	When the receiving channel's queue capacity drops below its specified STOP threshold it will issue and XOFF. When the receiving channel's queue empties out freeing up more than its specified START threshold it will issue an XON.
IRQ	When the receiving channel's queue capacity drops below its specified STOP threshold it will trigger an interrupt.
TRIGGER	When the receiving channel's queue capacity drops below specified STOP threshold it will yank on the specified "trigline".

EXAMPLES

```
SER2:REC:PACE XON
```

```
SER:REC:PACE?
```

SERial:RECeive:PACE:THReshold:STOP

The user specifies the minimum number of free buffers (Stop Threshold) expressed as a percentage. This means that when the number of buffers available falls "below" the STOP THReshold an XOFF will be issued.

The stop threshold is not allowed to be less than 0.017857142 and must be less than the start threshold.

[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:PACE:THReshold:STOP <percent>

DEFINITIONS

QUEUE	:	The VM6068 has a queue for each of its 4 channels.
BUFFER	:	Each Queue has 28 buffers.
BUFFER RAM	:	Each of the 28 Buffers can allocate a maximum of roughly 8 k and a minimum of 28 as specifically documented in the TRACe:POINts command.
DEFAULT	:	0.30

EXAMPLES

`SER:REC:PACE:THR:STOP 0.3`	*This command says when the number of free Buffers drops below 30% issue an XOFF*

NOTE: *The percentage ultimately refers to some number of buffers rounded to the nearest buffer. The following example demonstrates this:*

```
SER1:REC:PACE:THR:STOP 0.24

SER1:REC:PACE:THR:STOP?
0.250000

SER1:REC:PACE:THR:STOP 0
-221, "Parameter error; minimum Stop threshold is 0.017857"

SER1:REC:PACE:THR:STOP 0.94
-221, "Parameter error; Stop threshold must be less than Start
threshold"
```

SERial:RECeive:PACE:THReshold:STARt

The user specifies the maximum number of free buffers (Start Threshold) expressed as a percentage. This means that AFTER an XOFF has been issued AND subsequently the number of buffers available rises "above" the STARt THReshold an XON will be issued.

The start threshold must be greater than the stop threshold and less than or equal to 100%.

[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:PACE:THReshold:STARt <percent>

See Definitions above.

Default: 0.70

EXAMPLES

```
SER1:REC:PACE:THR:START 0.70
```

This command says when the number of free Buffers rises above 70% issue an XON

```
SER1:REC:PACE:THR:START?
0.714286

SER1:REC:PACE:THR:START 0.10
-221, "Parameter error; Start threshold must be greater than Stop
threshold"

SER1:REC:PACE:THR:START 1.10
-221, "Parameter error; Start threshold must not be greater than
100%"
```

SERial:RECeive:XON

User definable XON character associated with pacing the receiver.

[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:XON <n>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <n> is an 8 bit binary value, the default is 17

EXAMPLES

```
SER1:REC:XON #H11

SER1:REC:XON?
17
```

SERial:RECeive:XOFF

User definable XOFF character associated with pacing the receiver.

[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:XOFF <n>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <n> is an 8 bit binary value

EXAMPLES

```
SER1:REC:XOFF #H13

SER1:REC:XOFF ?
19
```

SERial:TRANsmit:PACE

This command is used to pace the transmitter. When this channel receives an XOFF this channel's transmitter will be disabled.

[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:PACE XON | NONE

SERial:TRANsmit:XON

User definable XON character associated with pacing the transmitter.

[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:XON <n>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <n> is an 8 bit binary Value the default is 17

SERial:TRANsmit:OFF

User definable XOFF character associated with pacing the transmitter.

[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:XOFF <n>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <n> is an 8 bit binary value; the default is 19

SERial:TRANsmit

Forces transmission of a character over the specified UART channel. This transmission will occur even if this transmit channel has been XOFFed.

[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit <n>

Where <channel> is 1 | 2 | 3 | 4 (default is Channel 1)

Where <n> is an 8 bit binary value

REGISTER ACCESS

The VM6068 module supports register access for very high speed data transfers.

LOADING DATA VIA THE HARDWARE FIFO INTERFACE

The VXI device-dependent register at offset 0x20 can be used for loading serial data via the Hardware FIFO interface. It must be ensured that a hardware-based data path for a particular channel has been established before the data can be loaded using the register.

In order to check if there is any space for the data bytes to be loaded into the transmit queue of the channel, the Status Byte at offset 0x22 must be read. If Bit 2 of the register is set to high, it indicates that there is room for at least one more byte in the transmit queue of the channel. If Bit 2 of the register is set to low, it indicates that the transmit queue of the channel is full and no more data can be loaded into it.

Since it may take some time between the loading of the data into the data register at offset 0x20 and the data being moved into the channel's transmit queue via the hardware based data path, enough time must be provided for the same before assuming that there is no further space in the transmit queue of the channel.

In order to set the End-of-data indicator, Bit 15 of the VXI device-dependent register must be set to high for the last data byte that is being loaded into the module.

READING DATA VIA THE HARDWARE FIFO INTERFACE

The serial data which has been received in the receive queue of the channel whose hardware-based data path has been enabled can be read via the VXI device-dependent register at offset 0x20. The lower 8 bits of the register return the data values while the upper 8 bits return the error values corresponding to the received data values. The bit layout of the upper 8 bits of the register is as given below.

Bit 15 End of message indicator
Bit 14 Buffer closed due to control character match (last byte)
Bit 13 The buffer was closed due to consecutive IDLes
Bit 12 Address match - only used in multi-drop mode; 0 for UADDR2, 1 for UADDR1
Bit 11 A break sequence was received while receiving data into this buffer
Bit 10 Parity error or framing error occurred on last byte
Bit 9 A receiver overrun occurred during message reception
Bit 8 Carrier detect signal lost during message reception

In order to check if valid data is actually available in the VXI device-dependent register at offset 0x20, the Status Byte register at offset 0x22 must be read. If Bit 1 of Status Byte register is set to high, it indicates that valid data is still available at offset 0x20. If Bit 1 of the Status Byte Register is set to low, it indicates that no more valid data is available in the data register at offset 0x20.

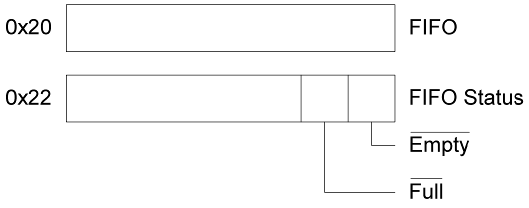


FIGURE 3-3: A16 REGISTER BITS

When a FEED ALL command is made, in addition to the receive data in the lower 8 bits (bits 0 through 7), the channel number is also indicated in bit 8 and 9:

Bit 9	Bit 8	Channel
0	0	RCH1
0	1	RCH2
1	0	RCH3
1	1	RCH4

For status data, bit 10 is the OR of the normal bits 8, 9, and 10, so bits 8 and 9 can be used for the channel indicator.

In FEED ALL, transmit data the lower 8 bits (bits 0 to 7) are still for data. The end indicator is bit 15. The user must place the channel indicator in bits 8 and 9:

Bit 9	Bit 8	Channel
0	0	TCH1
0	1	TCH2
1	0	TCH3
1	1	TCH4

Refer to page 120 for more information on the FEED:ALL command.

HDLC PROGRAMMING EXAMPLE

```
REM Program to demonstrate HDLC and address recognition

REM This program sends messages to 5 different addresses
REM Each channel should receive the message addressed to it and
REM the broadcast message

REM The physical connections are:
REM Channel 1 transmit is connected to channel 3 receive
REM Channel 2 transmit is connected to channel 4 receive
REM Channel 3 transmit is connected to channel 1 receive
REM Channel 4 transmit is connected to channel 2 receive

REM Include library declarations
'$INCLUDE: 'c:\nivxi\include\NIVXI.INC'

DECLARE SUB ibwrt (LA%, Cmmand$)
DECLARE SUB ibrd (add%, l$)
DECLARE FUNCTION getticks& ()
DECLARE SUB delayticks (ticks&)

REM Enable CTRL-C to break
KEY 20, CHR$(4) + CHR$(46)
ON KEY(20) GOSUB CTRLC:
KEY(20) ON

DIM ibcnt AS INTEGER
DIM l AS STRING * 100

REM Initialize the VXI library
ret% = InitVXIlibrary%

REM Locate a module
ret% = FindDevLA%("", -1, 261, -1, -1, -1, -1, LA%)

REM Confirm we have a card
IF ret% <> 0 THEN
    PRINT "No 6068 card found"
    GOTO CTRLC:
END IF

REM Set time-out to 1000 mS
ret% = WSsetTmo%(1000&, timo&)
IF ret% <> 0 THEN
    PRINT "Time out value not set"
END IF
```

```
REM Start in a known state
CALL ibwrt(LA%, "*rst")

REM Read error queue once
CALL ibwrt(LA%, "syst:err?")
CALL ibrd(LA%, l$)

REM Set the baud rates
CALL ibwrt(LA%, "BAUD1 100000,1")
CALL ibwrt(LA%, "BAUD2 100000,1")
CALL ibwrt(LA%, "BAUD3 100000,1")
CALL ibwrt(LA%, "BAUD4 100000,1")

REM Say what electrical standard to use
CALL ibwrt(LA%, "serial1:standard 449")
CALL ibwrt(LA%, "serial2:standard 449")
CALL ibwrt(LA%, "serial3:standard 449")
CALL ibwrt(LA%, "serial4:standard 449")

REM Use HDLC
CALL ibwrt(LA%, "serial1:protocol hdlc")
CALL ibwrt(LA%, "serial2:protocol hdlc")
CALL ibwrt(LA%, "serial3:protocol hdlc")
CALL ibwrt(LA%, "serial4:protocol hdlc")

REM Set channel 1's address to 16705 (#h4141) and mask to full compare
REM This makes the address appear as "AA" in the receive string
REM Also set to recognize an address of 8224 (#h2020)
CALL ibwrt(LA%, "serial1:rec:hmask #hffff")
CALL ibwrt(LA%, "serial1:rec:haddress 1,#h4141")
CALL ibwrt(LA%, "serial1:rec:haddress 2,#h4141")
CALL ibwrt(LA%, "serial1:rec:haddress 3,#h4141")
CALL ibwrt(LA%, "serial1:rec:haddress 4,#h2020")

REM Set channel 2's address to 16706 (#h4142) and mask to full compare
REM This makes the address appear as "AB" in the receive string
REM Also set to recognize an address of 65535
CALL ibwrt(LA%, "serial2:rec:hmask #hffff")
CALL ibwrt(LA%, "serial2:rec:haddress 1,#h4142")
CALL ibwrt(LA%, "serial2:rec:haddress 2,#h4142")
CALL ibwrt(LA%, "serial2:rec:haddress 3,#h4142")
CALL ibwrt(LA%, "serial2:rec:haddress 4,#h2020")

REM Set channel 3's address to 16707 (#h4143) and mask to full compare
REM This makes the address appear as "AC" in the receive string
REM Also set to recognize an address of 65535
CALL ibwrt(LA%, "serial3:rec:hmask #hffff")
CALL ibwrt(LA%, "serial3:rec:haddress 1,#h4143")
CALL ibwrt(LA%, "serial3:rec:haddress 2,#h4143")
CALL ibwrt(LA%, "serial3:rec:haddress 3,#h4143")
CALL ibwrt(LA%, "serial3:rec:haddress 4,#h2020")
```

```
REM Set channel 4's address to 16708 (#h4144) and mask to full compare
REM This makes the address appear as "AD" in the receive string
REM Also set to recognize an address of 65535
CALL ibwrt(LA%, "serial4:rec:hmask #hffff")
CALL ibwrt(LA%, "serial4:rec:haddress 1,#h4144")
CALL ibwrt(LA%, "serial4:rec:haddress 2,#h4144")
CALL ibwrt(LA%, "serial4:rec:haddress 3,#h4144")
CALL ibwrt(LA%, "serial4:rec:haddress 4,#h2020")

REM Build 5 messages, one for each address and one broadcast
msg1$ = "#0" + chr$(&H41) + chr$(&h41) + char$(info) + "Message 1"
msg2$ = "#0" + chr$(&H42) + chr$(&h41) + char$(info) + "Message 2"
msg3$ = "#0" + chr$(&H43) + chr$(&H41) + char$(info) + "Message 3"
msg4$ = "#0" + chr$(&H44) + chr$(&H41) + char$(info) + "Message 4"
msgb$ = "#0" + chr$(&H20) + chr$(&H20) + char$(info) + "Broadcast"

REM Set the data retrieval format
CALL ibwrt(LA%, "format 1 integer")
CALL ibwrt(LA%, "format 2 integer")
CALL ibwrt(LA%, "format 3 integer")
CALL ibwrt(LA%, "format 4 integer")


WHILE 1
   REM Send all 5 messages to all 4 channels
   FOR i% = 1 to 4
      tchan$ = "tch" + chr$(48+i%)
      CALL ibwrt(LA%, "trace:data " + tchan$ + "," + msg1$)
      CALL ibwrt(LA%, "trace:data " + tchan$ + "," + msg2$)
      CALL ibwrt(LA%, "trace:data " + tchan$ + "," + msg3$)
      CALL ibwrt(LA%, "trace:data " + tchan$ + "," + msg4$)
      CALL ibwrt(LA%, "trace:data " + tchan$ + "," + msgb$)
   NEXT

   REM Look at messages received and verifies they are correct

   REM Read channel 1 for data message
   CALL ibwrt(LA%, "trace:data? rch1")
   CALL ibrd(LA%, l$)

   REM Include all except the CRC in the test string
   test$ = left$(l$, ibcnt%-3)

   REM If the received data doesn't match, then error
   IF (test$ <> msg1$) THEN
      PRINT "Message doesn't match. Expected: "; msg1$; " Received: "; test$
      GOTO CTRLC
   END IF

   REM Read channel 1 for broadcast message
   CALL ibwrt(LA%, "trace:data? rch1")
   CALL ibrd(LA%, l$)
```

```
REM Include all except the CRC in the test string
test$ = left$(l$, ibcnt%-3)

REM If the received data doesn't match, then error
IF (test$ <> msgb$) THEN
    PRINT "Message doesn't match. Expected: "; msgb$; " Received: "; test$
    GOTO CTRLC
END IF

REM Read channel 2 for data message
CALL ibwrt(LA%, "trace:data? rch2")
CALL ibrd(LA%, l$)

REM Include all except the CRC in the test string
test$ = left$(l$, ibcnt%-3)

REM If the received data doesn't match, then error
IF (test$ <> msg2$) THEN
    PRINT "Message doesn't match. Expected: "; msg2$; " Received: "; test$
    GOTO CTRLC
END IF

REM Read channel 2 for broadcast message
CALL ibwrt(LA%, "trace:data? rch2")
CALL ibrd(LA%, l$)

REM Include all except the CRC in the test string
test$ = left$(l$, ibcnt%-3)

REM If the received data doesn't match, then error
IF (test$ <> msgb$) THEN
    PRINT "Message doesn't match. Expected: "; msgb$; " Received: "; test$
    GOTO CTRLC
END IF

REM Read channel 3 for data message
CALL ibwrt(LA%, "trace:data? rch3")
CALL ibrd(LA%, l$)

REM Include all except the CRC in the test string
test$ = left$(l$, ibcnt%-3)

REM If the received data doesn't match, then error
IF (test$ <> msg3$) THEN
    PRINT "Message doesn't match. Expected: "; msg3$; " Received: "; test$
    GOTO CTRLC
END IF
```

```
    REM Read channel 3 for broadcast message
    CALL ibwrt(LA%, "trace:data? rch3")
    CALL ibrd(LA%, l$)

    REM Include all except the CRC in the test string
    test$ = left$(l$, ibcnt%-3)

    REM If the received data doesn't match, then error
    IF (test$ <> msgb$) THEN
        PRINT "Message doesn't match. Expected: "; msgb$; " Received: "; test$
        GOTO CTRLC
    END IF

    REM Read channel 4 for data message
    CALL ibwrt(LA%, "trace:data? rch4")
    CALL ibrd(LA%, l$)

    REM Include all except the CRC in the test string
    test$ = left$(l$, ibcnt%-3)

    REM If the received data doesn't match, then error
    IF (test$ <> msg4$) THEN
        PRINT "Message doesn't match. Expected: "; msg4$; " Received: "; test$
        GOTO CTRLC
    END IF

    REM Read channel 1 for broadcast message
    CALL ibwrt(LA%, "trace:data? rch4")
    CALL ibrd(LA%, l$)

    REM Include all except the CRC in the test string
    test$ = left$(l$, ibcnt%-3)

    REM If the received data doesn't match, then error
    IF (test$ <> msgb$) THEN
        PRINT "Message doesn't match. Expected: "; msgb$; " Received: "; test$
        GOTO CTRLC
    END IF

    REM Indicate one successful pass
    print "*";
WEND


CTRLC:

REM Close down the VXI library
ret% = CloseVXIlibrary%

END
```

```
REM*******************************************************************
REM
REM Delay a specified number of ticks
REM
REM*******************************************************************


SUB delayticks (ticks&)
starttime& = getticks&
    WHILE (getticks& - starttime&) < ticks&
    WEND
END SUB 'delayticks


REM ******************************************************************
REM
REM Read the system timer ticks (18.2 per second)
REM
REM*******************************************************************

FUNCTION getticks&

    DEF SEG = 0

    tickl% = PEEK(&H46C)
    tickm% = PEEK(&H46D)
    tickh% = PEEK(&H46E)
    ticku% = PEEK(&H46F)

    WHILE tickl% <> PEEK(&H46C)
        tickl% = PEEK(&H46C)
        tickm% = PEEK(&H46D)
        tickh% = PEEK(&H46E)
        ticku% = PEEK(&H46F)
    WEND

    ticku% = ticku% AND &H7F
    getticks& = ticku% * &H1000000 + tickh% * &H10000& + tickm% * &H100& +
    tickl%

    DEF SEG

END FUNCTION 'getticks
```

```
REM ******************************************************************
REM
REM MXI Subroutines
REM
REM ******************************************************************


SUB ibrd (add%, l$)

    SHARED ibcnt AS INTEGER

    mode% = 1

    status% = WSrd(add%, l$, LEN(l$), mode%, ibcntr&)

    IF (status% AND 3) <> 3 THEN
        PRINT "Error in ibrd:"; status%
        ibcnt% = ibcntr&
    END IF

END SUB  'ibrd


SUB ibwrt (LA%, Cmmand$)

    CommandLength& = LEN(Cmmand$)

    REM print "IBWRT Sends: "+left$(Cmmand$,CommandLength&)
    ReturnStatus% = WSwrt%(LA%, Cmmand$, CommandLength&, 3, ReturnCount&)

    IF ReturnStatus% <> 7 THEN
        PRINT "Error in ibwrt:"; ReturnStatus%
    END IF

END SUB  'ibwrt

_
```

VXI*PLUG&PLAY* DRIVER EXAMPLE

```
/*
 *                      APPLICATION FUNCTION
 *                      --------------------
 */
/*****************************************************************************/
/****************************************************************************
Function:               vtvm6068_diagnostic

Formal Parameters       ViSession instrHndl
                        - A unique handle to the instrument.

                        ViPInt16    result
                        - Returns the result of diagnostic. The value of one
                        means diagnostic passed.
                        A value of zero means the diagnostic failed.

Return Values:          Returns VI_SUCCESS if successful.
                        else returns error value of the error encountered.

Description             For this diagnostic to be successful, the loop-back
                        connector must connect transmit channel 1 to receive
                        channel 3. This application function shows how to
                        group individual driver functions to transmit data on
                        one channel and receive the same data on another
                        channel using loop-back connector.
                        Please have a look at the source code of this function
                        to help you use the driver functions in your
                        application. If data transmitted on channel 1 is
                        received via the loop-back connector on channel 3 then
                        it means the diagnostic passed, otherwise the
                        diagnostic failed.
*****************************************************************************/
ViStatus _VI_FUNC vtvm6068_diagnostic(ViSession instrHndl, ViPInt16 result)
{
/*
 * Variable used to store the return status of the function
 */
    ViStatus status = VI_NULL;

    ViInt16  txData[100], rxData[200],  rxError[200];

    ViInt32             numBytesRxed = 0,
                        index = 0;
```

```
/*
 * Validating the input session handle
 */

status = vtvm6068_validSession(instrHndl);
    if (status < VI_SUCCESS)
        return status;

    if (result == VI_NULL)
        return VI_ERROR_PARAMETER2;

status = vtvm6068_reset (instrHndl);
    if (status < VI_SUCCESS)
        return status;

/* Setup baud rate of 250,000 BAUD for channel 1 */
status = vtvm6068_setupBaudRate (instrHndl, vtvm6068_GENERATOR_1,
                                 250000, vtvm6068_DIVISOR_16);
    if (status < VI_SUCCESS)
        return vtvm6068_ERROR_SETTING_BAUD_RATE;

/* Setup baud rate of 250,000 BAUD for channel 3 */
status = vtvm6068_setupBaudRate (instrHndl, vtvm6068_GENERATOR_3,
                                 250000, vtvm6068_DIVISOR_16);
    if (status < VI_SUCCESS)
        return vtvm6068_ERROR_SETTING_BAUD_RATE;

/* Setup Tx channel 1 parameters */
status = vtvm6068_setupTxRxChannel (instrHndl,vtvm6068_CHANNEL_1,
                                 vtvm6068_CONFIGURE_AS_TX,vtvm6068_PARITY_NONE,
                                 vtvm6068_NUM_BITS_8, vtvm6068_STOP_BITS_1,
                                 vtvm6068_INTERFACE_232, vtvm6068_PROT_UART);
    if (status < VI_SUCCESS)
        return vtvm6068_ERROR_SETTING_TX_CH_PARAMETERS;

/* Setup Rx channel 3 parameters */
    status = vtvm6068_setupTxRxChannel (instrHndl, vtvm6068_CHANNEL_3,
                                 vtvm6068_CONFIGURE_AS_RX, vtvm6068_PARITY_NONE,
                                 vtvm6068_NUM_BITS_8, vtvm6068_STOP_BITS_1,
                                 vtvm6068_INTERFACE_232, vtvm6068_PROT_UART);
    if (status < VI_SUCCESS)
        return vtvm6068_ERROR_SETTING_RX_CH_PARAMETERS;
```

```
/* Transmit 1000 bytes on channel 1 using Word Serial */
status = vtvm6068_loadDataViaWS (instrHndl, vtvm6068_TCH1, txData, 100);
    if (status < VI_SUCCESS)
        return vtvm6068_ERROR_LOADING_DATA_ON_TX_CHANNEL;

/* Read the receive queue for channel 3 using H/W FIFO */
status = vtvm6068_connectDisconnectHWFIFO (instrHndl, vtvm6068_CONNECT_HWFIFO,
                                 vtvm6068_RX_CHANNEL, vtvm6068_CHANNEL_3);
    if (status < VI_SUCCESS)
        return vtvm6068_ERROR_ENABLING_FIFO_FOR_RX_CHANNEL;

status = vtvm6068_readDataViaFIFO (instrHndl, rxData, rxError, &numBytesRxed);
    if (status < VI_SUCCESS)
        return vtvm6068_ERROR_READING_DATA_ON_RX_CHANNEL;

    if (numBytesRxed != 100)
    {
        *result = 0; return VI_SUCCESS;
    }

/* Compare the Tx data with the Rx data */
    for (index = 0; index < 100; index++)
        if (txData[index] != rxData[index])
        {
             *result = 0;
             return VI_SUCCESS;
        }

    *result = 1;

    return VI_SUCCESS;
}
```

SECTION 4

COMMAND DICTIONARY

INTRODUCTION

This section presents the instrument command set. It begins with an alphabetical list of all the commands supported by the VM6068 divided into three sections: IEEE 488.2 commands, the instrument specific SCPI commands and the required SCPI commands. With each command is a brief description of its function, whether the command's value is affected by the ***RST** command and its default value.

The remainder of this section is devoted to describing each command, one per page, in detail. The description is presented in a regular and orthogonal way assisting the user in the use of each command. Every command entry describes the exact command and query syntax, the use and range of parameters and a complete description of the command's purpose.

ALPHABETICAL COMMAND LISTING

The following tables provide an alphabetical listing of each command supported by the VM6068 along with a brief description. If an X is found in the column titled ***RST**, then the value or setting controlled by this command is possibly changed by the execution of the ***RST** command. If no X is found, then ***RST** has no effect. The Reset value column gives the value of each command's setting when the unit is powered up or when a ***RST** command is executed.

TABLE 4-1: IEEE 488.2 COMMON COMMANDS

Command	Description	*Rst	Reset Value
*CLS	Clear the Status Register.		N/A
*ESE	Set the Event Status Enable Register.		N/A
*ESR	Query the Standard Event Status Register.		N/A
*IDN?	Query the module identification string.		N/A
*OPC	Set the OPC bit in the Event Status Register.	X	0
*RST	Reset the module to a known state.		N/A
*SRE	Set the Service Request Enable Register.		N/A
*STB?	Query the Status Byte Register.		N/A
*TST?	Run a self-test and report the result.		N/A
*WAI	Wait for operations to complete.		N/A

TABLE 4-2: INSTRUMENT SPECIFIC SCPI COMMANDS

Command	Description	*Rst	Reset Value
BAUD	Sets the rate of a baud rate generator	X	9600, 16
FORMat:DATA	Sets the retrieved data format	X	ASCII
SERial:BITS	Sets the number of data bits	X	8
SERial:CLOCk	Sets the direction of the bi-directional clock	X	IN
SERial:CONTrol:CTS	Sets the use of hardware handshake lines	X	0
SERial:CRC	Selects CRC generation in HDLC mode	X	NONE
SERial:PROTocol	Sets the OSI level 2 protocol	X	UART
SERial:RECeive:CLOCk:DIVide	Selects baud clock divide ratio	X	Divide Ratio 16
SERial:RECeive:CLOCk:SOURce	Selects the baud rate clock source	X	CH1/INT1, CH2/INT2, etc.
SERial:RECeive:CODE	Sets the receive decoding method	X	NRZ
SERial:RECeive:ERRor:MASK	Sets a mask of what type of errors will be reported	X	1
SERial:RECeive:HADDress	Set HDLC address		N/A
SERial:RECeive:HMASk	Set HDLC address mask		N/A
SERial:RECeive:IDLe	Sets characters times before BD closes	X	1
SERial:RECeive:PACE	Sets the pace to the receiver	X	NONE
SERial:RECeive:PACE:THReshold:STARt	Sets the maximum number of free buffers expressed as a percentage	X	0.70
SERial:RECeive:PACE:THReshold:STOP	Sets the minimum number of free buffers expressed as a percentage	X	0.30
SERial:RECeive:PARity	Sets the receiver’s parity type.	X	NONE
SERial:RECeive:STATus?	Query for serial reception errors		N/A
SERial:RECeive:XOFF	Pacing the receiver	X	19
SERial:RECeive:XON	Pacing the receiver	X	17
SERial:STANdard	Sets the physical interface standard	X	OFF
SERial:TRANsmit	Forces transmission of a character		N/A
SERial:TRANsmit:CLOCk:DIVide	Selects the baud clock divide ratio	X	Divide Ratio 16
SERial:TRANsmit:CLOCk:SOURce	Selects baud rate clock source	X	CH1/INT1, CH2/INT2, etc
SERial:TRANsmit:CODE	Sets the transmit encoding method	X	NRZ
SERial:TRANsmit:PACE	Pace the transmitter		N/A
SERial:TRANsmit:PARity	Sets the transmitter’s parity type	X	NONE
SERial:TRANsmit:SBITs	Sets the transmitter’s number of stop bits	X	1
SERial:TRANsmit:XOFF	Pacing the transmitter	X	19
SERial:TRANsmit:XON	Pacing the transmitter	X	17
TRACe:CLOSe	Manually close a BD feature		N/A
TRACe:DATA	Transfers data into and out of the module.		N/A
TRACe:DATA:FEED	Establishes a hardware connection to a data queue	X	NONE
TRACe:FREE?	Queries the space left in a queue		N/A
TRACe:LENGth?	Queries the number of entries in a queue		N/A
TRACe:POINts	Sets the size of a queue	X	1/8 of buffer RAM

TABLE 4-3: SCPI REQUIRED COMMANDS

Command	Description	*Rst	Reset Value
STATus:OPERation:CONDition?	Queries the Operation Status Condition Register	X	
STATus:OPERation:ENABle	Sets the Operation Status Enable Register	X	
STATus:OPERation:EVENt?	Queries the Operation Status Event Register	X	
STATus:PRESet	Presets the Status Register	X	
STATus:QUEStionable: CONDition?	Queries the Questionable Status Condition Register	X	
STATus:QUEStionable:ENABle	Sets the Questionable Status Enable Register	X	
STATus:QUEStionable:EVENt?	Queries the Questionable Status Event Register	X	
SYSTem:ERRor?	Queries the Error Queue	X	Clears queue
SYSTem:VERsion?	Queries which version of the SCPI standard the module complies with		N/A

COMMAND DICTIONARY

The remainder of this section is devoted to the actual command dictionary. Each command is fully described on its own page. In defining how each command is used, the following items are described:

Purpose	Describes the purpose of the command.
Type	Describes the type of command such as an event or setting.
Command Syntax	Details the exact command format.
Command Parameters	Describes the parameters sent with the command and their legal range.
Reset Value	Describes the values assumed when the *RST command is sent.
Query Syntax	Details the exact query form of the command.
Query Parameters	Describes the parameters sent with the command and their legal range. The default parameter values are assumed the same as in the command form unless described otherwise.
Query Response	Describes the format of the query response and the valid range of output.
Description	Describes in detail what the command does and refers to additional sources.
Examples	Present the proper use of each command and its query (when available).
Related Commands	Lists commands that affect the use of this command or commands that are affected by this command.

IEEE 488.2 COMMON COMMANDS

*CLS

Purpose	Clears the Status Register	
Type	IEEE 488.2 Common Command	
Command Syntax	*CLS	
Command Parameters	None	
Reset Value	N/A	
Query Syntax	None	
Query Parameters	N/A	
Query Response	N/A	
Description	This command clears all event registers, clears the OPC flag and clears all queues (except the output queue).	
Examples	**Command / Query**	**Response / Descriptions**
	`*CLS`	*(Clears all status and event registers)*
Related Commands	None	

*ESE

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Sets the bits of the Event Status Enable Register</td></tr>
<tr><td><b>Type</b></td><td colspan="2">IEEE 488.2 Common Command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">*ESE <mask></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><mask> = numeric ASCII value from 0 to 255</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">*ESE?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">Numeric ASCII value from 0 to 255</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Event Status Enable command is used to set the bits of the Event Status Enable Register. See ANSI/IEEE 488.2-1987 section 11.5.1 for a complete description of the ESE register. A value of 1 in a bit position of the ESE register enables generation of the ESB (Event Status Bit) in the Status Byte by the corresponding bit in the ESR. If the ESB is set in the SRE register then an interrupt will be generated. See the *ESR? command for details regarding the individual bits. The ESE register layout is:

Bit 0 - Operation Complete
Bit 1 - Request Control (not used in the VM6068)
Bit 2 - Query Error
Bit 3 - Device Dependent Error (not used in the VM6068)
Bit 4 - Execution Error
Bit 5 - Command Error
Bit 6 - User Request (not used in the VM6068)
Bit 7 - Power On

The Event Status Enable query reports the current contents of the Event Status Enable Register.</td></tr>
<tr><td rowspan="3"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>*ESE 36</code></td><td></td></tr>
<tr><td><code>*ESE?</code></td><td>36 <i>(Returns the value of the event status enable register)</i></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">*ESR</td></tr>
</table>

*ESR?

Purpose	Queries and clears the Standard Event Status Register	
Type	IEEE 488.2 Common Command	
Command Syntax	None - Query Only	
Command Parameters	N/A	
Reset Value	N/A	
Query Syntax	ESR?	
Query Parameters	None	
Query Response	Numeric ASCII value from 0 to 255	
Description	The Event Status Register query - queries and clears the contents of the Standard Event Status Register. This register is used in conjunction with the ESE register to generate the ESB (Event Status Bit) in the Status Byte. The layout of the ESR is: Bit 0 - Operation Complete Bit 1 - Request Control (not used in the VM6068, always 0) Bit 2 - Query Error Bit 3 - Device Dependent Error (not used in the VM6068, always 0) Bit 4 - Execution Error Bit 5 - Command Error Bit 6 - User Request (not used in the VM6068, always 0) Bit 7 - Power On The Operation Complete bit is set by the VM6068 when it receives an *OPC command. The Query Error bit is set when data is over-written in the output queue. This could occur if one query is followed by another without reading the data from the first query. The Execution Error bit is set when an execution error is detected. See the section in the manual covering Error Messages for a list of execution error. Errors which range from -200 to -299 are execution errors. The Command Error bit is set when a command error is detected. See the section in this manual covering Error Messages for a list of command errors. Errors that range from -100 to -199 are command errors. The Power On bit is set when the module is first powered on or after it receives a reset via the VXI Control Register. Once the bit is cleared (by executing the *ESR? command) it will remain cleared.	
Examples	**Command / Query**	**Response *(Description)***
	`*ESR?`	4
Related Commands	*ESE	

*IDN?

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Queries the module for its identification string</td></tr>
<tr><td><b>Type</b></td><td colspan="2">IEEE 488.2 Common Command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">None - Query Only</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">*IDN?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">ASCII character string</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Identification query returns the identification string of the VM6068 module. The response is divided into four fields separated by commas. The first field is the manufacturer’s name, the second field is the model number, the third field is an optional serial number and the fourth field is the firmware revision number. If a serial number is not supplied, the third field is set to 0 (zero).</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>*IDN?</code></td><td>VXI Technology, Inc.,VM6068,0,1.xx
<i>(The revision listed here is for reference only; the response will always be the current revision of the instrument.)</i></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

*OPC

Purpose	Sets the OPC bit in the Event Status Register	
Type	IEEE 488.2 Common Command	
Command Syntax	*OPC	
Command Parameters	None	
Reset Value	N/A	
Query Syntax	*OPC?	
Query Parameters	None	
Query Response	1	
Description	The Operation Complete command sets the OPC bit in the Event Status Register when all pending operations have completed. The Operation Complete query will return a 1 to the output queue when all pending operations have completed.	
Examples	**Command / Query**	**Response *(Description)***
	`*OPC`	
	`*OPC?`	1
Related Commands	*WAI	

*RST

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Resets the module’s hardware and software to a known state</td></tr>
<tr><td><b>Type</b></td><td colspan="2">IEEE 488.2 Common Command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">*RST</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Reset command resets the module’s hardware and software to a known state. See the command index at the beginning of this chapter for the individual command settings associated with this command.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>*RST</code></td><td></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

*SRE

<table>
<tr><td><b>Purpose</b></td><td>Sets the Service Request Enable Register bits</td></tr>
<tr><td><b>Type</b></td><td>IEEE 488.2 Common Command</td></tr>
<tr><td><b>Command Syntax</b></td><td>*SRE <mask></td></tr>
<tr><td><b>Command Parameters</b></td><td><mask> = Numeric ASCII value from 0 to 255</td></tr>
<tr><td><b>Reset Value</b></td><td>N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td>*SRE?</td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Query Response</b></td><td>Numeric ASCII value from 64 to 255</td></tr>
<tr><td><b>Description</b></td><td>The Service Request Enable command is used to set the 8-bit Service Request Enable Register bits to generate a service request. If one of the bits is set and the corresponding bit in the Status Register becomes true, a Request True event will be sent. Bit 6 (Master Summary Status) is always set true regardless of what mask value is sent. See the IEEE 488.2 specification for additional information regarding the Service Request Enable Register and its use. The layout of the Service Request Enable Register is:

Bit 0 - Unused
Bit 1 - Unused
Bit 2 - Error Queue Has Data Enable
Bit 3 - Questionable Status Summary Enable (not used)
Bit 4 - Message Available Enable
Bit 5 - Event Status Bit Summary Enable
Bit 6 - Master Summary Status Enable (always 1)
Bit 7 - Operation Status Summary Enable

The Service Request Enable query fetches the current contents of the Service Request Enable Register.</td></tr>
<tr><td><b>Examples</b></td><td><table><tr><th>Command / Query</th><th>Response <i>(Description)</i></th></tr><tr><td><code>*SRE 4</code></td><td></td></tr><tr><td><code>*SRE?</code></td><td>4</td></tr></table></td></tr>
<tr><td><b>Related Commands</b></td><td>None</td></tr>
</table>

*STB?

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Queries the Status Byte Register</td></tr>
<tr><td><b>Type</b></td><td colspan="2">IEEE 488.2 Common Command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">None - Query Only</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">*STB?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">Numeric ASCII value from 0 to 255</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Read Status Byte query fetches the current contents of the Status Byte Register. See the IEEE 488.2 specification for additional information regarding the Status byte Register and its use. The layout of the Status Register is:

Bit 0 - Unused
Bit 1 - Unused
Bit 2 - Error Queue Has Data
Bit 4 - Questionable Status Summary (not used)
Bit 5 - Message Available
Bit 6 - Master Summary Status
Bit 7 - Operation Status Summary</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>*STB</code></td><td>16</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

*TST?

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Causes a self-test procedure to occur and queries the results</td></tr>
<tr><td><b>Type</b></td><td colspan="2">IEEE 488.2 Common Command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">None - Query Only</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">*TST?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">Numeric ASCII value from 0 to 143</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Self-Test query causes the VM6068 to run its self-test procedures and report on the results. The following tests are performed:

1. Each channel runs an internal loop-back self-test.
2. The buffer RAM runs a simple self-test.

The *TST? query returns a numeric ASCII value which has the following meaning:

Bit 0 - Channel 1 Failed
Bit 1 - Channel 2 Failed
Bit 2 - Channel 3 Failed
Bit 3 - Channel 4 Failed
Bit 4 - Unused
Bit 5 - Unused
Bit 6 - Unused
Bit 7 - RAM Test Failed

A bit value of 1 in any location indicates a failure while a 0 value indicates that the test passed. The RAM test failed bit indicates that the buffer RAM used for the data queues, failed to pass a simple pseudo random pattern test or an all zeros test.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>*TST</code></td><td>0</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">*TST? 0</td></tr>
</table>

*WAI

Purpose	Halts execution of commands and queries until the No Operation Pending message is true
Type	IEEE 488.2 Common Command
Command Syntax	*WAI
Command Parameters	None
Reset Value	N/A
Query Syntax	None
Query Parameters	N/A
Query Response	N/A
Description	The Wait to Continue command halts the execution of commands and queries until the No Operation Pending message is true. This command makes sure that all previous commands have been executed before processing. It provides a way of synchronizing the module with its master.
Examples	**Command / Query**: `*WAI` **Response *(Description)***:
Related Commands	*OPC

INSTRUMENT SPECIFIC SCPI COMMANDS

BAUD

<table>
<tr><td><b>Purpose</b></td><td>Sets the baud rate for a given baud rate generator</td></tr>
<tr><td><b>Type</b></td><td>Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td>[SYSTem:][COMMunicate:]BAUD[<generator>] <baud_rate>[,<divisor>]</td></tr>
<tr><td><b>Command Parameters</b></td><td><generator> = 1 | 2 | 3 | 4
<baud_rate> = numeric ASCII value from 367 to 3e6
<divisor> = 1 | 8 | 16 | 32</td></tr>
<tr><td><b>Reset Value</b></td><td><baud_rate> = 9615.384615 on all generators
<divisor> = 16 on all generators</td></tr>
<tr><td><b>Query Syntax</b></td><td>[SYSTem:][COMMunicate:]BAUD?<generator></td></tr>
<tr><td><b>Query Parameters</b></td><td><generator> = 1 | 2 | 3 | 4</td></tr>
<tr><td><b>Query Response</b></td><td>Returns the values currently set for the <baud_rate> and <divisor> parameters in the following format: <baud_rate>,<divisor>
<baud_rate> = Numeric ASCII value from 367 to 3e6
<divisor> = 1 | 8 | 16 | 32</td></tr>
<tr><td><b>Description</b></td><td>The Baud command sets the baud rate for one of four baud rate generators available in the VM6068. Generator 1 is used for Channel 1, generator 2 is used for Channel 2, etc. Each generator is a series of programmable dividers driven by the CPU clock operating at 24 MHz. The programmed baud rate is rounded to the nearest available baud rate.

Because the generator’s output may be divided by a receive or transmit channel, a divisor parameter is allowed which will take into account this clock division in calculating the desired baud rate. For example, if an asynchronous receive channel is to operate at 19.2 kbaud and uses the ÷ 16 mode, the baud rate generator would need to be programmed to 307.2 kbaud. The user would instead specify 19.2 kbaud and a ÷16 divisor to attain the desired baud rate.

The Baud query reports the selected baud rate after rounding off to the nearest available baud rate. This provides a means to check that the baud rate is within the required tolerance.

The product of the baud rate and the divisor must ≤ 24e6.

<b>NOTE</b> A divider value of “1” is not useful for UART operation. The line drivers are only good up to a 5 MHz bit rate.</td></tr>
<tr><td><b>Examples</b></td><td><table><tr><th>Command / Query</th><th>Response <i>(Description)</i></th></tr><tr><td><code>BAUD 2 38400</code></td><td></td></tr><tr><td><code>BAUD? 2</code></td><td>38400,16</td></tr></table></td></tr>
<tr><td><b>Related Commands</b></td><td>None</td></tr>
</table>

CALibration:SECure:STATe

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Secure/unsecure storing information in non-volatile memory</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">CALibration:SECure:STATe <boolean> , <security_code>
CALibration:SECure:STATe 1| ON</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><boolean> = 0 | 1 | OFF | ON
<security_code> = IEEE 488.2 definite or indefinite length block of the security code
Example: #16VM6068</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">1</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">CALibration:SECure:STATe?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">0 | 1</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The module powers up with the secure state enabled (or ON). While security is ON, no stores to non-volatile memory are allowed. This command turns the state ON or OFF. In order to disable the security state, the security code must be supplied. To turn ON security, the code does not need to be supplied. If it is supplied, the code is checked. The security code must be supplied in IEEE 488.2 definite or indefinite length arbitrary block format. The security code is “VM6068” and is case sensitive.
⚠ <b>Non-volatile storage commands should only be executed by qualified personnel. Changing these values incorrectly can cause the instrument to perform improperly</b></td></tr>
<tr><td><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td></td><td><code>CAL:SEC OFF #16VM6068</code></td><td><i>(Turn security off in preparation of a non-volatile memory store.)</i></td></tr>
<tr><td></td><td><code>CAL:SEC:STAT?</code></td><td>0 <i>(Indicates the calibration security is disabled so new information can be stored in non-volatile memory.)</i></td></tr>
<tr><td></td><td><code>CAL:SEC:STAT 1</code></td><td><i>(Turn calibration security on to prevent stores to non-volatile memory.)</i></td></tr>
<tr><td></td><td><code>CAL:SEC:STAT ?</code></td><td>1 <i>(Indicates the calibration security is enabled so that no new information can be stored in non-volatile memory.)</i></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">RS423FLAG</td></tr>
</table>

FORMat:DATA

Purpose	Sets the data format for retrieving received characters	
Type	Setting	
Command Syntax	FORMat[:DATA] <channel> <type>	
Command Parameters	<channel> = 1 | 2 | 3 | 4 <type> = ASCii | INTeger | HEXadecimal | OCTal | BINary	
Reset Value	N/A	
Query Syntax	FORMat[:DATA]? <channel>	
Query Parameters	<channel> = 1 | 2 | 3 | 4	
Query Response	Returns the currently set value of the <type> parameter	
Description	The Format Data command sets the data format for retrieving received characters.	
Examples	**Command / Query**	**Response *(Description)***
	`FORMAT 2 INT`	
	`FORMAT? 2`	INT
Related Commands	TRACe:DATA <trace_name>,(<block> | <NRf>{,<NRf>})	

FPGAREV ?

Purpose	Returns the current revision of the FPGA.	
Type	Query only	
Command Syntax	N/A	
Command Parameters	N/A	
Reset Value	N/A	
Query Syntax	FPGAREV ?	
Query Parameters	N/A	
Query Response	0 to 63	
Description	Report the current revision of the FPGA. First revision FPGAs are reported as 0. Second revision FPGAs are reported as 1.	
Examples	**Command / Query**	**Response *(Description)***
	FPGAREV ?	1
Related Commands	RS423FLAG?	

RS423FLAG

Purpose	Determine response to requests for RS-423	
Type	Setting	
Command Syntax	RS423FLAG <flag>	
Command Parameters	<flag> = Numeric ASCII value 0 to 2	
Reset Value	N/A	
Query Syntax	RS423FLAG ?	
Query Parameters	N/A	
Query Response	0 to 2	
Description	Older VM6068 boards contain an undocumented RS-423 mode which is not available on newer boards. Boards with an FPGAREV? of "0" are the only boards that allow this undocumented mode. To provide some measure of backward compatibility, the RS423FLAG is stored in non-volatile memory. The flag defaults to "0" if RS423FLAG has never been set. Older VM6068 boards respond to RS-423 requests based on the flag as follows: 0 - Allow the undocumented mode 1 - Don't allow the mode, issue an error 2 - Silently coerce the mode to RS-422 New VM6068 boards respond to RS-423 requests based on the flag as follows: 0 - Silently coerce the mode to RS-422 1 or 2 – Don't allow the mode, issue an error To use the command form, CAL:SEC:STAT must be "0".	
Examples	**Command / Query**	**Response *(Description)***
	`CAL:SEC:STAT 0, #16VM6068`	*(Turn security off)*
	`RS423FLAG 1`	*(Set the flag to 1)*
	`CAL:SEC:STAT 1`	*(Turn security back on)*
	`RS423FLAG?`	1 *(on a new VM6068 don't issue an error)*
Related Commands	CAL:SEC:STAT, FPGAREV?	

SERial:BITS

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Sets the number of transmit or receive data bits on the selected channel</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:BITS <bits></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<bits> = 5 | 6 | 7 | 8</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2"><bits> = 8 on all channels</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:BITS?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">Returns the currently set value of the <bits> parameter</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Serial Bits command sets the number of transmit and receive data bits on the selected channel. This command is only valid in UART mode. In non-UART mode, the query response is always 8. The query reports the number of transmit and receive data bits from the selected channel.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>SER4:BITS 7</code>
<code>SER4:BITS?</code></td><td>
7</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">SERial:PROTocol</td></tr>
</table>

SERial:CLOCk

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Sets the direction of the bi-directional clock</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:CLOCk <direction></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<direction> = IN | OUT</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">Bi-directional clocks are set for IN on all channels</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:CLOCk?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2"><direction> = IN | OUT</td></tr>
<tr><td><b>Description</b></td><td colspan="2">There is one bi-directional clock associated with each channel and one tri-state clock associated with each channel. The bi-directional clock is labeled RXCx on the connector pin out. At reset, the bi-directional clock is made an INput to the module. Under program control, this clock can be made an OUTput from the module. When a bi-directional clock is made an output, it sources the TXCx clock signal. When a bi-directional clock is made an input, it can be selected as a clock source for internal baud rate generation. A bi-directional clock is selected as a source with the designator EXTx.
<b>Note</b>: <i>There are certain restrictions on the use of the bi-directional clock as a clock source:</i>
<i>Channels 1 and 2 can only select EXT1 or EXT2</i>
<i>Channels 3 and 4 can only select EXT3 and EXT4</i>
<i>The tri-state clock is labeled TXC on the connector pin-out. At reset this clock is driven out. Under program control, this clock is tri-stated by setting SERial:CLOCk OUT.</i></td></tr>
<tr><td><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td></td><td><code>SER2:CLOC OUT</code></td><td></td></tr>
<tr><td></td><td><code>SER2:CLOC?</code></td><td>OUT</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">SERial:RECeive:CLOCk:SOURce
SERial:TRANsmit:CLOCk:SOURce</td></tr>
</table>

SERial:CONTrol:CTS

Purpose	Enables or disables the CTS handshaking on a serial channel
Type	Setting
Command Syntax	[SYSTem:][COMMunicate:]SERial[[<channel>]]:CONTrol:CTS<boolean>
Command Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)* <boolean> = 0 | 1 | OFF | ON Default value for Channel is 1
Reset Value	0 (All handshaking disabled OFF)
Query Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:CONTrol:CTS?
Query Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)*
Query Response	Returns the currently set value of the <boolean> parameter
Description	The Serial Control CTS command selects if CTS handshaking is to be used on a specific channel. The CTS input is a true hardware handshaking input and does not require CPU intervention to operate. If CTS handshaking is enabled, the input must be true for transmission to occur on the selected channel. If the input goes false mid-character, the current character is completed and transmission is stopped until the input is re-asserted. The Serial Control CTS query reports if this handshake mode is enabled or not.
Examples	**Command / Query** | **Response *(Description)***
	`SER3:COUN:CTS 1` |
	`SER3:CONT:CTS?` | 1
Related Commands	None

SERial:CRC

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Selects CRC generation in HDLC mode</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:CRC<type></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<type> = CCITT16 | CCITT32</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">At reset all channels are placed in the UART mode which has no CRC. When a channel is placed in HDLC mode, the CCITT16 CRC is selected.</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:CRC?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">CCITT16 | CCITT32 | NONE</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Serial Receive CRC command selects the type of automatic CRC generation and checking that the VM6068 performs when in HDLC mode. The CRC types are defined by the following polynomials:

CCITT16 = X16 +X12 + X5 + 1
CCITT32 = X32 + X26 + X23 + X22 + X16 +X12 +X11 +X10 + X8 +X7 +X5 + X4 + X2 + X1 + 1

Trying to set a CRC type when in UART mode will generate an error. Querying the CRC when in UART mode will return a value of NONE</td></tr>
<tr><td><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td></td><td><code>SER3:CRC CCITT32</code></td><td></td></tr>
<tr><td></td><td><code>SER3:CRC?</code></td><td>CCITT32</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">SERial:PROTocol</td></tr>
</table>

SERial:PROTocol

<table>
<tr><td><b>Purpose</b></td><td colspan="2">This command sets the serial interface OSI layer 2 protocol</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:PROTocol <type></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<type> = HDLC | UART</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2"><type> = UART</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:PROTocol?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2"><type> = HDLC | UART | UNKNOWN</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Serial Protocol command sets the serial interface OSI layer 2 protocol. The UNKNOWN response is what is returned if not in one of the other known protocols.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>SER2:PROT:HDLC</code>
<code>SER2:PROT?</code></td><td>
HDLC</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

SERial:RECeive:CLOCk:DIVide

Purpose	Selects the baud clock divide ratio used by the receiver
Type	Setting
Command Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:CLOCk:DIVide <ratio>
Command Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)* <ratio> = 1 | 8 | 16 | 32
Reset Value	All channels are set to a divide ratio of 16
Query Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:CLOCk:DIVide?
Query Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)*
Query Response	1 | 8 | 16 | 32
Description	The Serial Receive Clock Divide command sets the baud rate divider ratio used in receiver when sampling data. The divide ratio is normally set to 1 when synchronous clocking is selected and is normally set to 16 when asynchronous clocking is used. The other divide ratios are provided for further flexibility. It is important to consider the selected divide ratio when setting the desired baud rate. The supplied clock will have to operate at a rate equal to the desired baud rate times the divide ratio.
Examples	**Command / Query** **Response *(Description)*** `SER2:REC:CLOC:DIV 1` `SER2:REC:CLOC:DIV?` 1
Related Commands	SERial:RECeive:CLOCk:SOURce SERial:RECeive:BAUD

SERial:RECeive:CLOCk:SOURce

<table>
<tr><td><b>Purpose</b></td><td>Selects the baud rate clock source for a receiver</td></tr>
<tr><td><b>Type</b></td><td>Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td>[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:CLOCk:SOURce <source&gt;</td></tr>
<tr><td><b>Command Parameters</b></td><td>&lt;channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<source> = EXT1 | EXT2 | EXT3 | EXT4 | INT1 | INT2 | INT3 | INT4</td></tr>
<tr><td><b>Default Value</b></td><td>Channel 1 = INT1
Channel 2 = INT2
Channel 3 = INT3
Channel 4 = INT4</td></tr>
<tr><td><b>Query Syntax</b></td><td>[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:CLOCk:SOURce?</td></tr>
<tr><td><b>Query Parameters</b></td><td>&lt;channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td>EXT1 | EXT2 | EXT3 | EXT4 | INT1 | INT2 | INT3 | INT4</td></tr>
<tr><td><b>Description</b></td><td>The Serial Receive Clock Source command sets the baud rate clock source for a receiver. The clock source is either one of the four internal baud rate generators or an external source connected to the front panel mounted I/O connector. An external clock source is used when the channel is to be operated in synchronous mode. The receive channel will accept an externally provided source from the front panel when the EXT source is selected. Note: There are certain restrictions on which EXT can be used with which channels. Channels 1 and 2 can only be connected to EXT1 and EXT2. Channels 3 and 4 can only be connected to EXT3 and EXT4. There is no such restriction on the internal generators.</td></tr>
<tr><td><b>Examples</b></td><td><table><tr><th>Command / Query</th><th>Response <i>(Description)</i></th></tr><tr><td><code>SER3:REC:CLOC:SOUR EXT4</code></td><td></td></tr><tr><td><code>SER3:REC:CLOC:SOUR?</code></td><td>EXT4</td></tr></table></td></tr>
<tr><td><b>Related Commands</b></td><td>SERial:CLOCk
SERial:RECeive:CLOCk:DIVide
SERial:RECeive:BAUD</td></tr>
</table>

SERial:RECeive:CODE

<table>
<tr><td><b>Purpose</b></td><td>Sets the data decoding method for a receive channel</td></tr>
<tr><td><b>Type</b></td><td>Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td>[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:CODE <decode&gt;</td></tr>
<tr><td><b>Command Parameters</b></td><td>&lt;channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<decode> = NRZ | NRZM | NRZS | FM0 | FM1 | MANChester | DMANchester</td></tr>
<tr><td><b>Reset Value</b></td><td>All channels are set to NRZ</td></tr>
<tr><td><b>Query Syntax</b></td><td>[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:CODE?</td></tr>
<tr><td><b>Query Parameters</b></td><td>&lt;channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td>&lt;decode> = NRZ | NRZM | NRZS | FM0 | FM1 | MANC | DMAN | NONE
A query response of NONE indicates an unrecognized code type.</td></tr>
<tr><td><b>Description</b></td><td>Each channel contains a digital phase locked loop (DPLL) that can be programmed to decode a variety of different coding methods:

<b>NRZ</b> : Non-Return to Zero
<b>NRZM</b> : NRZI Mark
<b>NRZS</b> : NRZI Space
<b>FM0</b> : Reverse of FM1
<b>FM1</b> : Reverse of FM0
<b>MANC</b> : Manchester
<b>DMAN</b> : Differential Manchester (AKA Differential Biphase-L)

See previous section for more information on coding methods.

<b>Note</b>: <i>Here, when “levels” are mentioned, it refers to logical levels. Different electrical standards produce different voltage levels on the signal lines.</i></td></tr>
<tr><td><b>Examples</b></td><td><table>
<tr><th>Command / Query</th><th>Response <i>(Description)</i></th></tr>
<tr><td><code>SER3:REC:CODE MANC</code></td><td></td></tr>
<tr><td><code>SER3:REC:CODE?</code></td><td>MANC</td></tr>
</table></td></tr>
<tr><td><b>Related Commands</b></td><td>SERial:TRANsmit:CODE</td></tr>
</table>

SERial:RECeive:ERRor:MASK

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Masks reporting of selected errors</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:ERRor:MASK <bits></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<bits> = 0 | 1</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">1 = allows reporting of the error</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">None – Command Only</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Description</b></td><td colspan="2">This is a setup item that sets a mask of what types of errors will be reported by triggering the front panel LED. The mask has a direct correlation to the Buffer Descriptor. A bit of 1 allows the error, 0 masks the error.
Default mask : #H3B for UART mode
#HBF for HDLC mode
See this command in the previous section for error descriptions.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>SER3:REC:ERR:MASK #H00</code></td><td><i>(Masks all errors on Channel 3)</i></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:STATus? NEXT | ALL</td></tr>
</table>

SERial:RECeive:HADDress

<table>
<tr><td>**Purpose**</td><td colspan="2">Sets a receive channel's HDLC address</td></tr>
<tr><td>**Type**</td><td colspan="2">Setting</td></tr>
<tr><td>**Command Syntax**</td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:HADDress <pos>,<addr></td></tr>
<tr><td>**Command Parameters**</td><td colspan="2"><channel>= 1 | 2 | 3 | 4 *(default is Channel 1)*
<pos> = 1 | 2 | 3 | 4
<addr> = 0 to 65535</td></tr>
<tr><td>**Reset Value**</td><td colspan="2">N/A - all channels set to UART mode. When a channel is set to HDLC mode, all positions are set to 65535.</td></tr>
<tr><td>**Query Syntax**</td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:HADDress? <pos></td></tr>
<tr><td>**Query Parameters**</td><td colspan="2"><channel>= 1 | 2 | 3 | 4 *(default is Channel 1)*
<pos> = 1 | 2 | 3 | 4</td></tr>
<tr><td>**Query Response**</td><td colspan="2"><type> = ASCII number from 0 to 65535</td></tr>
<tr><td>**Description**</td><td colspan="2">This command is only valid in HDLC mode. When issued in another mode an error is generated.

In HDLC mode, each channel has five 16-bit registers for address recognition - one mask register and four address registers. This command deals with the address registers. As a frame is received the address is checked against the four address registers and then masked by the mask register. A one in the mask register represents a bit position for address comparison. A zero in the mask register represents a bit position that is not compared. Upon an address match, the address and data are stored in the buffer. If there is no address match, nothing is stored in the buffer.

Note: *For 8-bit addresses, the upper 8 bits of the mask register should be set to 0s. Only the low order 8 bits of the mask register and address registers are then used for address matching. If the mask register is set to all 0s, then all addresses are recognized. All address registers are used for address comparison all the time. If the user wants only one address to be recognized, all address registers must be set to that address.*

Note: *The least significant byte is received first. Example: A frame that begins with $7E (Flag), $68, $AA, ... is received. To recognize this as a 16 bit address, the mask should be set to $FFFF and one of the address register should be set to $AA68. To recognize this as an 8-bit address, the mask register should be set to $00FF and one of the address register should be set to $XX68 (the upper 8 bits don't matter).*</td></tr>
<tr><td rowspan="2">**Examples**</td><td>**Command / Query**</td><td>**Response *(Description)***</td></tr>
<tr><td>`SER2:REC:HADD 3,27`
`SER2:REC:HADD?`</td><td>
27</td></tr>
<tr><td>**Related Commands**</td><td colspan="2">SERial:RECeive:HMASk</td></tr>
</table>

SERial:RECeive:HMASk

Purpose	Sets a receive channel's HDLC address mask
Type	Setting
Command Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:HMASk <mask>
Command Parameters	<channel>= 1 | 2 | 3 | 4 *(default is Channel 1)* <mask> = 0 to 65535
Reset Value	N/A - All channels set to UART mode. When a channel is set to HDLC mode, the mask is set to 0.
Query Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:HMASk?
Query Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)*
Query Response	<mask> = ASCII number from 0 to 65535
Description	This command is only valid in HDLC mode. When issued in another mode an error is generated. In HDLC mode, each channel has five 16-bit registers for address recognition - one mask register and four address registers. This command deals with the mask register. As a frame is received the address is checked against the four address registers and then masked by the mask register. A one in the mask register represents a bit position for address comparison. A zero in the mask register represents a bit position that is not compared. Upon an address match, the address and data are stored in the buffer. If the address does not match, nothing is stored in the buffer. **Note**: *For 8-bit addresses, the upper 8 bits of the mask register should be set to 0s. Only the low order 8 bits of the mask register and address registers are then used for address matching. If the mask register is set to all 0s then all addresses are recognized. All address registers are used for address comparison all the time. If the user wants only one address to be recognized, all address registers must be set to that address.* **Note**: *The least significant byte of an address is received first. Example: A frame that begins with $7E (Flag), $68, $AA, ... is received. To recognize this as a 16-bit address, the mask should be set to $FFFF and one of the address register should be set to $AA68. To recognize this as an 8-bit address, the mask register should be set to $00FF and one of the address register should be set to $XX68 (the upper 8 bits don't matter).*
Examples	**Command / Query** | **Response *(Description)***
	`SER3:REC:HMAS 255` |
	`SER3:REC:HMAS?` | 255
Related Commands	SERial:RECeive:HADDress

SERial:RECeive:IDLe

<table>
<tr><td><b>Purpose</b></td><td>Specify character times/Buffer Descriptor</td></tr>
<tr><td><b>Type</b></td><td>Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td>[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:IDLe <idle_count></td></tr>
<tr><td><b>Command Parameters</b></td><td><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<idle_count> = 0 – 65535</td></tr>
<tr><td><b>Reset Value</b></td><td>1</td></tr>
<tr><td><b>Query Syntax</b></td><td>[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:IDLe?</td></tr>
<tr><td><b>Query Parameters</b></td><td><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td><idle_count> = 0 – 65535</td></tr>
<tr><td><b>Description</b></td><td>Programmable IDLe count Feature. The user can specify how many character times occur before a BD (Buffer Descriptor) is automatically closed. This is an unassigned integer value. Zero (0) specifies infinity.
For example: At 9600 baud, 1 start, 1 stop and 8 data bits the maximum idle of 65535 will take roughly 68 seconds to close. 104.166 µs per bit, times 10 = 1042 µs per character, times 65535 = 68 seconds.
<b>Note: This command operates in UART mode only.</b></td></tr>
<tr><td><b>Examples</b></td><td><table><tr><th>Command / Query</th><th>Response <i>(Description)</i></th></tr><tr><td><code>SER2:REC:IDL 65535</code></td><td></td></tr><tr><td><code>SER2:REC:IDL?</code></td><td>65535</td></tr></table></td></tr>
<tr><td><b>Related Commands</b></td><td>TRACe[:BUFFer]:CLOSe <channel></td></tr>
</table>

SERial:RECeive:PACE

Purpose	Used to pace the receiver	
Type	Setting	
Command Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:PACE NONE | XON | IRQ | RTS | TRIGGER <trigline>	
Command Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)* <trigline> = 0 - 7	
Reset Value	NONE	
Query Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:PACE?	
Query Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)*	
Query Response	NONE | XON | IRQ | RTS | TRIGGER <trigline>	
Description	This command is used to pace the receiver. A channel's receive queue THReshold is monitored. When a THReshold limit has been exceeded, the appropriate action will be taken.s	
Examples	**Command / Query**	**Response *(Description)***
	`SER1:REC:PACE XON`	
	`SER1:REC:PACE?`	XON
Related Commands	N/A	

SERial:RECeive:PACE:THReshold:STARt

Purpose	Specify the maximum number of free buffers	
Type	Setting	
Command Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:PACE:THReshold:STARt <percent>	
Command Parameters	<channel>= 1 | 2 | 3 | 4 *(default is Channel 1)* <percent> = Numeric value	
Reset Value	0.70	
Query Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:PACE:THReshold:STARt?	
Query Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)*	
Query Response	Numeric value	
Description	The user specifies the maximum number of free buffers (Start Threshold) expressed as a percentage. This means that AFTER an XOFF has been issued AND subsequently the number of buffers available rises "above" the STARt THReshold an XON will be issued. The start threshold must be greater than the stop threshold and less than or equal to 100%.	
Examples	**Command / Query**	**Response *(Description)***
	`SER1:REC:PACE:THR:START 0.70`	
	`SER1:REC:PACE:THR:START?`	0.714286
Related Commands	SERial:RECeive:PACE:THReshold:STOP	

SERial:RECeive:PACE:THReshold:STOP

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Specifies the minimum number of free buffers</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:PACE:THReshold:STOP <percent></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel> =1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<percent> =Numeric value</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">0.30</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:PACE:THReshold:STOP?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">Numeric value</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The user specifies the minimum number of free buffers (Stop Threshold) expressed as a percentage. This means that when the number of buffers available falls "below" the STOP THReshold an XOFF will be issued. The stop threshold is not allowed to be less than 0.017857142 and must be less than the start threshold.</td></tr>
<tr><td rowspan="3"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>SER2:REC:PACE:THR:STOP 0.24</code></td><td></td></tr>
<tr><td><code>SER2:REC:PACE:THR:STOP?</code></td><td>0.250000</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">SERial:RECeive:PACE:THReshold:STARt</td></tr>
</table>

SERial:RECeive:PARity

<table>
<tr><td><b>Purpose</b></td><td>Sets a receive channel's parity type</td></tr>
<tr><td><b>Type</b></td><td>Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td>[SYSTem:][COMMunicate:]SERial[<channel>][:RECeive]:PARity <type></td></tr>
<tr><td><b>Command Parameters</b></td><td><channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<type> = EVEN | ODD | NONE | IGN | ZERO | ONE</td></tr>
<tr><td><b>Reset Value</b></td><td><type> = N/A, parameter must be specified</td></tr>
<tr><td><b>Query Syntax</b></td><td>[SYSTem:][COMMunicate:]SERial[<channel>][:RECeive]:PARity?</td></tr>
<tr><td><b>Query Parameters</b></td><td><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td><type> = EVEN | ODD | NONE | IGN | ZERO | ONE | UNKNOWN</td></tr>
<tr><td><b>Description</b></td><td>The Serial Receive Parity command selects which parity mode to use on a selected receive channel. The following modes are supported:
<b>EVEN</b> : Received characters are checked for even parity.
<b>ODD</b> : Received characters are checked for odd parity.
<b>NONE</b> : No parity is checked on received characters. If a parity bit is sent to the receiver, it may cause a framing error. This also turns off parity for the transmitter.
<b>IGNore</b> : All parity errors on received data are ignored.
<b>ZERO</b> : Received characters are checked for a 0 parity bit.
<b>ONE</b> : Received characters are checked for a 1 parity bit.
<b>UNKNOWN</b> : This is what is returned in non-UART mode.
Enabling parity for the receiver (EVEN, ODD, ZERO or ONE) also enables parity for the transmitter. Disabling parity for the receiver (NONE) also disables parity for the transmitter. <b>Note: This command is only valid in UART mode.</b>
The Serial Receive Parity query reports the parity mode for the selected receive channel.</td></tr>
<tr><td><b>Examples</b></td><td><table><tr><th>Command / Query</th><th>Response <i>(Description)</i></th></tr><tr><td><code>SER2:REC:PAR EVEN</code></td><td></td></tr><tr><td><code>SER2:REC:PAR?</code></td><td>EVEN</td></tr></table></td></tr>
<tr><td><b>Related Commands</b></td><td>SERial:TRANsmit:PARity
SERial:PROTocol</td></tr>
</table>

SERial:RECeive:STATus?

Purpose	Queries for serial reception errors	
Type	Instrument specific	
Command Syntax	None – Query Only	
Command Parameters	N/A	
Reset Value	N/A	
Query Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:STATus? NEXT | ALL	
Query Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)*	
Query Response	Alpha-Numeric	
Description	Queries for serial reception errors. The “NEXT” or “ALL” Buffer Descriptors with data will be scanned for errors and a 16-bit word will be returned indicating the type of serial reception errors found. This word is the BD status word specifically. See **SERial:RECeive:ERRor:MASK** in previous section for error descriptions.	
Examples	**Command / Query**	**Response *(Description)***
	`SER2:REC:STAT? NEXT`	4, “Overrun; Channel 2”
Related Commands	[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:ERRor:MASK	

SERial:RECeive:XOFF

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Pace the receiver</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:XOFF <n></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<n> = 8 bit binary value</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:XOFF?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">8 bit binary value</td></tr>
<tr><td><b>Description</b></td><td colspan="2">User definable XOFF character associated with pacing the receiver.</td></tr>
<tr><td rowspan="3"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>SER1:REC:XOFF #H13</code></td><td></td></tr>
<tr><td><code>SER1:REC:XOFF?</code></td><td>19</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">N/A</td></tr>
</table>

SERial:RECeive:XON

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Pace the receiver</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:XON <n></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<n> = 8 bit binary value</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:XON?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">8 bit binary value</td></tr>
<tr><td><b>Description</b></td><td colspan="2">User definable XON character associated with pacing the receiver.</td></tr>
<tr><td rowspan="3"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>SER1:REC:XON #H11</code></td><td></td></tr>
<tr><td><code>SER1:REC:XON?</code></td><td>17</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">N/A</td></tr>
</table>

SERial:STANdard

Purpose	Sets the electrical interface standard for the selected channel
Type	Setting
Command Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:STANdard <standard>
Command Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)* <standard> = 232 | 422 | 449 | 485 | V.35 | 530 | OFF
Reset Value	<standard> = OFF
Query Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:STANdard?
Query Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)*
Query Response	<standard> = 232 | 422 | 449 | 485 | V.35 | 530 | OFF
Description	The Serial Standard command selects the desired physical interface standard for a given channel. The available standards are: RS-232, RS-422, RS-449, RS-485, EIA-530 and V.35. This command controls both the transmit and receive hardware. For additional information, refer to the section in this manual discussing the physical interface. The Serial Standard query reports the selected physical interface standard for a given channel. OFF means the drivers are tri-stated.
Examples	**Command / Query** — **Response *(Description)*** `SER2:STAN 422` `SER2:STAN?` — 422
Related Commands	N/A

SERial:TRANsmit

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Forces transmission of a character</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit <n></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<n> = 8 bit binary value</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">8 bit binary value</td></tr>
<tr><td><b>Description</b></td><td colspan="2">Forces transmission of a character over the specified UART channel. This transmission will occur even if this transmit channel has been set to XOFF.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td></td><td></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">N/A</td></tr>
</table>

SERial:TRANsmit:CLOCk:DIVide

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Selects the baud clock divide ratio used by the receiver</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:CLOCk:DIVide<ratio></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<ratio> = 1 | 8 | 16 | 32</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">All channels are set to a divide ratio of 16</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:CLOCk:DIVide?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">1, 8, 16, 32</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Serial Transmit Clock Divide command sets the baud rate divider ratio used in receiver when sampling data. The divide ratio is normally set to 1 when synchronous clocking is selected and is normally set to 16 when asynchronous clocking is used. The other divide ratios are provided for further flexibility.

It is important to consider the selected divide ratio when setting the desired baud rate. The supplied clock will have to operate at a rate equal to the desired baud rate times the divide ratio.</td></tr>
<tr><td rowspan="3"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>SER2:TRAN:CLOC:DIV 1</code></td><td></td></tr>
<tr><td><code>SER2:TRAN:CLOC:DIV?</code></td><td>1</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">SERial:TRANsmit:CLOCk:SOURce
BAUD</td></tr>
</table>

SERial:TRANsmit:CLOCk:SOURce

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Selects the baud rate clock source for a transmitter</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:CLOCk:SOURce <source></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<source>   = EXT1, EXT2, EXT3, EXT4, INT1, INT2, INT3, INT4</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">Channel 1 = INT1
Channel 2 = INT2
Channel 3 = INT3
Channel 4 = INT4</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:CLOCk:SOURce?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">EXT1, EXT2, EXT3, EXT4, INT1, INT2, INT3, INT4</td></tr>
<tr><td><b>Description</b></td><td colspan="2">This command sets the baud rate clock source for a transmitter. The clock source is one of the internal baud rate generators or an externally provided clock source. The external clock source is connected to the front panel mounted I/O connector allowing synchronous operation. . The receive channel will accept an externally provided source from the front panel when the EXT source is selected. Note: There are certain restrictions on which EXT can be used with which channels. Channels 1 and 2 can only be connected to EXT1 and EXT2. Channels 3 and 4 can only be connected to EXT3 and EXT4. There is no such restriction on the internal generators.</td></tr>
<tr><td rowspan="3"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>SER1:TRAN:CLOC:SOUR INT2</code></td><td></td></tr>
<tr><td><code>SER1:TRAN:CLOC:SOUR?</code></td><td>INT2</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">SERial:CLOCk
SERial:TRANsmit:CLOCk:DIVide
SERial:TRANsmit:BAUD</td></tr>
</table>

SERial:TRANSmit:CODE

Purpose	Sets the data encoding method for a transmit channel	
Type	Setting	
Command Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:CODE <encode>	
Command Parameters	<channel>= 1 | 2 | 3 | 4 *(default is Channel 1)* <encode> = NRZ | NRZM | NRZS | FM0 | FM1 | MANChester | DMANchester	
Reset Value	All channels are set to NRZ	
Query Syntax	[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:CODE?	
Query Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)*	
Query Response	<encode> = NRZ | NRZM | NRZS | FM0 | FM1 | MANC | DMAN | NONE A query response of NONE indicates an unrecognized code type.	
Description	Each channel contains a digital phase locked loop (DPLL) that can be programmed to decode a variety of different coding methods: **NRZ** : Non-Return to Zero **NRZM** : NRZI Mark **NRZS** : RZI Space **FM0** : Reverse of FM1 **FM1** : Reverse of FM0 **MANC** : Manchester **DMAN** : Differential Manchester (a.k.a. Differential Biphase-L) See SERial:RECeive:CODE in previous section for more information on coding methods. **Note**: *Here, when "levels" are mentioned, it refers to logical levels. Different electrical standards produce different voltage levels on the signal lines.*	
Examples	**Command / Query**	**Response *(Description)***
	`SER3:TRAN:CODE MANC`	
	`SER3:TRAN:CODE?`	MANC
Related Commands	SERial:RECeive:CODE	

SERial:TRANsmit:PACE

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Pace the transmitter</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:PACE <XON | NONE></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<XON | NONE> = Setting</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:PACE?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">XON or NONE</td></tr>
<tr><td><b>Description</b></td><td colspan="2">This command is used to pace the transmitter. When this channel receives an XOFF this channel’s transmitter will be disabled.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td></td><td></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">N/A</td></tr>
</table>

SERial:TRANsmit:PARity

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Sets a transmit channel’s parity type.</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting.</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:PARity <type></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<type> = EVEN | ODD | NONE | ZERO | ONE</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2"><type> = NONE</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:PARity?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2"><type> = EVEN | ODD | NONE | ZERO | ONE | UNKNOWN</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Serial Transmit Parity command selects which parity mode to use on a selected transmit channel. The following modes are supported:

<b>EVEN</b> : Transmitted characters are sent with an even parity bit.
<b>ODD</b> : Transmitted characters are sent with an odd parity bit.
<b>NONE</b> : No parity bit is sent on transmitted characters.
<b>ZERO</b> : Transmitted characters are sent with a 0 parity bit.
<b>ONE</b> : Transmitted characters are sent with a 1 parity bit.
<b>UNKNOWN</b> : This is what is returned in non-UART mode.

Enabling parity for the transmitter (EVEN, ODD, ZERO or ONE). Also enables parity for the receiver. Turning parity off (NONE) also disables parity for the receiver. This command is only valid in UART mode.

The Serial Transmit Parity query reports the selected parity mode for the selected transmit channel.</td></tr>
<tr><td><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td></td><td><code>SER2:TRAN:PAR ONE</code></td><td></td></tr>
<tr><td></td><td><code>SER2:TRAN:PAR?</code></td><td>ONE</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">SERial:RECeive:PARity
SERial:PROTocol</td></tr>
</table>

SERial:TRANsmit:SBITs

Purpose	Sets the number of stop bits on the selected transmit channel.	
Type	Setting.	
Command Syntax	[SYSTem:][COMMunicate:]SERial[<channel>][:TRANsmit]:SBITs <bits>	
Command Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)* <bits> = 1 | 2	
Reset Value	<bits> = 1	
Query Syntax	[SYSTem:][COMMunicate:]SERial[<channel>][:TRANsmit]:SBITs?	
Query Parameters	<channel> = 1 | 2 | 3 | 4 *(default is Channel 1)*	
Query Response	<bits> = 1 | 2	
Description	The Serial Transmit SBits command sets the number of stop bits on the selected transmit channel. The query reports the number of stop bits for the selected transmit channel. This command is not applicable for Rx channels. This command is only valid in the UART mode. For non-UART, the command is ignored.	
Examples	**Command / Query**	**Response *(Description)***
	`SER4:TRAN:SBITS 1`	
	`SER4:TRAN:SBITS?`	1
Related Commands	SERial:PROTocol	

SERial:TRANsmit:XOFF

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Pace the transmitter</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:XOFF <n></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<n> = 8 bit binary value</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">19</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:XOFF?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">8 bit binary value</td></tr>
<tr><td><b>Description</b></td><td colspan="2">User definable XOFF character associated with pacing the transmitter.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td></td><td></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">N/A</td></tr>
</table>

SERial:TRANsmit:XON

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Pace the transmitter</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Setting</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:XON &lt;n&gt;</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">&lt;channel>= 1 | 2 | 3 | 4 <i>(default is Channel 1)</i>
<n> = 8 bit binary value</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">17</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:TRANsmit:XON?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">&lt;channel> = 1 | 2 | 3 | 4 <i>(default is Channel 1)</i></td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">8 bit binary value</td></tr>
<tr><td><b>Description</b></td><td colspan="2">User definable XON character associated with pacing the transmitter.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td></td><td></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">N/A</td></tr>
</table>

SMARTREV ?

Purpose	Determine the current revision of the "Smart Application".	
Type	Query only	
Command Syntax	N/A	
Command Parameters	N/A	
Reset Value	N/A	
Query Syntax	SMARTREV ?	
Query Parameters	N/A	
Query Response	Software revision level of the "Smart Application".	
Description	Report the current revision of the "Smart Application".	
Examples	**Command / Query**	**Response** ***(Description)***
	SMARTREV ?	1.16 *(The "Smart Application" is revision 1.16)*
Related Commands		

TRACe:CLOSe

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Closes the Buffer Descriptor</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Instrument specific</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">TRACe[:BUFFer]:CLOSe <channel></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><channel> = 1 | 2 | 3 | 4</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">None – Command Only</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Description</b></td><td colspan="2">Manually close a Buffer Descriptor (BD). This feature is considered complimentary to the SERial:RECeive:IDLe command. If a user specifies zero (0) for an idle count, then this command would be used to close the BD. This typically would be done before a query.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>TRAC:BUFF:CLOS RCH1</code></td><td></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">[SYSTem:][COMMunicate:]SERial[<channel>]:RECeive:IDLe <idle_count></td></tr>
</table>

TRACe:DATA

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Loads or retrieves data to or from the specified queue using the word serial interface</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Data movement</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">TRACe:DATA<trace_name>,(<block> | <NRf> {,<NRf>})</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2"><trace_name> = TCH1, TCH2, TCH3, TCH4 for transmit queues
<block> = As defined in IEEE 488.2
<NRf> = As defined in IEEE 488.2</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">TRACe:DATA? <trace_name></td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><trace_name> = RCH1, RCH2, RCH3, RCH4 for receive queues</td></tr>
<tr><td><b>Query Default Value</b></td><td colspan="2"><trace_name> = RCH1</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">As set by the FORMat:DATA command</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Trace Data command is used to load and retrieve data to or from the transmit or receive queues using the word serial interface. Data may be loaded into a transmit queue using the block format or by using a series of comma separated values. See the FORMat:DATA command for details on data formats.

The Trace Data query is used to retrieve received data. The format of the received data is determined by the FORMat:DATA command. See the FORMat:DATA command for further details on data formatting.</td></tr>
<tr><td rowspan="3"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>TRAC:DATA TCH1,65,66,67</code></td><td></td></tr>
<tr><td><code>TRAC:DATA? RCH1</code></td><td>#31ABC</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">FORMat:DATA <channel>,<type></td></tr>
</table>

TRACe:DATA:FEED

Purpose	Used to establish a hardware FIFO based data path for a specified queue
Type	Setting
Command Syntax	TRACe:DATA:FEED<trace_name>,<data_handle> or TRACe:DATA:FEED ALL|NONE
Command Parameters	<trace_name> = TCH1, TCH2, TCH3, TCH4 for transmit queues <trace_name> = RCH1, RCH2, RCH3, RCH4 for receive queues <data_handle> = FIFO | NONE
Reset Value	NONE
Query Syntax	TRACe:DATA:FEED?
Query Parameters	None
Query Default Value	N/A
Query Response	ASCII string returns the <trace_name> of receiving data, a comma, and the <trace_name> transmitting data. Responses to TRACe:DATA:FEED ALL|NONE are ALL,ALL or NONE,NONE, respectively.
Description	The Trace Data command is used to establish a hardware-FIFO-based data path. This command sets up all the necessary hardware to move data written directly to the VXI device dependent register at offset 20_{16} into the desired queue. The data is written in binary format as an 8-bit byte. The register is actually word wide and the data should be right justified with the most significant bits set to indicate close of block. This command also allows the user to retrieve data through the hardware FIFO data path in a similar fashion to loading the queues. The data is read in binary format from the VXI device dependent register at offset 20_{16} with the 8-bit data right justified in the retrieved word. The most significant bits contains error and block end flags. See Reading Data via the Hardware FIFO in Section 3 for more information. The <data_handle> parameter is used to enable and disable this hardware data path. If FIFO is selected, the connection is established. If it is necessary to break the connection, resend the command with this parameter set to NONE. Sending the command prior to completing a data transfer will also break the connection and establish a new connection. Because the instrument must set internal registers and initialize a DMA channel, the user must use the *OPC command to determine that the connection is properly established prior to sending data to the VM6068. When a FEED ALL command is made, in addition to the receive data in the lower 8 bits (bits 0 to 7), the channel number is also indicated in bit 8 and 9: Bit 9 Bit 8 Channel 0 0 RCH1 0 1 RCH2 1 0 RCH3 1 1 RCH4

	For status data, bit 10 is the OR of the normal bits 8, 9, and 10, so bits 8 and 9 can be used for the channel indicator. In FEED ALL, transmit data the lower 8 bits (bits 0 to 7) are still for data. The end indicator is bit 15. The user must place the channel indicator in bits 8 and 9: Bit 9 Bit 8 Channel 0 0 TCH1 0 1 TCH2 1 0 TCH3 1 1 TCH4	
Examples	**Command / Query**	**Response *(Description)***
	`TRAC:DATA:FEED TCH1,FIFO`	
	`*OPC?`	1
	`TRAC:DATA:FEED?`	NONE,TCH1
Related Commands	None	

TRACe:FREE?

Purpose	Queries the amount of memory that is unused in a queue	
Type	Query	
Command Syntax	None - Query Only	
Command Parameters	N/A	
Reset Value	N/A	
Query Syntax	TRACe:FREE? <trace_name>	
Query Parameters	<trace_name>= TCH1, TCH2, TCH3, TCH4 for transmit queues <trace_name>= RCH1, RCH2, RCH3, RCH4 for receive queues	
Query Response	Numeric ASCII value from 0 to BUFFER SIZE	
Description	The Trace Free query reports the amount of available memory in the selected queue. The returned value reports the number of unused data bytes.	
Examples	**Command / Query**	**Response *(Description)***
	`TRAC:FREE? TCH4`	1024
Related Commands	TRACe:POINts<trace_name>,<points>	

TRACe:LENGth?

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Queries the number of characters in the specified queues</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Query</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">None - Query Only</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Default Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">TRACe:LENGth? <trace_name></td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2"><trace_name>= TCH1, TCH2, TCH3, TCH4 for transmit queues
<trace_name>= RCH1, RCH2, RCH3, RCH4 for receive queues</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">Numeric ASCII value from 0 to BUFFER SIZE</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Trace Length query reports the number of characters in the selected queue. This allows the user to calculate the number of additional characters that may be queued.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>TRAC:LENG? RCH4</code></td><td>128</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

TRACe:POINts

Purpose	Sets the size of a transmit or receive queue	
Type	Setting	
Command Syntax	TRACe:POINts <trace_name>,<points>	
Command Parameters	<trace_name>= TCH1, TCH2, TCH3, TCH4 for transmit queues <trace_name>= RCH1, RCH2, RCH3, RCH4 for receive queues <points> = numeric ASCII value from 2 to the size of the RAM installed	
Reset Value	<points> = 1024	
Query Syntax	TRACe:POINts? <trace_name>	
Query Parameters	<trace_name> = TCH1, TCH2, TCH3, TCH4 for transmit queues <trace_name> = RCH1, RCH2, RCH3, RCH4 for receive queues	
Query Response	Numeric ASCII value from 2 to the size of the buffer RAM installed	
Description	The Trace Points command sets the size of a transmit or receive queue. It allocates buffer RAM from an available pool to each queue. **Note**: *TRACe:POINts always rounds up to the next multiple of 28.* If the number of points specified exceeds the available memory, the maximum amount of memory is allocated to the queue and an error is generated. Note that any time the number of points in a queue is changed, the data in all queues is lost. Therefore, the size of any queue should not be changed while the VM6068 is active or if any desired data has not been sent by or retrieved from the instrument. The Trace Points query reports the size of a selected queue in bytes.	
Examples	**Command / Query**	**Response *(Description)***
	`TRAC:POIN TCH2,2048`	
	`TRAC:POIN? TCH2`	2048
Related Commands	TRACe:FREE? <trace_name>	

REQUIRED SCPI COMMANDS

STATus:OPERation:CONDition?

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Queries the Operation Status Condition Register</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Required SCPI command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">None - Query Only</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">STATus:OPERation:CONDition?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">0</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Operation Status Condition Register query is provided for SCPI compliance only. The VM6068 does not alter the state of any of the bits in this register and always reports a 0.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>STAT:OPER:COND?</code></td><td>0</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

STATus:OPERation:ENABle

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Sets the Operation Status Enable Register</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Required SCPI command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">STATus:OPERation:ENABle <NRf></td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">NRf = numeric ASCII value from 0 to 32767</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">NRf must be specified</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">STATus:OPERation:ENABle?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">Numeric ASCII value from 0 to 32767</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Operation Status Enable Register is included for SCPI compatibility and the VM6068 does not alter any of the bits in this register. The register layout is as follows:

Bit 0 - Calibrating
Bit 1 - Setting
Bit 2 - Ranging
Bit 3 - Sweeping
Bit 4 - Measuring
Bit 5 - Waiting for trigger
Bit 6 - Waiting for arm
Bit 7 - Correcting</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>STAT:OPER:ENAB 0</code>
<code>STAT:OPER:ENAB?</code></td><td>
0</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

STATus:OPERation:EVENt?

Purpose	Queries the Operation Status Event Register	
Type	Required SCPI command	
Command Syntax	None - Query Only	
Command Parameters	N/A	
Reset Value	N/A	
Query Syntax	STATus:OPERation[:EVENt]?	
Query Parameters	None	
Query Response	0	
Description	The Status Operation Event Register query is included for SCPI compliance. The VM6068 does not alter any of the bits in this register and always reports a 0.	
Examples	**Command / Query**	**Response *(Description)***
	`STAT:OPER?`	0
Related Commands	None	

STATus:PRESet

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Presets the Status Registers</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Required SCPI command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">STATus:PRESet</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">None - Command Only</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Status Preset command presets the Status Registers. The Operational Status Enable Register is set to 0 and the Questionable Status Enable Register is set to 0. This command is provided for SCPI compliance only.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>STAT:PRES</code></td><td></td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

STATus:QUEStionable:CONDition?

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Queries the Questionable Status Condition Register</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Required SCPI command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">None - Query Only</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">STATus:QUEStionable:CONDition?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">0</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Questionable Status Condition Register query is provided for SCPI compliance only. The VM6068 does not alter any of the bits in this register and a query always reports a 0.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>STAT:QUES:COND?</code></td><td>0</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

STATus:QUEStionable:ENABle

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Sets the Questionable Status Enable Register</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Required SCPI command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">STATus:QUEStionable:ENABle &lt;NRf&gt;</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">NRf = numeric ASCII value from 0 to 32767</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">NRf must be supplied.</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">STATus:QUEStionable:ENABle?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">Numeric ASCII value from 0 to 32767</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Status Questionable Enable command sets the bits in the Questionable Status Enable Register. This command is provided only to comply with the SCPI standard.

The Status Questionable Enable query reports the contents of the Questionable Status Enable Register. The VM6068 does not alter the bit settings of this register and will report the last programmed value.</td></tr>
<tr><td rowspan="3"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>STAT:QUES:ENAB 64</code></td><td></td></tr>
<tr><td><code>STAT:QUES:ENAB?</code></td><td>64</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

STATus:QUEStionable:EVENt?

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Queries the Questionable Status Event Register</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Required SCPI command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">None - Query Only</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">STATus:QUEStionable[:EVENt]?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">0</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The Questionable Status Event Register is provided for SCPI compliance only. The VM6068 does not alter the bits in this register and queries always report a 0.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>STAT:QUES?</code></td><td>0</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

SYSTem:ERRor?

Purpose	Queries the Error Queue
Type	Required SCPI command
Command Syntax	None - Query Only
Command Parameters	N/A
Reset Value	N/A
Query Syntax	SYSTem:ERRor?
Query Parameters	None
Query Response	ASCII string.
Description	The System Error query is used to retrieve error messages from the error queue. The error queue will maintain the two error messages. If additional errors occur, the queue will overflow and the second and subsequent error messages will be lost. In the case of an overflow, an overflow message will replace the second error message. See the SCPI standard Volume 2: Command Reference for details on errors and reporting them. Refer to the "Error Messages" section of this manual for specific details regarding the reported errors.
Examples	**Command / Query** `SYST:ERR?`
	Response *(Description)* -350,"Queue overflow"
Related Commands	None.

SYSTem:VERSion?

<table>
<tr><td><b>Purpose</b></td><td colspan="2">Queries the SCPI version number the VM6068 complies with</td></tr>
<tr><td><b>Type</b></td><td colspan="2">Required SCPI command</td></tr>
<tr><td><b>Command Syntax</b></td><td colspan="2">None - Query Only</td></tr>
<tr><td><b>Command Parameters</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Reset Value</b></td><td colspan="2">N/A</td></tr>
<tr><td><b>Query Syntax</b></td><td colspan="2">SYSTem:VERSion?</td></tr>
<tr><td><b>Query Parameters</b></td><td colspan="2">None</td></tr>
<tr><td><b>Query Response</b></td><td colspan="2">Numeric ASCII value</td></tr>
<tr><td><b>Description</b></td><td colspan="2">The System Version query reports version of the SCPI standard with which the VM6068 complies.</td></tr>
<tr><td rowspan="2"><b>Examples</b></td><td><b>Command / Query</b></td><td><b>Response <i>(Description)</i></b></td></tr>
<tr><td><code>SYST:VERS?</code></td><td>1994.0</td></tr>
<tr><td><b>Related Commands</b></td><td colspan="2">None</td></tr>
</table>

APPPENDIX A

*TST? 0 QUERY

The `*TST? 0` query is a loop-back test and requires a loop-back connector be connected to the VM6068 prior to executing the query (see Table 4-4 for connector details). To conduct the loop-back self-test, simply send the `*TST? 0` query. A successful self-test performance will result in a "0" response.

From			To		
Pin	Function	Input/Output	Pin	Function	Input/Output
1	TXD-	O	37	RXD-	I
2	TXD+	O	38	RXD+	I
3	RXD-	I	35	TXD-	O
4	RXD+	I	36	TXD+	O
5	RTS-	O	41	CTS-	I
6	RTS+	O	42	CTS+	I
7	CTS-	I	39	RTS-	O
8	CTS+	I	40	RTS+	O
9	DTR-	O	45	DSR-	I
10	DTR+	O	46	DSR+	I
11	DSR-	I	43	DTR-	O
12	DSR+	I	44	DTR+	O
13	TXC-	O	49	RXC-	I/O
14	TXC+	O	50	RXC+	I/O
15	RXC-	I/O	47	TXC-	O
16	RXC+	I/O	48	TXC+	O
17	GND		51	GND	Not conencted
18	GND		52	GND	Not conencted
19	TXD-	O	55	RXD-	I
20	TXD+	O	56	RXD+	I
21	RXD-	I	53	TXD-	O
22	RDX+	I	54	TXD+	O
23	RTS-	O	59	CTS-	I
24	RTS+	O	60	CTS+	I
25	CTS-	I	57	RTS-	O
26	CTS+	I	58	RTS+	O
27	DTR-	O	63	DSR-	I
28	DTR+	O	64	DSR+	I
29	DSR-	I	61	DTR-	O
30	DSR+	I	62	DTR+	O
31	TXC-	O	67	RXC-	I/O
32	TXC+	O	68	RXC+	I/O
33	RXC-	I/O	65	TXC-	O
34	RXC+	I/O	66	TXC+	O

TABLE 4-4: LOOP-BACK TEST CONNECTOR

`*TST? 0` tests the input and output capabilities by transmitting and receiving on alternate channels as follows:

Transmit	Receive
Channel 1	Channel 3
Channel 2	Channel 4
Channel 3	Channel 1
Channel 4	Channel 2

As with `*TST?`, a bit value of "1" in any location indicates a failure, while a "0" value indicates a successful test. If `*TST? 0` encounters a failure, the test is aborted. It then reports the standard being tested at the time of the failure as well as the test the standard failed.

The standard is reported in data bits 8 through 11. The standards are identified as follows:

Failed Standard	
Data Bits 8 - 11	
0001	RS-232
0010	RS-422
0011	RS-485
0100	RS-423
0101	RS-449
0110	RS-530
0111	V.35

There are two sets of tests run for each standard. The tests transmit data from one channel and recieve it on another channel. Since the unit cannot distinguish between the two, if either channel fails, the results will be the same. The first set of tests are reported in bits 0 through 2 with bit 3 set to 0, and the second set of tests are reported at data bits 0 through 2 with bit 3 set to 1. Data bits 4 through 7, and 11 through 15 are not used for test failure reporting.

If a test from the first set fails, it is reported at data bits 0 through 3 as:

First Test Set Failed	
Data Bits 0 - 3	
0000	TX1 out/RX3 in
0001	TX2 out/RX4 in
0010	TX3 out/RX1 in
0011	TX4 out/RX2 in
0100	TXC1 out/RXC3 in
0101	TXC2 out/RXC4 in
0110	TXC3 out/RXC1 in
0111	TXC4 out/RXC2 in

If a test from the second set fails, it is reported at data bits 0 through 3 as:

Second Test Set Failed	
Data Bits 0 - 3	
1000	DTR1 out/DSR3 in
1001	DTR2 out/DSR4 in
1010	DTR3 out/DSR1 in
1011	DTR4 out/DSR2 in
1100	RTS1 out/CTS3 in
1101	RTS2 out/CTS4 in
1110	RTS3 out/CTS1 in
1111	RTS4 out/CTS2 in

INDEX