VM7000

VMIP BREADBOARD

USER'S MANUAL

P/N: 82-0040-000
Rev. April 13, 2005

VXI Technology, Inc.

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

TABLE OF CONTENTS

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, Inc. 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
PRODUCT NAME	VMIP Breadboard
MODEL NUMBER(S)	VM7000
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.

April 2005

[illegible]

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

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 you suspect there is any damage to this product. ***Product should be inspected or serviced only by qualified personnel.***

Improper Use

The operator of this instrument is advised that if the equipment is used in a manner not specified in this manual, the protection provided by the equipment may be impaired. Conformity is checked by inspection.

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 VM7000 is a general-purpose breadboard designed for the VMIP™ (*VXI Modular Instrumentation Platform*) bus. It provides the user with complete direct register access to the board, eliminating the need for the user to spend time developing the VXI interface logic, allowing the user to concentrate on the custom product design.

The instrument uses the message-based word serial interface for programming and data movement, as well as supporting direct register access for very high-speed data throughput. The VM7000 command set conforms to the SCPI standard for consistency and ease of programming.

Since the VM7000 is a member of the VXI Technology, Inc. VMIP family, up to three separate custom products can be designed on a single C-size VXIbus card, or combined with other VMIP instruments to form a customized, highly integrated, multi-purpose instrument (see Figure 1-1). Each one of these instruments has its own unique logical address (ULA), and can have its own VXI*plug&play* driver. This allows the user to reduce system size and cost by combining the VM7000 with two other instrument functions in a single-wide C-size VXIbus module.

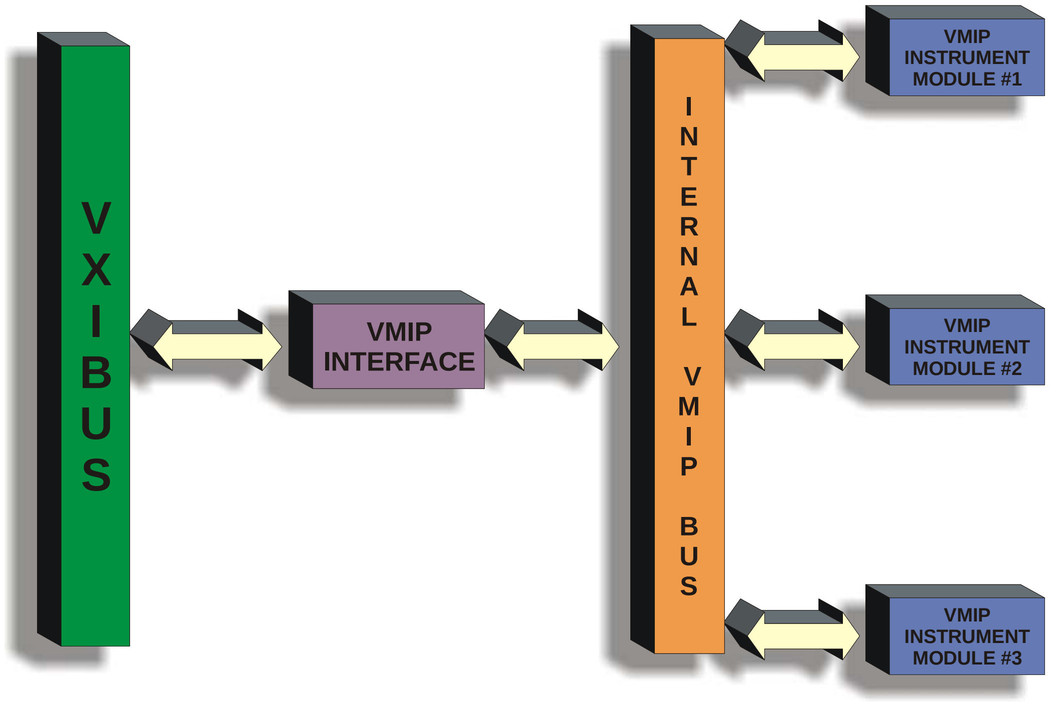

VXIBUS
VMIP INTERFACE
INTERNAL VMIP BUS
VMIP INSTRUMENT MODULE #1
VMIP INSTRUMENT MODULE #2
VMIP INSTRUMENT MODULE #3

FIGURE 1-1: VMIP™ PLATFORM

VM7000 FEATURES

- Provides 16 device-dependent registers
- Provides 32 digital I/O lines, configurable as input or output in groups of 8 bits. These can be used as input buffer or output latch
- Buffered data bus and other control lines are available for custom designs
- Access to all 8 VXI trigger input lines
- Access to all 8 VXI trigger output lines
- Access and fail LED’s are provided
- Access to VXI IRQ5
- Access to VXIbus
- Access to VXIbus power supply lines
- Programmable clock
- Active low and active high reset signals
- 68-pin high-density SCSI connector

DESIGN DESCRIPTION

The VM7000 provides 16 device-dependent registers that are located in the upper 32 bytes of the VXI instruments logical address space. The VM9000 base unit uses the lower 32 bytes. The VM7000 also provides 32-bit digital I/O on board, configurable as input or output in groups of 8 bits. This allows the user to develop simple functions controlled by I/O without having to understand the device-dependent registers.

All seven supply lines available on the VXIbus are brought out to the internal VMIP bus connectors for use by the VM7000. This allows virtually any type of instrument to be designed for the VXIbus, and does not limit the user to just the +5 V and ±12 V supply lines.

Each VMIP module has two connectors that go to the VMIP labeled P1 and P2. The lower connector is called P1 and the upper connector is called P2. Each connector has 60 pins. On schematics, the pins are generally shown as two groups of 30. The VMIP module has six connectors for the three modules. These connectors are called J1, J2, J3, J4, J5, and J6. A module's P1 connector mates with the VMIP's J1, J3, or J5 connector (depending on whether it is plugged into the top, middle or bottom position). A module's P2 connector mates with the VMIP's J2, J4, or J6 connector (depending on the module's position).

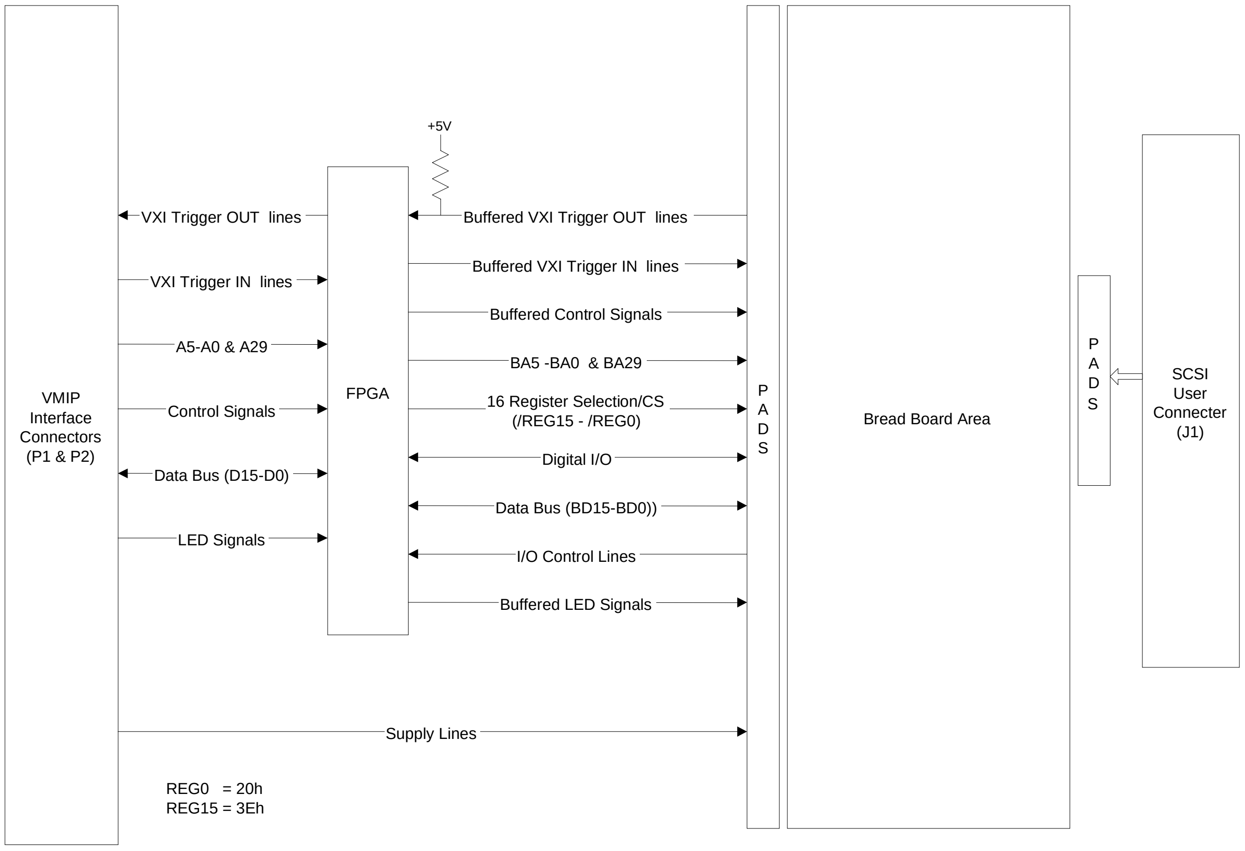


FIGURE 1-2: VM7000 - VMIP BREADBOARD

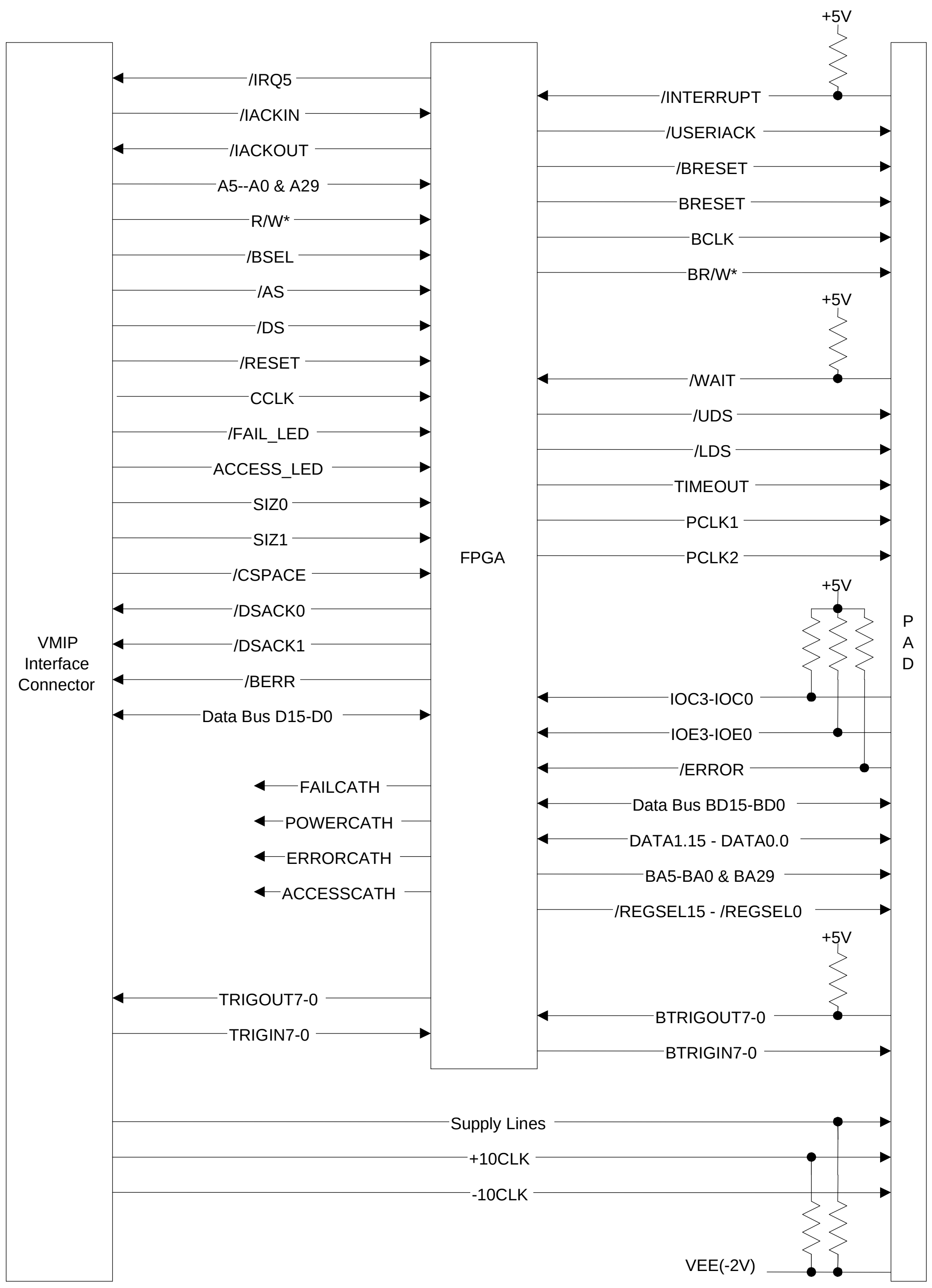


FIGURE 1-3: VMIP INTERFACE

VM7000 Outline

VM7000 Outline	
VMIP Interface Connector	
	This connector interfaces to the VMIP main board (VM9000).
Access/Fail Circuit	
	This generates the access/fail LED signal.
Supply Lines	
	The following supply lines are brought to the breadboard area with 40 mm traces: +5 V GND -5.2 V -2 V +12 V -12 V + 24 V -24 V In addition to this, +5V and GND layers are provided in the breadboard area.
Address Bus	
	The address lines, A5-A0 and A29, are used in the FPGA for register decoding. The buffered address lines BA5-BA0 and BA29 are brought to the breadboard area for the user.
Data Bus	
	The data lines, D15-D0, are buffered and used for FPGA and the on-board buffers/latches. The buffered data lines, BD15-BD0, are brought to the breadboard area for the user.
FPGA	
	The FPGA generates the 16 device-dependent register selection lines that are brought to the breadboard area. Two registers are used for buffers/latches. User signals are buffered in the FPGA and brought to the breadboard area for the user.
User Connector	
	The 68-pin SCSI-2 user connector pins are brought out to pads for easy soldering.
Control Signals	
	The following signals are brought to the FPGA from the VMIP: 1) CCLK 6) SIZ1 11) /CSPACE 16) IACKOUT 2) R/W* 7) /DSACK0 12) TRIGIN7-0 17) /FAIL_LED 3) /AS 8) /DSACK1 13) TRIGOUT7-0 18) ACCESS_LED 4) /DS 9) /RESET 14) IRQS 19) A5-A0 & A29 5) SIZ0 10) /BSEL 15) IACKIN 20) /BERR
Buffer/Latch	
	The VM7000 provides the user with 32 digital I/O lines that can be configured as input or output in groups of 8 bits. They can be configured as an input buffer, output buffer, or output latch. For many applications, the breadboard can be used with few external components. The FPGA generates chip select for these buffers in the device-dependent register area.
PCB Design	
	The PCB has four layers and is split into two sections: the VMIP interface area and the breadboard area. In the VMIP interface area, all four layers are used for signals. In the breadboard area, two layers are used for +5 V and GND, and two layers are left for the pads. Other supply lines (12 V, etc.) have 40 mm traces. The PCB includes: 1) De-coupling capacitors wherever required. 2) On-board ICs and the user +5 V supply lines are different. More than one +5 V lines come to the VMIP (P1 and P2) connectors. One of the +5 V lines is used for the on-board circuitry and a different line goes to the user pads. 3) The big circles on the breadboard area are for general use, the small circles are GND, and the small squares are +5 V.

SECTION 2

PREPARATION FOR USE

INSTALLATION

When the VM7000 is unpacked from its shipping carton, the contents should include the following items:

(1) VM7000 VXIbus module
(1) VM7000 Module User's Manual (this manual)

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

Once the VM7000 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 VM7000. Once the chassis is found to be adequate, the VM7000's logical address and the chassis' backplane jumpers should be configured prior to the VM7000'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.

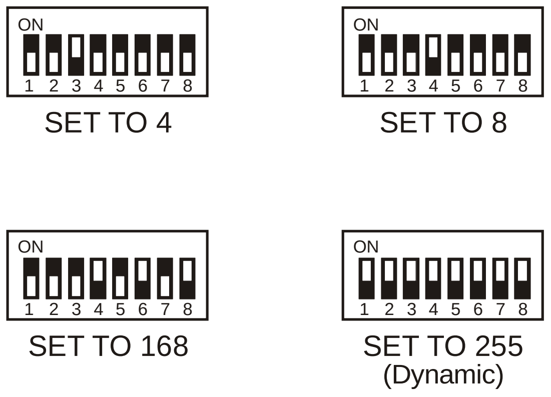
It should be noted that if the chassis cannot provide adequate power to the module, the instrument may not perform to specification or possibly not operate at all. In addition, if adequate cooling is not provided, the reliability of the instrument will be jeopardized and permanent damage may occur. Damage 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 manual for further details on setting the backplane jumpers.

SETTING THE LOGICAL ADDRESS

The logical address of the VM7000 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. Switches pushed toward the ON legend signify a logic 1; switches pushed away from the ON legend 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.

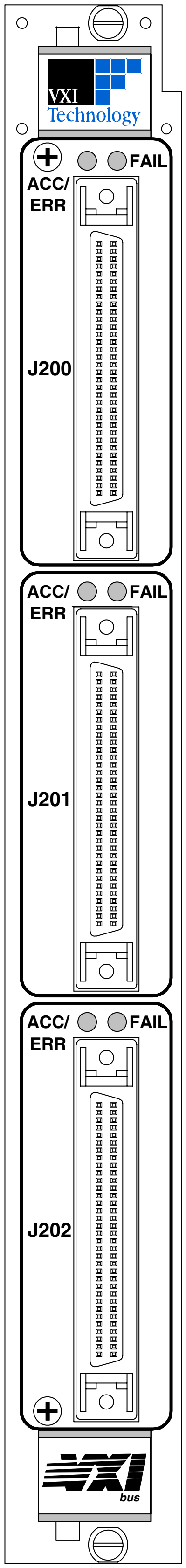


Switch Position	Switch Value
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128

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 VM7000-1has a connector labeled J201 that contains all signals for this instrument. The VM7000-2 has J201 and J202 provided, while the VM7000-3 has J200, J201, and J202. The wiring for each of these connectors is identical.

Regardless of whether the VM7000 is configured with other VM7000 modules or with other VMIP modules, each module is treated as an independent instrument in the VXIbus chassis. As such, each group has its own ACCESS and FAIL lights.

FIGURE 2-2: VM7000 FRONT PANEL

The mating connector for the VM7000 board is a 68-pin SCSI, and is available from the following company:

AMP P/N:	749621-7	Connector
AMP P/N:	749195-2	Back Shell

The pin locations for J200, J201 and J202 are shown in Figure 2-3.

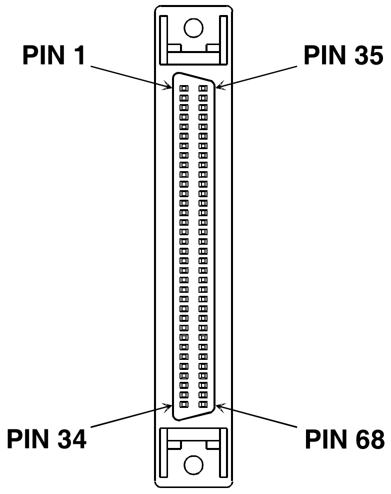


FIGURE 2-3: J200, J201 AND J202 PIN LOCATIONS

SECTION 3

DEVELOPMENT

REGISTER MAPPING

The VM7000 provides direct register access for faster data access. The VXI standard places certain restrictions on registers. All VXI cards are given 64 bytes in the upper quadrant of the A16 address space. The exact location is determined by the instrument's logical address:

base = 0xC000 + 64 * logical address

In a VXI system, each instrument is given a unique logical address. Usually, the logical address is assigned by an 8-position DIP-switch (see Section 2), which the user can set. This means each instrument has a unique 64-byte area.

Part of what the VXI standard requires is that certain registers need to be at certain locations in the 64-byte area. For register-based devices, the first 10 bytes of the 64-byte area are defined by the VXI standard. For message-based devices, the first 32 bytes are defined or reserved by the VXI standard. A VTI VM9000 base unit takes care of these first 32 bytes. If the user requires 32 bytes of registers, or less for the custom design, they should be grouped to fit in a 32-byte area.

The VM7000 uses the logical address of the module to map the upper 32 bytes of its A16 64-byte space into the VM9000's 68340-address space. The location of the 32 bytes is dependent on a module's position on the VMIP. The top module's registers appears in the 68340's address space at:

0x60000000 to 0x6000003F

The middle module's registers appear at:

0xA0000000 to 0xA000003F

and the bottom module's registers appear at:

0xE0000000 to 0xE000003F

By proper decoding and keeping user registers in a 32-byte area, these registers appear in the proper location in the backplane's A16 address space. From the external software perspective, one now deals with registers in a 32-byte area of A16. The VM7000 automatically takes care of this.

The VMIP provides a board select labeled **BSEL*** to each module on P2-P39. This goes active (low) when an access is made to a module. **BSEL*** goes low for the top module in the range:

0x40000000 to 0x7FFFFFFF

BSEL* goes low for the middle module in the range:

0x80000000 to 0xBFFFFFFF

and **BSEL*** goes low for the bottom module in the range:

0xC0000000 to 0xFFFFFFFF

Notice that 0x60000000 to 0x6000003F, which is one group of the registers mapped from the backplane, falls in the range of the top module. By using **BSEL*** in a module's decoding, the module works in any of the three positions (top, middle or bottom) without any jumper changes.

The VM7000 A16 Address Space register map is shown in table 3.1.

TABLE 3-1: VM7000 A16 MEMORY

Offset	Register Name	Description
3E	REGSEL15	16 bit I/O register for user circuits
3C	REGSEL14	16 bit I/O register for user circuits
3A	REGSEL13	16 bit I/O register for user circuits
38	REGSEL12	16 bit I/O register for user circuits
36	REGSEL11	16 bit I/O register for user circuits
34	REGSEL10	16 bit I/O register for user circuits
32	REGSEL9	16 bit I/O register for user circuits
30	REGSEL8	16 bit I/O register for user circuits
2E	REGSEL7	16 bit I/O register for user circuits
2C	REGSEL6	16 bit I/O register for user circuits
2A	REGSEL5	16 bit I/O register for user circuits
28	REGSEL4	16 bit I/O register for user circuits
26	REGSEL3	16 bit I/O register for user circuits
24	REGSEL2	16 bit I/O register for user circuits
22	REGSEL1	Used for on-board buffer/latch2
20	REGSEL0	Used for on-board buffer/latch1
1E		
1C		
1A		
18		
16	[A32 Pointer Low]	
14	[A32 Pointer High]	
12	[A24 Pointer Low]	
10	[A24 Pointer High]	
E	Data Low	
C	Data High	
A	Response [/Data Extended]	
8	Protocol [/Signal] Register	
6	[Offset Register]	
4	Status / Control Register	
2	Device Type	
0	ID Register	

REGSEL1 and **REGSEL0** are used for on-board digital I/O on the VM7000 and are also available on the pads. If the on-board functionality is not required, then these registers can also be used for the custom design. I/O control signals, **IOC3 - IOC0**, are provided to configure these registers for either input or output. I/O enable signals, **IOE3 - IOE0**, are provided to enable or disable these registers. If all 16 registers are required for the custom design, then **IOE3 - IOE0** can be used to disable internal registers **REGSEL1** and **REGSEL0**.

The VXI registers, from **00H** to **1FH**, are taken care of on the VMIP base unit so they are not decoded on the breadboard.

CONFIGURING ON-BOARD DIGITAL I/O

The four I/O control lines, **IOC3 - IOC0**, are used for configuring on-board digital I/O for input or output:

TABLE 3-2: I/O CONTROL

VXI Byte Registers	Function	Description
23	REGEL1 lower byte (DATA1.7 - DATA1.0) In/Out	IOC3 signal = 0 for input, 1 for output; default = 1
22	REGSEL1 higher byte (DATA1.15 - DATA1.8) In/Out	IOC2 signal = 0 for input, 1 for output; default = 1
21	REGSEL0 lower byte (DATA0.0 - DATA0.7) In/Out	IOC1 signal = 0 for input, 1 for output; default = 1
20	REGSEL0 higher byte (DATA0.15 - DATA0.8) In/Out	IOC0 signal = 0 for input, 1 for output; default = 1

The four I/O enable lines, IOE3 - IOE0, are used to enable or disable the digital I/O:

TABLE 3-3: I/O ENABLE/DISABLE

VXI Byte Registers	Function	Description
23	REGEL1 lower byte (DATA1.7 - DATA1.0)	IOE3 signal = 0 for disable, 1 for enable; default = 1
22	REGSEL1 higher byte (DATA1.15 - DATA1.8)	IOE2 signal = 0 for disable, 1 for enable; default = 1
21	REGSEL0 lower byte (DATA0.0 - DATA0.7)	IOE1 signal = 0 for disable, 1 for enable; default = 1
20	REGSEL0 higher byte (DATA0.15 - DATA0.8)	IOE0 signal = 0 for disable, 1 for enable; default = 1

DATA STROBE SIGNALS - /UDS AND /LDS

Two data strobe signals, **/UDS** and **/LDS**, are available on the pads for external interface. This is provided to access the upper byte (BD15-BD8) and the lower byte (BD7-BD0) correctly. For word access, both **/UDS** and **/LDS** are low. For byte access, **/UDS** is active low for the data bus higher byte (D15-D0), and **/LDS** is active low for the data bus lower byte (D7-D0). The **/UDS** and **/LDS** signals are generated from the bus cycle for byte access and word so that the external custom circuits can use **/UDS** and **/LDS** signals if required. Normally, chip select signals, **REGSEL0**, **REGSEL1**, etc., and **R/W*** are sufficient to interface to any external devices as long as the same Motorola convention is followed (i.e. D15-D8 is the upper byte, and D7-D0 is the lower byte). In other words, A0 is low for word access and upper byte access, and it is high for lower byte access. Since data strobe timing is taken care of in the FPGA, **/UDS** and **/LDS** may be left unused in the external interface.

/WAIT SIGNAL

This active low input signal is available on the pad for external interface. This is provided to interface a device that has different timing components. As soon as **REGSEL**, **BR/W*** and **DS** signals become active, the interface FPGA will look for the **/WAIT** signal to go high. If the external interface needs wait states, then the **/WAIT** signal can be pulled low as long as the Wait State is required. Once the external interface has read data from the data bus for the write cycle, or written to the data bus for the read cycle, the **/WAIT** signal can be made high to indicate the completion of the cycle. For on-board I/O access, the interface FPGA generates a proper **ACK** signal and will not look for the **/WAIT** signal. If the external circuit does not require a **/WAIT** state, then the **/WAIT** signal can be left untouched since this signal is normally pulled up on the board.

If the **/WAIT** signal is low for more than twenty clock periods, then the FPGA terminates the cycle and generates an active high **TIMEOUT** signal to indicate to the external interface that a bus time-out has occurred. Once the time-out occurs, this signal will remain high and will go low only on a system reset. However, the FPGA will allow the next bus access. The **TIMEOUT** signal can be used as a debugging signal.

READ CYCLE SEQUENCE

1. Address lines **BA4-BA0** are active, and **BR/W*** is high, on the rising edge of the clock.
2. One half clock later, the FPGA asserts one of the **REGSEL15-REGSEL0** lines and **/UDS** and **/LDS** becomes active.
3. The FPGA introduces a three clock period delay from the start of the cycle before it looks for a **/WAIT** signal.
4. The external device can place the data on the data bus and make the **/WAIT** signal high.
5. The FPGA latches the data on the next falling edge of the clock after the **/WAIT** signal becomes high.
6. Address lines **/UDS**, **/LDS** and **BR/W*** become inactive on the next rising edge of the clock.

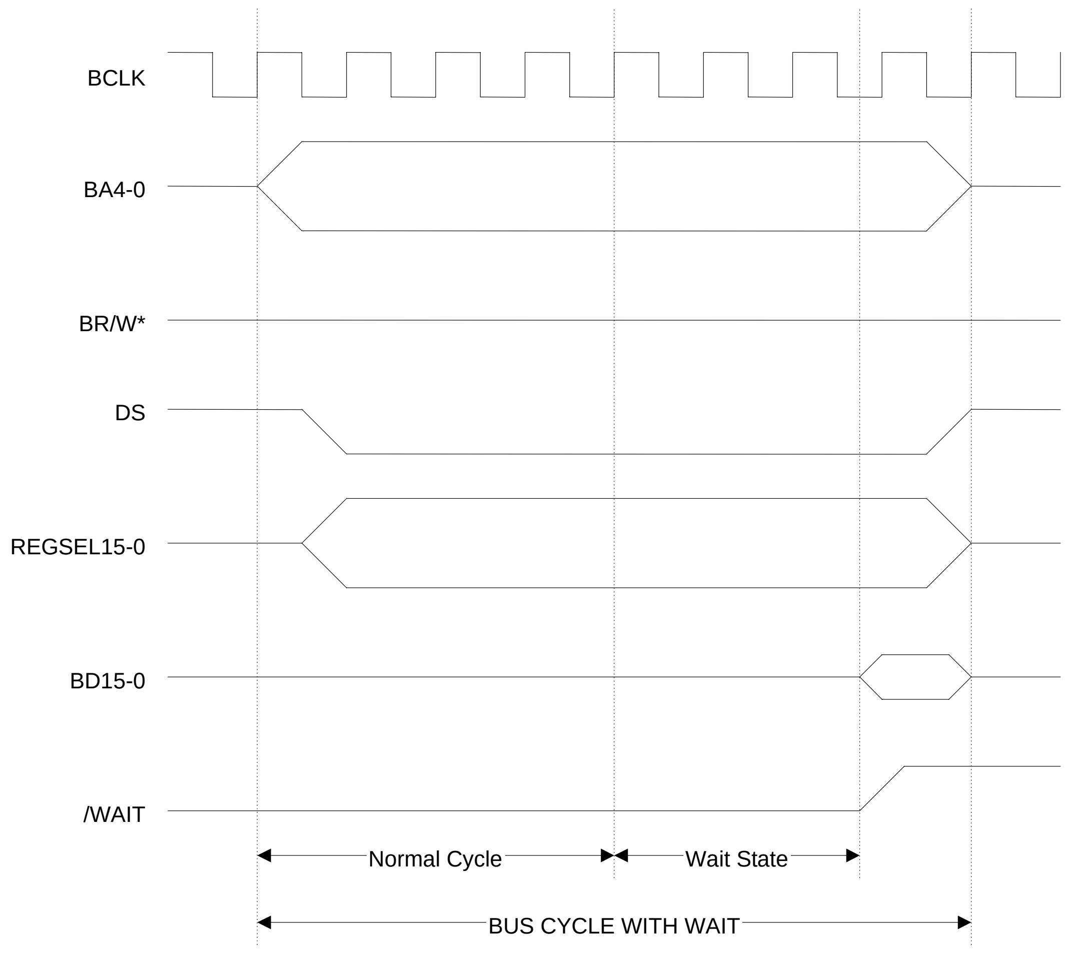


FIGURE 3-1: READ CYCLE

WRITE CYCLE SEQUENCE

1. Address lines **BA4-BA0** are active, and **BR/W*** I high, on the rising edge of the clock.
2. The FPGA places the data on the data bus after one clock.
3. The FPGA asserts one of the **REGSEL15-REGSEL0** lines, and the **/UDS** and **/LDS** signals become active after one and a half clocks.
4. The external device can read the data from the data bus and the **/WAIT** signal goes high.
5. Address lines **/UDS**, **/LDS** and **BR/W*** become inactive on the next rising edge of the clock.

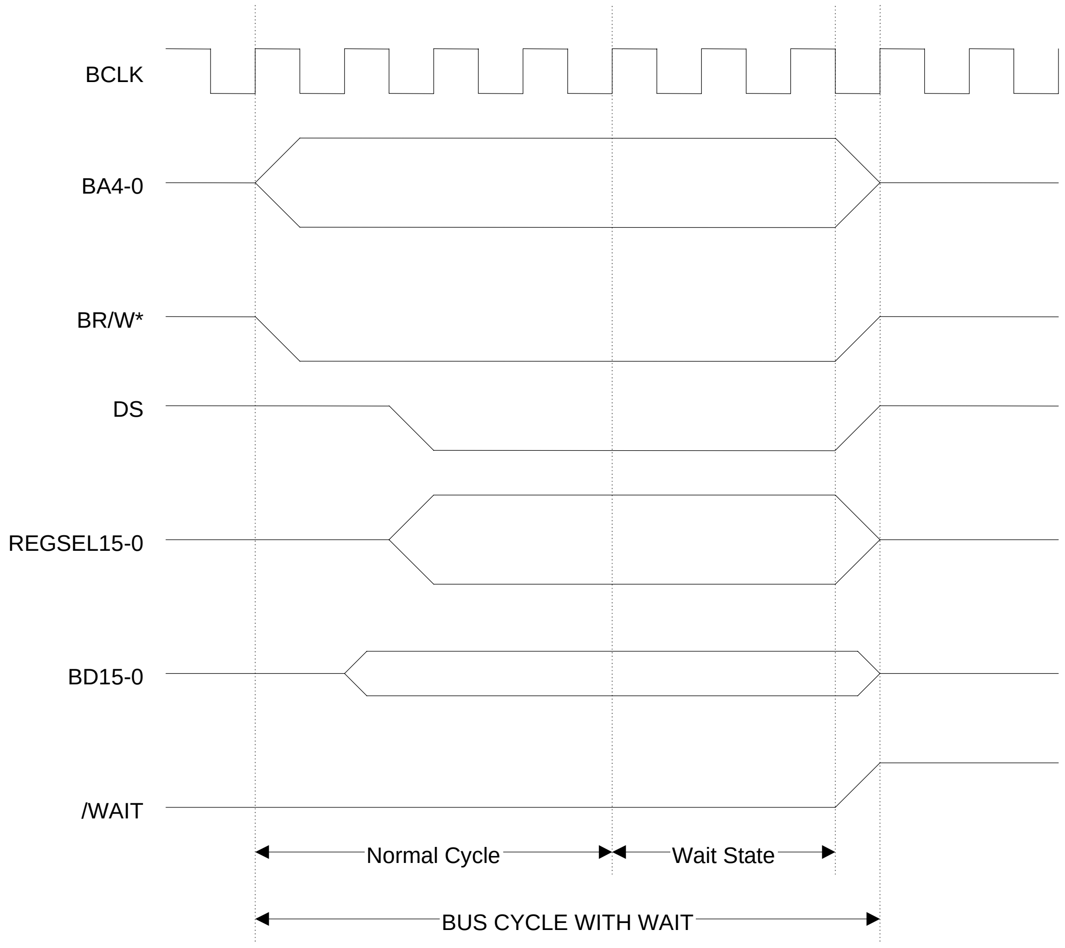


FIGURE 3-2: WRITE CYCLE

TABLE 3-4: READ / WRITE SIGNALS

Signals/ Event	REGSEL15 -REGSEL0	A0	/UDS	/LDS	BD15- BD0	BD15- BD8	BD7- BD0
Word READ	any one low	low	low	low	data READ		
Word WRITE	any one low	low	low	low	data WRITE		
Upper-byte READ	any one low	low	low	high		data READ	
Upper-byte WRITE	any one low	low	low	high		data WRITE	
Lower-byte READ	any one low	high	high	low			data READ
Lower-byte WRITE	any one low	high	high	low			data WRITE

/INTERRUPT SIGNAL

The external circuit can use the VXI interrupt **/IRQ5** if required. When the **/INTERRUPT** signal goes low, it generates the interrupt, **/IRQ5**, to the VMIP motherboard. Then the interface FPGA waits for the interrupt acknowledge cycle. When it completes the interrupt acknowledge cycle (from PC end software), the interface FPGA generates the active low **/USERIACK** pulse signal for three clock periods to indicate that the interrupt cycle is over. Ideally, the **/INTERRUPT** signal is normally high, and goes low for a minimum of three clock periods to generate the interrupt and then goes high after three clock periods, or goes high when it gets a low pulse on the **/USERIACK** signal. To generate the next interrupt, the **/INTERRUPT** signal should generate a low going pulse again.

On getting the interrupt, the base unit can execute one of the many pre-defined commands. Before using the interrupt, the user needs to configure the interrupt command. These commands are selected by word serial commands. For example, on getting the interrupt, the base unit sends a message to the VXI controller with the LA (Logical Address) signaling that the module at that LA has generated the interrupt. Or, it can send a message to the VXI controller with the LA and contents of a register (this register address is sent by a word serial command); then the front-end software can use that register. The PC and software receives a 16-bit status word for the interrupt. Bit0-Bit7 represent the LA, Bit8-Bit13 represent the user command, Bit14 is always set low, and Bit15 is always set high.

WORD SERIAL COMMANDS

To preserve the VXI device dependent registers for normal use by the external interface, the interrupt and programmable clock related configuration is done through word serial commands. On getting the interrupt, the PC end software will be notified with the LA of the module interrupted, along with 7 bits (the 8th bit is always set) of information that depends on the interrupt command set by the user. The following word serial commands are available for the user.

1. IRQ REG
   This command sets the option to read a register on getting an interrupt.

2. IRQ USER
   This command sets the option to send a user defined value on getting an interrupt.

3. IRQ:REG:BYTE <offset>
   This command specifies which byte register to read on getting an interrupt. For example:

   ```
   IRQ:REG:BYTE 3
   ```

   will read byte register 23H (only 6 bits are valid).

4. IRQ:STATus <status>
   This command specifies what user defined value to send on getting an interrupt. For example:

   ```
   IRQ:STAT 3FH
   ```

   will return **3FH** on getting an interrupt. (Actually, it will return **BFH** with the 7th and 8th bit set.)

5. IRQ?
   This command reads the interrupt configuration set by the user. For example, if the **IRQ REG** command had previously been set, then this command will return as **REG**.

6. IRQ:REG?
   This command reads the interrupt configuration set by the user. For example, if the **IRQ:REG:BYTE 3** command had previously been set, then this command will return as **BYTE 3**.

7. ACK
   This command generates an active low pulse on the **/USERIACK** output pin. This can be used to simulate the interrupt **ACK** signal for the external circuitry.

8. TIMER <value>
   This command loads the specified value into the counter to generate the programmable clock. For example, to set the counter to 6:

   ```
   tim 6
   ```

   For this command, **PCLK1** is high for one system clock, and low for five system clocks. **PCLK2** is high for six system clocks, and low for six system clocks.

/ERROR SIGNAL

The /ERROR signal is used for controlling the Error LED. A logic “0” turns on the Error LED, and with a logic “1” it is off.

LED INDICATIONS

There are two LEDs provided on the front panel. Each one has two colors - red and green. The LED on the right side of the front panel, labeled FAIL, is called the Power and System Fail LED. This LED goes to green when there is power to the module, and turns red when it gets a system fail indication from the back plane.

The left side LED, labeled ACC/ERR, is called the Access and Error LED. This LED turns green whenever there is a board access, and turns red when it gets an error signal from the user circuits. If no access or error has occurred, this LED will be off.

PROGRAMMABLE CLOCKS - PCLK1 AND PCLK2

There are many peripherals that need a clock lower than 25 MHz. This 25 MHz clock is divided by an internal 16-bit counter and provides the **PCLK1** and **PCLK2** output. The dividing factor is entered by word serial commands. For dividing up to 2, use **PCLK1**. For dividing by more than 2, use **PCLK1** and **PCLK2** as required. **PCLK1** provides an asymmetrical clock output, and **PCLK2** provides a symmetrical - 50% duty cycle - output.

If the counter was loaded with a value of 8: **PCLK1** is high for one system clock, and low for seven system clocks; **PCLK2** is high for eight system clocks, and low for eight system clocks.

If counter was loaded with a value of 10: **PCLK1** is high for one system clock, and low for nine system clocks; **PCLK2** is high for ten system clocks, and low for ten system clocks.

For a 50% duty cycle, load the counter with half the division number that is required, and use the **PCLK2** output.

USER SIGNALS

The following table describes the signals that are available on the pads for the user. The active low signals are marked with a /(slash) or a *(star) symbol.

TABLE 3-5: USER SIGNALS

Signal Name	Custom Circuits Input or Output	Description
/REGSEL0-/REGSEL15	input	16 device-dependent registers (20H - 3EH) on the upper 32 bytes of the logical address. **REGSEL0** and **REGSEL1** are used on the board; the remaining registers are available for general use. Byte and Word access is treated correctly for the registers.
BA5-BA0 and BA29	input	Buffered address lines for external use
BD15-BD0	input/output	Buffered data lines for external use
DATA0.15-DATA0.0 and DATA1.15-DATA1.0	input/output	On-board bi-directional digital I/O. Each group of 8 bits can be configured as input or output. Each port (group of 8 bits) can be configured as an input buffer, an output buffer, or an output latch. Byte and Word access is manipulated as per Motorola convention. For Byte access, each port is accessed as follows: DATA0.15 - DATA0.8 = port 0 - REGSEL0 upper byte DATA0.7 - DATA0.0 = port 1 - REGSEL0 lower byte DATA1.15 - DATA1.8 = port 2 - REGSEL1 upper byte DATA1.7 - DATA1.0 = port 3 - REGSEL1 lower byte For Word access, each port is accessed as follows: DATA0.0 - DATA0.15 = REGSEL0 DATA1.0 - DATA1.15 = REGSEL1
/BRESET	input	Buffered system reset - active low
BRESET	input	Buffered system reset - active high
BCLK	input	Buffered system clock, 25 MHz, for state machines
BR/W*	input	Buffered read/write signals
/UDS and /LDS	input	Data strobe signals for external interface to access the upper byte (BD15-BD8) and lower byte (BD7-BD0). **/UDS** is low for valid data on BD15-BD8, and **/LDS** is low for valid data on BD7-BD0. See table 3-4.
/WAIT	output	This signal is used to delay the bus cycle. This enables the wait state for low speed devices. This signal should be set low to introduce the wait state, and be set high when the wait state is not required.
TIMEOUT	input	This active high bus time-out signal is used to notify the external interface that the **WAIT** signal was low for more than twenty clock periods, and the FPGA terminated the cycle. Once the time-out occurs, this signal will remain high until the system is reset.

TABLE 3-5 USER SIGNALS – (CONT.)

Signal Name	Custom Circuits Input or Output	Description
/INTERRUPT	output	This active low signal generates an interrupt, **/IRQ5**, to the VMIP motherboard. When the motherboard completes the interrupt acknowledge cycle with the VXI PC end software, the interface FPGA generates the **/USERIACK** pulse signal to the external circuits. The interrupt command should be configured by word serial command before using the interrupt.
/USERIACK	input	This signal is used as described in the **/INTERRUPT** signal.
IOC0 IOC1 IOC2 IOC3	input	Configures REGSEL0 upper byte for I/O Configures REGSEL0 lower byte for I/O Configures REGSEL1 upper byte for I/O Configures REGSEL1 lower byte for I/O 1 = Output 0 = Input Unconnected = Output
IOE0 IOE1 IOE2 IOE3	input	Enables REGSEL0 upper byte Enables REGSEL0 lower byte Enables REGSEL1 upper byte Enables REGSEL1 lower byte 1 = Enable 0 = Disable Unconnected = Enabled
/ERROR	input	Error control signal, control the Error LED
BTRIGIN7 - BTRIGIN0	input	VXI Trigger Input lines
BTRIGOUT7 - BTRIGOUT0	output	VXI Trigger Output lines. A low on these pins drives low the respective trigger-out signal to the VXI bus, and a high puts the respective trigger-out signal in tri-state.
+10BCLK	output	+10 MHz clock for general use (ECL signal)
-10BCLK	output	-10 MHz clock for general use (ECL signal)
PCLK1	input	Programmable clock for asymmetrical output
PCLK2	input	Programmable clock for symmetrical output
+5V	N/A	+5 V supply
GND	N/A	Ground
-5.2V	N/A	-5.2 V supply
-2V	N/A	-2 V supply
+12V	N/A	+12 V supply
-12V	N/A	-12 V supply
+24V	N/A	+24 V supply
-24V	N/A	-24 V supply

TABLE 3-6: USER SIGNAL PAD MATRIX

	1	2	3	4	5
A	+5V (VCC)	-5.2V	-2V	+12V	-12V
B	+24V	-24V	+10CLK	-10CLK	/REGSEL0
C	/REGSEL1	/REGSEL2	/REGSEL3	/REGSEL4	/REGSEL5
E	/REGSEL6	/REGSEL7	/REGSEL8	/REGSEL9	/REGSEL10
F	/REGSEL11	/REGSEL12	/REGSEL13	/REGSEL14	/REGSEL15
G	DATA0.0	DATA0.1	DATA0.2	DATA0.3	DATA0.4
H	DATA0.5	DATA0.6	DATA0.7	DATA0.8	DATA0.9
I	DATA0.10	DATA0.11	DATA0.12	DATA0.13	DATA0.14
K	DATA0.15	DATA1.0	DATA1.1	DATA1.2	DATA1.3
L	DATA1.4	DATA1.5	DATA1.6	DATA1.7	DATA1.8
M	DATA1.9	DATA1.10	DATA1.11	DATA1.12	DATA1.13
N	DATA1.14	DATA1.15	BA0	BA1	BA2
O	---------	---------	BA3	BA4	BA5
Q	---------	---------	BA29	BCLK	BD0
S	---------	---------	BD1	BD2	BD3
T	BD4	BD5	BD6	BD7	BD8
V	BD9	BD10	BD11	BD12	BD13
W	BD14	BD15	BTRIGIN0	BTRIGIN1	BTRIGIN2
X	BTRIGIN3	BTRIGIN4	BTRIGIN5	BTRIGIN6	BTRIGIN7
Y	BTRIGOUT0	BTRIGOUT1	BTRIGOUT2	BTRIGOUT3	BTRIGOUT4
Z	BTRIGOUT5	BTRIGOUT6	BTRIGOUT7	IOE0	IOE1
AA	IOE2	IOE3	IOC0	IOC1	IOC2
AB	IOC3	/WAIT	/ERROR	/INTERRUPT	/UDS
AC	/LDS	BR/W*	/BRESET	/USERIACK	TIMEOUT
AD	PCLK1	PCLK2	BRESET	SPARE1	SPARE2
AE	N/C	N/C	N/C	N/C	N/C

Note: *On the breadboard area, the large circles are for general use, the small squares are +5 V and the small circles are ground.*

APPLICATION EXAMPLES

These applications are only intended to demonstrate how easy it is to develop and control a custom design, for the VMIP family, using the VM7000 breadboard approach.

16-CHANNEL RELAY DESIGN EXAMPLE

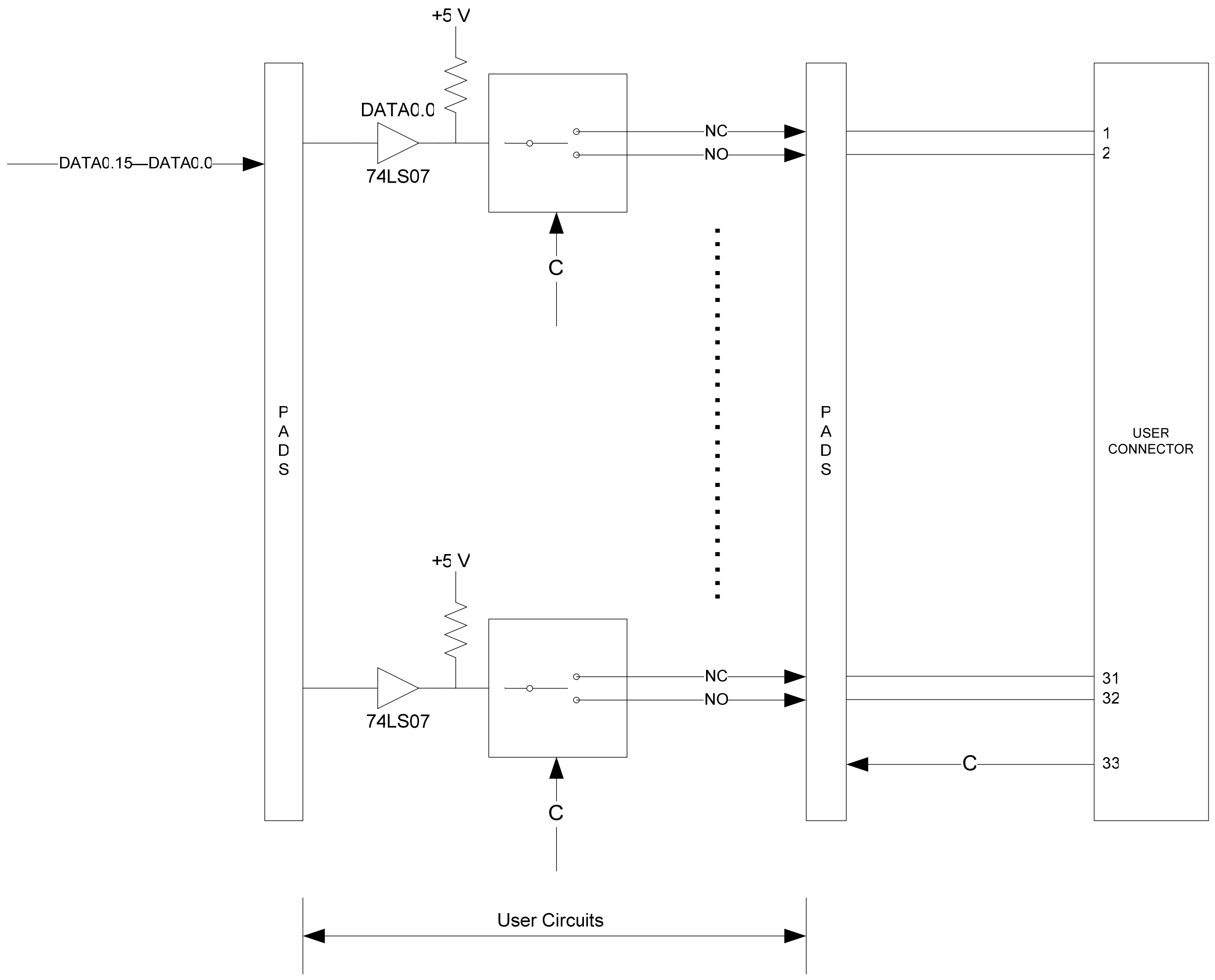


FIGURE 3-3: 16-CHANNEL RELAY

Additional Components Required:

1. 2 - octal open collector buffer
2. 16 - relays

The open collector buffers are interfaced to the on-board latch. The relays are connected to the open collector buffers. The open collector buffers and relays are soldered onto the breadboard area. The relay signals - NC, NO, and C, are brought to the pads near the user connector. Use protection circuits and LEDs as applicable.

Steps:

1. Assemble the components (open collector buffers and relays) on the breadboard area.
2. Connect the relay points to the user connector pads.
3. Configure the first 16 digital I/O lines as output ports (**IOC0** and **IOC1** should be high).
4. Use **REGSEL0** to set the relay status. Each bit represents one relay. Writing a 16-bit word to **REGSEL0** sets the relay status.

Programming:

1. Configure the I/O ports.
   IOC0 - IOC3 are high by default - this will configure the **REGSEL0** upper and lower bytes as output ports.
2. Relay Control - **REGSEL0**:

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
REGSEL0	K16	K15	K14	K13	K12	K11	K10	K9	K8	K7	K6	K5	K4	K3	K2	K1

Setting a bit to 0 opens the relay, and setting a bit to 1 closes it. For example:

- to close K2, write 0002H to **REGSEL0**
- to close K1 and K2, write 0003H to **REGSEL0**
- to close all the relays, write FFFFH to **REGSEL0**

32-Channel Relay Design Example

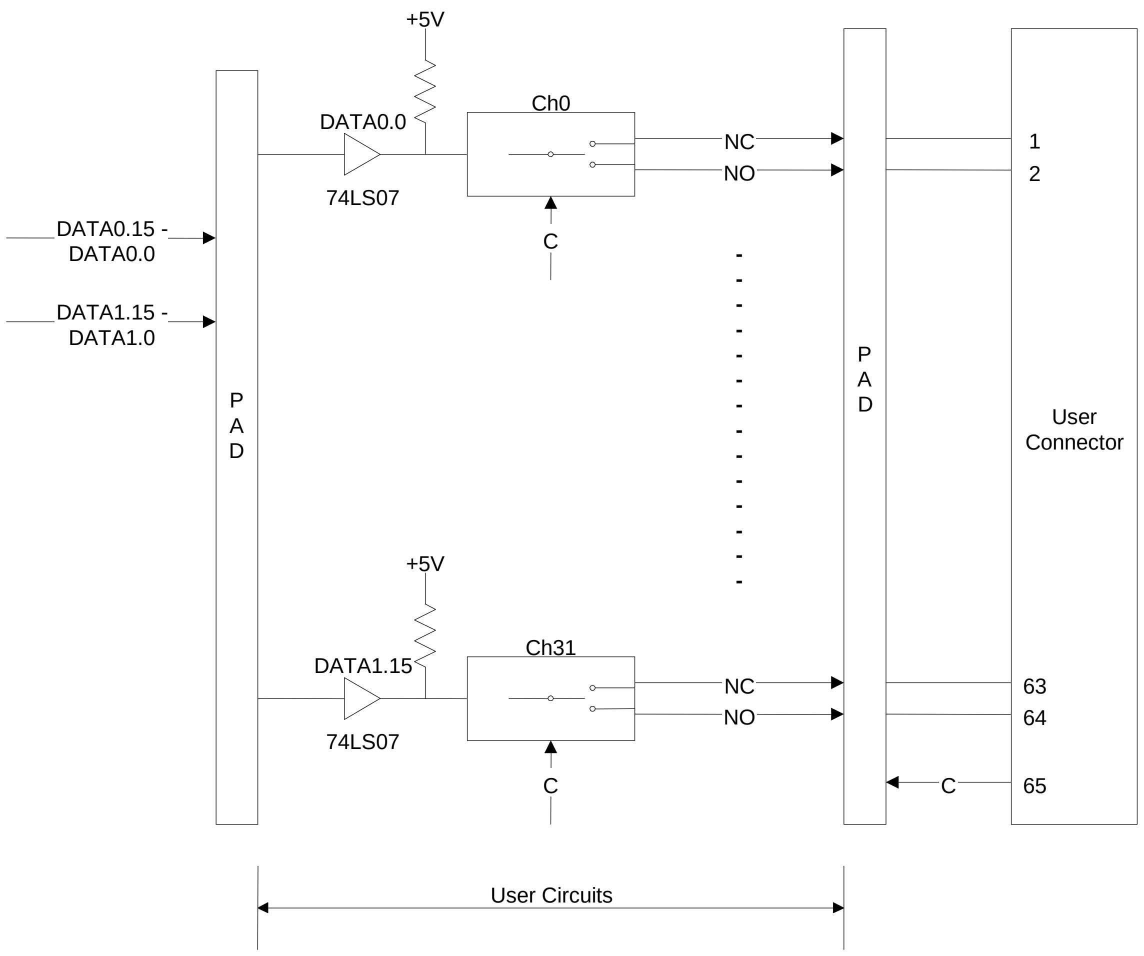


Figure 3-4: 32-Channel Relay

Additional Components Required:

1. 4 - octal open collector buffers
2. 32 - relays

The open collector buffers are interfaced to the on-board latch. The relays are connected to the open collector buffers. The open collector buffers and relays are soldered onto the breadboard area. The relay signals - NC, NO, and C, are brought to the pads near the user connector. Use protection circuits and LEDs as applicable.

Steps:

1. Assemble the components (open collector buffers and relays) on the breadboard area.
2. Connect the relay points to the user connector pads.
3. Configure 32 digital I/O lines as output ports (**IOC0**, **IOC1**, **IOC2**, and **IOC3** should be high).
4. Use **REGSEL0** and **REGSEL1** to set the relay status. Each bit represents one relay. Writing a 16-bit word to **REGSEL0** sets the status of the first 16 relays (K1-K16), and writing a 16-bit word to **REGSEL1** sets the status of the next 16 relays (K17-K32).

Programming:

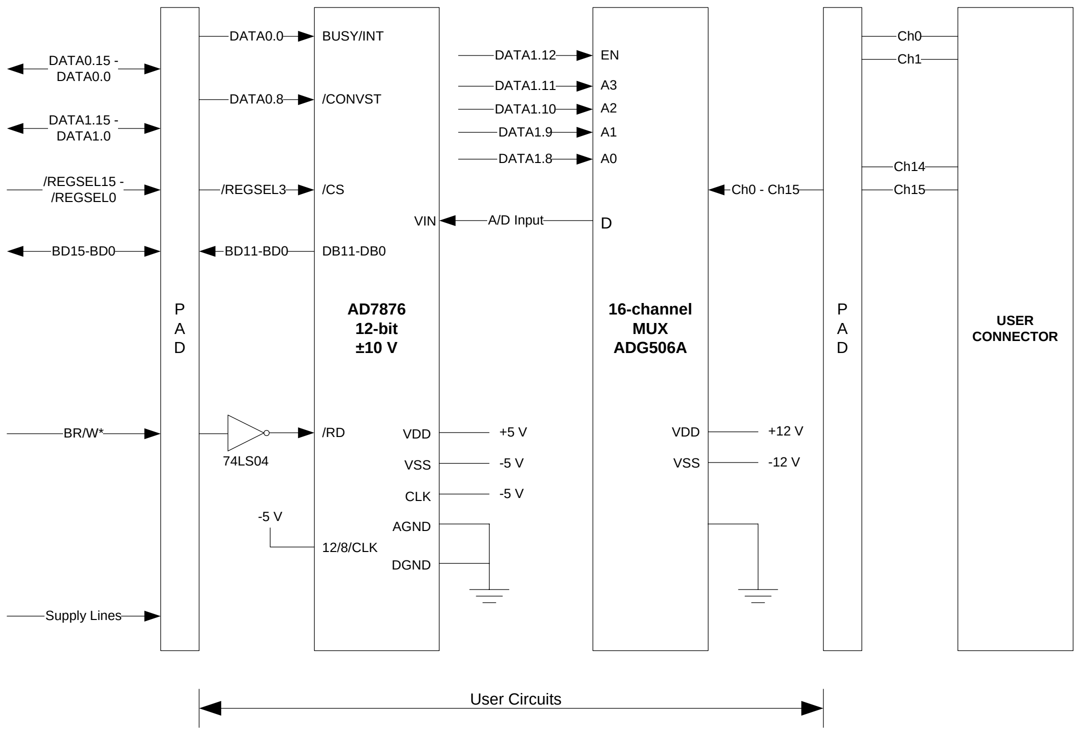
1. Configure the I/O ports.
   IOC0 - IOC3 are high by default - this will configure the **REGSEL0** and **REGSEL1** upper and lower bytes as output ports.
2. Relay Control - **REGSEL0** and **REGSEL1**:

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
REGSEL1	K32	K31	K30	K29	K28	K27	K26	K25	K24	K23	K22	K21	K20	K19	K18	K17
REGSEL0	K16	K15	K14	K13	K12	K11	K10	K9	K8	K7	K6	K5	K4	K3	K2	K1

Setting a bit to 0 opens the relay, and setting a bit to 1 closes it. For example:

- to close K2, write 0002H to **REGSEL0**
- to close K1 and K2, write 0003H to **REGSEL0**
- to close K1 through K16, write FFFFH to **REGSEL0**
- to close K18, write 0002H to **REGSEL1**
- to close K17 and K18, write 0003H to **REGSEL1**
- to close K17 through K32, write FFFFH to **REGSEL1**

16-CHANNEL A/D DESIGN EXAMPLE



Output Port = DATA0.15 – DATA0.8
Input Port = DATA0.7 – DATA0.0
Output Port = DATA1.15 – DATA1.8

FIGURE 3-5: 16-CHANNEL A/D

Additional Components Required:

1. 1 - 16-channel MUX
2. 1 - A/D
3. 1 - hex inverter

In this example, analog devices AD7876 and ADG506A are used.

AD7876 is a complete 12-bit A/D. It accepts ±10 V dc input. It needs the following signals:

CS	This is a chip select for the A/D. **REGSEL3** can be used for this.
CONVST	This is a conversion start signal for the A/D. The breadboard digital I/O signal, **DATA0.8**, can be used for this.
DB0-DB11	Result from the A/D. This is interfaced to the breadboard data bus.
RD	This is the READ signal for the A/D. The breadboard signal, **BR/W***, can be used for this. This signal is inverted through a hex inverter for an active low RD signal as required by the A/D. The results can be read with the CS (**REGSEL3**) and RD (**BR/W***) signals.

ADG506A is a 16-channel MUX. It needs the following signals:

EN	Enable signal for the MUX. The breadboard digital I/O signal, **DATA1.12**, can be used for this.
A0 to A3	Channel selection for the MUX. The breadboard digital I/O signals, **DATA1.11** - **DATA1.8** can be used for this.

Steps:

1. Assemble the components (A/D, MUX and hex inverter) on the breadboard area.
2. Connect the MUX input points (analog input) to the user connector pads.
3. Configure the on-board digital I/O lines as follows:

   - **REGSEL0** (port 0 = **DATA0.15 - DATA0.8**): Output port
   - **REGSEL0** (port 1 = **DATA0.7 - DATA0.0**): Input port
   - **REGSEL1** (port 2 = **DATA1.15 - DATA1.8**): Output port

4. Select the analog input channel by writing to port 2. Start the conversion by writing to port 0. Read the status from port 1. If the conversion is over, then read the result by reading **REGSEL3**.

Programming:

1. Set the logic for **IOC0** through **IOC3**:

    - **REGSEL0** (port 0 = **DATA0.15 - DATA0.8**): Output port→ **IOC0** logic high
    - **REGSEL0** (port 1 = **DATA0.7 - DATA0.0**): Input port→ **IOC1** logic low
    - **REGSEL1** (port 2 = **DATA1.15 - DATA1.8**): Output port→ **IOC2** logic high

2. Channels (port 2):

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
REGSEL1	x	x	x	EN	A3	A2	A1	A0	x	x	x	x	x	x	x	x

To select channels:

Bit	**12**	**11**	**10**	**9**	**8**
Ch0	1	0	0	0	0
Ch1	1	0	0	0	1
Ch2	1	0	0	1	0
Ch3	1	0	0	1	1
Ch4	1	0	1	0	0
Ch5	1	0	1	0	1
CH6	1	0	1	1	0
Ch7	1	0	1	1	1
Ch8	1	1	0	0	0
Ch9	1	1	0	0	1
Ch10	1	1	0	1	0
Ch11	1	1	0	1	1
Ch12	1	1	1	0	0
Ch13	1	1	1	0	1
Ch14	1	1	1	1	0
Ch15	1	1	1	1	1

For example:

- to select channel 0, write 10H to port 2
- to select channel 1, write 11H to port 2

3. Start conversion by writing 01H to port 0, then writing 00H to port 0 (it needs a start conversion pulse).

4. Read the conversion status in port 1. If bit0 in port 1 is 0, it indicates that the conversion is completed.

5. Read the result in **REGSEL3**. The results are in two's compliment form. **0000H** corresponds to **-10.0 V**, **07FFH** corresponds to **0.0 V**, and **0FFFH** corresponds to +**10.0 V**.

4-CHANNEL D/A DESIGN EXAMPLE

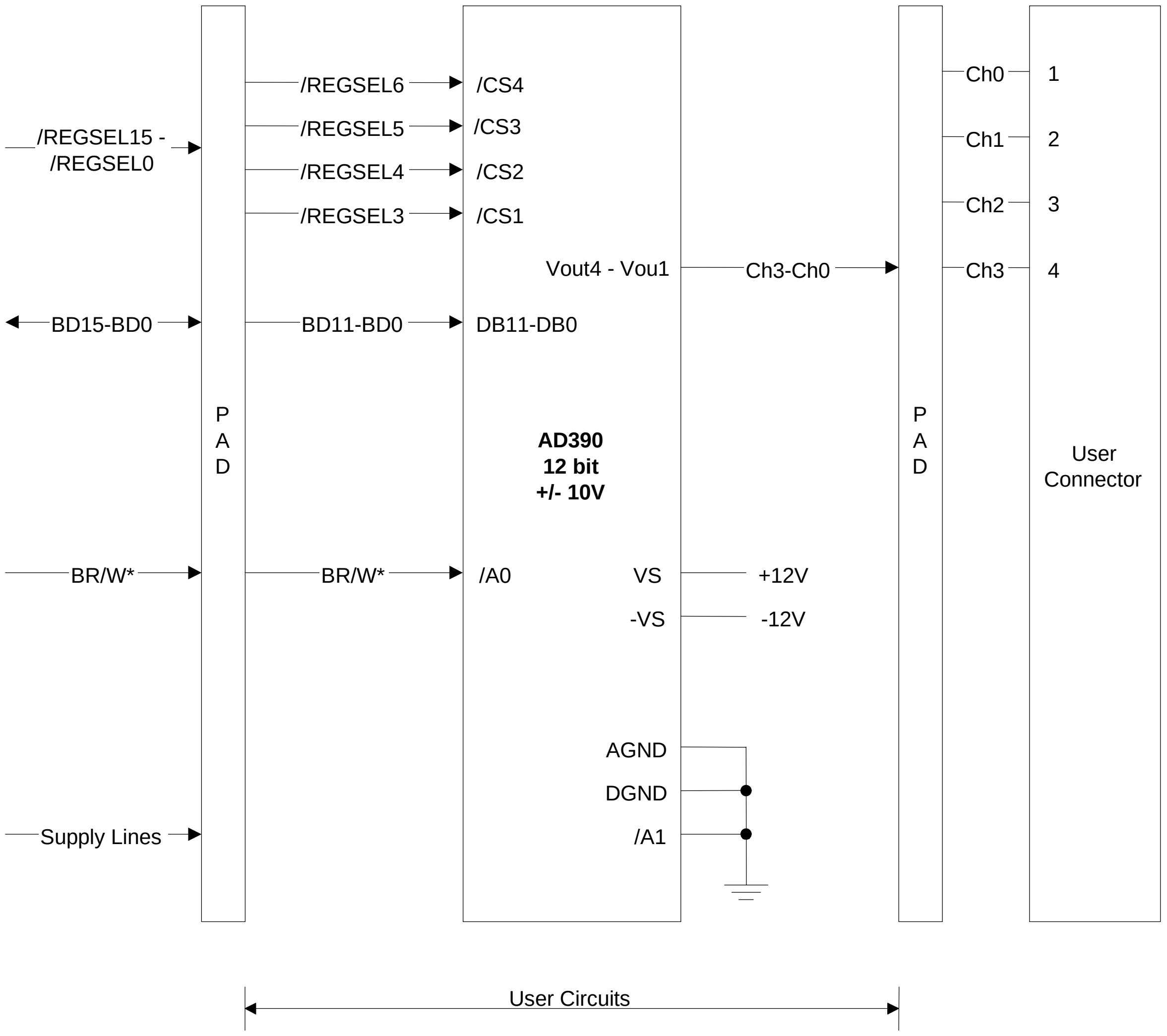


FIGURE 3-6: 4-CHANNEL D/A

Additional Components Required:

1. 1 - quad DAC

In this example, Analog Devices AD390 is used.

The AD390 is a complete 12-bit quad D/A converter. It gives four ±10 V dc outputs. It needs the following signals:

CS1 to CS4	This is a chip select/channel selection for the D/A. **REGSEL3** to **REGSEL6** can be used for this.
/A0	This is the WR signal for the D/A. The breadboard signal **BR/W*** can be used for this. The voltage level can be set with the CS (**REGSEL3 - REGSEL6**) and /A0 (**BR/W***) signals.
DB11 - DB0	Data lines for the D/A. This is interfaced to the breadboard data bus.

Steps:

1. Assemble the component (one D/A) onto the breadboard area.
2. Connect the voltage output points (analog output) to the user connector pads.
3. Select the analog output channel by writing to **REGSEL3 - REGSEL6**. **REGSEL3** represents channel 0 and **REGSEL6** represents channel 3.

Programming:

1. Select the voltage level and channel. **0000H** corresponds to **-10.0 V**, **07FFH** corresponds to **0.0 V**, and **0FFFH** corresponds to **+10.0 V**. For example:

   - to set channel 0 voltage to -10.0 V, write 0000H to **REGSEL3**

   - to set channel 1 voltage to +10.0 V, write 0FFFH to **REGSEL4**

   - to set channel 3 voltage to 0.0 V, write 07FFH to **REGSE6**

SINGLE CHANNEL UART DESIGN EXAMPLE

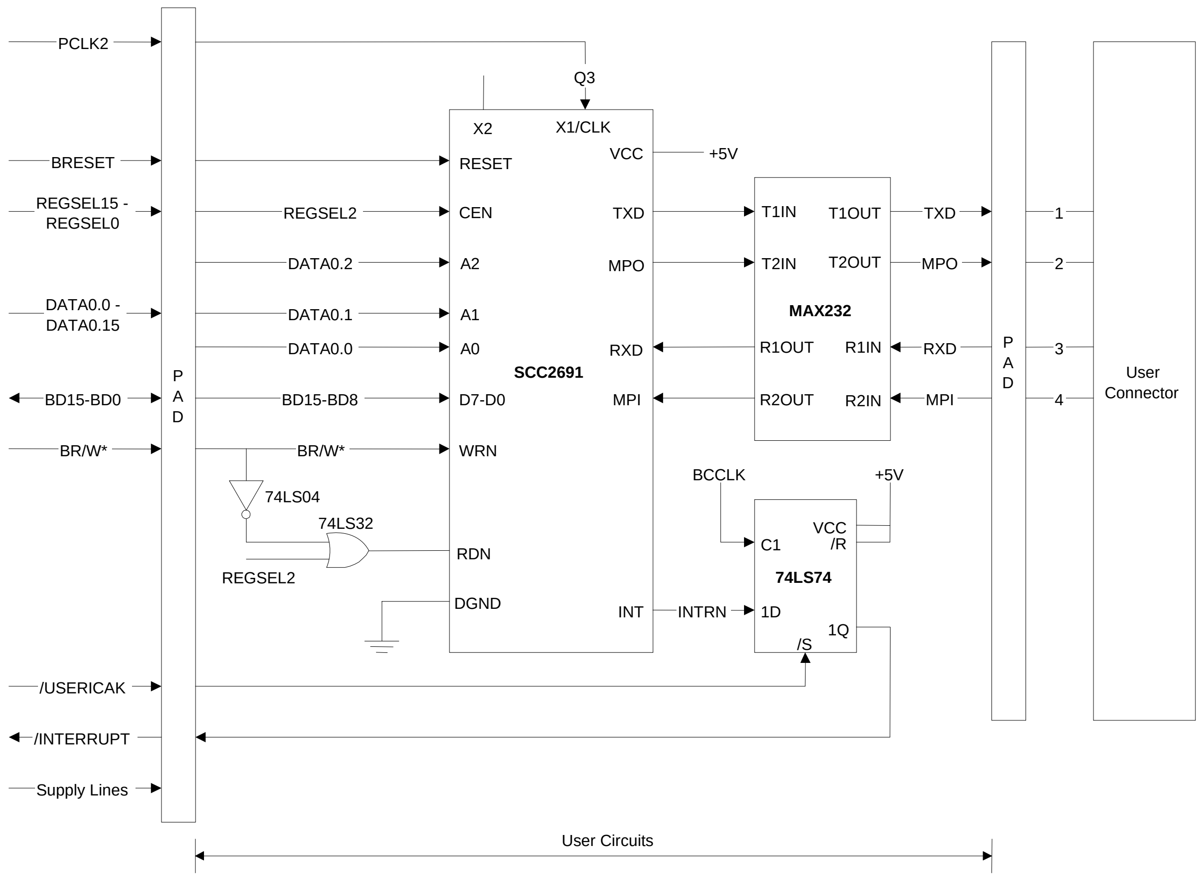


FIGURE 3-7: SINGLE CHANNEL UART

Additional Components Required:

1. 1 - UART
2. 1 – 5 V to RS-232 converter (four channel)
3. 1 - flip-flop for clearing interrupt
4. 1 - inverter buffer to get separate read/write signals
5. 1 - OR gate to generate a pulse read signal

In this example, a Philips Semiconductor SCC2691 UART and a MAXIM MAX232 TTL/RS232 interface is used.

The SCC2691 Universal Asynchronous Receiver/Transmitter is a single chip that provides a full duplex asynchronous receiver/transmitter. The interface is done between this chip and the VM7000 as follows:

X1/CLK	This is a clock input to the UART. The breadboard's **PCLK2** is connected to this signal. The system clock is divided by 6. By loading a 3 into the divide counter, a 50% duty cycle is obtained on **PCLK2** that is a division of 6.
X2	Since the external clock is given, this is left floating.
RESET	Positive system reset, **BRESET**, is connected to this pin.
CEN	Chip enable. **REGSEL2** is connected to this pin.
A2 - A0	UART internal register selection. The on-board digital I/O, **DATA0.0 - DATA0.2**, are connected to these pins.
D7 - D0	Data bus for transferring data between UART and the interface. The system data bus, **BD15 - BD8**, is used.
WRN	Active low write enable signal. The **BR/W*** signal is used for this.
RDN	Active low read enable. Since the UART needs a pulse read signal, the system read signal is combined with the CEN (**REGSEL2**) signal to generate the pulse read signal.
INTRN	This is an active high interrupt signal from the UART. The UART can generate a common interrupt on various conditions that are selected in the interrupt mask register. This signal is used in conjunction with a flip-flop, so that the interrupt going to the system is cleared when it receives a **/USERIACK** signal from the system.
MPI & MPO	These are general I/O pins that can be used for specific functions. They are not used in this example.
RXD & TXD	These are the receiving and transmit signals. These signals are TTL and they are interfaced to a RS232 converter.
MAX232	This is a signal chip, 5V, two-channel TTL/RS232 interface. It also needs a few capacitors that are not shown in the drawing.

Steps:

1. Assemble the components (UART, RS232 converter, flip-flop, inverter, and OR gate) onto the breadboard area.
2. Connect the signals as shown in the drawing.
3. Configure the on-board digital I/O lines as follows:

   REGSELO (port 0 and port 1 **= DATA0.15 - DATA0.0**): output port

4. Configure the interrupt command by word serial command.
5. Configure the counter value by word serial command.
6. The UART internal registers are selected from **DATA0.0 - DATA0.2**, and a read/write is done through **REGSEL2**.

The UART internal register selection (port 0) is done as follows:

DATA0.2	DATA0.1	DATA0.0	UART Register READ	UART Register WRITE
0	0	0	Mode Register 1 & Mode Register 2	Mode Register 1 & Mode Register 2
0	0	1	Status Register	Clock Select Register
0	1	0	Not Used	Command Register
0	1	1	Receiver Holding Register	Transmitter Holding Register
1	0	0	Not Used	Auxiliary Control Register
1	0	1	Interrupt Status Register	Interrupt Mask Register
1	1	0	Counter Upper Output Register	Counter Upper Preset Register
1	1	1	Counter Lower Output Register	Counter Lower Preset Register

Programming:

1. Set the logic high for **IOC0**, **IOC1**, **IOE0** and **IOE1**.

   - **REGSEL0** (port 0 = **DATA0.15 - DATA0.8**): Output port→ **IOC0** logic high
   - **REGSEL0** (port 1 = **DATA0.7 - DATA0.0**): Output port→ **IOC1** logic high

2. Send a word serial command to the divider as follows:

```
timer 3
```

3. Send a word serial command to configure the interrupt. On getting an interrupt, the PC end software will receive an indication that the interrupt has occurred.

```
IRQ USER
IRQ:STATUS 03
```

This will return 83H to the PC end software on getting an interrupt to the breadboard.

4. Mode Select Register 1

   Leave this register in default status, so there will be an interrupt for when the Receiving Holding Register is ready, and for Even parity.

5. Mode Select Register 2

   This is also left in the default status.

6. Clock Select Register is configured for 9.6 kbaud as follows:

   a) Write 01H to port 0 to select Clock Select Register.

   b) Write BB00H to **REGSEL2** to select 9.6k baud.

7. The Command Register is left in the default status.

8. The Auxiliary Control Register is left in the default status.

9. The Interrupt Mask Register is configured to receive an interrupt on receiving data:

   a) Write 05H to port 0 to select the Interrupt Mask Register.

   b) Write 0400H to **REGSEL2** to select Receive data ready interrupt.

10. The Counter Register is left in the default status; it is not used in this example.

11. When the PC end software gets an interrupt indication, read the Interrupt Status Register and the data as follows:

    a) Write 05H to port 0 to select the Interrupt Status Register.

    b) Read status from **REGSEL2** and check for the receive data ready.

    c) Write 03H to port 0 to select the Receiver Holding Register.

    d) Read data from **REGSEL2**.

12. Send data to the transmitter as follows:

    a) Write 01H to port 0 to select the Status Register.

    b) Read status from **REGSEL2** and check if the Transmitter Holding Register is empty.

    c) Write 03H to port 0 to select the Transmitter Holding Register.

    d) Write data to **REGSEL2** to transmit the data.

SECTION 4

COMMAND DICTIONARY

INTRODUCTION

This section presents the instrument command set. It includes an alphabetical list of all the commands supported by the VM7000 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.

Each command is described, 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.

PROGRAMMING

The VM7000 is a VXIbus message-based device whose command set is compliant with the Standard Command for Programmable Instruments (SCPI) programming language.

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, **MASK**, **SHIFt** and **STATus** are all branches off the **IRQ** trunk and can be combined as follows:

```
IRQ:MASK <value>;SHIFt <count>;STATus <status>
```

The above command is the same as the these three commands:

```
IRQ:MASK <value>
IRQ:SHIFt <count>
IRQ:STATus <status>
```

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 short and long form for the user). As an example, these commands are all correct and all have the same effect:

```
PROGram:MODule <ID>, <code>
PROGRAM:MODULE <ID>, <code>
program:module <ID>, <code>
PROG:MODule <ID>, <code>
PROG:MOD <ID>, <code>
prog:mod <ID>, <code>
```

The following command is **not** correct because it doesn't use the complete short form of **PROGram**:

```
pro:mod <ID>, <code>
```

(incorrect syntax - missing "g" - only prog or program is correct)

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 a question mark (?). Query responses do not include the command header. This means only the parameter is returned: no part of the command or "question" is returned.

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 to be used.

Parameters are enclosed by angle brackets (<>). Braces ({}), or curly brackets, 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.

ALPHABETICAL COMMAND LISTING

The following tables provide an alphabetical listing of each command supported by the VM7000 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 default 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	*RST Value
*CLS	Clear the Status Register.	X	
*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
*TRG	Causes a trigger event to occur.		N/A
*TST?	Starts and reports a self-test procedure.		N/A
*WAI	Halts execution and queries		N/A

TABLE 4-2: INSTRUMENT SPECIFIC SCPI COMMANDS

Command	Description	*RST	*RST Value
ACK	Generates an active low pulse on /USERIACK.		N/A
INPut:REGister:BYTE?	Performs an 8-bit read.		N/A
INPut:REGister[:WORD]?	Performs a 16-bit read.		N/A
IRQ	Sets the function to be performed upon receiving an interrupt.	*	USER
IRQ:INVert:MASK	Inverts the output of the user defined data.	*	All 0's
IRQ:MASK	Masks the output of the user defined data.	*	All 1's
IRQ:REGister?	Queries which register will be read upon receiving an interrupt.		N/A
IRQ:REGister:BYTE	Sets which register to read upon receiving an interrupt.	*	0
IRQ:REGister[:WORD]	Sets which register to read upon receiving an interrupt.	*	0
IRQ:SHIFt	Shifts the output of the user defined data.	*	0
IRQ:STATus	Sets the user defined value to send upon receiving an interrupt.	*	3D
MEMory:DATA	Writes data to a specified non-volatile field.		N/A
OUTPut:REGister:BYTE	Performs an 8-bit write.		N/A
OUTPut:REGister[:WORD]	Performs a 16-bit write.		N/A
PEEK:BYTE?	Performs an 8-bit read.		N/A
PEEK[:REGister]?	Performs a 16-bit read.		N/A
POKE:BYTE	Performs an 8-bit write.		N/A
POKE[:REGister]	Performs a 16-bit write.		N/A
PROGram:MODule	Sets user defined manufacturer ID and model code.		N/A
TIMer	Sets the programmable clock.	*	0

TABLE 4-3: REQUIRED SCPI COMMANDS

Command	Description	*RST	*RST Value
STATus:OPERation:CONDition?	Query the Operation Status Condition Register.	X	
STATus:OPERation:ENABle	Sets the Operation Status Enable Register.	X	
STATus:OPERation[:EVENt]?	Query the Operation Status Event Register.	X	
STATus:PRESet	Presets the Status Register.	X	
STATus:QUEStionable:CONDition?	Query the Questionable Status Condition Register	X	
STATus:QUEStionable:ENABle	Sets the Questionable Status Enable Register.	X	
STATus:QUEStionable[:EVENt]?	Query the Questionable Status Event Register	X	
SYSTem:ERRor?	Query the Error Queue	X	Clears queue
SYSTem:VERsion?	Query the version of the SCPI standard to which module complies.		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.
***RST 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	
***RST Value**	*RST performs all the functions of *CLS	
Query Syntax	None - command only	
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	
Related Commands	None	

*ESE

Purpose	Sets the bits of the Event Status Enable Register.	
Type	IEEE 488.2 Common Command	
Command Syntax	*ESE <mask>	
Command Parameters	<mask> = numeric ASCII value in the range of 0 to 255	
***RST Value**	N/A	
Query Syntax	*ESE?	
Query Parameters	None	
Query Response	Numeric ASCII value from 0 to 255	
Description	The Event Status Enable command is used to set the bits of the Event Status Enable Register. See ANSI/IEEE488.2-1987 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) Bit 2 - Query Error Bit 3 - Device Dependent Error (not used) Bit 4 - Execution Error Bit 5 - Command Error Bit 6 - User Request (not used) Bit 7 - Power On The Event Status Enable query reports the current contents of the Event Status Enable Register.	
Examples	**Command / Query**	**Response / Descriptions**
	*ESE 36	
	*ESE?	36
Related Commands	*ESR?	

*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
***RST 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) Bit 2 - Query Error Bit 3 - Device Dependent Error (not used) Bit 4 - Execution Error Bit 5 - Command Error Bit 6 - User Request (not used) Bit 7 - Power On The Operation Complete bit is set by the VM7000 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. Errors that range from -200 to -299 are execution errors. The Command Error bit is set when a command error is detected. 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** — *ESR? **Response / Descriptions** — 4
Related Commands	*ESE

*IDN?

Purpose	Queries the module for its identification string.	
Type	IEEE 488.2 Common Command	
Command Syntax	None - query only	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	*IDN?	
Query Parameters	None	
Query Response	ASCII character string	
Description	The Identification query returns the identification string of the VM7000 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).	
Examples	**Command / Query**	**Response / Descriptions**
	*IDN?	VXI Technology Inc.,VM7000,0,1.00 *(The revision listed here is for reference only; the response will always be the current revision of the instrument.)*
Related Commands	None	

*OPC

Purpose	Sets the OPC bit in the Event Status Register.	
Type	IEEE 488.2 Common Command	
Command Syntax	*OPC	
Command Parameters	None	
***RST Value**	*RST removes any pending *OPC request	
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 / Descriptions**
	*OPC	
	*OPC?	1
Related Commands	*WAI	

*RST

Purpose	Resets the module’s hardware and software to a known state.	
Type	IEEE 488.2 Common Command	
Command Syntax	*RST	
Command Parameters	None	
***RST Value**	N/A	
Query Syntax	None - command only	
Query Parameters	N/A	
Query Response	N/A	
Description	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 default parameter values set by this command.	
Examples	**Command / Query**	**Response / Descriptions**
	*RST	
Related Commands	None	

*SRE

Purpose	Set the service request enable register.	
Type	IEEE 488.2 Common Command	
Command Syntax	*SRE <mask>	
Command Parameters	<mask> = Numeric ASCII value in the range of 0 to 255	
***RST Value**	N/A	
Query Syntax	*SRE?	
Query Parameters	None	
Query Response	Numeric ASCII value from 0 to 255	
Description	The service request enable mask is used to control which bits in the status byte generate back plane interrupts. If a bit is set in the mask that newly enables a bit set in the status byte and interrupts are enabled, the module will generate a REQUEST TRUE event via an interrupt. See the *STB? Command for the layout of bits. **Note**: Bit 6 is always internally cleared to zero as required by IEEE 488.2 section 11.3.2.3.	
Examples	**Command / Query**	**Response / Descriptions**
	*SRE 4	
	*SRE?	4
Related Commands	None	

*STB?

Purpose	Queries the Status Byte Register.	
Type	IEEE 488.2 Common Command	
Command Syntax	None - query only	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	*STB?	
Query Parameters	None	
Query Response	Numeric ASCII value from 0 to 255	
Description	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 Byte Register is: Bit 0 - Unused Bit 1 - Unused Bit 2 - Error Queue Has Data Bit 3 - Questionable Status Summary (not used) Bit 4 - Message Available Bit 5 - Event Status Bit (ESB) Bit 6 - Master Summary Status Bit 7 - Operation Status Summary	
Examples	**Command / Query**	**Response / Descriptions**
	*STB?	16
Related Commands	None	

*TRG

Purpose	Causes a trigger event to occur.	
Type	IEEE 488.2 Common Command	
Command Syntax	*TRG	
Command Parameters	None	
***RST Value**	N/A	
Query Syntax	None - command only	
Query Parameters	N/A	
Query Response	N/A	
Description	The Trigger command causes a trigger event to occur.	
Examples	**Command / Query**	**Response / Descriptions**
	*TRG	
Related Commands	None	

*TST?

Purpose	Causes a self-test procedure to occur and queries the results.	
Type	IEEE 488.2 Common Command	
Command Syntax	None - query only	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	*TST?	
Query Parameters	None	
Query Response	Numeric ASCII value from 0 to 255	
Description	The Self-Test query causes the VM7000 to run a self-test.	
Examples	**Command / Query**	**Response / Descriptions**
	*TST?	0 *(A return of 0 indicates self-test passed.)*
Related Commands	None	

*WAI

Purpose	Halts execution of additional commands and queries until the No Operation Pending message is true.	
Type	IEEE 488.2 Common Command	
Command Syntax	*WAI	
Command Parameters	None	
***RST Value**	N/A	
Query Syntax	None - command only	
Query Parameters	N/A	
Query Response	N/A	
Description	The Wait to Continue command halts the execution of additional commands and queries until the No Operation Pending message is true. This command makes sure that all previous commands have been executed before proceeding.	
Examples	**Command / Query**	**Response / Descriptions**
	*WAI	
Related Commands	*OPC	

INSTRUMENT SPECIFIC SCPI COMMANDS

ACK

Purpose	Generates an active low pulse on /USERIACK.	
Type	Event	
Command Syntax	ACK	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	N/A	
Query Parameters	N/A	
Query Response	N/A	
Description	The ACK command generates an active low pulse on the /USERIACK pin. This can be used to simulate the interrupt ACK signal for the external circuitry.	
Examples	**Command / Query**	**Response / Descriptions**
	ACK	
Related Commands	IRQ	

INPut:REGister:BYTE?

Purpose	Performs an 8-bit read.	
Type	Query	
Command Syntax	N/A	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	INPut:REGister:BYTE? <offset>	
Query Parameters	<offset> = number for register offset	
Query Response	<value> = returns the value stored in the register	
Description	The Input Register Byte query reads an 8-bit value from a specified register. Performs the same function as the **PEEK:BYTE?** query.	
Examples	**Command / Query**	**Response / Descriptions**
	INP:REG:BYTE? 4	12 *(Reports the value for register 4 (offset 24).)*
Related Commands	INPut:REGister[:WORD]? OUTPut:REGister:BYTE OUTPut:REGister[:WORD] PEEK:BYTE? PEEK[:REGister]? POKE:BYTE POKE[:REGister]	

INPut:REGister[:WORD]?

Purpose	Performs a 16-bit read.	
Type	Query	
Command Syntax	N/A	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	INPut:REGister[:WORD]? <offset>	
Query Parameters	<offset> = number for register offset	
Query Response	<value> = returns the value stored in the register	
Description	The Input Register Word query reads a 16-bit value from a specified register. Performs the same function as the **PEEK[:REGister]?** query. The following table illustrates the relationship between the <value> and the register offset it corresponds to: See Register Mapping in Section 3 for an alternate method.	
Examples	**Command / Query**	**Response / Descriptions**
	INP:REG? 4	1234 *(Reports the values for Register 4 (offsets 28 and 29).)*
Related Commands	INPut:REGister:BYTE? OUTPut:REGister:BYTE OUTPut:REGister[:WORD] PEEK:BYTE? PEEK[:REGister]? POKE:BYTE POKE[:REGister]	

Value	Offset	Value	Offset
0	0x20	8	0x30
1	0x22	9	0x32
2	0x24	10	0x34
3	0x26	11	0x36
4	0x28	12	0x38
5	0x2A	13	0x3A
6	0x2C	14	0x3C
7	0x2E	15	0x3E

IRQ

Purpose	Sets the function to be performed upon receiving an interrupt.	
Type	Setting	
Command Syntax	IRQ <mode>	
Command Parameters	<mode> = REG | USER	
***RST Value**	USER	
Query Syntax	IRQ?	
Query Parameters	N/A	
Query Response	REG | USER	
Description	The IRQ command set the option to read a register, or an user defined value, upon getting an interrupt. The default setting is to read an user defined value.	
Examples	**Command / Query**	**Response / Descriptions**
	IRQ REG	*(Sets the mode to read a register upon receiving an interrupt.)*
	IRQ?	REG *(Verifies that the IRQ mode is set for a REGister read.)*
Related Commands	IRQ:INVert:MASK IRQ:MASK IRQ:REGister? IRQ:REGister:BYTE IRQ:REGister[:WORD] IRQ:SHIFt IRQ:STATus	

IRQ:INVert:MASK

Purpose	Inverts the output of the user defined data.
Type	Setting
Command Syntax	IRQ:INVert:MASK <value>
Command Parameters	<value> = a number that determines which bits to invert
***RST Value**	All 0's (no inverting)
Query Syntax	IRQ:INVert:MASK?
Query Parameters	N/A
Query Response	Returns the number that shows which bits are inverted
Description	The IRQ Invert Mask command inverts the output of the user-defined data. For example:

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
	0	0	0	0	0	1	0	1	*inverts the 1st & 3rd bit*
	0	0	0	0	0	0	1	0	*inverts the 2nd bit*
	1	0	0	0	0	0	0	0	*inverts the 7th bit*

Examples	**Command / Query**	**Response / Descriptions**
	IRQ:INV:MASK 5	*(Will invert the first and third bits of the user-defined value.)*
	IRG:INV:MASK?	*(Verifies that the first and third bits will be inverted.)*

Related Commands	IRQ IRQ:MASK IRQ:REGister? IRQ:REGister:BYTE IRQ:REGister[:WORD] IRQ:SHIFt IRQ:STATus

IRQ:MASK

Purpose	Masks the output of the user defined data.
Type	Setting
Command Syntax	IRQ:MASK <value>
Command Parameters	<value> = a number that determines which bits to mask
***RST Value**	All 0's (no masking)
Query Syntax	IRQ:MASK?
Query Parameters	N/A
Query Response	returns the number that shows which bits are masked
Description	The IRQ Mask command masks the output of the user defined data (hides specified bits).

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
	0	0	0	0	0	1	0	1	*masks the 1st & 3rd bit*
	0	0	0	0	0	0	1	0	*masks the 2nd bit*
	1	0	0	0	0	0	0	0	*masks the 7th bit*

Examples

Command / Query	**Response / Descriptions**
IRQ:MASK 5	*(Will mask (hide) the first and third bits of the user-defined value.)*
IRQ:INV:MASK?	5 *(Verifies that the first and third bits will be masked.)*

Related Commands

IRQ
IRQ:INVert:MASK
IRQ:REGister?
IRQ:REGister:BYTE
IRQ:REGister[:WORD]
IRQ:SHIFt
IRQ:STATus

IRQ:REGister?

Purpose	Queries which register will be read upon receiving an interrupt.	
Type	Query	
Command Syntax	N/A	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	IRQ:REGister?	
Query Parameters	N/A	
Query Response	WORD <offset> | BYTE <offset>	
Description	The IRQ Register query reads and returns the interrupt configuration set by the user. The information will indicate whether the setting is to read a WORD or a BYTE, as well as the register that will be read.	
Examples	**Command / Query**	**Response / Descriptions**
	IRQ:REG:BYTE 3	*(Sets the interrupt configuration to read the data from register 3 (offset 23).)*
	IRQ:REG?	BYTE 3 *(Verifies that the interrupt configuration is set to read a byte of data from register 3 (offset 23).)*
Related Commands	IRQ:REGister:BYTE IRQ:REGister[:WORD]	

IRQ:REGister:BYTE

Purpose	Sets which register to read upon receiving an interrupt.	
Type	Setting	
Command Syntax	IRQ:REGister:BYTE <offset>	
Command Parameters	<offset> = number for register offset	
***RST Value**	0	
Query Syntax	IRQ:REGister?	
Query Parameters	N/A	
Query Response	BYTE <offset> | WORD <offset>	
Description	The IRQ Register Byte command set the interrupt configuration to read a byte of data from a specified register.	
Examples	**Command / Query**	**Response / Descriptions**
	IRQ:REG:BYTE 2	*(Sets the interrupt configuration to read the data from register 2 (offset 22).)*
	IRQ:REG?	BYTE 2 *(Verifies that the interrupt configuration is set to read a byte of data from register 2 (offset 22).)*
Related Commands	IRQ:REGister? IRQ:REGister[:WORD]	

IRQ:REGister[:WORD]

Purpose	Sets which register to read upon receiving an interrupt.	
Type	Setting	
Command Syntax	IRQ:REGister[:WORD] <offset>	
Command Parameters	<offset> = number for register offset	
***RST Value**	0	
Query Syntax	IRQ:REGister?	
Query Parameters	N/A	
Query Response	BYTE <offset> | WORD <offset>	
Description	The IRQ Register Word command set the interrupt configuration to read a word of data from a specified register.	
Examples	**Command / Query**	**Response / Descriptions**
	IRQ:REG 2	*(Sets the interrupt configuration to read the data from register 2 (offset 22). Will actually read 2 and 3 (offset 22 and 23).)*
	IRQ:REG?	WORD 2 *(Verifies that the interrupt configuration is set to read a word of data from register 2 (offset 22). Will actually read 2 and 3 (offset 22 and 23).)*
Related Commands	IRQ:REGister? IRQ:REGister:BYTE	

IRQ:SHIFt

Purpose	Shifts the output of the user defined data.
Type	Setting
Command Syntax	IRQ:SHIFt <count>
Command Parameters	<count> = a number that determines how many places to shift
***RST Value**	0
Query Syntax	IRQ:SHIFt?
Query Parameters	N/A
Query Response	Returns the number that shows how many places the bits are shifted
Description	The IRQ Shift command shifts the output of the user-defined data. The parameter entered determines the number of places the bits are shifted to the right. For example, if the command entered was **IRQ:SHIFt 1**, the input and output would look like this:

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
	0	0	0	0	1	1	1	0	*bits 1, 2 & 3 set high (input)*

Bits	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	
	0	0	0	0	0	1	1	1	*bits 1, 2 & 3 set high, shifted to the right one place (output)*

Examples	**Command / Query**	**Response / Descriptions**
	IRQ:SHIF 1	*(Will shift the out bit to the right by one place.)*
	IRQ:SHIF?	1 *(Verifies that output is set to be shifted over one place.)*

Related Commands	IRQ IRQ:INVert:MASK IRQ:MASK IRQ:REGister? IRQ:REGister:BYTE IRQ:REGister[:WORD] IRQ:STATus

IRQ:STATus

Purpose	Sets the user defined value to send upon receiving an interrupt.	
Type	Setting	
Command Syntax	IRQ:STATus <status>	
Command Parameters	<status> = user defined value	
***RST Value**	3D	
Query Syntax	IRQ:STATus?	
Query Parameters	N/A	
Query Response	user defined value	
Description	The IRQ Status command specifies what user defined value to send upon getting an interrupt.	
Examples	**Command / Query**	**Response / Descriptions**
	IRQ:STAT 3	*(Sets the value.)*
	IRQ:STAT?	3F *(Would actually return BF (the 7th bit is always low and the 8th bit is always high).)*
Related Commands	IRQ IRQ:INVert:MASK IRQ:MASK IRQ:REGister? IRQ:REGister:BYTE IRQ:REGister[:WORD] IRQ:SHIFt	

MEMory:DATA

Purpose	Writes data to a specified non-volatile field.	
Type	Setting	
Command Syntax	MEMory:DATA <index>,<data>	
Command Parameters	<index> = memory location <data> = definite or indefinite length arbitrary block format	
***RST Value**	N/A	
Query Syntax	MEMory:DATA? <index>	
Query Parameters	<index> = memory location	
Query Response	<data>	
Description	The Memory Data command writes data to specified non-volatile field.	
Examples	**Command / Query**	**Response / Descriptions**
	MEM:DATA 99, #2109876543210	*(Writes user specified data to a specific memory location.)*
	MEM:DATA? 99	#2109876543210
Related Commands	N/A	

OUTPut:REGister:BYTE

Purpose	Performs an 8-bit write.	
Type	Setting	
Command Syntax	OUTPut:REGister:BYTE <offset>,<value>	
Command Parameters	<offset> = register offset <value> = value to write to the register	
***RST Value**	N/A	
Query Syntax	N/A	
Query Parameters	N/A	
Query Response	N/A	
Description	The Output Register Byte command writes an 8-bit value to a specified register. Performs the same function as **POKE:BYTE** command.	
Examples	**Command / Query**	**Response / Descriptions**
	OUTP:REG:BYTE 4,12	*(Writes the value 12 (0001 0010) to register 4 (offset 24).)*
	INP:REG:BYTE? 4	12 *(Returns the value stored in register 4 (offset 24).)*
Related Commands	INPut:REGister:BYTE? INPut:REGister[:WORD]? OUTPut:REGister[:WORD] PEEK:BYTE? PEEK[:REGister]? POKE:BYTE POKE[:REGister]	

OUTPut:REGister[:WORD]

Purpose	Performs a 16-bit write.
Type	Setting
Command Syntax	OUTPut:REGister[:WORD] <offset>,<value>
Command Parameters	<offset> = register offset <value> = value to write to the register
***RST Value**	N/A
Query Syntax	N/A
Query Parameters	N/A
Query Response	N/A
Description	The Output Register Word command writes an 16-bit value to a specified register. Performs the same function as **POKE[:REGister]** command. The following table illustrates the relationship between the <value> and the register offset it corresponds to:

Value	Offset	Value	Offset
0	0x20	8	0x30
1	0x22	9	0x32
2	0x24	10	0x34
3	0x26	11	0x36
4	0x28	12	0x38
5	0x2A	13	0x3A
6	0x2C	14	0x3C
7	0x2E	15	0x3E

See Register Mapping in Section 3 for an alternate method.

Examples

Command / Query	**Response / Descriptions**
OUTP:REG 4,1234	*(Writes the value 1234 (0001 0010 0011 0100) to Register 4 (offsets 28 and 29).)*
INP:REG? 4	1234 *(Returns the value stored in Register 4 (offsets 28 and 29).)*

Related Commands

INPut:REGister:BYTE?
INPut:REGister[:WORD]?
OUTPut:REGister:BYTE
PEEK:BYTE?
PEEK[:REGister]?
POKE:BYTE
POKE[:REGister]

PEEK:BYTE?

Purpose	Performs an 8-bit read.	
Type	Query	
Command Syntax	N/A	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	PEEK:BYTE? <offset>	
Query Parameters	<offset> = number for register offset	
Query Response	<value> = returns the value stored in the register	
Description	The Peek Byte query reads an 8-bit value from a specified register. Performs the same function as the **INPut:REGister:BYTE?** query.	
Examples	**Command / Query**	**Response / Descriptions**
	PEEK:BYTE? 4	12 *(Reports the value for register 4 (offset 24).)*
Related Commands	INPut:REGister:BYTE? INPut:REGister[:WORD]? OUTPut:REGister:BYTE OUTPut:REGister[:WORD] PEEK[:REGister]? POKE:BYTE POKE[:REGister]	

PEEK[:REGister]?

Purpose	Performs a 16-bit read.
Type	Query
Command Syntax	N/A
Command Parameters	N/A
***RST Value**	N/A
Query Syntax	PEEK[:REGister]? <offset>
Query Parameters	<offset> = number for register offset
Query Response	<value> = returns the value stored in the register
Description	The Peek Register query reads a 16-bit value from a specified register. Performs the same function as the **INPut:REGister[:WORD]?** query. The following table illustrates the relationship between the <value> and the register offset it corresponds to:

Value	Offset	Value	Offset
0	0x20	8	0x30
1	0x22	9	0x32
2	0x24	10	0x34
3	0x26	11	0x36
4	0x28	12	0x38
5	0x2A	13	0x3A
6	0x2C	14	0x3C
7	0x2E	15	0x3E

See Register Mapping in Section 3 for an alternate method.

Examples	**Command / Query**	**Response / Descriptions**
	PEEK? 4	1234 *(Reports the values for Register 4 (offsets 28 and 29).)*

Related Commands	INPut:REGister:BYTE? INPut:REGister[:WORD]? OUTPut:REGister:BYTE OUTPut:REGister[:WORD] PEEK:BYTE? POKE:BYTE POKE[:REGister]

POKE:BYTE

Purpose	Performs an 8-bit write.	
Type	Setting	
Command Syntax	POKE:BYTE <offset>,<value>	
Command Parameters	<offset> = register offset <value> = value to write to the register	
***RST Value**	N/A	
Query Syntax	N/A	
Query Parameters	N/A	
Query Response	N/A	
Description	The Poke Byte command writes an 8-bit value to a specified register. Performs the same function as **OUTPut:REGister:BYTE** command.	
Examples	**Command / Query**	**Response / Descriptions**
	POKE:BYTE 4,12	*(Writes the value 12 (0001 0010) to register 4 (offset 24).)*
	PEEK:BYTE? 4	12 *(Returns the value stored in register 4 (offset 24).)*
Related Commands	INPut:REGister:BYTE? INPut:REGister[:WORD]? OUTPut:REGister:BYTE OUTPut:REGister[:WORD] PEEK:BYTE? PEEK[:REGister]? POKE[:REGister]	

POKE[:REGister]

Purpose	Performs a 16-bit write.
Type	Setting
Command Syntax	POKE[:REGister] <offset>,<value>
Command Parameters	<offset> = register offset <value> = value to write to the register
***RST Value**	N/A
Query Syntax	N/A
Query Parameters	N/A
Query Response	N/A
Description	The Poke Register command writes an 16-bit value to a specified register. Performs the same function as **OUTPut:REGister[:WORD]** command. The following table illustrates the relationship between the <value> and the register offset it corresponds to:

Value	Offset	Value	Offset
0	0x20	8	0x30
1	0x22	9	0x32
2	0x24	10	0x34
3	0x26	11	0x36
4	0x28	12	0x38
5	0x2A	13	0x3A
6	0x2C	14	0x3C
7	0x2E	15	0x3E

See Register Mapping in Section 3 for an alternate method.

Examples

Command / Query	Response / Descriptions
POKE 4,1234	*(Writes the value 1234 (0001 0010 0011 0100) to Register 4 (offsets 28 and 29).)*
PEEK? 4	1234 *(Returns the value stored in Register 4 (offsets 28 and 29).)*

Related Commands

INPut:REGister:BYTE?
INPut:REGister[:WORD]?
OUTPut:REGister:BYTE
OUTPut:REGister[:WORD]
PEEK:BYTE?
PEEK[:REGister]?
POKE:BYTE

PROGram:MODule

Purpose	Sets user defined manufacturer ID and model code.	
Type	Setting	
Command Syntax	PROGram:MODule <ID>,<code>	
Command Parameters	<ID> = user defined manufacturer's ID <code> = user defined model code	
***RST Value**	N/A	
Query Syntax	PROGram:MODule?	
Query Parameters	N/A	
Query Response	<ID>,<code>	
Description	The Program Module command set a user-defined manufacturer's I.D. and model code.	
Examples	**Command / Query**	**Response / Descriptions**
	PROG:MOD VXI,VM7000	
Related Commands	N/A	

TIMer

Purpose	Sets the programmable clock.	
Type	Setting	
Command Syntax	TIMer <value>	
Command Parameters	<value>	
***RST Value**	0	
Query Syntax	TIMer?	
Query Parameters	N/A	
Query Response	<value>	
Description	The Timer command set the programmable clock. There is a 25 MHz clock that is divided by an internal 16-bit counter and provides a **PCLK1** and a **PCLK2** output.	
Examples	**Command / Query**	**Response / Descriptions**
	TIM 8	*(**PLCK1** will be high for **1** system clock, and low for **7** system clocks.* * **PCLK2** will be high for **8** system clocks, and low for **8** system clocks.)*
	TIM 10	*(**PCLK1** will be high for **1** system clock, and low for **9** system clocks.* * **PCLK2** will be high for **10** system clocks, and low for **10** system clocks.)*
Related Commands	N/A	

REQUIRED SCPI COMMANDS

STATus:OPERation:CONDition?

Purpose	Queries the Operation Status Condition Register.	
Type	Required SCPI command	
Command Syntax	None - query only	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	STATus:OPERation:CONDition?	
Query Parameters	None	
Query Response	0	
Description	The Operation Status Condition Register query is provided for SCPI compliance only. The VM7000 does not alter the state of any of the bits in this register and always reports a 0.	
Examples	**Command / Query**	**Response / Descriptions**
	STAT:OPER:COND?	0
Related Commands	None	

STATus:OPERation:ENABle

Purpose	Sets the Operation Status Enable Register.	
Type	Required SCPI command	
Command Syntax	STATus:OPERation:ENABle <NRF>	
Command Parameters	<NRF> = numeric ASCII value from 0 to 32767	
***RST Value**	N/A	
Query Syntax	STATus:OPERation:ENABle?	
Query Parameters	None	
Query Response	Numeric ASCII value from 0 to 32767	
Description	The Operation Status Enable Register is included for SCPI. The register layout is as follows: Bit 0 - Calibrating (not used on the VM7000) Bit 1 - Setting (not used on the VM7000) Bit 2 - Ranging (not used on the VM7000) Bit 3 - Sweeping (not used on the VM7000) Bit 4 - Measuring (not used on the VM7000) Bit 5 - Waiting for trigger (not used on the VM7000) Bit 6 - Waiting for arm (not used on the VM7000) Bit 7 - Correcting (not used on the VM7000)	
Examples	**Command / Query**	**Response / Descriptions**
	STAT:OPER:ENAB 0	
	STAT:OPER:ENAB?	0
Related Commands	None	

STATus:OPERation:EVENt?

Purpose	Queries the Operation Status Event Register.	
Type	Required SCPI command	
Command Syntax	None - query only	
Command Parameters	N/A	
***RST 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 register layout is as follows: Bit 0 - Calibrating (not used on the VM7000) Bit 1 - Settling (not used on the VM7000) Bit 2 - Ranging (not used on the VM7000) Bit 3 - Sweeping (not used on the VM7000) Bit 4 - Measuring (not used on the VM7000) Bit 5 - Waiting for trigger (not used on the VM7000) Bit 6 - Waiting for arm (not used on the VM7000) Bit 7 - Correcting (not used on the VM7000)	
Examples	**Command / Query**	**Response / Descriptions**
	STAT:OPER?	0
Related Commands	None	

STATus:PRESet

Purpose	Presets the Status Registers.	
Type	Required SCPI command	
Command Syntax	STATus:PRESet	
Command Parameters	None	
***RST Value**	N/A	
Query Syntax	None - command only	
Query Parameters	N/A	
Query Response	N/A	
Description	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.	
Examples	**Command / Query**	**Response / Descriptions**
	STAT:PRES	
Related Commands	None	

STATus:QUEStionable:CONDition?

Purpose	Queries the Questionable Status Condition Register.	
Type	Required SCPI command	
Command Syntax	None - query only	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	STATus:QUEStionable:CONDition?	
Query Parameters	None	
Query Response	0	
Description	The Questionable Status Condition Register query is provided for SCPI compliance only. The VM7000 does not alter any of the bits in this register and a query always reports a 0.	
Examples	**Command / Query**	**Response / Descriptions**
	STAT:QUES:COND?	0
Related Commands	None	

STATus:QUEStionable:ENABle

Purpose	Sets the Questionable Status Enable Register.
Type	Required SCPI command
Command Syntax	STATus:QUEStionable:ENABle <NRF>
Command Parameters	NRF = numeric ASCII value from 0 to 32767
***RST Value**	N/A
Query Syntax	STATus:QUEStionable:ENABle?
Query Parameters	None
Query Response	Numeric ASCII value from 0 to 32767
Description	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 VM7000 does not alter the bit settings of this register and will report the last programmed value.
Related Commands	None

Examples

Command / Query	**Response / Descriptions**
STAT:QUES:ENAB 64	
STAT:QUES:ENAB?	64

STATus:QUEStionable:EVENt

Purpose	Queries the Questionable Status Event Register.	
Type	Required SCPI command	
Command Syntax	None - query only	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	STATus:QUEStionable [:EVENt]?	
Query Parameters	None	
Query Response	0	
Description	The Questionable Status Event Register is provided for SCPI compliance only. The VM7000 does not alter the bits in this register and queries always report a 0.	
Examples	**Command / Query**	**Response / Descriptions**
	STAT:QUES?	0
Related Commands	None	

SYSTem:ERRor?

Purpose	Queries the Error Queue.	
Type	Required SCPI command	
Command Syntax	None - query only	
Command Parameters	N/A	
***RST 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 up to ten error messages. If additional errors occur, the queue will overflow and the tenth and subsequent error messages will be lost. In the case of an overflow, an overflow message will replace the tenth error message. See the SCPI standard Volume 2: Command Reference for details on errors and reporting them.	
Examples	**Command / Query**	**Response / Descriptions**
	SYS:ERR?	-350 *(No error.)*
Related Commands	None	

SYSTem:VERSion?

Purpose	Queries the SCPI version number with which the VM7000 complies.	
Type	Required SCPI command	
Command Syntax	None - query only	
Command Parameters	N/A	
***RST Value**	N/A	
Query Syntax	SYSTem:VERSion?	
Query Parameters	None	
Query Response	Numeric ASCII value	
Description	The System Version query reports version of the SCPI standard with which the VM7000 complies.	
Examples	**Command / Query**	**Response / Descriptions**
	SYST:VERS?	1994.0
Related Commands	None	

SECTION 5

DOCUMENTATION

LIST

The documents referenced below can be found at the rear of this manual, after the index. If an 11x17 copy of the schematics are desired, simply open the *VM7000 User's Manual* PDF found on the VXI*plug&play* Drivers and Product User's Guide CD that comes with the VM7000 and print the desired pages, making the appropriate paper size selections for the printer being used.

VTI Part No.	**Document Type**	**Description**
52-0135-000PL	Parts List	VM7000 Breadboard
50-0095-000	Schematic	VM7000 Breadboard
99-0011-000	Mechanical Specification	VMIP Daughter Board

INDEX

Engineering Parts List

Released: 12/17/98

ITEM	VTI PART#	QTY	DESCRIPTION	CAGE	MANUFCTR'S P/N	NOTES
1	01-0010-471	4	RES, CHIP, 470 OHM 5%, 0.12W, 1206	91637	CRCW1206-471J	R1, R2, R4, R6
2	01-0076-103	1	RES, NET, 10K 2%, 20 PINS, 19 RES, .31" SMD	91637	SOMC-2001-103G	RP1
3	05-0010-104	7	CAP, CHIP, 0.1uF 20% 50V Z5U, 1206	04222	12065E104MATMA	DC1, DC1A, DC1B, DC1C, DC1D, DC1E, DC1F
4	11-0024-000	1	LED, DUAL T-1 BI-COLOR RED/GREEN, R/A PCB MOUNT	030B9	PL308-2GR	D1
5	21-0038-001	1	IC, FPGA, PROG, QL2007-1, 208 PIN, VM7000, U1	03LB1	21-0038-001 (FROM -000)	U1
6	27-0062-068	1	CONN, FEMALE AMPLIMATE .050 SERIES, R/A, 68 PINS	00779	787171-7	J1
7	27-0068-060	2	CONN, PLUG, VMIP INTERFACE, 60 PIN	52072	628-030	P1, P2
8	33-0107-000	1	PCB, VMIP, VM7000 BREADBOARD	03LB1	33-0107-000	

VXI Technology, Inc. 17912 Mitchell, Irvine CA 92614 (949) 955-1894 Fax: (949) 955-3041	Title: ASSY, PCB, VM7000, VMIP BREADBOARD	Doc No: 52-0135-000PL Rev: A Sheet 1 of 1

REVISIONS

REV	ECN	DESCRIPTION	DATE	APPROVED
A	0184	Release	12/18/98	T.F.

Rework / Assembly instructions for 52-0135-000 - Rev. A (with 33-0107-000 - Rev. B):

1. Do not solder R3 and R5.

2. Make the following trace cuts:

   a) Cut trace from left pad-top of R4 to via - topside.

   b) Cut trace from right pad-right of R4 to via - topside. Scrape off some of the solder mask on the via side of the cut etch (this lead to U1-195).

   c) Cut trace from left pad-top of R6 to via - topside.

   d) Cut trace from right pad-bottom of R6 to via - topside. Scrape off some of the solder mask on the via side of the cut etch (this lead to U1-197).

3. Solder the following jumpers using 30GA wire:

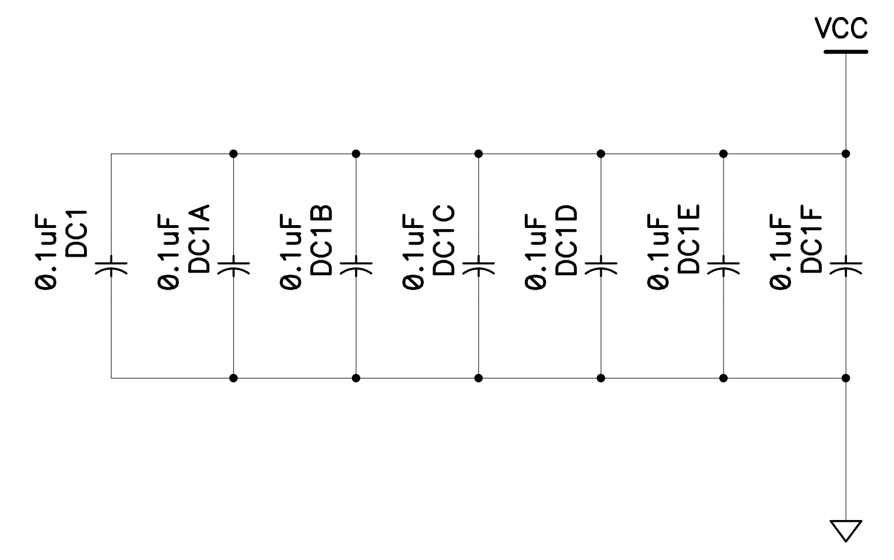
   a) Solder left pad of R4 to the exposed trace leading to U1-195 - topside.

   b) Solder left side of R6 to the exposed trace leading to U1-197 - topside.

UNLESS OTHERWISE SPEC'D DIMENSIONS ARE IN INCHES TOLERANCES ARE:	CONTRACT	VXI Technology, Inc. 17912 Mitchell, Irvine CA 92614 (949) 955-1894		
2 PLCS ± 0.010 / 3 PLCS ± 0.005 / ANGLES ± 1 DEG		ASSY, PCB, VM7000, VMIP BREADBOARD		
MATERIAL:	DRAWN: T. Fairbanks			
	CHECKED:			
	ENGR:	SIZE A	CAGE 03LB1	DOC NO: 52-0135-000RW / REV: A
	APPROVED:			
	APPROVED:	SCALE: NONE	FILE: 52-0135-000RW-a.doc	Page 1 of 1

REVISIONS

REV:	ECO:	DESCRIPTION:	DATE:	APPROVAL:
A		RELEASE FOR PROTOTYPE	12/01/98	M.C.
B	0184	RELEASE FOR PRODUCTION	12/10/98	M.C.
C	3318	REVISED TO CHANGE PAD DESIGNATORS	02.24.2005	R.G.



6. FOR PRINTED CIRCUIT BOARD, REFER TO: 33-0107-000.
5. RESISTORS WITH 3 SIGNIFICANT DIGITS ARE 0.12W 1%.
4. RESISTORS WITH 2 SIGNIFICANT DIGITS ARE 0.06W 5%.
3. RESISTOR VALUES ARE IN OHMS.
2. CAPACITOR VALUES 1.0uF OR GREATER ARE 35V 20% TANTALUM.
1. CAPACITOR VALUES LESS THAN 1.0uF ARE 50V 20% MONOLYTHIC CERAMIC.

NOTES: UNLESS OTHERWISE SPECIFIED.

REV: C SH: 1 DRW NO: 50-0095-000

VXI Technology, Inc.
17912 MITCHELL
IRVINE CA 92614 (949) 955-1894

APPROVAL:		DATE:
DRN	M.C.	12/01/98
CHK		
ENGR		
Q.A.		
CUST		

TITLE: SCHEMATIC, VM7000 VMIP BREADBOARD

SIZE	CAGE CODE	DWG NO.	REV.
D	03LB1	50-0095-000	C
SCALE NONE	FILE 0095-00b.sch	SHEET 1 OF 2	

P1:A
VMIP_P1
P1:B
VMIP_P1
P2:A
VMIP_P2
P2:B
VMIP_P2
U1
QL2007_0PQ208C
VCC
470E
R1
470E
R2
D1:A
LOWER LED
GRN
RED
470E
R4
D1:B
UPPER LED
GRN
RED
470E
R6
POWERCATH
FAILCATH
ACCESSCATH
ERRORCATH
RP1
10K
J2:A
CON68A
J1:A
CONN_68FM_SCSI2
J2:B
CON68A
J1:B
CONN_68FM_SCSI2
PAD

VXI Technology, Inc.
17912 MITCHELL
IRVINE CA 92614 (949) 955-1894

TITLE:
SCHEMATIC, VM7000 VMIP BREADBOARD

SIZE	CAGE CODE	DWG NO.	REV.
D	03LB1	50-0095-000	C
SCALE NONE	FILE 0095-00b.sch		SHEET 2

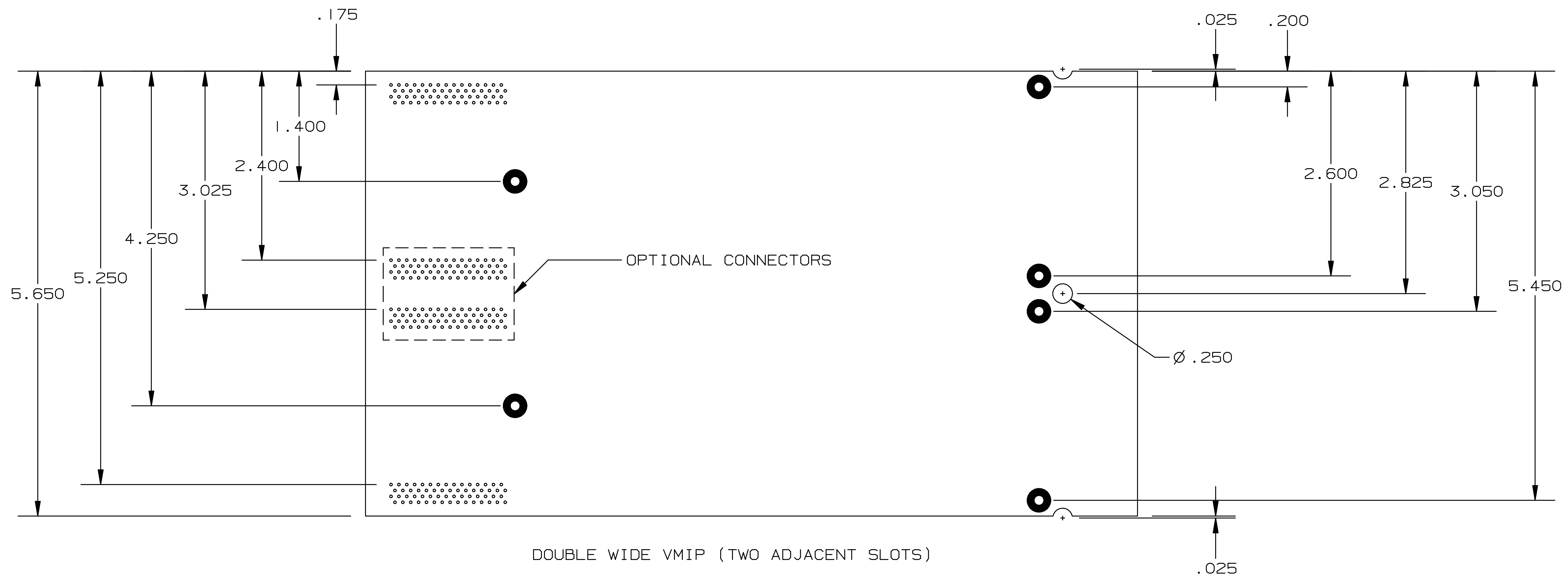
DRW NO: 50-0095-000 SH: 2 REV: C

PROPRIETARY NOTICE

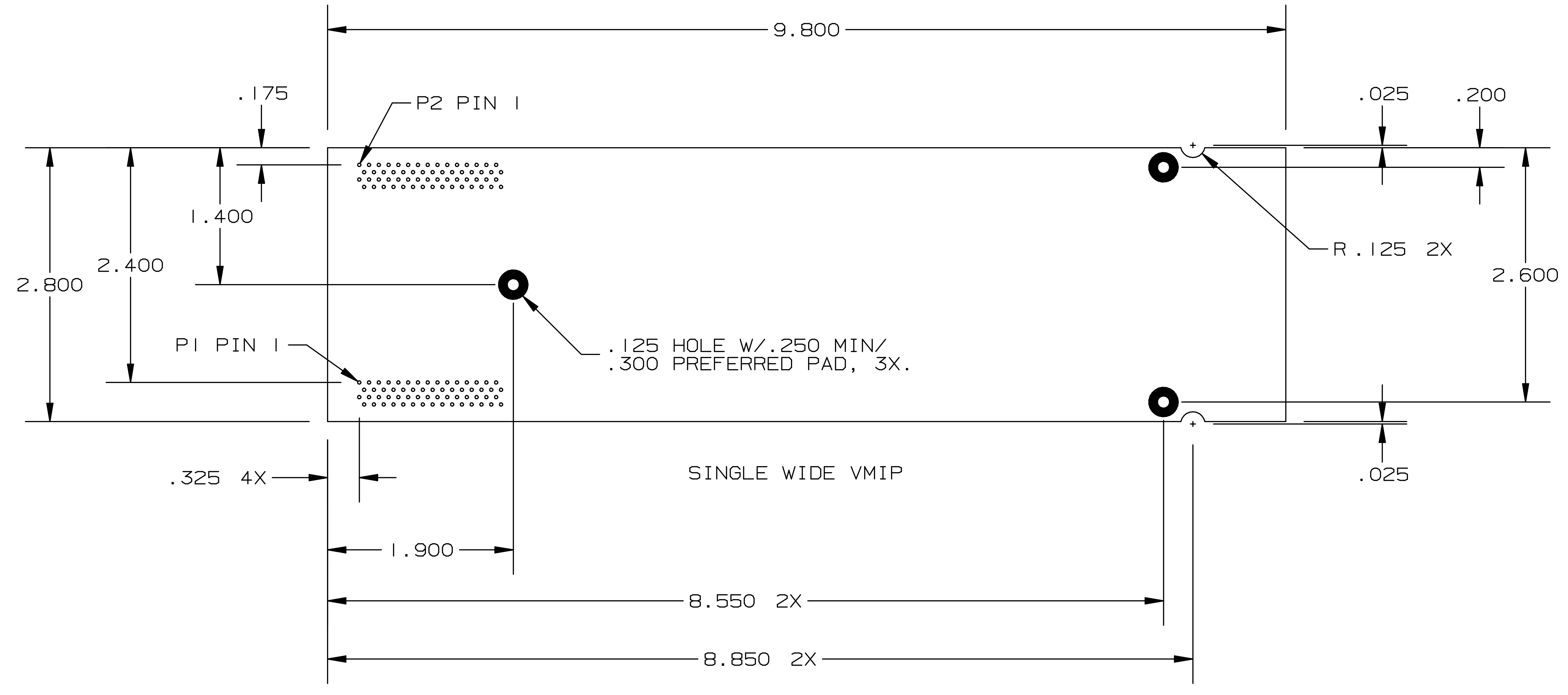
THIS DOCUMENT AND THE TECHNICAL DATA DISCLOSED ARE PROPRIETARY TO VXI TECHNOLOGY, INC. AND SHALL NOT, WITHOUT EXPRESS WRITTEN PERMISSION OF VXI TECHNOLOGY, INC. BE USED, RELEASED OR DISCLOSED IN WHOLE OR PART OR USED TO SOLICIT QUOTATIONS FROM A COMPETITIVE SOURCE OR USED FOR MANUFACTURE BY ANYONE OTHER THAN VXI TECHNOLOGY, INC. THE INFORMATION HERE ON HAS BEEN DEVELOPED AT PRIVATE EXPENSE AND MAY ONLY BE USED FOR PURPOSES OF ENGINEERING EVALUATION AND FOR INCORPORATION INTO TECHNICAL SPECIFICATINS AND OTHER DOCUMENTS WHICH SPECIFY PROCUREMENT FROM VXI TECHNOLOGY, INC.

REVISION HISTORY

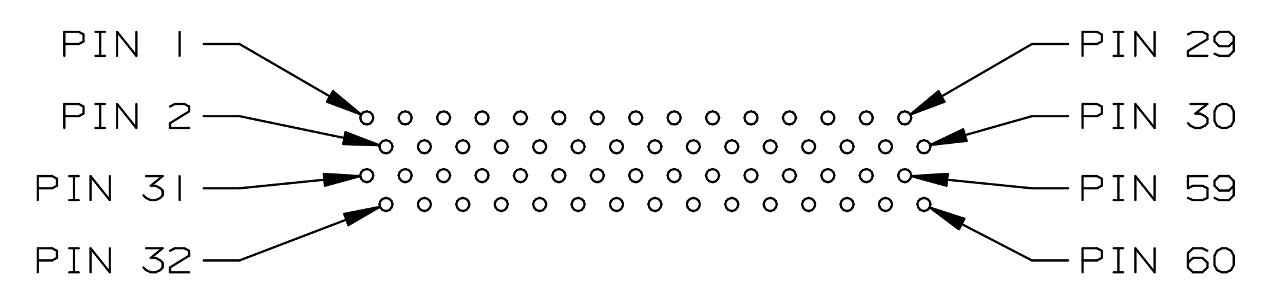
REV	ECN	DESCRIPTION	DATE	APPROVED
A		RELEASE FOR PRODUCTION.	01/29/98	A.J.G.
B	0344	CORRECT THE LOCATION OF P1 CONNECTOR.	08/14/98	A.J.G.



DOUBLE WIDE VMIP (TWO ADJACENT SLOTS)



SINGLE WIDE VMIP



P1 AND P2 PIN ORIENTATION

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES
TOLERANCES
.XX = ±0.01 .XXX = ±0.005
ANGLES = ±1.0°

THIRD ANGLE PROJECTION

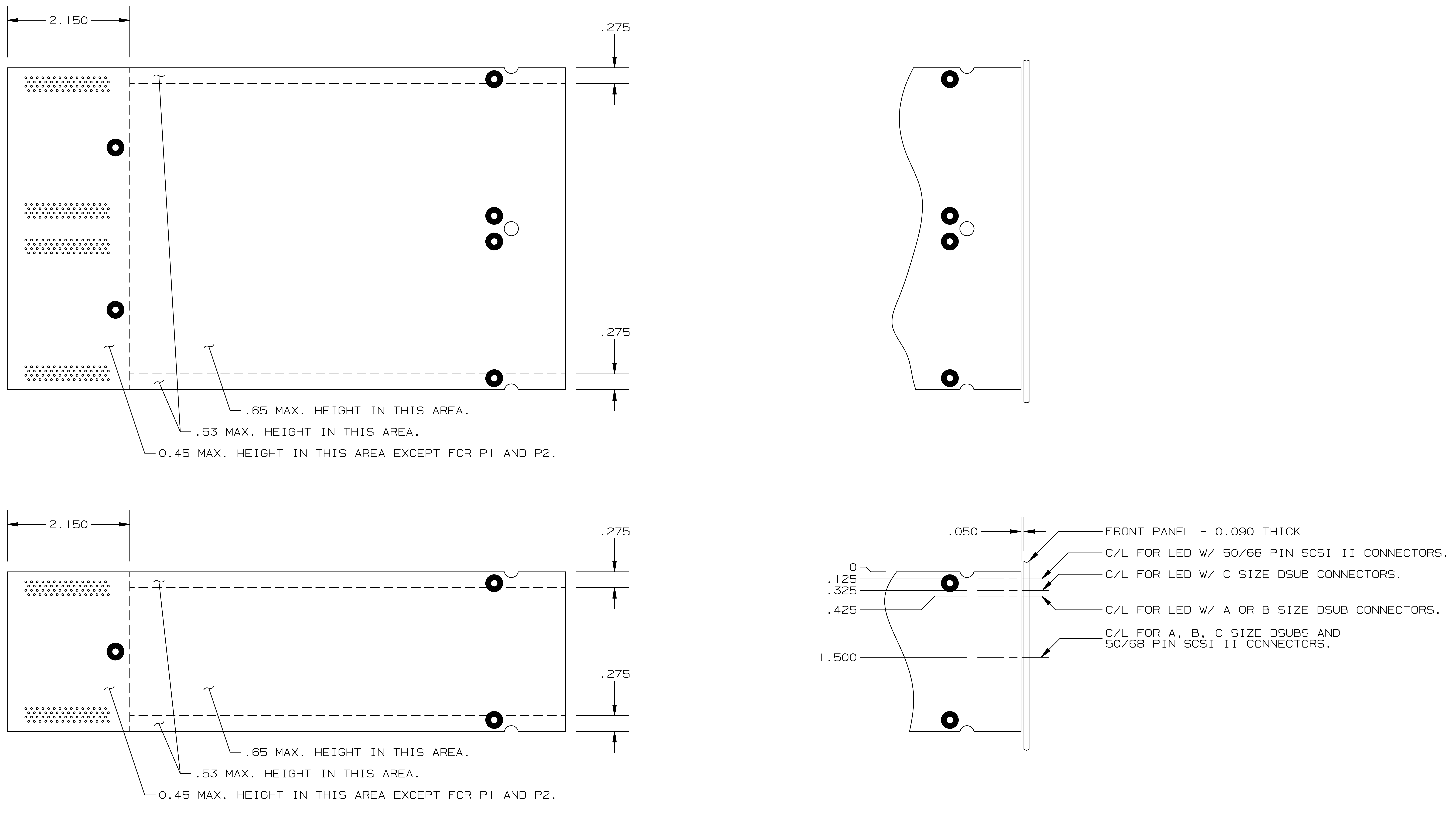
APPROVAL:		DATE
DRN	A.J.G.	01/29/98
CHK		
ENGR	A.J.G.	01/29/98
Q.A.		
CUST		

VXI Technology
17912 MITCHELL, IRVINE CA 92614 (714) 955-1894

TITLE
SPEC, MECHANICAL, VMIP DAUGHTER BOARD, SGL/DBL

SIZE	CAGE CODE	DWG NO	REV
D	03LB1	99-0011-000	B

SCALE 1:1 LAYER 10 SHEET 1 OF 2


2.150
.275
.275
.65 MAX. HEIGHT IN THIS AREA.
.53 MAX. HEIGHT IN THIS AREA.
0.45 MAX. HEIGHT IN THIS AREA EXCEPT FOR P1 AND P2.
2.150
.275
.275
.65 MAX. HEIGHT IN THIS AREA.
.53 MAX. HEIGHT IN THIS AREA.
0.45 MAX. HEIGHT IN THIS AREA EXCEPT FOR P1 AND P2.
.050
FRONT PANEL - 0.090 THICK
C/L FOR LED W/ 50/68 PIN SCSI II CONNECTORS.
C/L FOR LED W/ C SIZE DSUB CONNECTORS.
C/L FOR LED W/ A OR B SIZE DSUB CONNECTORS.
C/L FOR A, B, C SIZE DSUBS AND 50/68 PIN SCSI II CONNECTORS.
0
.125
.325
.425
1.500

VXI Technology 17912 MITCHELL, IRVINE CA 92614 (714) 955-1894			
TITLE SPEC, MECHANICAL, VMIP DAUGHTER BOARD, SGL/DBL			
SIZE D	CAGE CODE 03LB1	DWG NO 99-0011-000	REV B
SCALE 1:1		LAYER 10	SHEET 2